File No. S360-25
Order No. GC28-6817-4

OS

IBM Systems Reference Library

IBM System/360 Operating System

FORTRAN IV (G and H) Programmer's Guide

Program Numbers 360S-FO-500
360S-FO-520

This publication explains how to use the IBM System/360 Operating System to compile, linkage edit, and execute programs written in the IBM System/360 FORTRAN IV language. In addition, it contains information on program optimization, processing efficiency, extended error handling, and Assembler language subroutine linkage conventions. A section on programming factors of special interest to users of the IBM System/360 Models 91 and 195 is also included.

This publication is directed primarily to programmers familiar with the FORTRAN IV language. Previous knowledge of the IBM System/360 Operating System is not required.

Fifth Edition (September 1973)

This is a major revision of, and makes obsolete, the previous editions GC28-6817-2, -3, and Technical Newsletters GN28-0590 and GN28-0591.

All changes to the text, and small changes to illustrations, are indicated by a vertical line to the left of the change.

Changes are periodically made to the specifications herein. Before using this publication in connection with the operation of IBM systems, consult the latest IBM System/360 and System/370 Bibliography, Order No. GA22-6822 for the editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM Branch Office serving your locality.

Address comments concerning the contents of this publication to IBM Corporation, Programming Publications, 1271 Avenue of the Americas, New York, New York 10020. Comments become the property of IBM.

This publication is directed to programmers using either the IBM System/360 FORTRAN IV (G) or FORTRAN IV (H) compiler. It explains how to compile, linkage edit, and execute programs under control of the IBM System/360 Operating System. The FORTRAN IV language is described in the publication IBM System/360 and System/370 FORTRAN IV Language, Order No. GC28-6515, which is a corequisite to this publication.

Most of the information contained in this guide is common to both the FORTRAN IV (G) and FORTRAN IV (H) compilers. Where differences exist, they are clearly marked.

Paragraphs or sections applicable to the (G) compiler, but not the (H), are designated throughout this publication by the symbol:

```
┌───────┐
|G ONLY|
└───────┘
```

Conversely, paragraphs or sections applicable to the (H) compiler, but not the (G), are designated by the symbol:

```
┌───────┐
|H ONLY|
└───────┘
```

The programmer's guide is designed to provide programmers with information at three levels of complexity.

1. Programmers who will use the cataloged procedures as provided by IBM should read the "Introduction" and "Job Control Language" sections to understand the job control statements, the "FORTRAN Job Processing" section to understand the use of cataloged procedures, the "Programming Considerations" section to be able to use the FORTRAN language correctly and efficiently, and the "System Output" section to understand the listings, maps, and messages generated by the compiler, the linkage editor, and a load module.

2. Programmers who, in addition, are concerned with creating and retrieving data sets, optimizing the use of I/O devices, or temporarily modifying IBM-supplied cataloged procedures should read the entire programmer's guide.

3. Programmers who are concerned with making extensive use of the operating system facilities, such as writing their own cataloged procedures, modifying the FORTRAN library, or calculating region sizes for operating in an MVT environment, should also read the entire programmer's guide in conjunction with the following publications, as required:

IBM System/360 Operating System: Job Control Language Reference, Order No. GC28-6704

IBM System/360 Operating System: Job Control Language User's Guide, Order No. GC28-6703

IBM System/360 Operating System: Introduction, Order No. GC28-6534

IBM System/360 Operating System: System Programmer's Guide, Order No. GC28-6550

IBM System/360 Operating System: Supervisor and Data Management Services, Order No. GC28-6646

IBM System/360 Operating System: Supervisor and Data Management Macro Instructions, Order No. GC28-6647

IBM System/360 Operating System: Utilities, Order No. GC28-6586

IBM System/360: FORTRAN IV Library: Mathematical and Service Subprograms, Order No. GC28-6818

IBM System/360 Operating System: Linkage Editor and Loader, Order No. GC28-6538

IBM System/360 Operating System: System Generation, Order No. GC28-6554

IBM System/360 Operating System: Operator's Guide, Order No. GC28-6540

IBM System/360 Operating System: Messages and Codes, Order No. GC28-6608

IBM System/360 Operating System: Programmer's Guide to Debugging, Order No. GC28-6670

IBM System/360 Operating System: Storage Estimates, Order No. GC28-6551

This publication contains appendixes that provide the programmer with the following information:

- Descriptions and explanations of compiler invocation from a problem program.
- Examples of job processing.
- Descriptions and explanations for the preparation of subprograms written in assembler language for use with a main program written in FORTRAN.
- Descriptions of the diagnostic messages produced during compilation and load module execution.
- A list of American National Standard carriage control characters.
- A list of input/output unit types.
- A description of the FORTRAN IV (H) optimization features.
- A description of the FORTRAN IV (G) debug facility.
- A discussion of FORTRAN programming considerations for the user of the IBM System/360 Models 91 and 195.

For easier reading, the titles of publications referred to in this publication are abbreviated. For example, references to the publication IBM System/360 Operating System: Linkage Editor and Loader are abbreviated to Linkage Editor and Loader publication.

Summary of Amendments — Number 1

Date of Publication: September 1973

Order No. of Publication: Revision GC28-6817-4

New System Diagnostics

Maintenance: Programming and Documentation

Two new FORTRAN IV (G) compiler diagnostics messages (IEY045I and IEY046I) and one new FORTRAN IV (H) compiler diagnostic message (IEK790I) have been added. Furthermore, the text of FORTRAN IV library diagnostic message IHC240I has been revised.

Miscellaneous Changes

Maintenance: Documentation Only

SIZE Compiler Option

The description of the FORTRAN IV (H) compiler option SIZE has been expanded and clarified.

Linkage Editor Input

The discussion of primary input to the linkage editor now includes members of partitioned data sets as being acceptable.

Use of DD Statements for Direct Access Data Sets

Clarifications have been made to the discussion of direct access data set characteristics and the unique relationships between such specifications on DEFINE FILE and DD statements.

Label Information

The discussion of the LABEL subparameters IN and OUT have been revised; a reference to partitioned data set retrieval for read purposes has been added.

Tape Recording Density Information

Magnetic tape density default values are now listed for 7-track tape, 9-track tape without dual density, and 9-track tape with dual density.

Record Format Specifications

Specification of standard blocks (S) and the track overflow feature (T) are mutually exclusive for fixed length records. Examples of the subparameter RECFM have been corrected to indicate this fact.

Programming Considerations

Clarifications have been made to the following areas of discussion: conditional branching, use of embedded blanks in FORTRAN programs, DO loop optimization, data initialization, and compiler data set concatenation. A further entry has also been added to the list of FORTRAN IV (G) and FORTRAN IV (H) compiler restrictions.

Extended Error Handling Facility

Descriptions of the specifications inoal and inomes in the CALL ERRSET statement have been revised and expanded. Changes have also been made to the current list of option table default values.

Assembler Language Subprograms

The description of assembler language subprogram linkage conventions has been updated.

System Diagnostics

The description of FORTRAN IV (H) compiler informative messages has been updated to conform to the current version of compiler output.

Program Optimization Considerations

A further consideration has been added to the items already specified under "Programming Considerations Using the Optimizer."

Editorial changes that have no technical significance are not noted here.

Specific changes to the text made as of this publishing date are indicated by a vertical bar to the left of the text. These bars will be deleted at any subsequent republication of the page affected.

ILLUSTRATIONS

FIGURES

TABLES

The IBM System/360 Operating System consists of a control program and processing programs. The control program supervises execution of all processing programs, such as the FORTRAN compiler, and all problem programs, such as a FORTRAN program. Therefore, to execute a FORTRAN program, the programmer must first communicate with the operating system. The medium of communication between the programmer and the operating system is the job control language.

The programmer uses job control statements to define two units of work -- the job and the job step -- to the operating system and to define the files (data sets) used in these jobs and job steps. He defines a job to the operating system by using a JOB statement; a job step by using an EXEC statement; and a data set by using a DD statement.

JOB AND JOB STEP RELATIONSHIP

To the operating system, a job consists of executing one or more job steps. In the simplest case, a job consists of one job step. For example, executing a FORTRAN main program to invert a matrix is a job consisting of one job step.

In more complex cases, one job may consist of a series of job steps. For example, a programmer is given a tape containing raw data from a rocket firing: he must transform this raw data into a series of graphs and reports. Three steps may be defined:

1. Compare the raw data to projected data and eliminate errors which arise because of intermittent errors in gauges and transmission facilities.

2. Use the redefined data and a set of parameters as input to a set of equations, which develop values for the production of graphs and reports.

3. Use the values to plot the graphs and print the reports.

Figure 1 illustrates the rocket firing job with three job steps.

In the previous example, each step could be defined as a separate job with one job step in each job. However, designating related job steps as one job is more efficient: processing time is decreased because only one job is defined, and interdependence of job steps may be stated. (The interdependence of jobs cannot be stated.)

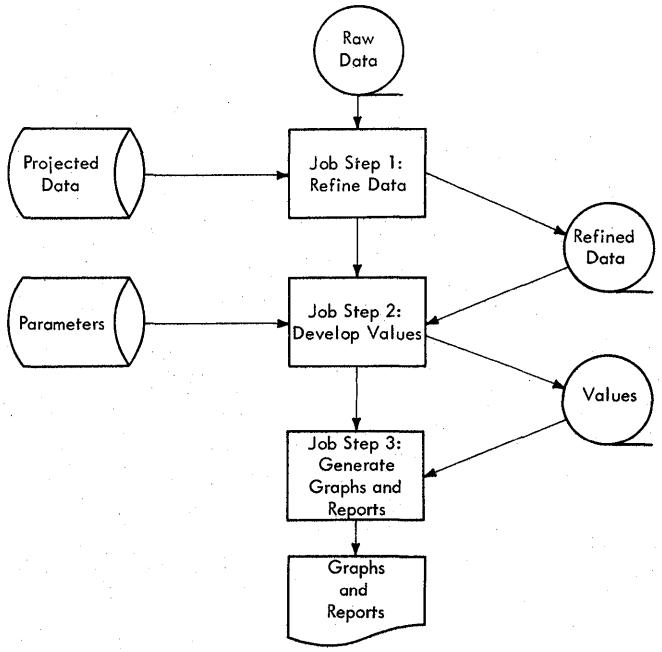


Figure 1. Rocket Firing Job

FORTRAN PROCESSING AND CATALOGED PROCEDURES

When a programmer writes a FORTRAN program, the objective is to obtain a problem solution. However, before the program can provide this solution, the program itself must undergo processing. The source program (source module) is compiled to give an object module; and the object module is linkage edited to give a load module.[1] This load module is then executed to give the desired problem solution.

If each of the three steps involved in processing a FORTRAN module is a job step in the same job, a set of job control statements that consists of one EXEC

---

[1]As an alternative, the object module may be edited and then automatically executed by the loader, another IBM-supplied program. Details on the use of the loader can be found in the section "Loader Processing."

statement and one or more DD statements is required for each step. Because writing these job control statements can be time-consuming work for the programmer, IBM supplies cataloged procedures to aid in the processing of FORTRAN modules. A cataloged procedure consists of a procedure step or a series of procedure steps. Each step contains the necessary set of job control statements to compile or to linkage edit or to execute a FORTRAN module. (Note: A JOB statement cannot be cataloged.)

For each compiler, IBM provides five cataloged procedures. The procedures and their uses are shown in Table 1.

Any of the cataloged procedures can be invoked by an EXEC statement in the input stream. In addition, each of the procedures can be temporarily modified by this EXEC statement and any DD statements in the input stream; this temporary modification is called overriding.

Table 1. Cataloged Procedure-Names and Functions

Procedure-Name		
FORTRAN G	FORTRAN H	Function
FORTGC	FORTHC	compile
FORTGCL	FORTHCL	compile and linkage edit
FORTGLG	FORTHLG	linkage edit and execute
FORTGCLG	FORTHCLG	compile, linkage edit, and execute
FORTGCLD	FORTHCLD	compile and load

DATA SETS

For FORTRAN processing, a programmer uses DD statements to define the particular data set(s) required for a compile, linkage edit, or execute step. In the operating system, a data set is a named, organized collection of one or more records that are logically related. For example, a data set may be a source module, a library of mathematical functions, or the data processed by a load module.

Data Set Organization

A data set in FORTRAN may be one of three types: sequential, partitioned or direct-access.

A sequential data set is one in which records are accessed solely on the basis of their successive physical positions. A sequential data set may reside on cards, tape, or disk. The compiler, linkage editor, and load modules process sequential data sets. The compiler uses the queued sequential access method (QSAM) for such processing, and load modules use the basic sequential access method (BSAM) for object time I/O operations. (For additional information on access methods, see the Supervisor and Data Management Services publication, Order No. GC28-6646.)

A partitioned data set (PDS) is composed of named, independent groups of sequential data and resides on a direct-access volume. A directory index resides in the PDS and directs the operating system to any group of sequential data. Each group of sequential data is called a member. Partitioned data sets are used for storage of any type of sequentially organized data. In particular, they are used for storage of source and load modules (each module is a member). In fact, a load module can be executed only if it is a member of a partitioned data set. A PDS of load modules is created by either the linkage editor or a utility program. A PDS is accessible to the linkage editor; however, only individual members of a PDS are accessible to the compiler. Members of a PDS are accessible to a FORTRAN load module; however, concurrent processing of two or more members of the same PDS is not supported. Sequential processing of two or more members is permitted if one member is closed before the other is processed. See the discussion "Partitioned Data Set Processing" for details on accessing partitioned data sets.

The FORTRAN library is a cataloged PDS that contains the library subprograms in the form of load modules. SYS1.FORTLIB is the name given to this PDS.

To process a member of a partitioned data set, the programmer must use the DD statement to provide information about the data set and the member. The programmer must specify (in the DSNAME parameter) both the name of the data set and of the member, and must indicate (in the LABEL parameter) if the member is to be created or retrieved. However, if the programmer requests the FORTRAN compiler to process a partitioned data set (for example, to compile a source deck stored as a member of

a partitioned data set) no LABEL information need be specified.

Note that the processing of a partitioned data set is limited to READ or WRITE operations only. The programmer is not permitted both to READ and WRITE the same data set in a single program.

A direct-access data set contains records that are read or written by specifying the position of the record within the data set. When the position of the record is indicated in a FIND, READ, or WRITE statement, the operating system goes directly to that position in the data set and either retrieves, reads, or writes the record. For example, with a sequential data set, if the 100th record is read or written, all records preceding the 100th record (records 1 through 99) must be transmitted before the 100th record can be transmitted. With a direct-access data set the 100th record can be transmitted directly by indicating in the I/O statement that the 100th record is to be transmitted. However, in a direct-access data set, records can be transmitted by FORTRAN direct-access I/O statements only; they cannot be transmitted by FORTRAN sequential I/O statements. Records in a direct-access data set can be transmitted sequentially by using the associated variable in direct-access I/O statements.

A direct-access data set must reside on a direct-access volume. Direct-access data sets are processed by FORTRAN load modules; the compiler and linkage editor cannot process direct-access data sets. Load modules process data sets of this type with the basic direct-access method (BDAM).

Data Set Labels

Data sets that reside on direct-access volumes have standard labels only; data sets that reside on magnetic tape volumes can have standard labels or no labels. Information, such as a data set identifier, volume sequence number, record format, density, etc., is stored in the data set labels. The information required in a DD statement used to retrieve a labeled data set is substantially less than in one used to retrieve an unlabeled data set.

Data Set Cataloging

To relieve the programmer of the burden of remembering the volume on which a particular data set resides, the operating system provides a cataloging facility. When a data set is cataloged, the serial number of its volume is associated in the catalog with the data set name. A programmer can refer to this data set without specifying its physical location. Any data set residing on a direct-access or magnetic tape volume can be cataloged.

Sequential, partitioned, and direct-access data sets can be cataloged; however, an individual member of a PDS cannot be cataloged because a member is not a data set.

The FORTRAN programmer uses the job control statements shown in Table 2 in compiling, linkage editing, and executing programs.

JOB MANAGEMENT

Job control statements are processed by a group of operating system routines known collectively as job management. Job management routines interpret control statements, control the flow of jobs, and issue messages to both the operator and the programmer. Job management has two major components: a job scheduler and a master scheduler.

Table 2. Job Control Statements

Statement	Function
JOB	Indicates the beginning of a new job and describes that job
EXEC	Indicates a job step and describes that job step; indicates the cataloged procedure or load module to be executed
DD	Describes data sets, and controls device and volume assignment
delimiter	Separates data sets in the input stream from control statements; it appears after each data set in the input stream
comment	Contains miscellaneous remarks, annotations, etc., written by the programmer; it can appear before or after any control statement.

The specific facilities available through the job scheduler and the master scheduler depend on the scheduling level the installation selects during system generation. Schedulers are available at two levels -- the sequential scheduler and the priority scheduler.

Sequential schedulers process job steps one at a time in the order of their appearance in the input stream. Operating systems with a primary control program (PCP) use sequential schedulers.

Priority schedulers are used by systems that provide multiprogramming with a fixed number of tasks (MFT) or multiprogramming with a variable number of tasks (MVT). Priority schedulers process complete jobs according to their relative priority within job classes. Priority schedulers can accept input data from more than one input stream.

CODING JOB CONTROL STATEMENTS

Except for the comment statement, control statements contain two identifying characters (// or /*) in card columns 1 and 2. The comment statement is identified by the initial characters //* in card columns 1, 2, and 3. Control statements may contain four fields -- name, operation, operand, and comments (see Figure 2).

NAME FIELD

The name field contains between one and eight alphameric characters, the first of which must be alphabetic or the characters #, &, or $. The name field begins in card column 3 and is followed by one or more

FORMAT	APPLICABLE CONTROL STATEMENTS
//Name Operation Operand [Comment]	JOB,EXEC,DD
// Operation Operand [Comment]	EXEC,DD
/* [Comment]	delimiter
//*[Comment]	comment

Figure 2. Job Control Statement Formats

blanks to separate it from the operation field. The name field is used:

1. To identify the control statement to the operating system.
2. To enable other control statements in the job to refer to information contained in the named statement.
3. To relate DD statements to I/O statements in the load module.

OPERATION FIELD

The operation field contains one of the following operation codes:

JOB
EXEC
DD

or, if the statement is a delimiter or comment statement, the operation field is blank. The operation code is preceded and followed by one or more blanks.

OPERAND FIELD

The operand field contains the parameters that provide required and optional information to the operating system. Parameters are separated by commas, and the operand field is ended by placing one or more blanks after the last parameter. There are two types of parameters, positional and keyword.

Positional Parameters: Positional parameters are placed first in the operand field and must appear in the specified order. If a positional parameter is omitted and other positional parameters follow, the omission must be indicated by a comma.

Keyword Parameters: Keyword parameters follow positional parameters in the operand field. (If no positional parameters appear, a keyword parameter can appear first in the operand field; no leading comma is required.) Keyword parameters may appear in any order. If a keyword parameter is omitted, a comma is not required to indicate the omission.

Subparameters: Subparameters are either positional or keyword and are noted as such in the definition of control statements.

Positional subparameters appear first in a parameter and must appear in the specified order. If a positional subparameter is omitted and other positional subparameters follow, the omission must be indicated by a comma.

Keyword subparameters follow positional subparameters in a parameter. (If no positional subparameters appear, a keyword subparameter can appear first in the parameter; no leading comma is required.) Keyword subparameters may appear in any order. If a keyword subparameter is omitted, a comma is not required to indicate the omission.

COMMENTS FIELD

The comments field can contain any information considered helpful by the programmer. Comments, except for those on a comment statement, must be separated from the operand field (or the * in a delimeter statement) by at least one blank; they may appear in the remaining columns up to and including column 71.

Continuing Control Statements

Except for the comment statement, which is contained in columns 1 through 80, control statements are contained in columns 1 through 71 of cards or card images. If the total length of a statement exceeds 71 columns, or if parameters are to be placed on separate cards, operating system continuation conventions must be followed. To continue an operand field:

1. Interrupt the field after a complete parameter or subparameter (including the comma that follows it) at or before column 71.
2. Include comments, if desired, by following the interrupted field with at least one blank.
3. Optionally, code any nonblank character in column 72. If a character is not coded in column 72, the job scheduler treats the next statement as a continuation statement if the conventions outlined in points 4 and 5 are followed.
4. Code the identifying characters // in columns 1 and 2 of the following card or card image.
5. Continue the interrupted operand beginning in any column from 4 through 16.

Note: Excessive continuation cards should be avoided whenever possible to reduce processing time for the control program.

Comments can be continued onto additional cards after the operand has been completed. To continue a comments field:

1. Interrupt the comment at a convenient place.
2. Code a nonblank character in column 72.
3. Code the identifying characters // in columns 1 and 2 of the following card or card image.
4. Continue the comments field beginning in any column after column 3.

Note: The comment statement cannot be continued.

NOTATION FOR DEFINING CONTROL STATEMENTS

The notation used in this publication to define control statements is described in the following paragraphs.

1. The set of symbols listed below are used to define control statements, but are never written in an actual statement.

a.	hyphen	-
b.	or	|
c.	underscore	_
d.	braces	{ }
e.	brackets	[ ]
f.	ellipsis	...
g.	superscript	i

   The special uses of these symbols are explained in paragraphs 4-10.

2. Uppercase letters and words, numbers, and the set of symbols listed below are written in an actual control statement exactly as shown in the statement definition. (Any exceptions to this rule are noted in the definition of a control statement.)

a.	apostrophe	'
b.	asterisk	*
c.	comma	,
d.	equal sign	=
e.	parentheses	( )
f.	period	.
g.	slash	/

3. Lowercase letters, words, and symbols appearing in a control statement definition represent variables for which specific information is substituted in the actual statement.

   Example: If name appears in a statement definition, a specific value (e.g., ALPHA) is substituted for the variable in the actual statement.

4. Hyphens join lowercase letters, words, and symbols to form a single variable.

   Example: If member-name appears in a statement definition, a specific value (e.g., BETA) is substituted for the variable in the actual statement.

5. Stacked items or items separated from each other by the "or" symbol represent alternatives. Only one such alternative should be selected.

   Example: The two representations

   ```
   A
   B  and A|B|C
   C
   ```

   have the same meaning and indicate that either A or B or C should be selected.

6. An underscore indicates a default option. If an underscored alternative is selected, it need not be written in the actual statement.

   Example: The two representations

   ```
   A
   B  and A|B|C
   C
   ```

   have the same meaning and indicate that either A or B or C should be selected; however, if B is selected, it need not be written, because it is the default option.

7. Braces group related items, such as alternatives.

   Example: ALPHA={{A|B|C},D)

   indicates that a choice should be made among the items enclosed within the braces. If A is selected, the result is ALPHA=(A,D). If C is selected, the result can be either ALPHA=(,D) or ALPHA=(C,D).

8. Brackets also group related items; however, everything within the brackets is optional and may be omitted.

Example: ALPHA=([A|B|C],D)

indicates that a choice can be made among the items enclosed within the brackets or that the items within the brackets can be omitted. If B is selected, the result is ALPHA=(B,D). If no choice is made, the result is ALPHA=(,D).

9. An ellipsis indicates that the preceding item or group of items can be repeated more than once in succession.

   Example: ALPHA[,BETA]...

   indicates that ALPHA can appear alone or can be followed by ,BETA optionally repeated any number of times in succession.

10. A superscript refers to a description in a footnote.

    Example: $\left\{\begin{matrix}\text{NEW}\\\text{OLD}\\\text{MOD}\\\text{SHR}\end{matrix}\right\}^{1}$

    indicates that additional information concerning the grouped items is contained in footnote number 1.

11. Blanks are used to improve the readability of control statement definitions. Unless otherwise noted, blanks have no meaning in a statement definition.

JOB STATEMENT

The JOB statement (Figure 3) is the first statement in the sequence of control statements that describe a job. The JOB statement contains the following information:

1. Name of the job.
2. Accounting information relative to the job.
3. Programmer's name.
4. Whether the job control statements are printed for the programmer.
5. Conditions for terminating the execution of the job.
6. A job priority assignment.
7. Output class for priority scheduler messages.
8. Specification of main storage requirements for a job.
9. Specification of the maximum amount of time to be allotted for a job.

Examples of the JOB statement are shown in Figure 4.

NAME FIELD

The "jobname" must always be specified; it identifies the job to the operating system. No two jobs being handled concurrently by a priority scheduler should have the same "jobname".

OPERAND FIELD

Job Accounting Information

The first positional parameter can contain the installation account number and any parameters passed to the installation accounting routines. These routines are written by the installation and inserted in the operating system when it is generated. The format of the accounting information is specified by the installation.

As a system generation option with sequential schedulers, the account number can be established as a required parameter. With priority schedulers, the requirement can be established with a cataloged procedure for the input reader. (Information on the cataloged procedure for the input reader and on how to write an accounting routine may be found in the System Programmer's Guide, Order No. GC28-6550.) Otherwise, the account number is optional.

Programmer's Name

The "programmer name" is the second positional parameter. If no job accounting information is supplied, its absence must be indicated by a comma preceding the programmer's name. If neither job accounting information nor programmer's name is present, commas need not be used to indicate their absence.

This parameter is optional unless it is made mandatory at the installation in the same way as job accounting information is made mandatory.

Name	Operation	Operand
		Positional Parameters
//jobname	JOB	[([account-number][,accounting-information])[1] [2] [3]]
		[,programmer-name][4] [5] [6]
		Keyword Parameters
		[MSGLEVEL=(x,y)][7]
		[COND=((code,operator)[,(code,operator)]...[8])[9]]
		[PRTY=nn][10]
		[MSGCLASS=x][10]
		[REGION=({nnnnnK / value$_0$K} [,value$_1$K])][11]
		[TIME=(minutes, seconds)][10]

[1]If the information specified ("account-number" and/or "accounting-information") contains blanks, or any special characters other than hyphens, it must be delimited by apostrophes instead of parentheses.
[2]If only "account-number" is specified, the delimiting parentheses may be omitted.
[3]The maximum number of characters allowed between the delimiting parentheses or apostrophes is 142.
[4]If "programmer-name" contains blanks, or any special characters other than periods, it must be enclosed within apostrophes.
[5]When an apostrophe is contained within "programmer-name", the apostrophe must be shown as two consecutive apostrophes.
[6]The maximum number of characters allowed for "programmer-name" is 20.
[7]The symbol x represents a job control language message code and may be specified as 0, 1, or 2; y represents a job scheduler allocation message code and may be specified as 0 or 1.
[8]The maximum number of repetitions allowed is 7.
[9]If only one test is specified, the outer pair of parentheses may be omitted.
[10]This parameter is used with priority schedulers only. The sequential scheduler ignores it.
[11]This parameter is used with MVT priority schedulers only.

Figure 3. JOB Statement

Sample Coding Form

1-10 | 11-20 | 21-30 | 31-40 | 41-50 | 51-60 | 61-70 | 71-80

```
Example 1
//PROGRAM JOB (215,819,46W),'J.SMITH',COND=(7,LT),MSGLEVEL=1
Example 2
//PROG2 JOB 1087F-21,COND=(7,LT),PRTY=10,REGION=100K
```

Figure 4. Sample JOB Statements

Statement, Allocation, and Termination Messages

The MSGLEVEL parameter indicates the type of messages the programmer wishes to receive from the control program.

MSGLEVEL=(x,y)

The letter x represents a job control language message code. The value of x may be 0, 1, or 2. When x=0, only the JOB statement, control statement errors, and diagnostics appear on SYSOUT. When x=1, input statements, cataloged procedure statements, and symbolic substitutions of parameters appear. When x=2, only input statements appear.

The letter y represents an allocation message code. The value of y may be 0 or 1. When y=0, no allocation or termination messages appear, if the program completes execution. In the event of an abnormal termination, only termination messages appear. When y=1, allocation, termination, and recovery messages all appear.

If MSGLEVEL is omitted, the default values assigned are those established at system generation time for PCP or from the reader procedure in a multiprogramming environment.

Conditions for Terminating a Job

At the completion of a job step, a code is issued indicating the outcome of that job step. This generated code is tested against the conditions stated in control statements.

The COND parameter of the JOB statement specifies conditions under which a job is terminated. Up to eight different tests, each consisting of a code and an operator, may be specified to the right of the equal sign. The code may be any number between 0 and 4095. The operator indicates the mathematical relationship between the code placed in the JOB statement and the codes issued by completed job steps. If the relationship is true, the job is terminated. The six operators and their meanings are:

Operator	Meaning
GT	greater than
GE	greater than or equal to
EQ	equal to
NE	not equal to
LT	less than
LE	less than or equal to

For example, if a code 8 is returned by the compiler and the JOB statement contains:

COND=(7,LT)

the job is terminated.

If more than one condition is indicated in the COND parameter and any condition is satisfied, the job is terminated.

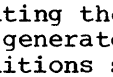
For the FORT step of both the FORTGCLG and FORTHCLG cataloged procedures, the compilers issue one of the following error codes:

0 - No errors or warnings detected.

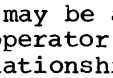
4 - Possible errors (warnings) detected, execution should be successful.

8 - Errors detected, execution may fail. Compilation continues but the linkage editor job step is not executed unless the programmer has increased the error code acceptable to the linkage editor. (The discussion "Condition for Bypassing a Job Step" later in this section describes the method for specifying the acceptable error code.)

If the LOAD option has been specified, an object module will be supplied.



If the error is found in an executable statement, the statement is replaced by a call to the IBERH routine (IHCIBERH). If the resulting load module is executed, IBERH is called and execution is terminated.



12 - Severe errors detected, execution is impossible.

16 - Terminal errors detected, compiler terminated abnormally.

Assigning Job Priority (PRTY)
(Used with Priority Schedulers Only)

To assign a priority other than the default job priority (as established in the input reader procedure), the parameter PRTY=nn must be coded in the operand field of the JOB statement. The "nn" is to be replaced with a decimal number from 0 through 13 (the highest priority that can be assigned is 13).

Whenever possible, avoid using priority 13. This is used by the system to expedite processing of jobs in which certain errors were diagnosed. It is also intended for other special uses by future features of systems with priority schedulers.

If the PRTY parameter is omitted, the default job priority is assumed.

Requesting a Message Class (MSGCLASS)
(Used with Priority Schedulers Only)

With a quantity and diversity of data in the output stream, an installation may want to separate different types of output data into different classes. Each class is directed to an output writer associated with a specific output unit.

The MSGCLASS=x parameter allows the messages issued by the priority scheduler to be routed to an output class other than the normal message class, A. The "x" is to be replaced with an alphabetic or numeric character. An output writer, assigned to process this class, transfers the data to a specific device.

If the MSGCLASS parameter is omitted, the job scheduler messages are routed to the standard output class, A.

Specifying Main Storage Requirements for a Job (REGION)
(Systems with MVT Only)

The REGION parameter is used to specify:

- The maximum amount of main storage to be allocated to the job. This figure must include the size of those components that are required by the user's program and that are not resident in main storage.
- The amount of main storage to be allocated to the job, and in which storage hierarchy or hierarchies the space is to be allocated. This request should be made only if main storage hierarchy support is specified during system generation.

To request the maximum amount of main storage required by the job, REGION=nnnnnK is coded in the operand field of the JOB statement. The term nnnnn is replaced with the number of contiguous 1024-byte areas to be allocated to the job, e.g., REGION=52K. This number can range from one to five digits, but cannot exceed 16383. It should be specified as an even number. (If an odd number is specified, the system treats it as the next highest even number.)

If the REGION parameter is omitted or if a region size smaller than the default region size is requested, the default value (as established in the input reader procedure) is assumed.

Note: If different region sizes are to be specified for each step in the job code, the REGION parameter must be specified in the EXEC statement associated with each step, as described in the section "EXEC Statement."

Main storage hierarchy support provides for storage hierarchies 0 and 1. If IBM 2361 Core Storage, Model 1 or 2, is present in the system, processor storage is referred to as hierarchy 0 and 2361 Core Storage is referred to as hierarchy 1. If 2361 Core Storage is not present but main storage hierarchy support was specified during system generation, a 2-part region is established in processor storage when a region is defined to exist in two hierarchies. The two parts are not necessarily contiguous.

When main storage hierarchy support is included in the system, the REGION parameter can be used to request both the maximum amount of storage to be allocated to the job and the hierarchy or hierarchies in which the storage is to be allocated.

To specify a region size and the hierarchy desired, REGION=($value_0$K,$value_1$K) is coded in the operand field of the JOB statement. The term "$value_0$" is replaced with the number of contiguous 1024-byte areas to be allocated to the job in hierarchy 0; the term "$value_1$" is replaced with the number of contiguous 1024-byte areas to be allocated in hierarchy 1, e.g., REGION= (60K,200K). When processor storage includes hierarchies 0 and 1, the combined values of $value_0$ and $value_1$ cannot exceed 16383. If 2361 Core Storage is present, value cannot exceed 16383, and $value_1$ cannot exceed 1024, if using a single Model 1, or 2048, if using a single Model 2. Each value specified should be an even number. (If an odd number is specified, the system treats it as the next highest even number.)

In systems with main storage hierarchy support, either subparameter can be omitted to request storage in only one hierarchy.

If storage is requested only in hierarchy 1, a comma must be coded to indicate the absence of the first subparameter, e.g., REGION=(,52K). If storage is requested only in hierarchy 0, the parentheses need not be coded, e.g., REGION=70K.

If the REGION parameter is omitted, or if a region size smaller than the default region size is requested, the default value (as established in the input reader procedure) is assumed. When the default region size is assumed, storage is always allocated in hierarchy 0.

Notes:

- If different region sizes are to be specified for each step in the job, code the REGION parameter in the EXEC statement associated with each step, as described in the section "EXEC Statement."
- If main storage hierarchy support is not included and regions are requested in both hierarchies, the region sizes are combined and an attempt is made to allocate a single region from processor storage. If a region is requested entirely from hierarchy 1, an attempt is made to allocate the region from processor storage.
- For information on storage requirements to be considered when specifying a region size, see the Storage Estimates publication.

Setting a Job Time Limit (TIME)

(Used by Priority Schedulers Only)

To limit the computing time used by a single job, a maximum time for its completion can be assigned. Such an assignment is useful in a multiprogramming environment where more than one job has access to the computing system.

The time is coded in minutes and seconds. The number of minutes cannot exceed 1439. The number of seconds cannot exceed 59. If the job is not completed in the assigned time, it is terminated. If the job execution time is expected to exceed 1439 minutes (24 hours), TIME=1440 can be coded to eliminate job timing. If the TIME parameter is omitted, the default job time limit (as established in the cataloged procedure for the reader/interpreter) is assumed.

EXEC STATEMENT

The EXEC statement (Figure 5) indicates the beginning of a job step and describes that job step. The statement can contain the following information:

1. Name of job step or procedure step.
2. Name of the cataloged procedure or load module to be executed.
3. Compiler and/or linkage editor options passed to the job step.
4. Accounting information relative to this job step.
5. Conditions for bypassing the execution of this job step.
6. A time limit for the job step or an entire cataloged procedure.
7. Specification of main storage requirements for a job step or an entire cataloged procedure.

Name	Operation	Operand
//[stepname][1]	EXEC	Positional Parameter {PROC=cataloged-procedure-name cataloged-procedure-name PGM=program-name PGM=*.stepname.ddname PGM=*.stepname.procstep.ddname} Keyword Parameters [{PARM PARM.procstep[2]}=(option[,option]...)[3 4 5]] [{ACCT ACCT.procstep[2]}=(accounting-information)[3 6 7]] [{COND COND.procstep[2]}=((code,operator[,stepname[.procstep]]) [,(code,operator[,stepname[.procstep]])]...[8])[9]] [{TIME TIME.procstep[2]}=(minutes,seconds)][10 11] [{REGION REGION.procstep[2]}=({nnnnnK value$_0$K} [,value$_1$K])] [10 12] [{DPRTY DPRTY.procstep[2]}=(value 1[,value 2])][13]

[1]If information from this control statement is referred to in a later job step, "stepname" is required.
[2]If this format is selected, it may be repeated in the EXEC statement, once for each step in the cataloged procedure.
[3]If the information specified contains blanks, parentheses, or equal signs, either the keyword subparameter must be enclosed by apostrophes or the entire PARM field must be delimited by apostrophes instead of parentheses.
[4]If only one option is specified and it does not contain any blanks, parentheses, or equal signs, the delimiting parentheses may be omitted.
[5]The maximum number of characters allowed between delimiting parentheses is 100. If the option list is enclosed in apostrophes, however, the parameter must be coded on one card.
[6]If "accounting-information" does not contain commas, blanks, parentheses, or equal signs, the delimiting parentheses may be omitted.
[7]The maximum number of characters allowed between the delimiting apostrophes or parentheses is 142.
[8]The maximum number of repetitions allowed is 7.
[9]If only one test is specified, the outer pair of parentheses may be omitted.
[10]This parameter is used with priority schedulers only. Sequential schedulers ignore it.
[11]If only minutes are given, the parentheses need not be used. If only seconds are given, the parentheses must be used and a comma must precede the seconds.
[12]If only value$_0$K is given, the parentheses need not be used. If only value$_1$K is given, the parentheses must be used and a comma must precede value$_1$K.
[13]If only value 1 is given, the parentheses need not be used. If only value 2 is given, the parentheses must be used and a comma must precede value 2.

Figure 5. EXEC Statement

Example 1 of Figure 6 shows EXEC statements used to execute programs. The program names used are the (G) and (H) compiler names. Example 2 in Figure 6 shows, for each compiler, an EXEC statement used to execute a cataloged procedure.

NAME FIELD

The "stepname" is the name of the job step or procedure step. It is required when information from this job step is referred to in a later job step. No two steps in the same job should have the same "stepname."

OPERAND FIELD

Positional Parameter

The first parameter of an EXEC statement must specify either the name of the cataloged procedure or program to be executed. Each program (load module) to be executed must be a member of a library (PDS). The library can be the system library (SYS1.LINKLIB), a private library, or a temporary library created to store a program from a previous job step of the same job.

Specifying a Cataloged Procedure:

{PROC=cataloged-procedure-name}
{cataloged-procedure-name }

indicate that a cataloged procedure is invoked. The "cataloged procedure-name" is the name of the cataloged procedure. For example,

```
// EXEC PROC=FORTHC
      or
// EXEC FORTHC
```

indicates that the FORTRAN IV (H) cataloged procedure FORTHC is to be executed.

Specifying a Program in a Library:

PGM=program-name

indicates that a program is executed. The "program name" is the member name of a load module in the system library (SYS1.LINKLIB) or private library. For example,

```
// EXEC PGM=IEWL
```

indicates that the load module IEWL is executed. (A load module in a private

Sample Coding Form

```
1-10      11-20     21-30     31-40     41-50     51-60     61-70     71-80
Example 1 FORTRAN IV (G) COMPILER
// EXEC PGM=IEYFORT,ACCT=(896,427),COND=(7,LT),TIME=200,REGION=1024K
Example 1 FORTRAN IV (H) COMPILER
// EXEC PGM=IEKAA00,ACCT=(896,427),COND=(7,LT),TIME=200,REGION=228K
Example 2 FORTRAN IV (G) COMPILER
//STEP4 EXEC FORTGCLG,                                                  1
//          PARM.FORT='DECK,MAP,LIST',                                  2
//          PARM.LKED=XREF,                                             3
//          COND.LKED=(7,LT,STEP4.FORT),                                4
//          COND.GO=((7,LT,STEP4.LKED),(7,LT,STEP4.FORT)),              5
//          ACCT=1024LA
Example 2 FORTRAN IV (H) COMPILER
//STEP4 EXEC FORTHCLG,                                                  1
//          PARM.FORT='DECK,LINECNT=80,MAP,ID,EDIT',                    2
//          PARM.LKED=XREF,                                             3
//          COND.LKED=(7,LT,STEP4.FORT),                                4
//          COND.GO=((7,LT,STEP4.LKED),(7,LT,STEP4.FORT)),              5
//          ACCT=1024LA
```

Figure 6. Sample EXEC Statements

library is executed by concatenating that private library with the system library through the use of a JOBLIB DD statement. See the discussion concerning JOBLIB under "Data Definition (DD) Statement" in this section.)

Specifying a Program Described in a Previous Job Step:

PGM=*.stepname.ddname
indicates that the name of the program to be executed is taken from a DD statement of a previous job step. The * indicates the current job; "stepname" is the name of a previous step within the current job; and "ddname" is the name of a DD statement within that previous job step. (The "stepname" cannot refer to a job step in another job.) The program referred to must be a member of a PDS. For example, in the following statements, statement STEP5 indicates that the name of the program is taken from the DD statement SYSLMOD in job step STEP4. Consequently, the load module ARCTAN in the PDS MATH is executed.

```
//MCLX JOB ,JOHNSMITH,COND=(7,LT)
              .
              .
              .
//STEP4 EXEC PGM=IEWL
//SYSLMOD DD DSNAME=MATH(ARCTAN)
              .
              .
              .
//STEP5 EXEC PGM=*.STEP4.SYSLMOD
```

Specifying a Program Described in a Cataloged Procedure:

PGM=*.stepname.procstep.ddname
indicates that the name of the program to be executed is taken from a DD statement of a previously executed step of a cataloged procedure. The * indicates the current job; "stepname" is the name of the job step that invoked the cataloged procedure; "procstep" is the name of a step within the procedure; "ddname" is the name of a DD statement within the procedure step. (The "stepname" cannot refer to a job step in another job.) For example, consider a cataloged procedure FORT,

```
//COMPIL   EXEC PGM=IEKAA00
//SYSPUNCH DD   UNIT=SYSCP
//SYSPRINT DD   SYSOUT=A
//SYSLIN   DD   DSNAME=LINKINP
              .
              .
              .
//LKED     EXEC PGM=IEWL
//SYSLMOD  DD   DSNAME=RESULT(ANS)
              .
              .
              .
```

Furthermore, assume the following statements are placed in the input stream.

```
//XLIV     JOB  ,SMITH,COND=(7,LT)
//S1       EXEC PROC=FORT
              .
              .
              .
//S2       EXEC PGM=*.S1.LKED.SYSLMOD
//FT03F001 DD   UNIT=PRINTER
//FT01F001 DD   UNIT=INPUT
```

Statement S2 indicates that the name of the program is taken from the DD statement SYSLMOD. The statement is located in the procedure step LKED of the cataloged procedure FORT, which was invoked by statement S1. Consequently, the load module ANS in the PDS RESULT is executed.

Keyword Parameters

The keyword parameters may refer to a program, to an entire cataloged procedure, or to a step within a cataloged procedure.

Options for the Compiler and Linkage Editor:

The PARM parameter is used to pass options to the compiler or linkage editor. (PARM has no meaning to a FORTRAN load module.)

PARM
passes options to the compiler or linkage editor, when either is invoked by the PGM parameter in an EXEC statement, or to the first step in a cataloged procedure.

PARM.procstep
passes options to a compiler or linkage editor step within the named cataloged procedure step.

The formats for compiler options and those linkage editor options most applicable to the FORTRAN programmer are shown in Figure 7.

Note: If a subparameter expression in the list of the PARM parameter contains special characters, either of two methods may be used to delimit the expression:

1. Enclose the entire subparameter list in apostrophes. For example:

   PARM = 'LIST,MAP,NAME=MYMAIN,DECK'

2. Enclose the subparameter expression in apostrophes and the entire subparameter list in parentheses. Thus, the above example can be coded as:

   PARM = (LIST,MAP,'NAME=MYMAIN',DECK)

Since a list enclosed in apostrophes cannot be continued onto another control statement, the second method should be used when the PARM parameter must be interrupted.

Detailed information concerning compiler and linkage editor options is given in the section "FORTRAN Job Processing."

Condition for Bypassing a Job Step:

This COND parameter (unlike the one in the JOB statement) determines if the job step defined by the EXEC statement is bypassed.

COND
: states conditions for bypassing the execution of a program or an entire cataloged procedure.

COND.procstep
: states conditions for bypassing the execution of a specific cataloged procedure step "procstep".

The subparameters for the COND parameter are of the form:

(code,operator[,stepname])

The subparameters "code" and "operator" are the same as the code and operator described for the COND parameter in the JOB statement. The subparameter "stepname" identifies the previous job step that issued the code. For example, the COND parameter

COND=((5,LT,FORT),(5,LT,LKED))

indicates that the step in which the COND parameter appears is bypassed if 5 is less than the code returned by either of the steps FORT or LKED.

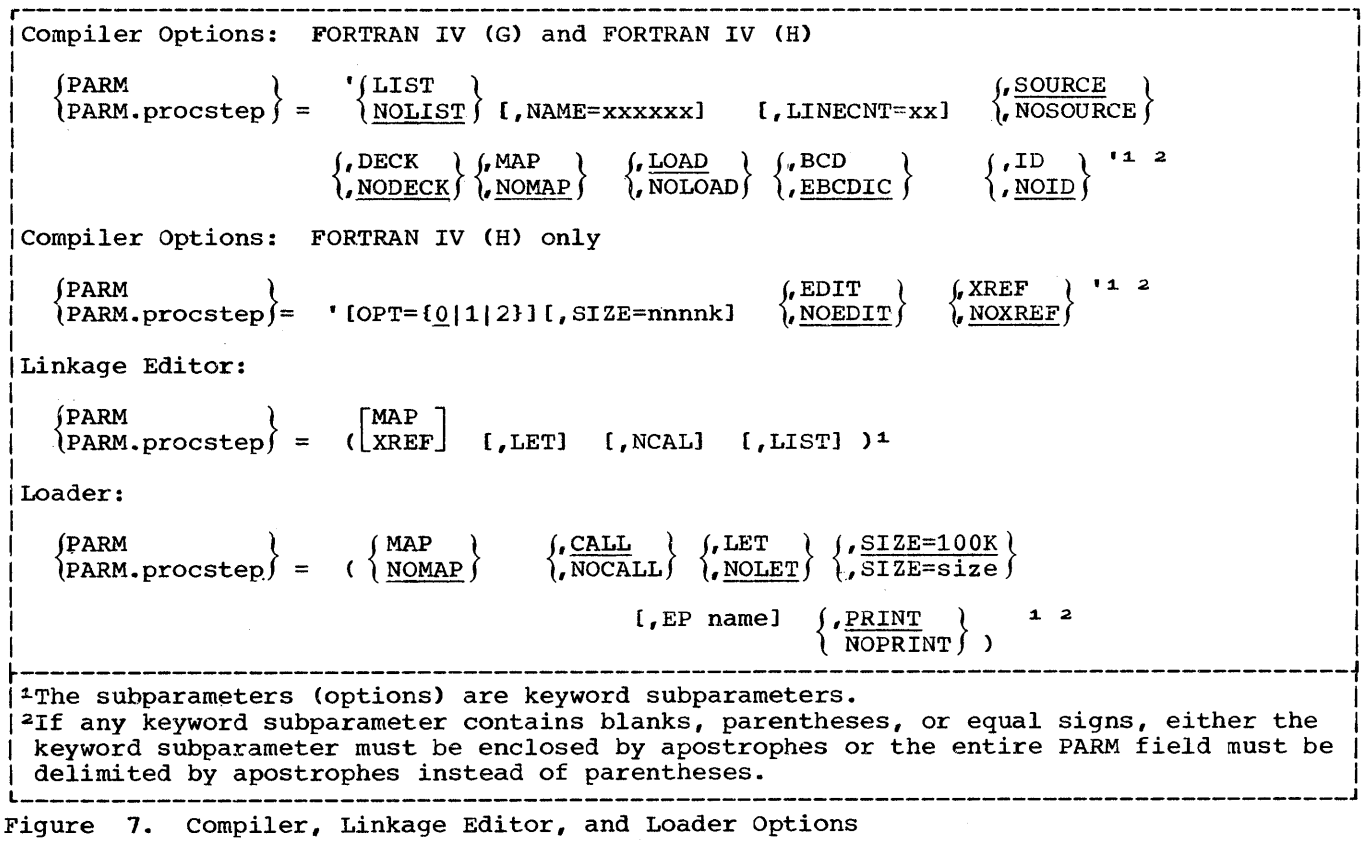


Figure 7. Compiler, Linkage Editor, and Loader Options

If a step in a cataloged procedure issued the code, "stepname" must qualify the name of the procedure step; that is,

(code,operator[,stepname.procstep])

If "stepname" is not given, "code" is compared with all codes issued by previous job steps.

Accounting Information:

The ACCT parameter specifies accounting information for a job step within a job.

ACCT
is used to pass accounting information to the installation accounting routines for this job step.

ACCT.procstep
is used to pass accounting information for a step within a cataloged procedure.

If both the JOB and EXEC statements contain accounting information, the installation accounting routines decide how the accounting information shall be used for the job step.

Setting Job Step Time Limits (TIME):
(Used with MVT Priority Schedulers Only)

To limit the computing time used by a single job step or cataloged procedure, a maximum time for its completion can be assigned. If the job step is not completed in this time, the entire job is terminated. Assignment of such a time limit is particularly useful in a multiprogramming environment where more than one job has access to the computing system.

The time is coded in minutes and seconds. The number of minutes cannot exceed 1439 (24 hours); the number of seconds cannot exceed 59. (If the job step execution time is expected to exceed 1439 minutes, TIME=1440 can be coded to eliminate job step timing.)

If the TIME parameter is omitted, the default job step time limit (as established in the cataloged procedure for the input reader) is assumed.

TIME
is used to assign a time limit for a job step or for an entire cataloged procedure. For a cataloged procedure, this parameter overrides all TIME parameters that may have been specified in the procedure.

TIME.procstep
is used to assign a time limit for a single step of a cataloged procedure. This parameter overrides, for the named step, any TIME parameter which is present. One parameter of this form can be written for each step in the procedure.

Specifying Main Storage Requirements for a Job Step (REGION)
(Systems with MVT Only)

The REGION parameter is used to specify:

- The maximum amount of main storage to be allocated to the job step. This figure must include the size of those components that are required by the user's program and that are not resident in main storage.
- The amount of main storage to be allocated to the job, step and in which storage hierarchy or hierarchies the space is to be allocated. This request should be made only if main storage hierarchy support is specified during system generation.

To request the maximum amount of main storage required by the job step, REGION=nnnnnK is coded in the operand field of the JOB statement. The term nnnnn is replaced with the number of contiguous 1024-byte areas to be allocated to the job, e.g., REGION=52K. This number can range from one to five digits, but cannot exceed 16383. It should be specified as an even number. (If an odd number is specified, the system treats it as the next highest even number.)

If the REGION parameter is omitted or if a region size smaller than the default region size is requested, the default value (as established in the input reader procedure) is assumed.

Main storage hierarchy support provides for storage hierarchies 0 and 1. If IBM 2361 Core Storage, Model 1 or 2, is present in the system, processor storage is referred to as hierarchy 0 and 2361 Core Storage is referred to as hierarchy 1. If 2361 Core Storage is not present but main storage hierarchy support was specified during system generation, a 2-part region is established in processor storage when a region is defined to exist in two hierarchies. The two parts are not necessarily contiguous.

When main storage hierarchy support is included in the system, the REGION parameter can be used to request both the maximum amount of storage to be allocated to the job step and the hierarchy or hierarchies in which the storage is to be allocated.

To specify a region size and the hierarchy desired, REGION=(value$_0$ K, value$_1$K) is coded in the operand field of the JOB statement. The term "value$_0$" is replaced with the number of contiguous 1024-byte areas to be allocated to the job step in hierarchy 0; the term "value$_1$" is replaced with the number of contiguous 1024-byte areas to be allocated in hierarchy 1, e.g., REGION= (60K,200K). When processor storage includes hierarchies 0 and 1, the combined values of value$_0$ and value$_1$ cannot exceed 16383. If 2361 Core Storage is present, value$_0$ cannot exceed 16383, and value$_1$ cannot exceed 1024, if using a single Model 1, or 2048, if using a single Model 2. Each value specified should be an even number. (If an odd number is specified, the system treats it as the next highest even number.)

In systems with main storage hierarchy support, either subparameter can be omitted to request storage in only one hierarchy. If storage is requested only in hierarchy 1, a comma must be coded to indicate the absence of the first subparameter, e.g., REGION=(,52K). If storage is requested only in hierarchy 0, the parentheses need not be coded, e.g., REGION=70K.

If the REGION parameter is omitted, or if a region size smaller than the default region size is requested, the default value (as established in the input reader procedure) is assumed. When the default region size is assumed, storage is always allocated in hierarchy 0.

Notes:

- If the REGION parameter has been specified on the JOB statement, REGION parameters on the job's EXEC statements are ignored.
- If main storage hierarchy support is not included and regions are requested in both hierarchies, the region sizes are combined and an attempt is made to allocate a single region from processor storage. If a region is requested entirely from hierarchy 1, an attempt is made to allocate the region from processor storage.
- When the job step uses a cataloged procedure, a region size can be requested for a single procedure step by including, as part of the REGION parameter, the procedure step name, i.e., REGION.procstep. This specification overrides the REGION parameter in the named procedure step, if one is present. As many parameters of this form can be coded as there are steps in the cataloged procedure.
- To request a single region size for an entire cataloged procedure, code the REGION parameter without a procedure step name. This specification overrides all REGION parameters in the procedure, if any are present.
- For information on storage requirements to be considered when specifying a region size, see the Storage Estimates publication.

Establishing a Dispatching Priority (DPRTY) (Systems with MVT only)

The DPRTY parameter specifies the dispatching priority of a job step's tasks. The dispatching priority determines the order in which a job step's tasks will use main storage and CPU resources. Unless the DPRTY parameter is coded, each job step is assigned the same dispatching priority as the job.

To assign a dispatching priority to a job step, the keyword parameter:

DPRTY = (value 1, value 2)

is coded in the operand field of the EXEC statement. The terms value 1 and value 2 may each be assigned a number from 0 through 15. The higher the number, the higher the dispatching priority will be. (Whenever possible, assigning a number of 15 to value 1 should be avoided since this number is used for certain system tasks.) The number assigned to value 1 is converted by the system to determine an internal priority. The number assigned to value 2 is added to the internal priority to form the dispatching priority. If a number is not assigned to value 1, a default value of zero is assumed; for value 2 a default value of 11 is assumed.

DPRTY
: is used to assign a dispatching priority for a job step or for an entire cataloged procedure. For a cataloged procedure, this specification overrides all DPRTY parameters that may have been specified in the procedure.

DPRTY.procstep
: is used to assign a dispatching priority to a single procedure step in a cataloged procedure. This parameter overrides, for the named step, any DPRTY parameter which is present. One parameter of this form can be written for each step in the cataloged procedure.

Note: A detailed discussion of dispatching priorities can be found in the Introduction publication listed in the Preface.

DATA DEFINITION (DD) STATEMENT

The DD statement (Figure 8) describes data sets. The DD statement can contain the following information:

1. Name of the data set to be processed.
2. Type and number of I/O devices for the data set.
3. Volume(s) on which the data set resides.
4. Amount and type of space allocated on a direct access volume.
5. Label information for the data set.
6. Disposition of the data set after execution of the job step.
7. Allocation of data sets with regard to channel optimization.
8. Whether a particular data set may be used only for input or only for output.

NAME FIELD

ddname
is used:

1. To identify data sets defined by this DD statement to the compiler or linkage editor.
2. To relate data sets defined by this DD statement to data set reference numbers used by the programmer in his source module.
3. To identify this DD statement to other control statements in the input stream.

The "ddname" format is given in "FORTRAN Job Processing."

procstep.ddname
is used to override DD statements in cataloged procedures. The step in the cataloged procedure is identified by "procstep". The "ddname" identifies either:

1. A DD statement in the cataloged procedure that is to be modified by the DD statement in the input stream, or
2. A DD statement that is to be added to the DD statements in the procedure step.

JOBLIB and STEPLIB
are used to concatenate a private library with the system library, SYS1.LINKLIB; that is, the operating system library and the data sets specified in the JOBLIB or STEPLIB DD statement are temporarily combined to form one library. Use of JOBLIB results in concatenation for the duration of a job; use of STEPLIB, for the duration of a job step.

The JOBLIB DD statement must appear immediately after the JOB statement of the job to which it pertains, and its operand field, at a minimum, must contain the DSNAME and DISP parameters. The DISP parameter must be coded either DISP=(NEW,PASS) or DISP=(OLD,PASS) or DISP=(SHR,PASS) so that the library remains available throughout the job. (See the discussion of the DISP parameter under "Operand Field."

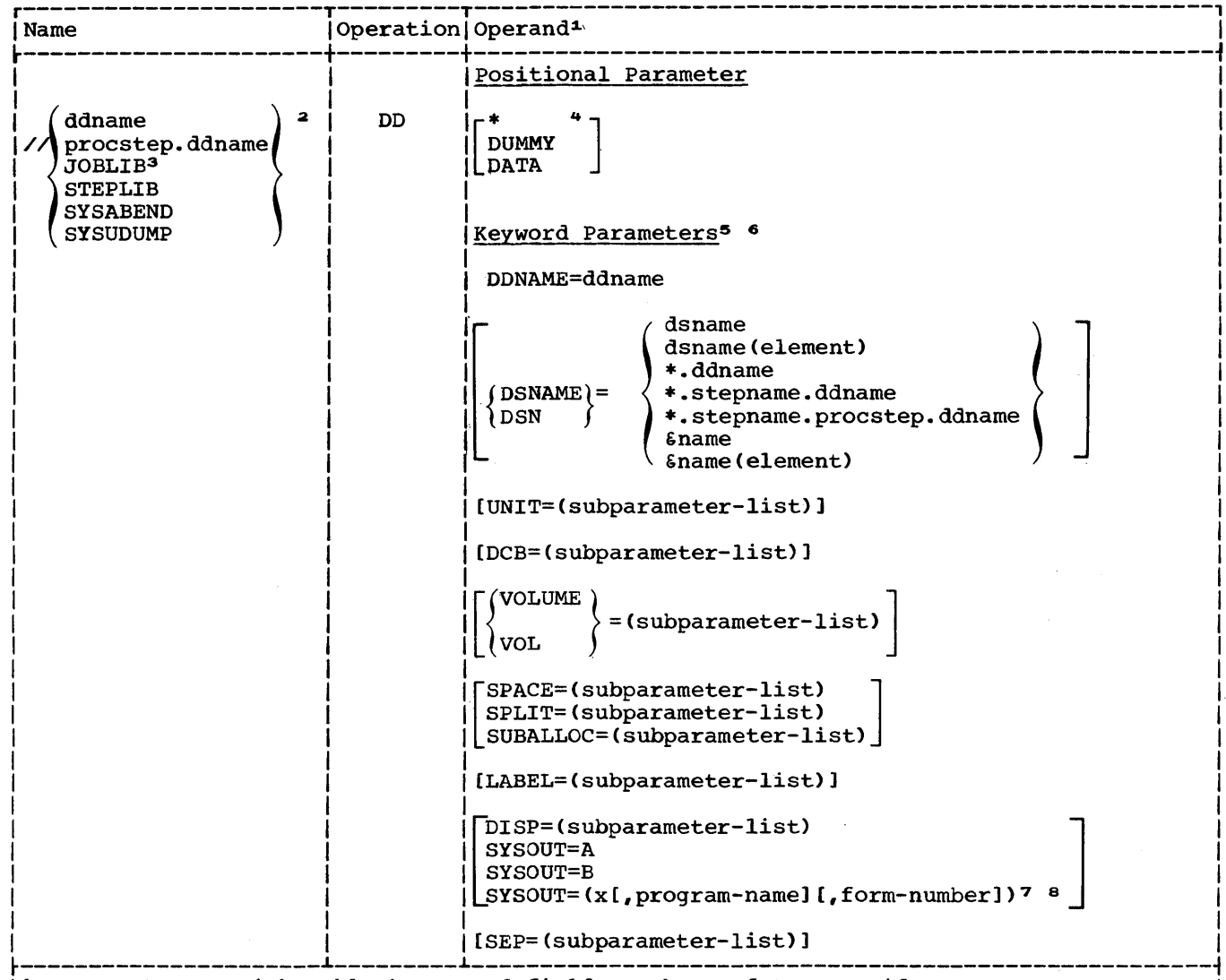
The STEPLIB DD statement may appear in any position among the DD statements for the step. The data set defined should be OLD. If the private library is not cataloged and is to be referred to in a later step (or steps), DISP=(OLD,PASS) or DISP=(SHR,PASS) should be coded; a later step may then refer to it by coding DSNAME=*.stepname.STEPLIB, DISP=(OLD,PASS) on the STEPLIB DD statement for the later step.

For additional information on the use of JOBLIB and STEPLIB DD statements, see the Job Control Language Reference publication, Order No. GC28-6704.

SYSABEND and SYSUDUMP
are special DD names used to define a data set on which a system abnormal termination dump can be written. The dump is provided for job steps subject to abnormal termination.

The dump provided when the SYSABEND DD statement is used includes the system nucleus, the problem program storage area, and a trace table, if the trace table option was requested at system generation. The SYSUDUMP DD statement provides a dump of only the problem program area.

A full discussion of SYSABEND and SYSUDUMP DD statements, with an example of use, appears in the Job Control Language Reference publication, Order No. GC28-6704.



[1]A DD statement with a blank operand field can be used to override parameters specified in cataloged procedures. (See "Overriding and Adding DD Statements" in the section "Cataloged Procedures".)
[2]The name field is blank when concatenating data sets. (Note the exception for the use of JOBLIB.)
[3]The JOBLIB statement precedes any EXEC statements in the job. (See the discussion concerning JOBLIB under "Name Field" in this section.)
[4]If either the * or DATA positional parameter is specified, no keyword parameters other than DCB=BLKSIZE and DCB=BUFNO can be specified.
[5]If "subparameter-list" consists of only one subparameter and no leading comma (indicating the omission of a positional subparameter) is required, the delimiting parentheses may be omitted.
[6]If "subparameter-list" is omitted, the entire parameter must be omitted.
[7]This form of the parameter is used only with priority schedulers.
[8]If "program-name" and "form-number" are omitted, the delimiting parentheses can be omitted. If only the form number is given, the parentheses must be used and two commas must precede the form number.

Figure 8. Data Definition Statement

BLANK NAME FIELD

If the name field is blank, the data set defined by the DD statement is concatenated with the data set defined in the preceding DD statement. In effect, these two data sets are combined into one data set. (Private libraries, i.e., partitioned data sets, may also be concatenated with the library specified in the JOBLIB DD statement. Therefore, several libraries can be concatenated with the system library. Individual members of a partitioned data set, however, cannot be concatenated.)

Note: Handling of data sets whose records are of different lengths and/or different formats is a function of the program being executed. Data sets designated for concatenation may not be in the input stream.

OPERAND FIELD

For purposes of discussion, parameters for the DD statement have been divided into seven functions. Parameters are used to:

- Specify data in the input stream.
- Specify unit record data sets.
- Retrieve a previously created and cataloged data set.
- Retrieve a data set created in a previous job step in the current job and passed to the current job step.
- Retrieve a data set created but not cataloged in a previous job.
- Create data sets that reside on magnetic tape or direct access volumes.
- Optimize I/O operations.

The following text describes the DD statement parameters that apply to:

- Processing unit record data sets.
- Retrieving data sets created in previous job steps.
- Retrieving data sets created and cataloged in previous jobs.

See Figure 9 for applicable parameters.

Parameters shown in Figure 8 and not mentioned in this section are used to create data sets and optimize I/O operations in job steps. These parameters are discussed in the sections "Creating Data Sets" and "Programming Considerations."

Specifying Data in the Input Stream:

*
: indicates that a data set (e.g., a source module or data) immediately follows this DD statement in the input stream (see Figure 10). If the EXEC statement for the job step invokes a cataloged procedure, a data set may be placed in the input stream for each procedure step. If the EXEC statement specifies execution of a program, only one data set may be placed in the input stream. The DD * statement must be the last DD statement for the procedure step or program. The end of the data set must be indicated by a delimiter statement. The data itself cannot contain job control statements (neither the comment statement nor any statements with // or /* in columns 1 or 2). Note, too, that if * is specified, no keyword parameters other than DCB=BLKSIZE or DCB=BUFNO may be specified.

DATA
: also indicates data in the input stream. The restrictions and use of the DATA parameter are the same as the * parameter, except that // may appear in the first and second positions of a record.

```
 {*   }1
 {DATA}

                  ( dsname                         )
                  ( dsname(element)                )
                  ( *.ddname                       )
 {DSNAME}=        { *.stepname.ddname              }
 {DSN   }         ( *.stepname.procstep.ddname     )
                  ( &name                          )
                  ( &name(element)                 )

 UNIT=(name[,{n|P}2])3

 DCB=( {MODE=E} {,STACK=1}  )
       {MODE=C} {,STACK=2}

   SYSOUT=A
   SYSOUT=B
   SYSOUT=(x[,program-name]
             [,form-number])4 5

           (SHR ) [,DELETE6 ]  [,DELETE  ]
 DISP=(    (OLD ) [,KEEP    ]  [,KEEP    ]  )7
           (NEW ) [,PASS    ]  [,CATLG   ]
           (MOD ) [,CATLG   ]  [,UNCATLG ]
                  [,UNCATLG ]

 LABEL=(subparameter-list)8

 {VOLUME}
 {      }=(subparameter-list)8
 {VOL   }
```

[1]If * is specified, no keyword parameters other than DCB=BLKSIZE or DCB=BUFNO may be specified.
[2]If neither "n" nor "P" is specified, 1 is assumed.
[3]If only "name" is specified, the delimiting parentheses may be omitted.
[4]This form of the parameter is used only with priority schedulers.
[5]If "program-name" and "form-number" are omitted, the delimiting parentheses can be omitted.
[6]The assumption for the second subparameter is discussed in "Specifying the Disposition of a Data Set" in this section.
[7]The subparameters are positional.
[8]See the section "Creating Data Sets."

Figure 9. DD Statement

UNIT Parameter:

UNIT=(name[,{n|P}])
specifies the name and number of I/O devices for a data set (see Figure 10). When the system is generated, the "name" is assigned by the operating system or the installation and represents a device address, a device type, or a device class. (See the System Generation publication.)

The programmer can use only the assigned names in his DD statements. For example,

UNIT=190, UNIT=2311, UNIT=TAPE

where 190 is a device address, 2311 is a device type, and TAPE is a device class.

n|P
specifies the number of devices allocated to the data set. If a number "n" is specified, the operating system assigns that number of devices to the data set. "P" is used with cataloged data sets when the required number of volumes is unknown. The control program assigns a device for each volume required by the data set.

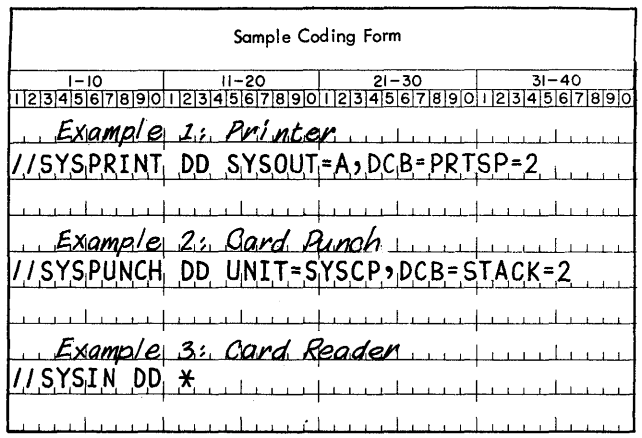


Figure 10. Examples of DD Statements for Unit Record Devices

DCB Parameter:

DCB=({MODE=E} {,STACK=1})
     {MODE=C} {,STACK=2}

specify options for the card read punch. The MODE subparameter indicates whether the card is transmitted in column binary or EBCDIC mode; C specifies column binary, and E specifies EBCDIC.

The STACK subparameter indicates stacker selection for the card read punch.

Routing a Data Set To An Output Stream (SYSOUT): With the SYSOUT parameter, output data sets can be routed to a system output stream and handled much the same as system messages.

SYSOUT=A
can be used with sequential schedulers to indicate that the data set is to be

written on the system output device. No parameter other than the DCB parameter has any meaning when SYSOUT=A is used. This form of the SYSOUT parameter may be specified for printer data sets.

SYSOUT=B
can be used with sequential schedulers to indicate the system card punch unit. The priority scheduler routes the output data set to class B.

SYSOUT=(x[,program-name][,form-number])
is used with priority schedulers. When priority schedulers are used, a data set is normally written on an intermediate direct access device during program execution and later routed through an output stream to a system output device. The "x" is to be replaced by an alphabetic or numeric character that specifies the system output class to be used. Output writers route data from the output classes to system output devices. The DD statement for this data set can also include a unit specification that describes the intermediate direct access device and an estimate of the space required. If these parameters are omitted, the job scheduler provides default values as the job is read and processed.

If there is a special installation program to handle output operations, its "program-name" should be specified. "Program-name" is the member name of the program, which must reside in the system library.

If the output data set is to be printed or punched on a specific type of output form, a 4-digit "form-number" should be specified. This form number is used to instruct the operator, in a message issued at the time the data set is to be printed, of the form to be used.

Retrieving Previously Created Data Sets

If a data set is created with standard labels and cataloged in a previous job, all information for the data set, such as record format, density, volume sequence number, device type, etc., is stored in the catalog and labels. This information need not be repeated in the DD statement used to retrieve the data set; only the name (DSNAME) and disposition (DISP) is required.

If a data set was created in a previous job step in the current job and its disposition was specified as PASS, all the information in the previous DD statement is available to the control program, and is accessible by referring to the previous DD statement by name. To retrieve the data set, a pointer to a data set created in a previous job step is specified by the DSNAME parameter. The disposition (DISP) of the data set is also specified, along with the UNIT parameter if more than one unit is to be allocated.

If a data set is created with standard labels in a previous job but not cataloged, information for the data set, such as record format, density, volume sequence number, etc., is stored in the labels; the device type information is not stored. To retrieve the data set, the name (DSNAME), disposition (DISP), volume serial number (VOLUME), and device (UNIT) must be specified.

If a data set is created with no labels and cataloged, device type information is stored in the catalog. To retrieve the data set, the name (DSNAME), disposition (DISP), volume serial number (VOLUME), and the LABEL and DCB parameters must be specified.

Examples of the use of DD statements to retrieve previously created data sets are shown in Figure 11.

IDENTIFYING A CREATED DATA SET: The DSNAME parameter indicates the name of a data set or refers to a data set defined in the current or a previous job step.

Specifying a Cataloged Data Set by Name:

DSNAME=dsname
the name of the data set is indicated by "dsname." If the data set was previously created and cataloged, the control program uses the catalog to find the data set and instructs the operator to mount the required volumes.

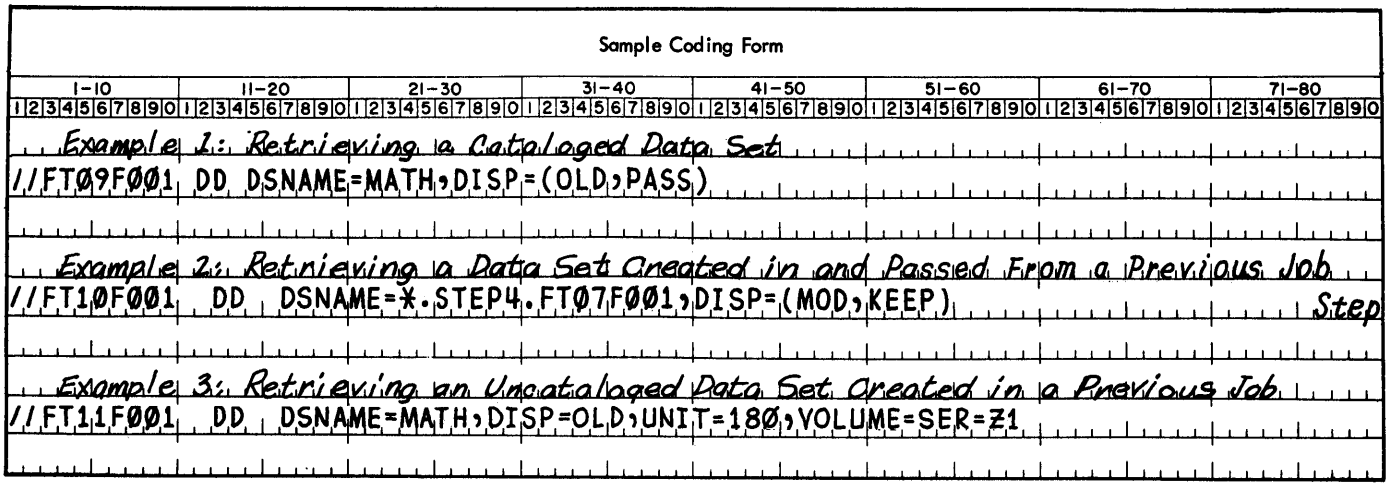


Figure 11. Retrieving Previously Created Data Sets

Specifying a Generation Data Group or PDS:

DSNAME=dsname(element)
indicates either a generation data set contained in a generation data group, or a member of a partitioned data set. The name of the generation data group or partitioned data set is indicated by "dsname"; if "element" is either 0 or a signed integer, a generation data set is indicated. For example,

```
DSNAME=FIRING(-2)
```

indicates the thirdmost recent member of the generation data group FIRING. (See the Data Management publication for the complete description of generation data sets.) If "element" is a name, a member of a partitioned data set is indicated.

Note: Members of a partitioned data set may be read as input to a FORTRAN object program or created as output from a FORTRAN object program, but only if the member name and either LABEL=(,,,IN) or LABEL=(,,,OUT) are specified in an associated DD statement.

Referring to a Data Set in the Current Job Step:

DSNAME=*.ddname
indicates a data set that is defined previously in a DD statement in this job step. The * indicates the current job. The name of the data set is copied from the DSNAME parameter in the DD statement named "ddname".

Referring to a Data Set in a Previous Job Step:

DSNAME=*.stepname.ddname
indicates a data set that is defined in a DD statement in a previous job step in this job. The * indicates the current job, and "stepname" is the name of a previous job step. The name of the data set is copied from the DSNAME parameter in the DD statement named "ddname". For example, in the following control statements the DD statement FT08F001 in job step S2 indicates that the data set name (TIME) is copied from the DD statement FT09F001 in job step S1.

```
//LAUNCH   JOB
//JOBLIB   DD DSNAME=FIRING,DISP=(OLD,PASS)
//S1 EXEC PGM=ROCKET
//FT01F001 DD DSNAME=RATES(+1),DISP=OLD
//FT09F001 DD DSNAME=TIME,DISP=(OLD,PASS)
//S2 EXEC PGM=DISTANCE
//FT08F001 DD DSNAME=*.S1.FT09F001,        X
//            DISP=OLD
//FT05F001 DD *
            .
            .
            .
```

Referring to a Data Set in a Cataloged Procedure:

DSNAME=*.stepname.procstep.ddname
indicates a data set that is defined in a cataloged procedure invoked by a previous job step in this job. The * indicates the current job; "stepname" is the name of a previous job step that invoked the cataloged procedure; "procstep" is the name of a step in the cataloged procedure. The name of the data set is copied from the DSNAME parameter in the DD statement named "ddname".

Assigning Names to Temporary Data Sets:

DSNAME=&name
assigns a name to a temporary data set. The control program assigns the data set a unique name which exists only until the end of the current job. The data set is accessible in subsequent job steps by specifying "&name". This option is useful in passing an object module from a compiler job step to a linkage editor job step.

DSNAME=&name(element)
assigns a name to a member of a temporary PDS. The name is assigned in the same manner as for the DSNAME=&name option. The "&name(element)" option is useful in storing load modules that will be executed in a later job step in the current job.

SPECIFYING THE DISPOSITION OF A DATA SET:
The DISP parameter is specified for both previously created data sets and data sets being created in this job step. It contains three subparameters.

```
DISP=( SHR  [,DELETE ]  [,DELETE ]
       NEW  [,KEEP   ]  [,KEEP   ]
       OLD  [,PASS   ]  [,CATLG  ]
       MOD  [,CATLG  ]  [,UNCATLG]
            [,UNCATLG]             )
```

The first subparameter indicates the status of the data set at the beginning of or during the job step.

SHR
indicates that the data set resides on a direct-access volume and is used as input to a job whose operations do not prevent simultaneous use of the data set as input to another job. This parameter has meaning only in a multiprogramming environment for existing data sets. If it is omitted in a multiprogramming environment, the data set is considered unusable by any other concurrently operating job. If it is coded in other than a multiprogramming environment, the system assumes that the disposition of the data set is OLD.

NEW
indicates that the data set is created in this step. NEW is discussed in more detail in the section "Creating Data Sets."

OLD
indicates that the data set was created by a previous job or job step.

MOD
indicates that the data set was created in a previous job or job step, and that additional records are to be added to it. Before the first input/output operation for the data set occurs, the data set is automatically positioned after the last record in the data set. If MOD is specified and no volume information (e.g., volume serial number) is available for the data set, the system assumes the data set does not yet exist and creates the data set for the job step. (Volume information is considered available if it is coded in the DD statement, passed with the data set from a previous step, or contained in the catalog.)

The second subparameter indicates the disposition of the data set at normal job step termination.

DELETE
causes the space occupied by the data set to be released and made available at the end of the current job step. If the data set was cataloged, it is removed from the catalog.

KEEP
insures that the data set is kept intact until a DELETE parameter is specified in a subsequent job or job step. KEEP is used to retain uncataloged data sets for processing in future jobs. KEEP does not imply PASS.

PASS
indicates that the data set is referred to in a later job step. When a subsequent reference to the data set is made, its PASS status lapses unless another PASS is issued. The final disposition of the data set should be stated in the last job step that uses the data set. When a data set is in PASS status, the volume(s) on which it is mounted is retained. If demounting is necessary, the control program issues a message to mount the volume(s) when needed. PASS is used to pass data sets among job steps in the same job.

If a data set on an unlabeled tape is being passed, the volume serial number must be specified in the VOLUME=SER= parameter of the DD statement that passed the data set.

Note: The PASS status of the private library specified in a JOBLIB DD statement always remains in effect for the duration of a job.

CATLG
causes the creation of a catalog entry that points to the data set. The data set can then be referred to in subsequent jobs or job steps by name (CATLG implies KEEP).

UNCATLG
causes references to the data set to be removed from the catalog at the end of the job step.

If the second subparameter is not specified, no action is taken to alter the status of the data set. If the data set was created in this job, it is deleted at the end of the current job step. If the data set existed before this job, it exists after the end of the job.

The third subparameter indicates the disposition of the data set if the job step terminates abnormally. This is the conditional disposition of the data set. Explanations for DELETE, KEEP, CATLG, and UNCATLG are the same as those for normal termination.

Notes:

- If a conditional disposition is not specified and the job step abnormally terminates, the requested disposition (the second subparameter of the DISP keyword) is performed.
- Data sets that were passed but not received by subsequent steps because of abnormal termination will assume the conditional disposition specified the last time they were passed. If a conditional disposition was not specified then, all data sets that were new when initially passed are deleted. All other data sets are kept.
- A conditional disposition other than DELETE for a temporary data set is invalid, and the system assumes DELETE.

Effect of DISP Parameter at End of FORTRAN Job: In a FORTRAN job that is terminated by a STOP or CALL EXIT statement all data sets that were used by the job will be closed. The closing operation will position the volume in accordance with the DISP parameter, as follows:

DISP Parameter	Positioning Action
PASS	Forward space to end of data set
DELETE	Rewind
KEEP, CATLG, UNCATLG	Rewind and unload

DELIMITER STATEMENT

The delimiter statement (see Figure 12) is used to separate data from subsequent control statements in the input stream, and is placed after each data set in the input stream.

Name	
/*	

Figure 12. Delimiter Statement

The delimiter statement contains a slash in column 1, an asterisk in column 2, and a blank in column 3. The remainder of the card may contain comments.

COMMENT STATEMENT

The comment statement (see Figure 13) is used to enter any information considered helpful by the programmer. It can be inserted before or after any control statement that follows the JOB statement. Comments can be coded in columns 4 through 80. The comments cannot be continued onto another statement. (If the comment statement appears on a system output listing, it can be identified by the appearance of *** in columns 1 through 3.)

Name	
//*	

Figure 13. Comment Statement

The comment statement contains a slash in column 1, a slash in column 2, and an asterisk in column 3. The rest of the card can contain comments.

To process a FORTRAN source module from compilation through execution, three steps are required: to compile the source module to obtain an object module, to linkage edit the object module to obtain a load module,[1] and to execute the load module. For each of these three steps, job control statements are required to indicate the program or procedure to be executed, to specify options for the compiler and linkage editor, to specify conditions for termination of processing, and to define the data sets used during processing. Because writing these job control statements can be time-consuming work for the programmer, IBM supplies, for each compiler, four cataloged procedures to aid in the processing of FORTRAN modules. The use of cataloged procedures minimizes the number of job control statements that must be supplied by the programmer.

USING CATALOGED PROCEDURES

When a programmer uses cataloged procedures, he must supply the following job control statements.

1. A JOB statement.
2. An EXEC statement that indicates the cataloged procedure to be executed.
3. A procstep.SYSIN DD statement that specifies the location of the source module(s) or the object module(s) to the control program. (Note: If the source module(s) and/or object module(s) are placed in the input stream, a delimiter statement is required at the end of each data set.)

In addition, a GO.SYSIN DD statement can be used to define data in the input stream for load module execution. (A delimiter statement is required at the end of the data.)

The job control statements needed to invoke the procedures, and deck structures used with the procedures are described in the following text.

---

[1]As an alternative, the object module may be edited and then automatically executed by the loader, another IBM-supplied program. Details on the use of the loader can be found in the Linkage Editor and Loader publication.

COMPILE

The FORTRAN IV (G) cataloged procedure for compilation is FORTGC; the FORTRAN IV (H), FORTHC.[2] These cataloged procedures consist of the control statements shown in Figures 49 and 54, respectively.

Figures 14, 15, and 16 show control statements that can be used, as programming needs dictate, to invoke for either compiler the cataloged procedure for compilation. For both compilers, control statements and control statement fields are identical, except for the procedure-name specified on the EXEC statement: FORTGC is specified for the (G) compiler; FORTHC, for the (H) compiler. In the control statement sequences shown, the SYSIN data set containing the source module is defined as data in the input stream for the compiler. Note that a delimiter statement follows the FORTRAN source module.

```
//jobname JOB
// EXEC FORTGC or FORTHC
//FORT.SYSIN DD *
┌---------------------------------------┐
|          FORTRAN Source Module        |
└---------------------------------------┘
/*
```

Figure 14. Invoking the Cataloged Procedure FORTGC or FORTHC

Single Compile: A sample deck structure to compile a single source module is shown in Figure 15.

```
//JOBSC JOB 00,FORTRANPROG,MSGLEVEL=1
//EXECC EXEC PROC=FORTGC or PROC=FORTHC
//FORT.SYSIN DD *
┌---------------------------------------┐
|          FORTRAN Source Module        |
└---------------------------------------┘
/*
```

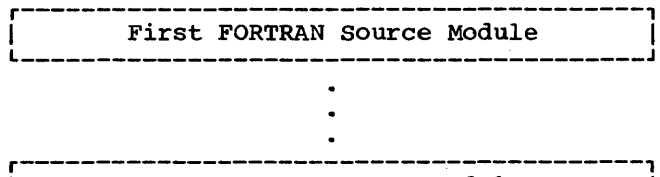
Figure 15. Compiling a Single Source Module

---

[2]For FORTRAN IV (H), if the EDIT option is specified, a SYSUT1 data set must be defined as a work data set for the compiler; if the compiler XREF option is specified, a SYSUT2 data set must be defined as a work data set.

Batched compile: A sample deck structure to batch compile is shown in Figure 16.

```
//JOBBC JOB 00,FORTRANPROG,MSGLEVEL=1
//EXECC EXEC PROC=FORTGC or PROC=FORTHC
//FORT.SYSIN DD *
```




```
/*
```

Figure 16. Compiling Several Source Modules

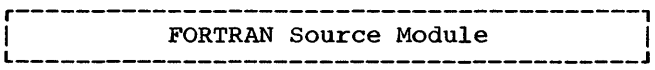
When several source modules are entered in the SYSIN data set for one job step, the compiler recognizes the FORTRAN END statement. If the next card is a delimiter statement, control returns to the control program at the end of the compilation. If the next card is a FORTRAN statement, control remains with the FORTRAN compiler.

COMPILE AND LINKAGE EDIT

For FORTRAN IV (G), the cataloged procedure to compile FORTRAN source modules and linkage edit the resulting object modules is FORTGCL; for FORTRAN IV (H), FORTHCL. These cataloged procedures consist of the control statements shown in Figures 50 and 55, respectively.

Figure 17 shows control statements that can be used to invoke FORTGCL or FORTHCL. The control statements are identical for both compilers, except for the procedure-name specified on the EXEC statement: FORTGCL is specified for the (G) compiler; FORTHCL, for the (H) compiler.

```
//jobname JOB
// EXEC FORTGCL or FORTHCL
//FORT.SYSIN DD *
```




```
/*
```

Figure 17. Invoking the Cataloged Procedure FORTGCL or FORTHCL

LINKAGE EDIT AND EXECUTE

For FORTRAN IV (G), the cataloged procedure to linkage edit FORTRAN object modules and execute the resulting load module is FORTGLG; for FORTRAN IV (H), FORTHLG. These cataloged procedures consist of the control statements shown in Figures 51 and 56, respectively.

Figure 18 shows control statements that can be used to invoke FORTGLG or FORTHLG. The control statements are identical for both compilers, except for the procedure name specified on the EXEC statement: FORTGLG is specified for the (G) compiler; FORTHLG, for the (H) compiler.

```
//jobname JOB
// EXEC FORTGLG or FORTHLG
//LKED.SYSIN DD *
```

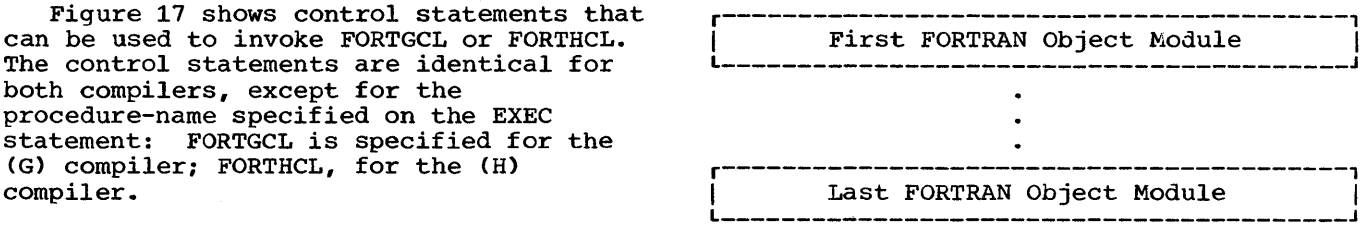

```
/*
```

Figure 18. Invoking the Cataloged Procedure FORTGLG or FORTHLG

A sample deck structure to linkage edit and execute, as one load module, several object modules entered in the input stream is shown in Figure 19.

The object module decks were created by the DECK compiler option. The linkage editor recognizes the end of one module and the beginning of another, and resolves references between them.

```
//JOBBLG JOB 00,FORTPROG,MSGLEVEL=1
//EXECLG EXEC PROC=FORTGLG or PROC=FORTHLG
//LKED.SYSIN DD *
```




```
/*
//GO.SYSIN DD *
```

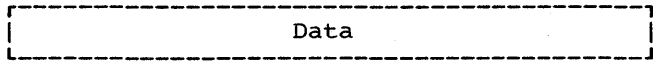


```
/*
```

Figure 19. Linkage Edit and Execute

A sample deck structure is shown in Figure 20 to linkage edit and execute, as one load module, object modules that are members of the cataloged sequential data set, OBJMODS, which resides on a tape volume. In addition, a data set in the input stream is processed using the SYSIN data set.

```
//JOBBLG JOB 00,FORTPROG,MSGLEVEL=1
//EXECLG EXEC FORTGLG or FORTHLG
//LKED.SYSIN DD DSNAME=OBJMODS,DISP=OLD
//GO.SYSIN DD *
```




```
/*
```

Figure 20. Linkage Edit and Execute Object Modules in a Cataloged Data Set

COMPILE, LINKAGE EDIT, AND EXECUTE

The FORTRAN IV (G) cataloged procedure FORTGCLG and the FORTRAN IV (H) cataloged procedure FORTHCLG each pass a source through three procedure steps -- compile, linkage edit, and go (execute). These cataloged procedures consist of the control statements shown in Figures 52 and 57, respectively.

Figures 21, 22, and 23 show control statements used to invoke FORTGCLG or FORTHCLG. For both compilers, control statements and control statement fields are identical, except for the procedure name specified on the EXEC statement: FORTGCLG is specified for the (G) compiler; FORTHCLG, for the (H).

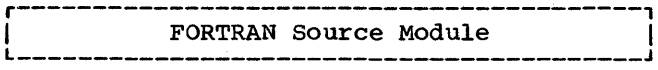
```
//jobname JOB
// EXEC PROC=FORTGCLG or PROC=FORTHCLG
//FORT.SYSIN DD *
```


```
/*
```

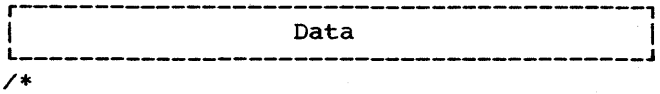
Figure 21. Invoking the Cataloged Procedure FORTGCLG or FORTHCLG

Single Compile, Linkage Edit, and Execute: Figure 22 shows a sample deck structure to compile, linkage edit, and execute a single source module.

```
//JOBSCLG JOB 00,FORTPROG,MSGLEVEL=1
//EXECC EXEC FORTGCLG or FORTHCLG
//FORT.SYSIN DD *
```




```
/*
//GO.SYSIN DD *
```

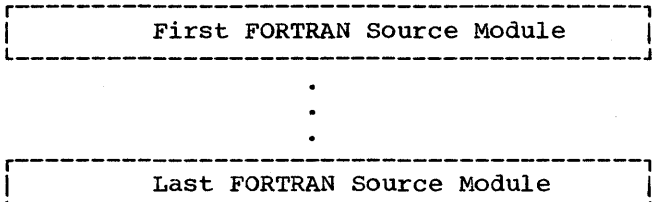



```
/*
```

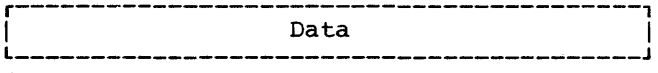
Figure 22. Single Compile, Linkage Edit, and Execute

Batched Compile, Linkage Edit, and Execute: Figure 23 shows a sample deck structure to batch compile, linkage edit, and execute a FORTRAN main program and its subprograms. The source modules are placed in the input stream along with a data set that is read using the SYSIN data set.

```
//JOBBCLG JOB 00,FORTPROG,MSGLEVEL=1
//EXECCLG EXEC FORTGCLG or FORTHCLG
//FORT.SYSIN DD *
```




```
/*
//GO.SYSIN DD *
```




```
/*
```

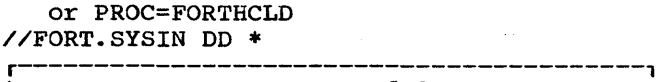
Figure 23. Batched Compile, Linkage Edit, and Execute

COMPILE AND LOAD

The FORTRAN IV (G) cataloged procedure FORTGCLD and the FORTRAN IV (H) cataloged procedure FORTHCLD compile FORTRAN source modules and load the resulting object modules. The load step combines the function of the linkage editor with execution of the edited module.

Figure 24 shows control statements that can be used to invoke FORTGCLD or FORTHCLD.

```
//jobname JOB
//EXECLD EXEC PROC=FORTGCLD
   or PROC=FORTHCLD
//FORT.SYSIN DD *
```




```
/*
```

Figure 24. Invoking the Cataloged Procedure FORTGCLD or FORTHCLD

Single Compile and Load: Figure 25 shows control statements that can be used to compile and load a single source module.

```
//jobname JOB 00,'SOURCE A',MSGLEVEL=1
//EXECA EXEC FORTGCLD or FORTHCLD
//FORT.SYSIN DD *
┌---------------------------------------┐
|       FORTRAN Source Module           |
└---------------------------------------┘
/*
//GO.SYSIN DD *
┌---------------------------------------┐
|                Data                   |
└---------------------------------------┘
/*
```

Figure 25. Single Compile and Load

Batched Compile and Load: Figure 26 shows control statements that can be used to batch compile and load a FORTRAN main program and its subprograms. The source modules are placed in the input stream along with a data set that is read using the SYSIN data set.

```
//jobname JOB 00,'SOURCE B',MSGLEVEL=1
//EXECB EXEC PROC=FORTGCLD or PROC=FORTHCLD
//FORT.SYSIN DD *
┌---------------------------------------┐
|       First FORTRAN Source Module     |
└---------------------------------------┘
                    .
                    .
                    .
┌---------------------------------------┐
|       Last FORTRAN Source Module      |
└---------------------------------------┘
/*
//GO.SYSIN DD *
┌---------------------------------------┐
|                Data                   |
└---------------------------------------┘
/*
```

Figure 26. Batched Compile and Load

COMPILER PROCESSING

The names for DD statements (ddnames) relate I/O statements in the compiler with data sets used by the compiler. These ddnames must be used for the compiler. When the system is generated, names for I/O device classes are also established and must be used by the programmer.

Compiler Name

The program name for the FORTRAN IV (G) compiler is IEYFORT; for the FORTRAN IV (H) compiler, IEKAA00. If either compiler is to be executed without using the supplied cataloged procedures, an EXEC statement of the following form must be used:

```
// EXEC PGM=IEYFORT or // EXEC PGM=IEKAA00
```

(For more information on procedures and options in invoking IEYFORT or IEKAA00, see "Appendix A: Invoking the FORTRAN Compiler.")

Compiler ddnames

The compiler can use seven data sets. To establish communication between the compiler and the programmer, each data set is assigned a specific ddname. Each data set has a specific function and device requirement. Table 3 lists the ddnames, functions, and device requirements for the data sets.

To compile a FORTRAN source module, two of these data sets are necessary -- SYSIN and SYSPRINT, along with the direct access volume(s) that contains the operating system. However, with these two data sets, only the source listing is generated by the compiler. If an object module is to be punched and/or written on a direct-access or magnetic tape volume, a SYSLIN and/or SYSPUNCH DD statement must be supplied.[1]

For the DD statements SYSIN, SYSABEND, SYSUDUMP, or SYSPRINT, an intermediate storage device may be specified instead of the card reader or printer. The intermediate storage device can be magnetic tape or a direct-access device.

If an intermediate device is specified for SYSIN, the compiler assumes that the source module deck was written on intermediate storage by a previous job or job step. If an intermediate device is specified for SYSPRINT, the map, listing, and error/warning messages are written on intermediate storage; a new job or job step can print the contents of the data set. When the SYSPRINT data set is written on intermediate storage, carriage control characters are placed in the records.

---

[1]For FORTRAN IV (H), if a structured source listing is to be generated, a SYSUT1 DD statement must be supplied. If a cross reference listing is to be generated by the compiler, a SYSUT2 DD statement must be supplied.

Table 3. Compiler ddnames

ddname	Function	Device Requirements
FORTRAN IV (G) and FORTRAN IV (H)		
SYSIN	reading the source program	•card reader •intermediate storage
SYSPRINT	writing the storage map, listing, label map, and messages	•printer •intermediate storage
SYSPUNCH	punching the object module deck	•card punch[1] •direct access •magnetic tape
SYSLIN	output data set for the object module, used as input to the linkage editor	•direct access •magnetic tape •card punch[1]
FORTRAN IV (H) Only		
SYSUT1	work data set for the structured source listing	•direct access •magnetic tape
SYSUT2	work data set for the compiler cross reference listing	•direct access •magnetic tape
SYSABEND or SYSUDUMP	writing the dump for an abnormal termination	•printer •intermediate storage

[1]These must not be the same card punches.

Compiler Device Classes

Names for input/output device classes used for compilation are also specified by the operating system when the system is generated. The class names, functions, and types of devices are shown in Table 4.

The data sets used by the compiler must be assigned to the device classes listed in Table 5.

Compiler Data Set Assumptions

Standard assumptions are made for the DCB parameter of the data sets used by the FORTRAN IV (G) and (H) compilers. Table 6 contains the values set for the logical record length, record format, and blocksize for the FORTRAN IV (G) compiler. Table 7 contains the values set for the logical record length, record format, and blocksize for the FORTRAN IV (H) compiler. Of the values in these two tables, only the values for blocksize may be overridden with a DD statement.

In addition, the programmer may specify the number of buffers to be used for the compiler data sets. If this information is missing, the queued sequential access method (QSAM) default is used. This default is three buffers for an IBM 2540 card read punch and two buffers for all other devices.

Table 4. Device Class Names

Class Name	Class Functions	Device Type
SYSSQ	writing, reading, and backspacing (sequential)	•magnetic tape •direct access
SYSDA	writing, reading, backspacing, and updating records in place (direct)	•direct access
SYSCP	punching cards	•card punch
A	SYSOUT output	•printer •magnetic tape
B	SYSOUT card image output	•card punch •magnetic tape

Table 5. Correspondence Between Compiler ddnames and Device Classes

ddname	Possible Device Classes (H)	Possible Device Classes (G)
SYSIN	SYSSQ, or the input stream device (specified by DD * or DD DATA), or a device specified as the card reader	SYSSQ, or the input stream device (specified by DD * or DD DATA), or a device specified as the card reader
SYSPRINT	A,SYSSQ	A,SYSSQ
SYSPUNCH	B,SYSCP[1],SYSSQ,SYSDA	B,SYSCP
SYSLIN	SYSSQ,SYSDA,SYSCP[1]	SYSSQ,SYSDA
SYSUT1	SYSSQ	
SYSUT2	SYSSQ	
SYSABEND or SYSUDUMP	A,SYSSQ	

[1]SYSPUNCH and SYSLIN must not be assigned to the same card punch.

Compiler Options

Options may be passed to the FORTRAN IV (G) or (H) compiler through the PARM parameter in the EXEC statement (see Figure 27).

The following information may be specified for both compilers:

1. Whether a listing of an object module is printed.
2. The name assigned to the program.
3. The number of lines per page for the source listing.
4. Whether the source module is coded in Binary Coded Decimal (BCD) or Extended Binary Coded Decimal Interchange Code (EBCDIC).
5. Whether a list of the source statements, with their associated internal statement numbers, is printed.
6. Whether an object module is punched.
7. Whether a storage map of variable names used in the source mdoule is printed.
8. Whether the compiler writes the object module on external storage for input to the linkage editor.
9. Whether traceback information is to be inserted into the source module.

Table 6. DCB Assumptions for the (G) Compiler Data Sets

ddname	LRECL	RECFM	BLKSIZE
SYSIN	80	FB	80
SYSPRINT	120	FBSA	120
SYSLIN	80	FBS	80
SYSPUNCH	80	FBSA	80

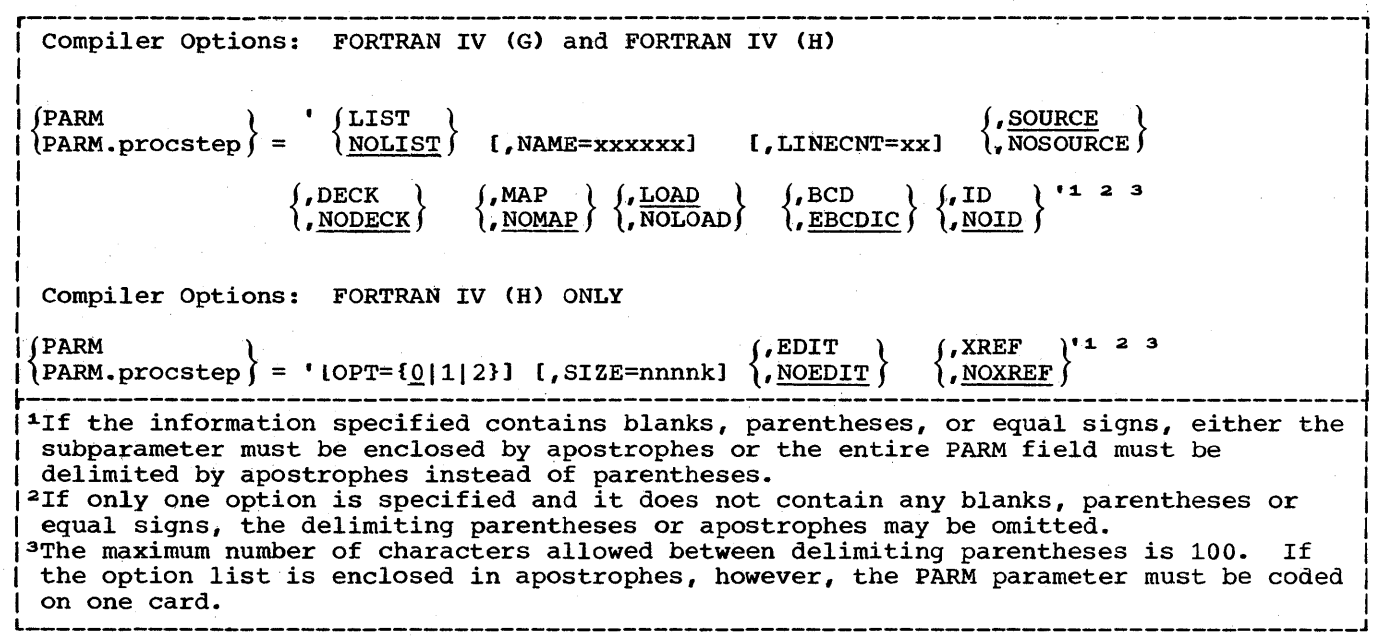
Note: The values specified for LRECL and RECFM cannot be changed by the FORTRAN programmer. The value for BLKSIZE may be changed.

For fixed-length records (F), S indicates standard blocks, with no truncated blocks or unfilled tracks within the data set.

The following information may be specified for the (H) compiler only:

1. Whether a storage map of labels used in the source module is printed.
2. The type of optimization, if any, desired by the programmer.
3. Whether a structured source listing is written.
4. Whether a cross reference listing is printed.

Compiler options in the PARM parameter need not be coded in any specific order.



[1]If the information specified contains blanks, parentheses, or equal signs, either the subparameter must be enclosed by apostrophes or the entire PARM field must be delimited by apostrophes instead of parentheses.
[2]If only one option is specified and it does not contain any blanks, parentheses or equal signs, the delimiting parentheses or apostrophes may be omitted.
[3]The maximum number of characters allowed between delimiting parentheses is 100. If the option list is enclosed in apostrophes, however, the PARM parameter must be coded on one card.

Figure 27. Compiler Options

Table 7. DCB Assumptions for the (H) Compiler Data Sets

ddname	LRECL	RECFM	BLKSIZE[1]
SYSIN	80	FB	80[1]
SYSPRINT	137	VBA	141[1]
SYSLIN	80	FB	80[1]
SYSPUNCH	80	FB	80[1]
SYSUT1	105	FB	1050[2]
SYSUT2	1024-4096[3]	FB	1024-4096[2]

[1]This value may be increased by overriding the present value, either through a DCB BLKSIZE parameter in the associated DD statement or through the DSCB blocksize information for a preallocated data set -- if the overriding value is a multiple of LRECL.
[2]This value is fixed by the compiler and may not be overridden. If BLKSIZE is provided either through a DCB parameter in the DD statement or through a DSCB for a preallocated data set, it is ignored.
[3]The value is within this range, and the actual value is calculated during execution. The size of one of the tables used by the compiler (the address constant table) is compared with the tracksize of the device specified by SYSUT2, and the LRECL and BLKSIZE fields are equated to the smaller value.

LIST or NOLIST

The LIST option indicates that the object module listing is written in the data set specified by the SYSPRINT DD card. (The statements in the object module listing are in a pseudo assembler language format.) The NOLIST option indicates that no object module listing is written. A description of the object module listing is given in the section "System Output."

NAME=xxxxxx

The NAME option specifies the name (xxxxxx) assigned to a module (main program only) by the programmer. If NAME is not specified or the main program is not the first module in a compilation, the compiler assumes the name MAIN for the main program. The name of a subprogram is always the name specified in the SUBROUTINE or FUNCTION statement.

The name appears in the source listing, map, and object module listing. (See "Multiple Compilation Within a Job Step" in this section for additional considerations concerning the NAME option.)

LINECNT=xx

The LINECNT option specifies the maximum number of lines (xx) per page for a source listing. The specified number (xx) may be any in the range 01 to 99. If LINECNT is not specified, a default of 50 lines per page is provided. (The LINECNT option is effective only at compile time.)

SOURCE or NOSOURCE

The SOURCE option specifies that the source listing is written in the data set specified by the SYSPRINT DD statement. The NOSOURCE option indicates that no source listing is written. A description of the source listing is given in the section "System Output."

DECK or NODECK

The DECK option specifies that an object module card deck is punched as specified by the SYSPUNCH DD statement. The object module deck can be used as input to the linkage editor in a subsequent job. NODECK specifies that no object module deck is punched. A description of the deck is given in the section "System Output."

MAP or NOMAP

The MAP option specifies that a table of names is written in the data set specified by the SYSPRINT DD statement. The type and location of each name is listed. Included in the table of names for FORTRAN IV (H) is a table of labels appearing in the input stream. A description of the table is given in the section "System Output." The NOMAP option specifies that the table of names is not written.

LOAD or NOLOAD

The LOAD option indicates that the object module is written in the data set specified by the SYSLIN DD statement. This option must be used if the cataloged procedure to compile and linkage edit, or to compile, linkage edit, and execute is used; i.e., the object module is used as input to the linkage editor in the current job. The NOLOAD option indicates that the object module is not written on external storage. This option can only be used if the cataloged procedure to compile is used.

BCD or EBCDIC

The BCD option indicates that the source module is written in Binary Coded Decimal; EBCDIC indicates Extended Binary Coded Decimal Interchange Code. To intermix BCD and EBCDIC in the source module, BCD should be specified.

Notes:

1. If the EBCDIC option is selected, statement numbers passed as arguments must be coded as

   &n

   However, if the BCD option is selected, statement numbers passed as arguments must be coded as

   $n

   and the $ must not be used as an alphabetic character in the source module.

   (The n represents the statement number.)

2. The compiler does not support BCD characters either in literal data or as print control characters. Such characters are treated as EBCDIC. For example, a BCD + used as a carriage control character will not cause printing to continue on the same line. Programs keypunched in BCD, therefore, should be carefully screened if errors relating to literal data and print control characters are to be avoided.

ID or NOID

The ID option specifies that internal statement numbers (ISN) are to be generated for statements that call subroutine or contain external function references. Calls to IBCOM are not affected. An additional four bytes are required for each linkage.

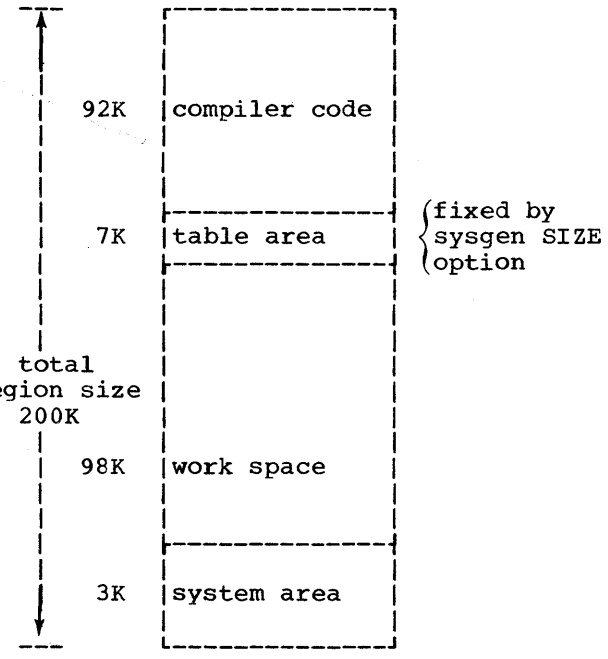
The ISN is used by the traceback in the event of an error in the called subprogram. See the discussion on "Load Module Output" in the section on "System Output."

```
┌------┐
|H ONLY|OPT={0|1|2}
└------┘
```

The OPT=0 option indicates that the compiler uses no optimizing techniques in producing an object module. The OPT=1 option indicates that the compiler treats each source module as a single program loop and optimizes the loop with regard to register allocation and branching. The OPT=2 option indicates that the compiler treats each source module as a collection of program loops and optimizes each loop with regard to register allocation, branching, common expression elimination, and replacement of redundant computations. The options OPT=1 and OPT=2 are described in more detail in the section "Appendix H: FORTRAN IV (H) Optimization Facilities."

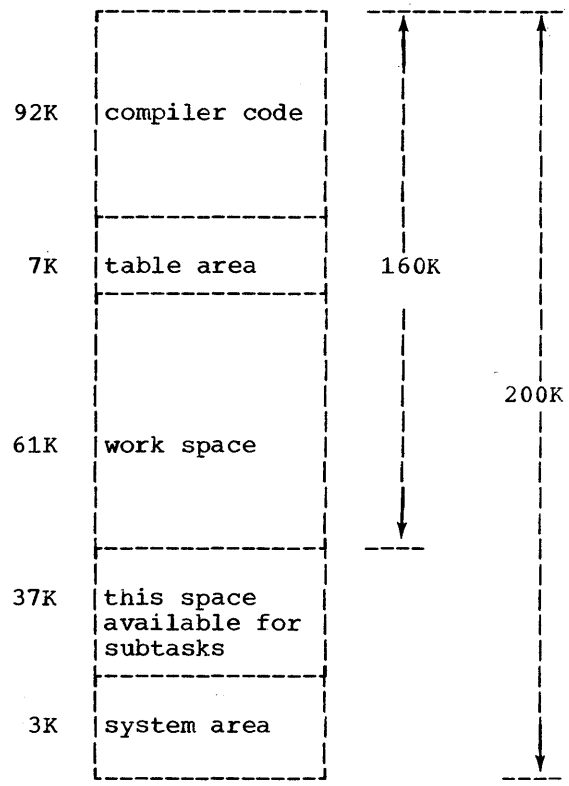
H ONLY | SIZE=nnnnK

The SIZE compiler option is used to limit the amount of main storage used by the compiler. In normal instances the total amount of main storage available to the compiler depends on the region size in an MVT environment, the partition size in an MFT environment, or the machine size in a PCP environment. Of this available space, compiler code takes up a fixed amount of storage, followed by compiler work table area; the size of the work table area is fixed at system generation time via the SIZE specification in the FORTRAN system generation macro. (The SIZE specification at SYSGEN time should not be confused with the SIZE compiler option.) The remainder of available main storage is used by the compiler for work space except for 3K bytes which are left for non-resident system routines. For example, a REGION specification of 200K would be typically allocated as follows:



In certain instances however, a programmer may wish to limit the amount of main storage used by the compiler. The prime example is when the FORTRAN H compiler is executed as the original task in a multitasking environment. Unless the compiler is confined to a portion of the region or partition, no subtasks could be created by the multitasking monitor.

The programmer may limit the amount of main storage available to the compiler by specifying SIZE=nnnnK in the operand field of the EXEC statement. The value nnnn can range from 115 to 9999 and is equal to the size of the compiler code, table area, and work space; the first two quantities are fixed, while the last is dependent on the SIZE value coded. Using the same REGION specification of 200K as in the first example, a compile-time specification of SIZE=160K would typically result in the following allocation:



The compiler prints two informational messages that refer to the specification of the SIZE option. The first appears as follows:

OPTIONS IN EFFECT option ,...,SIZE=nnnnK

The value listed for nnnn represents the user-specified SIZE value. If the SIZE option was not coded, nnn is listed as 0.

The second informational message is:

nnnnK BYTES OF CORE NOT USED

This message is produced if more than 10K bytes of available work space remained unused during compilation. If the SIZE option was indicated, the term nnnn indicates how much smaller the SIZE value could have been specified for optimal storage use. If the SIZE option was not coded, this message indicates how much smaller the region size could have been specified.

The size of the region or partition in which the compiler is running must be at least 10K bytes larger than the specified SIZE value. If the SIZE option is specified incorrectly, the compiler diagnostic message IEK410I (INVALID SIZE PARAMETER) is produced and the SIZE option is ignored.

H ONLY EDIT or NOEDIT

The EDIT option specifies that a structured source listing is written in the data set specified by the SYSPRINT DD statement. This listing indicates the loop structure and the logical continuity of the source program. If this option is used, OPT=2 must be specified and a DD statement with the name SYSUT1 must be supplied. The following is a typical DD statement for a utility data set:

```
//SYSUT1   DD DSNAME=&UT1,UNIT=SYSSQ,
               SPACE=(TRK,(40))
```

&UT1
specifies a temporary data set.

UNIT=SYSSQ
specifies that the data set is to reside in a sequential device class.

SPACE=(TRK,(40))
specifies that if the data set is assigned to a direct access device, 40 tracks are to be allocated to the data set.

The NOEDIT option specifies that no structured source listing is written. A description of the structured source listing is given in the section "System Output."

H ONLY XREF or NOXREF

The XREF option specifies that a cross reference listing of variables and labels is written in the data set specified by the SYSPRINT DD statement. This listing indicates the internal statement number of every statement in which a variable or label is used. If this option is specified, a DD statement with the name SYSUT2 must be supplied. The NOXREF option specifies that no cross reference listing is written. A description of the compiler cross reference listing is given in the section "System Output."

Note: The default compiler options shown in this publication are standard IBM defaults; however, during system generation, an installation can choose its own set of default options.

Multiple Compilation Within a Job Step

Several compilations may be performed within one job step. The compiler recognizes the FORTRAN END statement in a source module, compiles the program, and determines if another source module follows the END statement. If there is another source module, another compilation is initiated (see Figure 28).

```
//JOBRA JOB  ,'FORTRAN PROG'
//STEP1 EXEC FORTGC or FORTHC
//FORT.SYSIN DD *
   1 READ (8,10)A,B,C
      .
      .
      .
     END
     SUBROUTINE CALC
      .
      .
     END
/*
```

Figure 28. Multiple Compilation Within a Job Step

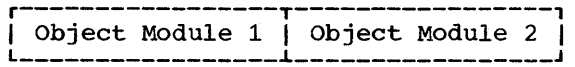
Only one EXEC statement may be used to initiate a job step; therefore, compiler options can be stated only once for all compilations in a job step.

In a multiple compilation, only the first program (if it is a main program) is given the name specified in the NAME option; all subsequent main programs are given the name MAIN. If the first program is a subprogram, the name specified in the NAME option is not used. If the NAME option is not specified, all main programs in a multiple compilation are given the name MAIN. For example, in the multiple compilation,

```
//MULCOM   JOB
// EXEC FORTGC or
     FORTHC,PARM.FORT='NAME=IOR'
//FORT.SYSIN DD *
 READ(1,10)ALP,BETA
   .
   .
   .
  END
 SUBROUTINE INVERT(A,B)
   .
   .
   .
 END
 READ(5)P,Q,R
   .
   .
   .
 END
/*
```

the first main program is given the name IOR; the third program is given the name MAIN. The second program is assigned the name INVERT.

When a multiple compilation is performed, the SYSLIN data set contains all the object modules, because only one SYSLIN DD statement may be supplied for compiler output. If tape or direct-access output is specified for the compiler, the object modules are written sequentially on the volume:



LINKAGE EDITOR PROCESSING

The linkage editor processes FORTRAN object modules, resolves any references to subprograms, and prepares a load module for execution.[1] To communicate with the linkage editor, the programmer supplies an EXEC statement and DD statements that define all required data sets; he may also supply linkage editor control statements.

Linkage Editor Names

Five linkage editor programs are available with the operating system. The program names for these linkage editors and the minimum storage in which they are designed to operate are:

	PCP and MFT	MVT-System
IEWLE150	15K	24K
IEWLE180	18K	26K
IEWLF440	44K	54K
IEWLF880	88K	96K
IEWLF128	128K	136K

(Where K=1024 Bytes)

All facilities described for the linkage editor in this publication are available with all five linkage editors, except that blocking of primary input/output is available only with the higher level linkage editors: IEWLF440, IEWLF880, and IEWLF128.

---

[1]Another IBM-supplied program, the loader, combines -- into one job step -- the functions of the linkage editor with execution of the edited module. Details on the use of the loader can be found in the Linkage Editor and Loader publication.

For simpler programming, the linkage editors have been assigned the alias program name IEWL. If the programmer specifies the parameter

```
PGM=IEWL
```

in the EXEC statement, the highest level linkage editor provided in the installation's operating system is executed. If he wants to execute a specific linkage editor, he must specify the specific program name of that linkage editor.

Linkage Editor Input and Output

There are two types of input to the linkage editor: primary and secondary.

Primary input is a sequential data set that contains object modules and linkage editor control statements. (A member of a PDS can be the primary input.) Any external references among object modules in the primary input are resolved by the linkage editor as the primary input is processed. Furthermore, the primary input can contain references to the secondary input. These references are linkage editor control statements and/or external references in the FORTRAN modules.

Secondary input resolves the references and is separated into two types: automatic call library and additional input specified by the programmer. The automatic call library should always be the FORTRAN library (SYS1.FORTLIB), which is the PDS that contains the FORTRAN library subprograms. Through the use of DD statements the automatic call library can be concatenated with other partitioned data sets. Three types of additional input may be specified by the programmer:

- An object module used as the main program in the load module being constructed. This object module, which can be accompanied by linkage editor control statements, is either a member of a PDS or is a sequential data set. The first record in the primary input data set must be a linkage editor INCLUDE control statement that tells the linkage editor to insert the main program.
- An object module or a load module used to resolve external references made in another module. The object module, which can be accompanied by linkage editor control statements, is a sequential data set or is a member of a PDS. The load module, which is a

member of a PDS, cannot be accompanied by linkage editor control statements. An INCLUDE statement that defines the data set must be given.

- A module used to resolve external references made in another module. The load module or object module, which can be accompanied by linkage editor control statements, is a member of PDS. A linkage editor LIBRARY control statement that defines the data set to the linkage editor must be given.

In addition, the secondary input can contain external references and linkage editor control statements. The automatic call library and any of the three types of additional input may be used to resolve references in the secondary input.

The load module created by the linkage editor is always placed in a PDS. Error messages and optional diagnostic messages are written on intermediate storage or a printer. In addition, a work data set is required by the linkage editor to do its processing. Figure 29 shows the I/O flow in linkage editor processing.

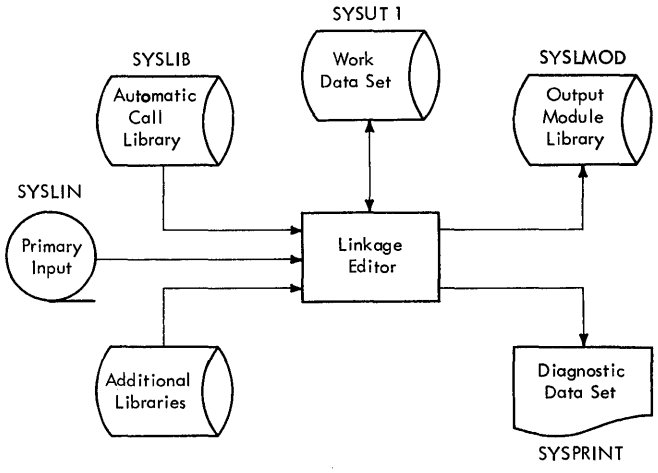


Figure 29. Linkage Editor Input and Output

Any data sets specified by SYSLIB or SYSLMOD must be partitioned data sets. The ddname for the DD statement that retrieves any additional libraries is written in INCLUDE and LIBRARY statements and is not fixed by the linkage editor.

Linkage Editor ddnames and Device Classes

The programmer communicates data set information to the linkage editor through DD statements identified by specific ddnames (similar to the ddnames used by the compiler). The ddnames, functions, and requirements for data sets are shown in Table 8.

The device classes used by the compiler (see Table 4) must also be used with the linkage editor. The data sets used by linkage editor may be assigned to the device classes listed in Table 9.

Table 8. Linkage Editor ddnames

ddname	Function	Device Requirements
SYSLIN	primary input data, normally the output of the compiler	•direct access •magnetic tape •card reader
SYSLIB	automatic call library (SYS1.FORTLIB)	•direct access
SYSUT1	work data set	•direct access
SYSPRINT	diagnostic messages	•printer •intermediate storage device
SYSLMOD	output data set for the load module	•direct access
user-specified	additional libraries and object modules	•direct access •magnetic tape

Table 9. Correspondence Between Linkage Editor ddnames and Device Classes

ddname	Possible Device Classes
SYSLIN	SYSSQ,SYSDA,or the input stream device (specified by DD * or DD DATA), or a device specified as the card reader
SYSLIB	SYSDA
SYSUT1	SYSDA
SYSLMOD	SYSDA
SYSPRINT	A,SYSSQ
user-specified	SYSDA,SYSSQ

Additional Input

The INCLUDE and LIBRARY statements are used to specify additional secondary input to the linkage editor. Modules neither specified by INCLUDE or LIBRARY statements nor contained in the primary input are retrieved from the automatic call library.

INCLUDE Statement:

Operation	Operand
INCLUDE	ddname[(member-name [,member-name]...)] [,ddname[(member-name [,member-name]...)]]...

The INCLUDE statement is used to include either members of additional libraries or a sequential data set. The "ddname" specifies a DD statement that defines either a library containing object modules and control statements or just load modules, or defines a sequential data set containing object modules and control statements. The "member name" is not used when a sequential data set is specified.

The linkage editor inserts the object module or load module in the output load module when the INCLUDE statement is encountered.

LIBRARY Statement:

Operation	Operand
LIBRARY	ddname(member-name [,member-name]...) [,ddname(member-name [,member-name]...)]...

The LIBRARY statement is used to include members of additional libraries. The "ddname" must be the name of a DD statement that specifies a library that contains either object modules and linkage editor control statements, or just load modules. The "member name" is an external reference that is unresolved after primary input processing is complete.

The LIBRARY statement differs from the INCLUDE statement: external references specified in the LIBRARY statement are not resolved until all other processing, except references reserved for the automatic call library, is completed by the linkage editor. (INCLUDE statements resolve external references when the INCLUDE statement is encountered.)

Example: Two subprograms, SUB1 and SUB2, and a main program, MAIN, are compiled by separate job steps. In addition to the FORTRAN library, a private library, MYLIB, is used to resolve external references to the symbols X, Y, and Z. Each of the object modules is placed in a sequential data set by the compiler, and passed to the linkage editor job step.

Figure 30 shows the control statements for this job. (Cataloged procedures are not used.) In this job, an additional library, MYLIB, is specified by the LIBRARY statement and the ADDLIB DD statement. SUB1 and SUB2 are included in the load module by the INCLUDE statements and the DD statements DD1 and DD2. The linkage editor input stream, SYSLIN, is two concatenated data sets: the first data set is the sequential data set &GOFILE which contains the main program; the second data set is the two INCLUDE statements and the LIBRARY statement. After linkage editor execution, the load module is placed in the PDS PROGLIB and given the name CALC.

Note: This example shows the use of the FORTRAN IV (H) compiler (program name IEKAA00). An example showing the use of the (G) compiler would be identical except for program name; PGM=IEYFORT would be coded, where appropriate, instead of PGM=IEKAA00.

```
//JOBX      JOB
//STEP1     EXEC      PGM=IEKAA00,PARM='NAME=MAIN,LOAD'
                          .
                          .
                          .
//SYSLIN    DD        DSNAME=&GOFILE,DISP=(,PASS),UNIT=SYSSQ
//SYSIN     DD        *
            Source module for MAIN
/*
//STEP2     EXEC      PGM=IEKAA00,PARM='NAME=SUB1,LOAD'
                          .
                          .
                          .
//SYSLIN    DD        DSNAME=&SUBPROG1,DISP=(,PASS),UNIT=SYSSQ
//SYSIN     DD        *
            Source module for SUB1
/*
//STEP3     EXEC      PGM=IEKAA00,PARM='NAME=SUB2,LOAD'
                          .
                          .
                          .
//SYSLIN    DD        DSNAME=&SUBPROG2,DISP=(,PASS),UNIT=SYSSQ
//SYSIN     DD        *
            Source module for SUB2
/*
//STEP4     EXEC      PGM=IEWL
                          .
                          .
                          .
//SYSLIB    DD        DSNAME=SYS1.FORTLIB,DISP=OLD
//SYSLMOD   DD        DSNAME=PROGLIB(CALC),UNIT=SYSDA
//ADDLIB    DD        DSNAME=MYLIB,DISP=OLD
//DD1       DD        DSNAME=*.STEP2.SYSLIN,DISP=OLD
//DD2       DD        DSNAME=*.STEP3.SYSLIN,DISP=OLD
//SYSLIN    DD        DSNAME=*.STEP1.SYSLIN,DISP=OLD
//          DD        *
            INCLUDE   DD1
            INCLUDE   DD2
            LIBRARY   ADDLIB(X,Y,Z)
/*
```

Figure 30. Linkage Editor Example -- (H) Compiler

Linkage Editor Priority

If modules with the same name appear in a single data set, only the module encountered first is inserted in the output load module.

Other Linkage Editor Control Statements

In addition to the LIBRARY and INCLUDE statements, other control statements are available for use with the linkage editor. These statements enable the user to: specify different names for load modules (ALIAS), replace modules within a load module (REPLACE), change program names (CHANGE), and name entry points (ENTRY). In addition, two statements (OVERLAY and INSERT) enable the programmer to overlay load modules. For a detailed description of these control statements, see the Linkage Editor and Loader publication, Order No. GC28-6538.

Options for Linkage Editor Processing

The linkage editor options are specified in an EXEC statement. The options that are most applicable to the FORTRAN programmer are:

```
{PARM          }   [MAP ]
{PARM.procstep}=([XREF] [,LET] [,NCAL]
                 [,LIST])
```

MAP or XREF

The MAP option informs the linkage editor to produce a map of the load module; this map indicates the relative location and length of main programs and subprograms. If XREF is specified, a map of the load module is produced and a cross reference list indicating all external references in each main program and subprogram is generated. The map or map and cross reference list are written in the data set specified by the SYSPRINT DD statement. If neither option is specified, the system generation option for the procedure for the linkage editor is put into effect. Descriptions of the map and cross reference listing are given in "System Output."

LET

The LET option informs the linkage editor to mark the load module executable even though error conditions, which could cause execution to fail, have been detected.

NCAL

The NCAL option informs the linkage editor that the libraries specified in the SYSLIB DD statement or specified in LIBRARY statements are not used to resolve external references. (The SYSLIB DD statement need not be specified.) The subprograms in the libraries are not inserted in the load module; however, the load module is marked executable.

LIST

The LIST option indicates that linkage editor control statements are listed in card-image format in the diagnostic output data set specified by the SYSPRINT DD statement.

Other options can also be specified for the linkage editor. For a detailed description of all linkage editor options, see the Linkage Editor and Loader publication, Order No. GC28-6538.

LOAD MODULE EXECUTION

When "PGM=program name" is used to indicate the execution of a load module, the module must be in either the system library (SYS1.LINKLIB) or a private library. When the module is in a private library, a JOBLIB DD statement must be supplied to indicate the name of the private library. For example, assume that the load modules CALC and ALGBRA in the library MATH and the load module MATRIX in the library MATRICES are executed in the following job:

```
//JOBN JOB 00,FORTPROG
//JOBLIB DD DSNAME=MATH,DISP=(OLD,PASS)
// DD DSNAME=MATRICES,DISP=(OLD,PASS)
//STEP1 EXEC PGM=CALC
         .
         .
         .
//STEP2 EXEC PGM=MATRIX
         .
         .
         .
//STEP3 EXEC PGM=ALGBRA
         .
         .
         .
```

The JOBLIB DD statement concatenates the private library MATH with the system library. The private library MATRICES is concatenated with the system library, by concatenating the second DD statement with the JOBLIB DD statement.

Execution ddnames

In the source module, data set reference numbers are used to identify data sets. Data sets processed by a FORTRAN load module must be defined by DD statements. The correspondence between a data set reference number and a DD statement is made by a ddname.

The ddname format that must be used for load module execution is

```
FTxxFyyy
```

where xx is the data set reference number, and yyy is a FORTRAN sequence number.

Data Set Reference Number (xx): When the system is generated, the upper limit for data set reference numbers is specified by the installation; it must not exceed 99. This upper limit does not correspond to the number of input/output devices.

If an installation specifies an upper limit of 99 for its data set reference numbers, the ddnames and data set reference numbers correspond as shown in Table 10. Note that 0 is not a valid data set reference number.

FORTRAN Sequence Number (yyy): The FORTRAN sequence number is used to refer to separate data sets that are read or written using the same data set reference number. For the first data set, the sequence number is 001; for the second 002; etc. This sequence number is incremented when (1) an END FILE statement is executed and a subsequent WRITE is issued with the same data set reference number or (2) the "END=" exit is taken following a READ and a subsequent READ or WRITE is issued with the same data set reference number.

A DD statement with the required ddname must be supplied every time the WRITE, END FILE, WRITE sequence occurs. If the FORTRAN statements in the following example are executed, DD statements with the ddnames indicated by the arrows must be supplied for the corresponding WRITE statements.

```
   Statements                                  ddnames

15 FORMAT(3F10.3,I7)
10 FORMAT(3F10.3)
   DO 20 I=1,J
     .
     .
     .
20 WRITE(17,10)A,B,C ----------->  FT17F001
     .
     .
     .
   END FILE 17
   DO 30 I=1,N
     .
     .
     .
30 WRITE(17,15)X,Y,Z,K --------->  FT17F002
   END FILE 17
   DO 40 I=1,M,2
     .
     .
     .
40 WRITE(17,10)A,B,C ----------->  FT17F003
     .
     .
     .
   END FILE 17
```

If the preceding instructions are used to write a tape, the output tape (unlabeled) has the appearance shown in Figure 31. The tapemarks are written by execution of the ENDFILE statements. Successful execution of ENDFILE always includes writing an end-of-data indicator.

Reference Numbers for Data Sets Specified in DEFINE FILE Statements

The characteristics of any data set to be used during a direct-access input/output operation must be described by a DEFINE FILE statement.

The data set reference number specified in any DEFINE FILE statement may refer to only one data set. In other words, the method described previously concerning references to separate data sets that are read or written using the same data set reference number is prohibited. For example, the statement

```
DEFINE FILE 2(50,100,L,I2)
```

establishes a data set reference number of 02. All subsequent input/output statements must refer to only one data set with the DD name of FT02F001. (For a more detailed explanation of the DEFINE FILE statement, refer to the FORTRAN IV Language publication, Order No. GC28-6515.)

Retrieving Data Sets Written with Varying FORTRAN Sequence Numbers

To retrieve the data sets shown in Figure 31, the data set sequence number in the LABEL parameter must be supplied in the DD statement. The LABEL parameter is described in detail in the section "Creating Data Sets."

```
LABEL=([data-set-sequence-number]{,NL }  )
                                 {,SL }
                                 {,BLP}
```

The "data set sequence number" indicates the position of the data set on a sequential volume. (This sequence number is cataloged.) For the first data set on the volume, the data set sequence number is 1; for the second, it is 2; etc.

Table 10. Load Module ddnames

Data Set Reference Numbers	ddnames
1	FT01Fyyy
2	FT02Fyyy
.	.
.	.
.	.
13	FT13Fyyy
.	.
.	.
.	.
99	FT99Fyyy

If one of the data sets shown in Figure 31 is read in the same job step in which it is written, an END FILE statement must be issued after the last WRITE instruction. If the data set is to be read by the same data set reference number, DD statement FT17F004 is used to read the data set. The execution of a READ statement following an END FILE increments the FORTRAN sequence number by 1. For example, the following DD statements are used to write the three data sets shown in Figure 31 and then read the second data set:

```
//FT17F001 DD UNIT=TAPE,LABEL=(,NL),       X
//    DISP=(,PASS)
//FT17F002 DD UNIT=TAPE,LABEL=(2,NL),      X
//    VOLUME=REF=*.FT17F001
//FT17F003 DD UNIT=TAPE,LABEL=(3,NL),      X
//    VOLUME=REF=*.FT17F001
//FT17F004 DD VOLUME=REF=*.FT17F001,       X
//    DISP=OLD,LABEL=(2,NL),               X
//    DSNAME=*.FT17F002,UNIT=TAPE
```

The VOLUME parameter indicates that the data set resides on the same volume as the data set defined by DD statement FT17F001. DD statement FT17F004 refers to the data set defined by DD statement FT17F002.

If the data set is read by a different data set reference number, for example, data set reference number 18; then, the DD statement FT17F004 is replaced by the statement.

```
//FT18F001 DD VOLUME=REF=*.FT17F002,       X
//    DISP=OLD,LABEL=(2,NL)
```

If the data sets shown in Figure 31 are cataloged for later reading, and if the following DD statements are used to write the data sets,

```
//FT17F001 DD DSNAME=N1,LABEL=(1,NL),      X
//    DISP=(,CATLG),UNIT=TAPE,             X
//    VOLUME=SER=163K
//FT17F002 DD DSNAME=N2,LABEL=(2,NL),      X
//    DISP=(,CATLG),VOLUME=REF=*.FT17F001
//FT17F003 DD DSNAME=N3,LABEL=(3,NL),      X
//    DISP=(,CATLG),VOLUME=REF=*.FT17F002
```

the information necessary to retrieve the data sets is the DSNAME, the LABEL, and the DISP parameters. For example, if data set reference number 10 is used to retrieve data set N1, the following DD statement is required.

```
//FT10F001 DD DSNAME=N1,DISP=OLD,          X
//            LABEL=(,NL)
```

If the data set is not cataloged and then retrieved in a later job, the VOLUME, UNIT, and LABEL information is needed to retrieve the data set. When the data set is created, the programmer must assign a specific volume to it.

Assume the data sets shown in Figure 28 were assigned the volume identified by the volume serial number A11111 when the data sets were created. If the second data set written on the volume is retrieved by data set reference number 10 in a later job, the following DD statement is needed.

```
//FT10F001 DD VOLUME=SER=A11111,DISP=OLD, X
//              LABEL=(2,NL),UNIT=SYSSQ
```

END Exit: Data sets written using the same data set reference number can be retrieved in the same job or job step by using a facility provided in the FORTRAN language -- the "END=" exit in a READ statement. After the last data set is written and the END FILE is executed, a REWIND may be issued. A subsequent READ using the same data set reference number resets the FORTRAN sequence number to 001. When the last record of a data set has been read, an additional READ causes the END exit to be taken. On the next READ, the sequence number is incremented by 1. The data sets shown in Figure 31 can be read by using the following sequence of statements.

Note: The DD statements used to create the data sets also suffice for retrieving the data sets. No additional DD statements are required.

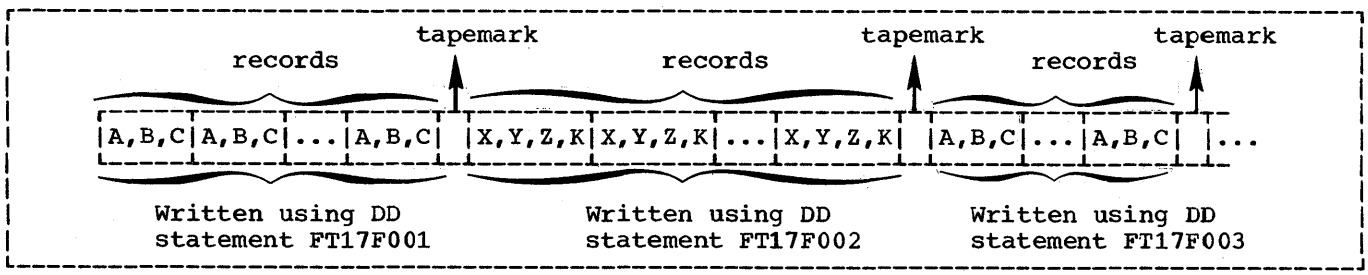


Figure 31. Tape Output for Several Data Sets Using Same Data Set Reference Number

```
    REWIND 17
       .
       .
       .
100 READ(17,10,END=200)A,B,C  ---->FT17F001
       .
       .
       .
    GO TO 100
       .
       .
       .
200 READ(17,15,END=300)X,Y,Z,K---->FT17F002
       .
       .
       .
    GO TO 200
       .
       .
       .
300 READ(17,10,END=350)A,B,C  ---->FT17F003
       .
       .
       .
    GO TO 300
       .
       .
       .
350 REWIND 17
```

Concatenation: The data sets shown in Figure 31 can be concatenated and read as a single data set. The information necessary (assume cataloged data sets) to retrieve the data sets is the DSNAME, LABEL, and DISP parameters. For example, if data set reference number 16 is used to retrieve the data sets, the following DD statements are required.

```
//FT16F001 DD  DSNAME=N1,DISP=OLD,          X
//              LABEL=(,NL)
//         DD  DSNAME=N2,DISP=OLD,          X
//              LABEL=(2,NL)
//         DD  DSNAME=N3,DISP=OLD,          X
//              LABEL=(3,NL)
```

The ERR option the FORTRAN READ statement may be used to give control to the problem program if an uncorrectable I/O error occurs on a magnetic tape or direct-access device. This parameter is not effective for data sets on unit record devices.

Note: Concatenation of data sets with unlike attributes is not supported. Partitioned data sets with like attributes may be concatenated for input only. Concatenation of two or more members of the same PDS is not supported.

Partitioned Data Set Processing

FORTRAN load modules may access two or more members of the same partitioned data set (PDS); however, only sequential processing is permitted. The PDS must be closed for one member before attempting to read or write another member.

PDS Processing Using "END=" Option: One method of sequentially processing two or more members of the same PDS is by using the "END=" option in a FORTRAN sequential READ statement. When the "END=" option is executed and a subsequent READ or WRITE statement is issued with the same data set reference number, the FORTRAN sequence number is incremented by one. This allows another member of the PDS referenced by the same data set reference number to be processed.

The following FORTRAN program illustrates how this method is put into effect:

```
          INTEGER *4 X(20),Y(20)
10        READ (2,1,END=98) X
1         FORMAT (20A4)
          GO TO 10
98        READ (2,1,END=99) Y
          GO TO 98
99        WRITE (6,2) X,Y
          STOP
          END
```

Execution of statement 10 results in the processing of the first PDS member which is referenced by the FORTRAN sequence number 001. If this member has the name MEMBER1 and resides in the cataloged partitioned data set named PDS, the DD statement that must be supplied is:

```
//FT02F001  DD  DSN=PDS(MEMBER1),
                LABEL(,,,IN),DISP=OLD
```

When the "END=" option is executed in statement 10 and the next READ statement, statement 98, is encountered, the FORTRAN sequence number becomes 002. This closes the PDS for the first member. Another member may then be processed. If its name is MEMBER5, the DD statement that must be supplied is:

```
//FT02F002  DD  DSN=PDS(MEMBER5),
                LABEL=(,,,IN),DISP=OLD
```

Note: For PDS processing, the "END=" option specification is the only method of incrementing the FORTRAN sequence number. The END FILE statement methods described earlier in the section "FORTRAN Sequence Number" cannot be implemented since END FILE statements cannot be used for partitioned data sets.

PDS Processing Using REWIND: A second method of processing two or more members of the same PDS is the use of the REWIND statement in the FORTRAN program. This statement should be of the form:

```
REWIND    a
```

where a is an unsigned integer constant or variable representing a data set reference number. Execution of the REWIND statement closes the data set represented by the integer a. Any subsequent READ or WRITE statement opens the data set again.

The following example illustrates the use of the REWIND statement for the reading of two members of the same PDS:

```
        INTEGER *4 X(20),Y(20)
        READ (2,1) X
        REWIND 2
        READ (3,1) Y
        WRITE (6,2) X,Y
1       FORMAT (20A4)
2       FORMAT (' ',20A4)
        STOP
        END
```

Execution of the first READ statement results in the processing of the first PDS member which is referenced by the FORTRAN sequence number 001. If the member has the name MEMBER1 and resides in the cataloged partitioned data set named PDS, the DD statement that must be supplied is:

```
//FT02F001  DD  DSN=PDS(MEMBER1),
                LABEL=(,,,IN),DISP=OLD
```

When the REWIND statement is executed, the PDS is closed for MEMBER1. The next READ statement reopens the data set for another PDS member. If the next member name is MEMBER5, the DD statement that must be supplied is:

```
//FT03F001  DD  DSN=PDS(MEMBER5),
                LABEL=(,,,IN),DISP=OLD
```

The following example illustrates the use of the REWIND statement for the writing of two PDS members:

```
        INTEGER *4 X(20)
        DO 3 I=1,20
3       X(I)=I
        WRITE (2,1)X
1       FORMAT (' ',20A4)
        REWIND 2
        WRITE (3,1)X
        STOP
        END
```

Here, the use of the REWIND statement for the data set reference number 2 closes the PDS. It is reopened for the next member by the reference to data set reference number 3. The DD statements that must be supplied are the same as those in the previous example; however, LABEL=(,,,OUT) must be specified to indicate output processing.

REWIND and BACKSPACE Statements

The REWIND and BACKSPACE statements force execution of positioning operations by the control program.

A REWIND statement instructs the control program to position the volume on the device so that the next record read or written is the first record transmitted for that data set reference number on that volume, irrespective of data set sequence numbers. The space acquired dynamically for I/O buffers for a data set is released as part of the REWIND operation. For this reason, a program that uses many data sets may conserve main storage by issuing REWIND statements after processing is completed. Since a REWIND statement closes the data set, any subsequent READ or WRITE statement opens the data set again. For a data set where DISP=MOD was specified, the READ or WRITE statement causes positioning at the end of the data set before the statement is executed.

The BACKSPACE statement causes a backward skip of one logical record for each BACKSPACE issued. The records may be blocked or unblocked and of any valid type (F,U,V). Note that the default selection for FORTRAN data sets is U-type (undefined)records which can not be blocked. If a BACKSPACE statement requests backward movement past the load point or first record of the data set, that request is ignored. Since BACKSPACE is not supported across reels of a multi-reel data set on tape, a BACKSPACE request made under such conditions is treated as an attempt to move backward past the load point. The user is not made aware of input/output errors that have occurred during a BACKSPACE operation until he issues his next READ or WRITE request.

Notes:

1. REWIND, BACKSPACE or END FILE statements specified for data sets defined in direct-access statements are ignored.

2. BACKSPACE statements should not be directed to the data set defined as SYSIN.

3. At end-of-file, if the programmer wishes to access the file, he should issue at least two BACKSPACE statements. The first statement causes his file to be positioned before the tapemark; the second positions the file at the beginning of the last logical record.

Error Message Data Set

When the system is generated, the installation assigns a data set reference number so that execution error messages and information for traceback, DUMPs, and PDUMPs can be written on a data set. This data set is automatically opened at library initialization time. The programmer must define a data set, using a DD statement with the ddname for that data set reference number. This data set should be defined using the SYSOUT=A parameter. If the error message data set is on tape, the DD statement should contain DCB parameters for BLKSIZE=133 and RECFM=UA. (The System Generation publication, Order No. GC28-6554, explains the method of assigning the data set reference number. See the description of the OBJERR parameter in the section on the FORTLIB macro instruction.)

Execution Device Classes

For load module execution, the programmer can use the same names assigned to device classes used by the compiler (shown in Table 4). However, additional names for specific devices and device classes can be assigned by the installation. The programmer can choose which device to use for his data sets, and can specify the name of the device or class of devices in the UNIT parameter of the DD statement.

DCB Parameter

The DCB parameter may be specified for data sets when a load module is executed. For information concerning the DCB parameter, see the section "Creating Data Sets."

LOADER PROCESSING

The loader combines into one job step the functions of the linkage editor with execution of the edited module. It processes FORTRAN object or load modules, resolves any references to subprograms, loads the module, and executes the loaded module. The loader does not produce load modules for program libraries. For detailed information on the loader, see the Linkage Editor and Loader publication.

Loader Name

The program name for the loader is IEWLDRGO. An alias program name, LOADER, has been assigned to the loader for simpler programming. If the loader is executed as a job step, the parameter PGM=LOADER or PGM=IEWLDRGO is used in the EXEC statement of that job step.

Loader Input and Output

The primary input to the loader is in the form of object modules and/or load modules. While processing an input module, the loader finds any references to subprograms in the input module and resolves them.

The output of the loader consists of error and diagnostic messages and an optional storage map of the loaded program. The output is written on either an intermediate storage device or a printer. The loader does not require intermediate work data sets.

Loader ddnames and Device Classes

The programmer communicates data set information to the loader through DD statements identified by specific ddnames. (These ddnames can be changed during system generation.) The ddnames, functions, and requirements for data sets are shown in Table 11. Only the SYSLIN DD statement is required; the other two are optional. In addition, any DD statements required for execution of the loaded program must be included in the job step. (These DD statements are described in the section "Load Module Execution.")

Table 11. Loader ddnames

ddname	Function	Device Requirements
SYSLIN	Primary input data, normally the output of the compiler.	direct access magnetic tape card reader
SYSLIB	Automatic call library (SYS1.FORTLIB)	direct access
SYSLOUT	Diagnostic messages and storage map.	printer intermediate storage device
user-specified	Data required for execution of the loaded program.	any device

The device classes used by the compiler (see Table 3) must also be used with the loader. The data sets used by the loader may be assigned to the device classes listed in Table 12.

Table 12. Correspondence Between Loader ddnames and Device Classes

ddname	Possible Device Classes
SYSLIN	SYSSQ, SYSDA, or the input stream device (specified by DD * or DD DATA), or a device specified as the card reader.
SYSLIB	SYSDA
SYSPRINT	A, SYSSQ
user-specified	SYSDA, SYSSQ

Loader Priority

If modules with the same name appear in the input to the loader, the loader accepts only the first module which appears.

Options for Loader Processing

The loader and loaded program options are specified in the PARM field of the EXEC statement as follows:

```
{PARM          } = ( {MAP  } {,CALL  }
{PARM.procstep }     {NOMAP} {,NOCALL}

                     {,LET  } {,SIZE=100K}
                     {,NOLET} {,SIZE=size}

                   [,EP=name] {,PRINT  })
                              {,NOPRINT}
```

MAP or NOMAP

The MAP option informs the loader to produce a map of the loaded program; this map lists external names and their absolute storage addresses on the data set specified by the SYSLOUT DD statement. (If the SYSLOUT DD statement is not used in the input deck, this option is ignored.) The NOMAP option specifies that the map of the loaded program is not to be written.

CALL or NOCALL or NCAL

The CALL option specifies that an automatic search of the data set specified on the SYSLIB DD statement is to be made. (If the SYSLIB DD statement is not in the input deck, this option is ignored.) The NOCALL or NCAL option specifies that an automatic search of the SYSLIB data set is not to be made.

LET or NOLET

The LET option informs the loader to try to execute the object program even though a severity 2 error condition is found. (A severity 2 error condition is one that could make execution of the loaded program impossible.) The NOLET option informs the loader not to try to execute the loaded program when a severity 2 error condition is found.

SIZE=size

The SIZE option specifies the size, in bytes, of dynamic main storage that can be used by the loader. The size of the program to be loaded must be included in this figure.

EP=name

The EP option specifies the external name to be assigned as the entry point of the loaded program.

PRINT or NOPRINT

The PRINT option informs the loader to produce diagnostic messages on the SYSLOUT data set. The NOPRINT option informs the loader not to produce diagnostic messages on the SYSLOUT data set; SYSLOUT will not be opened.

Note: The default options are: NOMAP, CALL, NOLET, SIZE=100K, and PRINT. Other default options, however, can be specified with the LOADER macro instruction during system generation.

The following are examples of the EXEC statement specified for loader processing:

```
//LOAD   EXEC   PGM=LOADER
//LOAD   EXEC   PGM=IEWLDRGO,PARM=(MAP,      X
//                'EP=FIRST')
//LOAD   EXEC   PGM=IEWLDRGO,PARM=(MAP,LET)
//LOAD   EXEC   PGM=LOADER,PARM=NOPRINT
```

Programming Example

Figure 32 shows the control statements used in a job invoking the loader. Two subprograms, SUB1 and SUB2, and a main program, MAIN, are compiled in separate job steps. In addition to the FORTRAN library, a private library, MYLIB, is used to resolve external references. Each of the object modules is placed in a sequential data set by the compiler and passed to the loader step.

It should be noted that cataloged procedures are not used in this job. The private library, MYLIB, is concatenated with the SYSLIB DD statement. SUB1 and SUB2 are included in the program to be loaded by concatenating them with the SYSLIN DD statement. The loaded program requires the FT01F001 and FT10F001 DD statements for execution.

```
//JOBX     JOB
//STEP1    EXEC PGM=IEKAA00,                X
//              PARM='NAME=MAIN,LOAD'
                .
                .
                .
//SYSLIN   DD   DSNAME=&GOFILE,             X
//              DISP=(,PASS),               X
//              UNIT=SYSSQ
//SYSIN    DD   *
     Source Module for MAIN
/*
//STEP2    EXEC PGM=IEKAA00,                X
//              PARM='NAME=SUB1,LOAD'
                .
                .
                .
//SYSLIN   DD   DSNAME=&SUBPROG1,           X
//              DISP=(,PASS),               X
//              UNIT=SYSSQ
//SYSIN    DD   *
     Source Module for SUB1
/*
//STEP3    EXEC PGM=IEKAA00,                X
//              PARM='NAME=SUB2,LOAD'
                .
                .
                .
//SYSLIN   DD   DSNAME=&SUBPROG2,           X
//              DISP=(,PASS),               X
//              UNIT=SYSSQ
//SYSIN    DD   *
     Source Module for SUB2
/*
//STEP4    EXEC PGM=LOADER
//SYSLOUT  DD   SYSOUT=A
//SYSLIB   DD   DSNAME=SYS1.FORTLIB,        X
//              DISP=SHR
//         DD   DSNAME=MYLIB,DISP=SHR
//SYSLIN   DD   DSNAME=*.STEP1.SYSLIN,      X
//              DISP=OLD
//         DD   DSNAME=*.STEP2.SYSLIN,      X
//              DISP=OLD
//         DD   DSNAME=*.STEP3.SYSLIN,      X
//              DISP=OLD
//FT01F001 DD   DSNAME=PARAMS,DISP=OLD
//FT10F001 DD   SYSOUT=A
/*
```

Figure 32. Loader Example

DEDICATED WORK DATA SETS

Under MVT, installations can provide preallocated or dedicated work data sets. If an installation has provided these data sets, the programmer can use them as an alternative to creating his work data sets. Use of dedicated work data sets is more efficient than creating work data sets by specifying a disposition of NEW,DELETE on the work data set DD statement.

The system allocates these data sets at start initiator time (when input/output device requirements for a job step are analyzed by the system). The number of data sets to be allocated is based on the number of work data set DD statements in a cataloged procedure known as the initiator procedure. The initiator procedure is supplied by IBM, and can be modified or rewritten by the installation.

To use a dedicated work data set, DSNAME=&&name or DSNAME=&name must be coded on a DD statement along with all other parameters used to define a new data set (see Figure 34). Every DD statement in a job with a "name" identical to a ddname on a DD statement in the initiator procedure is assigned the corresponding dedicated data set. If the system cannot assign this dedicated data set, it uses the parameters coded on the DD statement to create a temporary data set.

Note: This facility does not support tape files.

The following rules apply to the parameters of DD statements associated with dedicated work data sets:

1. DSNAME -- The temporary name from the initiator procedure replaces that specified in the DD statement.

2. DISP -- The disposition specification cannot cause deletion of a dedicated work data set. Disposition will appear to allocation as OLD,KEEP or OLD,PASS, only.

3. UNIT -- Specification of UNIT=AFF=DDNAME and DEFER on DD statements are ignored if they apply to dedicated work data sets.

4. VOLUME -- Volume information on the DD statement is overridden by the volume information in the initiator procedure. A specification of REF=*.stepname.ddname is not valid since the initiator procedure may contain only one step.

5. EXPDT/RETPD -- Expiration date or retention is ignored if it is specified on the DD statement.

6. SUBALLOC=stepname.ddname must not be specified since the initiator may contain only one job step. A specification of RLSE will be ignored. All other space parameters are allowed.

Note: The units, primary space, secondary space, and directory quantities on the DD statement are compared with those in the dedicated data set. The data set will be assigned as long as it is equal to or greater than the parameter specified.

7. DCB -- Information specified in the DCB parameter overrides the DCB specification in the initiator procedure.

For detailed information on pre-allocated or dedicated data sets, see the chapter "System Reader, Initiator and Writer Cataloged Procedures" in the System Programmer's Guide publication.

Data sets are created by specifying parameters in the DD statement or by using a data set utility program. This section discusses the use of the DD statement to create data sets. (The Utilities publication, Order No. GC28-6586, discusses data set utility programs.) No consideration is given to optimizing I/O operations; this information is given in the section "Program Optimization."

To create data sets, the DSNAME, UNIT, VOLUME, SPACE, LABEL, DISP, SYSOUT, and DCB parameters are of special significance (see Figure 34). These parameters specify:

DSNAME - name of the data set

UNIT - class and number of devices used for the data set

VOLUME - volume on which the data set resides

LABEL - label specification

DISP - the disposition of the data set after the completion of the job step

SYSOUT - ultimate device for unit record data sets

DCB - tape density, record format, record length

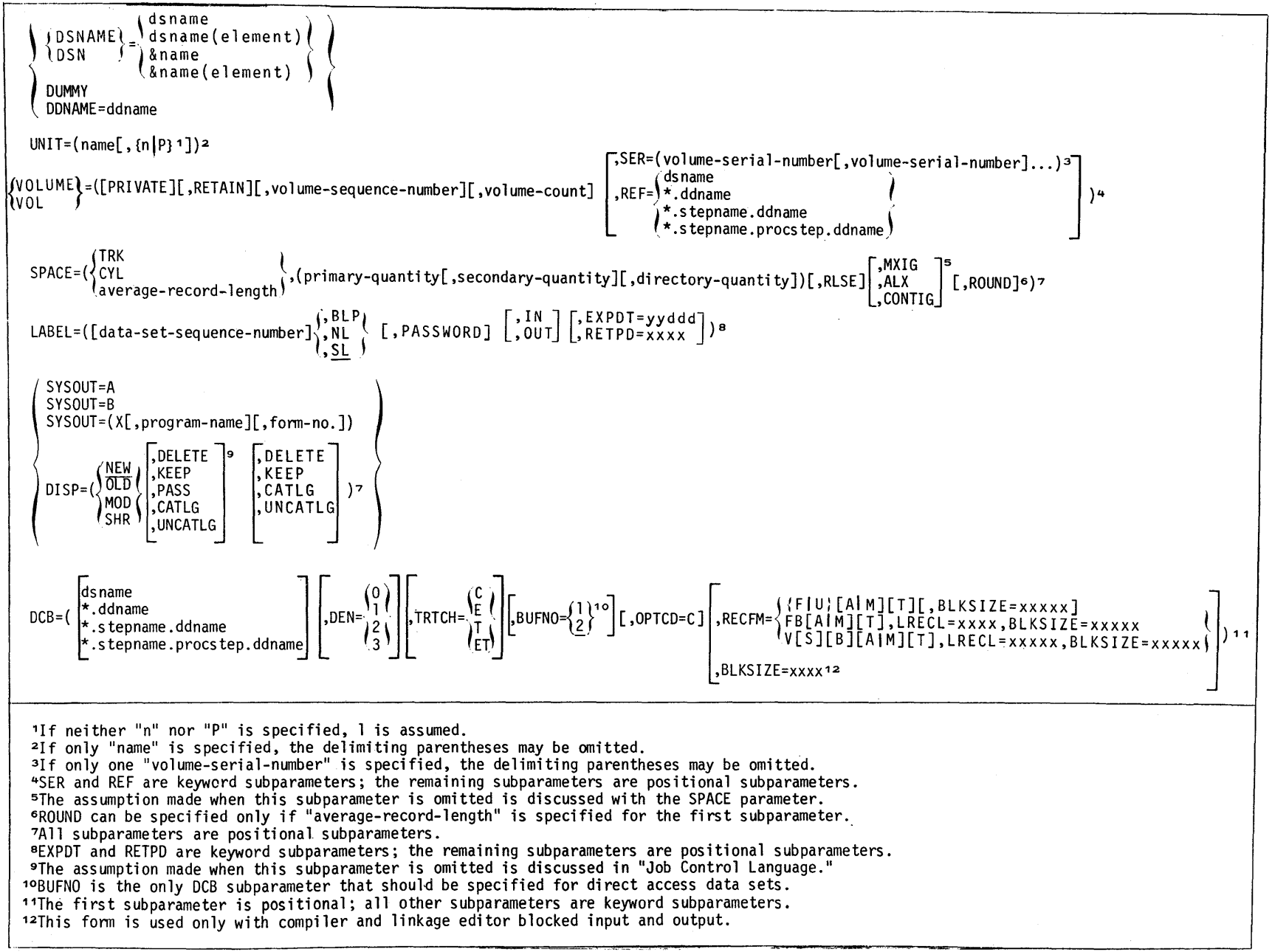
Examples of DD statements used to create data sets are shown in Figure 33.

Sample Coding Form

```
1-10      11-20     21-30     31-40     41-50     51-60     61-70     71-80
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
Example 1: Creating a Cataloged Data Set
//FT31F001 DD DSNAME=MATRIX,DISP=(NEW,CATLG),LABEL=(,SL,EXPDT=67031),                              1
//         UNIT=DACLASS,VOLUME=(PRIVATE,RETAIN,SER=AA69),                                         2
//         SPACE=(300,(100,100),,CONTIG,ROUND),                                                   3
//         DCB=(RECFM=VB,LRECL=604,BLKSIZE=1212)

Example 2: Creating a Data Set for a Job
//FT89F001 DD DSNAME=&TEMP,UNIT=(TAPECLS,3),DISP=(NEW,PASS),                                      1
//         VOLUME=(,RETAIN,1,9,SER=(777,888,999,444)),                                            2
//         DCB=(DEN=2,RECFM=U,BLKSIZE=2500)

Example 3: Specifying a SYSOUT Data Set for the Compiler
//SYSPRINT DD SYSOUT=A,DCB=(BLKSIZE=144,DEN=2,TRTCH=C)

Example 4: Creating a Data Set That is Kept But Not Cataloged
//FT31F001 DD DSNAME=CHEM,DISP=(,KEEP),UNIT=2400-2,                                               1
//         DCB=(DEN=2,TRTCH=ET,RECFM=U,BLKSIZE=1000),                                             2
//         VOL=SER=A605
```

Figure 33. Examples of DD Statements

Figure 34. DD Parameters for Creating Data Sets



[1]If neither "n" nor "P" is specified, 1 is assumed.
[2]If only "name" is specified, the delimiting parentheses may be omitted.
[3]If only one "volume-serial-number" is specified, the delimiting parentheses may be omitted.
[4]SER and REF are keyword subparameters; the remaining subparameters are positional subparameters.
[5]The assumption made when this subparameter is omitted is discussed with the SPACE parameter.
[6]ROUND can be specified only if "average-record-length" is specified for the first subparameter.
[7]All subparameters are positional subparameters.
[8]EXPDT and RETPD are keyword subparameters; the remaining subparameters are positional subparameters.
[9]The assumption made when this subparameter is omitted is discussed in "Job Control Language."
[10]BUFNO is the only DCB subparameter that should be specified for direct access data sets.
[11]The first subparameter is positional; all other subparameters are keyword subparameters.
[12]This form is used only with compiler and linkage editor blocked input and output.

USE OF DD STATEMENTS FOR DIRECT-ACCESS DATA SETS

Data sets that are referred to in FORTRAN direct-access input/output statements must first be defined in the DEFINE FILE statement. However, the DD statement may be used in conjunction with the DEFINE FILE statement for designating other characteristics of the data set.

For example, a direct-access data set containing fifty formatted records of maximum length 100 bytes can be defined as follows:

```
DEFINE FILE 2(50,100,E,12)
```

The DD statement associated with this data set may be specified with additional information such as data set disposition and data set name. For example:

```
//FT02F001 DD DSNAME=BOOL,DISP=(NEW,CATLG)1
//       LABEL=(,SL),UNIT=SYSDA,           2
//       VOLUME=(PRIVATE,RETAIN)           3
//       SPACE=(100,(50,30),,CONTIG),      4
//       DCB=(RECFM=F,BLKSIZE=100)
```

Caution must be taken when specifying the name field and parameters in the DD statement. The name field must contain FTxxF001, where xx is the data set reference number specified in the DEFINE FILE statement. The DD statement parameters must conform to the specifications in the DEFINE FILE statement. Specifically, the SPACE parameter, which must be indicated for all direct-access data sets, must specify the same record length and number of records as in the DEFINE FILE statement. In the DCB parameter, the subparameters DEN and TRTCH should not be specified since they apply only to data sets residing on magnetic tape volumes. In addition, the DUMMY parameter should not be specified, because of a conflict in specifications. The conflict arises because the disposition of a direct-access data set is always checked and a dummy data set has no disposition.

DATA SET NAME

The DSNAME parameter specifies the name of the data set. Only four forms of the DSNAME parameter are used to create data sets.

{DSNAME=dsname
{DSNAME=dsname(element)}

specify names for data sets that are created for permanent use.

Note: Members of a partitioned dataset may be read as input to a FORTRAN object program or created as output from a FORTRAN object program, but only if the member name and either LABEL = (,,,IN) or LABEL = (,,,OUT) are specified in an associated DD statement.

{DSNAME=&name
{DSNAME=&name(element)}

specify data sets that are temporarily created for the execution of a single job or job step.

DUMMY

is specified in the DD statement to inhibit I/O operations specified for the data set. A WRITE statement is recognized, but no data is transmitted. (When the programmer specifies DUMMY in a DD statement used to override a cataloged procedure, all parameters in the cataloged DD statement are overridden.)

Note: A DUMMY data set should only be read if the "END= " option is specified in the FORTRAN READ statement. If the option is not specified, a read causes an end of data set condition, and termination of execution of the load module.

DDNAME=ddname

indicates a DUMMY data set that will assume the characteristics specified in a following DD statement "ddname". The DD statement identified by "ddname" then loses its identity; that is, it cannot be referred to by an *....ddname parameter. The statement in which the DDNAME parameter appears may be referenced by subsequent *....ddname parameters. If a subsequent statement identified by "ddname" does not appear, the data set defined by the DD statement containing the DDNAME parameter is assumed to be an unused statement. The DDNAME parameter can be used five times in any given job step or procedure step, but no two uses can refer to the same "ddname". The DDNAME parameter is used mainly for cataloged procedures.

SPECIFYING INPUT/OUTPUT DEVICES

The name and number of input/output devices are specified in the UNIT parameter,

```
UNIT=(name[,{n|P}] )
```

name
is given to the input/output device when the system is generated.

n|P
specifies the number of devices allocated to the data set. If a number "n" is specified, the operating system assigns that number of devices to the data set. "P" is used with cataloged data sets when the required number of volumes is unknown. The control program assigns a device for each volume required by the data set.

Note: See Appendix F for a list of input/output unit types.

SPECIFYING VOLUMES

The programmer indicates the volumes used for the data set in the VOLUME parameter.

```
VOLUME=( [PRIVATE][,RETAIN]
         [,volume-sequence-number]
         [,volume-count]
        [,SER=(volume-serial-number
               [,volume-serial-number]...)
        ,REF= dsname
              *.ddname
              *.stepname.ddname
              *.stepname.procstep.ddname ] )
```

identifies the volume(s) assigned to the data set.

PRIVATE
indicates that the assigned volume is to contain only the data set defined by this DD statement. PRIVATE is overridden when the DD statement for a data set requests the use of the private volume with the SER or REF subparameter. The volume is demounted after its last use in the job step, unless RETAIN is specified.

RETAIN
indicates that this volume is to remain mounted after the job step is completed. Volumes are retained so that data may be transmitted to or from the data set, or so that other data sets may reside on the volume. If the data set requires more than one volume, only the last volume is retained; the other volumes are demounted when the end of volume is reached. If each job step issues a RETAIN for the volume, the retained status lapses when execution of the job is completed.

volume-sequence-number
is a 1- to 4-digit decimal number that specifies the sequence number of the first volume of the data set that is read or written. The volume sequence number is meaningful only if the data set is cataloged and volumes lower in sequence are omitted.

volume-count
specifies the number of volumes required by the data set. Unless the SER or REF subparameter is used, this subparameter is required for every multi-volume output data set.

SER
specifies one or more serial numbers for the volumes required by the data sets. A volume serial number consists of one to six alphameric characters. If it contains less than six characters, the serial number is left-adjusted and padded with blanks. If SER is not specified, and DISP is not specified as NEW, the data set is assumed to be cataloged and serial numbers are retrieved from the catalog, or inherited from passed data sets in a previous step. A volume serial number is not required for new output data sets.

REF
indicates that the data set is to occupy the same volume(s) as the data set identified by "dsname", "*.ddname", "*.stepname.ddname", or *.stepname.procstep.ddname. Table 13 shows the data set references.

When the data set resides on a tape volume and REF is specified, the data set is placed on the same volume, behind the data set referred to by this subparameter. If this subparameter is used, the UNIT parameter, if specified, is ignored.

If SER or REF is not specified, the control program allocates any non-private volume that is available.

Table 13. Data Set References

Option	Refers to
REF=dsname	a data set named "dsname"
REF=*.ddname	a data set indicated by DD statement "ddname" in the current job step
REF=*.stepname.ddname	a data set indicated by DD statement "ddname" in the previous job step "stepname"
REF=*.stepname.procstep.ddname	a data set indicated by DD statement "ddname" in the procedure step "procstep" invoked in the previous job step "stepname"

SPECIFYING SPACE ON DIRECT-ACCESS VOLUMES

The programmer indicates the amount of space for a data set in the SPACE parameter.

```
SPACE=({TRK                          }
       {CYL                          }
       {average-record-length        }

       ,(primary-quantity

       [,secondary-quantity]

       [,directory-quantity])

       [,RLSE][,MXIG ][,ROUND])
              [,ALX  ]
              [,CONTIG]
```

The SPACE parameter specifies:

1. Units of measurement in which space is allocated.
2. Amount of space allocated.
3. Whether unused space can be released.
4. In what format space is allocated.

{TRK / CYL / average-record-length}
: specifies the units of measurement in which storage is assigned. The units may be tracks (TRK), cylinders (CYL), or records (average record length in bytes expressed as a decimal number less than or equal to 65,535).

(primary-quantity[,secondary-quantity] [directory-quantity])
: specifies the amount of space allocated for the data set. The "primary quantity" indicates the number of records, tracks, or cylinders to be allocated when the job step begins. The "secondary quantity" indicates how much space is to be allocated each time previously allocated space is exhausted. (Note: The maximum number of times secondary allocation will be made is 15.)

For example, by specifying:

```
SPACE=(120,(400,100))
```

space is reserved for 400 records; the average record length is 120 characters. Each time space is exhausted, space for 100 additional records is allocated.

The "directory quantity" is used only in writing a PDS; it specifies the number of 256-byte blocks to reserve for the PDS directory.

By specifying:

```
SPACE=(CYL,(20,2,5))
```

20 cylinders are allocated to the data set. When previously allocated space is exhausted, two additional cylinders are allocated. Furthermore, space is reserved for five 256-byte blocks in the directory of a PDS.

Note: When the FORTRAN programmer uses a direct-access data set, he must allocate space on the direct-access volume in two places: the DEFINE FILE statement in the source module and a DD statement at load module execution. He must also make certain that the DD statement SPACE parameter contains an adequate SPACE allocation, based on the value specified in the DEFINE FILE statement.

RLSE
: indicates that all unused external storage assigned to a data set is to be released when the data set is closed in a job step.

Note: The RLSE subparameter is ignored for any file for which END FILE is specified, or for which a BACKSPACE statement is issued.

[MXIG]
[ALX]
[CONTIG]

specify the format of the space allocated to the data set, as requested in the "primary quantity".

MXIG
requests the largest single block of contiguous storage that is greater than or equal to the space requested in the "primary quantity".

ALX
requests all available storage on the volume as long as there is at least as much space as specified in the "primary quantity". The operating system must be able to allocate at least the amount specified as the "primary quantity" by using, at most, five noncontiguous areas of storage.

CONTIG
requests that the space indicated in the "primary quantity" be contiguous.

If the subparameter is not specified, or if any option cannot be fulfilled, the operating system attempts to assign contiguous space. If there is not enough contiguous space, up to five noncontiguous areas are allocated.

ROUND
indicates that allocation of space for the specified number of records is to begin and end on a cylinder boundary.

Note: If a data set might be written on a direct access volume, the SPACE parameter must be specified in the DD statement.

LABEL INFORMATION

The label parameter (LABEL) is used to specify the type and contents of a data set label.

LABEL=([data-set-sequence-number]{,NL / ,SL / ,BLP}
[,PASSWORD] [,OUT / ,IN] [,EXPDT=yyddd] [,RETPD=xxxx])

data-set-sequence-number
is a 4-digit number that identifies the relative location of the data set with respect to the first data set on a tape volume. (For example, if there are three data sets on a magnetic tape volume, the third data set is identified by data set sequence number 3.) If the data set sequence number is not specified, the operating system assumes 1.

{NL / SL / BLP}

specifies data set label information. SL indicates standard labels. NL indicates no labels (applicable only to data sets residing on a tape volume). BLP indicates that label processing should be bypassed.

The feature that allows bypassing of label processing is a system generation option (OPTIONS=BYLABEL). If this option has not been specified and BLP is coded, the system assumes NL.

PASSWORD
is used to secure a data set from unauthorized access. The operating system assigns security protection to the data set. Subsequently, whenever the data set is retrieved, the operator must respond to a message by issuing the correct password. Detailed information on the use of the PASSWORD subparameter can be found in the Job Control Language and the Data Management and Supervisor Services publications. Note that prudence should be exercised in specifying this option; indiscriminate use of PASSWORD can result in operation inefficiencies.

[OUT]
[IN]

The subparameters IN, OUT are used to control data sets that are to be processed as input or output only. Thus a form of read/write protection is offered by these parameters.

For input data sets, the IN subparameter allows:

- Access to members of a partitioned data set (for read purposes only).
- A means of avoiding operator intervention when reading a data set that is protected by either a high expiration date or by the absence of the write ring (file-protected tape).

For output data sets, the OUT subparameter allows a member of a partitioned data set to be created.

IN
specifies that the data set is to be processed for input only. IN will be recognized only if the first input/output operation specifying the

data set is a READ; any subsequent WRITE issued to the data set will be treated as an error, and the job will be terminated. If the first operation is not READ, the IN subparameter has no effect and both READ/WRITE operations are allowed.

Specification of the IN subparameter allows access to members of a partitioned data set for read purposes. Additionally, the specification of IN permits the reading of a password-protected data set (if the correct password is supplied) and avoids the need of operator intervention when reading a data set protected by either a high expiration date or the absence of a write-ring.

OUT
: specifies that the data set defined by the DD statement is to be processed for output only. OUT will be recognized only if the first input/output operation specifying the data set is a WRITE; any subsequent READ issued to the data set will be treated as an error, and the job will be terminated. If the first operation is not WRITE, the OUT subparameter has no effect and both READ/WRITE operations are allowed.

  The OUT subparameter permits the creation of a partitioned data set but only if the first input/output specification is a WRITE.

Note: If neither subparameter is specified, the data set will be opened for both input and output processing necessitating the use of a write-ring.

EXPDT=yyddd
RETPD=xxxx
: specifies how long the data set shall exist. The expiration date, EXPDT=yyddd, indicates the year (yy) and the day (ddd) the data set can be deleted. The period of retention, RETPD=xxxx, indicates the period of time, in days, that the data set is to be retained. If neither is specified, the retention period is assumed to be zero.

DISPOSITION OF A DATA SET

The disposition of a data set is specified by the DISP parameter; see "Data Definition (DD) Statement". The same options are used for both creating data sets and retrieving previously created data sets. When a data set is created, the subparameters used are NEW, MOD, KEEP, PASS, and CATLG.

WRITING A UNIT RECORD DATA SET ON AN INTERMEDIATE DEVICE

With the SYSOUT parameter, output data sets can be routed to a system output stream and handled much the same as system messages.

SYSOUT=A
: can be used with sequential schedulers to indicate that the data set is to be written on the system output device. No parameter other than the DCB parameter has any meaning when SYSOUT=A is used. This form of the SYSOUT parameter may be specified for printer data sets.

SYSOUT=B
: can be used with sequential schedulers to indicate the system card punch unit. The priority scheduler routes the output data set to class B.

SYSOUT=(x[,program-name] [,form-number])
: indicates that the data set is normally written on an intermediate direct access device during program execution, and later routed through an output stream to a system output device. The "x" is to be replaced by an alphabetic or numeric character that specifies the system output class to be used. Output writers route data from the output classes to system output devices. The DD statement for this data set can also include a unit specification that describes the intermediate direct access device and an estimate of the space required. If these parameters are omitted, the job scheduler provides default values as the job is read and processed.

  If there is a special installation program to handle output operations, its "program-name" should be specified. "Program-name" is the member name of the program, which must reside in the system library.

  If the output data set is to be printed or punched on a specific type of output form, a four-digit "form number" should be specified. This form number is used to instruct the operator, in a message issued at the time the data set is to be printed, of the form to be used.

Note: If the DEN subparameter is explicitly specified for SYSOUT data sets, only DEN=2 is allowed in the DCB parameter. In addition, TRTCH=C must be specified in the DCB parameter, when the SYSOUT data set (1) is written on 7-track tape and (2) is composed of variable-length records or contains binary information.

DCB PARAMETER

For load module execution, the FORTRAN programmer may specify record formats and record lengths for sequentially organized data sets that reside on magnetic tape or direct access volumes. The DCB information is placed in the labels for these data sets.

```
DCB=([dsname                          ]
     [*.ddname                        ]
     [*.stepname.ddname               ]
     [*.stepname.procstep.ddname      ]

     [,DEN={0|1|2|3}][,TRTCH={C|E|T|ET}]

[,BUFNO={1|2}][,OPTCD=C]

[,RECFM={{F|U}[A|M][T][,BLKSIZE=xxxx]                    }]
        {FB[A|M][T],LRECL=xxxx,BLKSIZE=xxxx,             }
        {V[S][B][A|M][T],LRECL=xxxx,                     }
        { BLKSIZE=xxxx                                   }

[,BLKSIZE=xxxx]
```

REFERRING TO PREVIOUSLY SPECIFIED DCB INFORMATION

The first subparameter

```
[dsname                          ]
[*.ddname                        ]
[*.stepname.ddname               ]
[*.stepname.procstep.ddname      ]
```

is used to copy DCB information from the data set label of a cataloged data set or from a preceding DD statement. The copied information is used for processing the data set defined by the DD statement in which the subparameter appears. Any subparameters that follow this subparameter override any copied DCB subparameters.

dsname
: indicates that the DCB subparameters of a cataloged data set "dsname" are copied. The data set indicated by "dsname" must be currently mounted and it must reside on a direct access volume.

*.ddname
: indicates that the DCB subparameters in a preceding DD statement "ddname" in the current job step are copied.

*.stepname.ddname
: indicates that the DCB subparameters in a DD statement "ddname" that occurs in a previous job step "stepname" in the current job are copied.

*.stepname.procstep.ddname
: indicates that the DCB subparameters in the DD statement "ddname" are copied from a previous step "procstep" in a cataloged procedure. The procedure was invoked by the EXEC statement "stepname" in the current job.

DENSITY AND CONVERSION

The second subparameter indicates the density and conversion for data sets residing on magnetic tape volumes.

DENSITY: Density is specified for data sets residing on any magnetic tape volume.

DEN={0|1|2|3}
: indicates the density used to write a data set (see Table 14).

Table 14. DEN Values

DEN Value	Tape Recording Density (bits/inch)	
	Model 2400	
	7-Track	9-Track
0	200	-
1	556	-
2	800	800
3	-	1600

If DEN is not specified, 800 bits per inch is assumed for 7-track tape, 800 bits-per-inch for 9-track tape without dual density, and 1600 bits-per-inch for 9-track tape with dual density.

CONVERSION: Conversion is used only for data sets residing on 7-track tape volumes.

TRTCH={C|E|T|ET}
indicates which conversion type is used:

C - data conversion feature is used

E - even parity is used

T - translation from BCD to EBCDIC is required

ET - even parity is used and translation from BCD to EBCDIC is required.

NUMBER OF BUFFERS FOR SEQUENTIAL DATA SETS

The number of buffers required to read or write any data set is specified by

BUFNO=x

where x=1 or 2

CHAINED SCHEDULING

Chained scheduling may be requested by specifying OPTCD=C as a DCB subparameter in the DD statement. Although chained scheduling is not used for direct-access I/O itself, it does produce faster formatting of direct-access data sets. Note that when chained scheduling is specified, the system makes use of about 2K additional bytes of main storage to provide the feature.

RECORD FORMAT

Formatted Control	Unformatted Control
RECFM=U[A|M][T] RECFM=V[B][A|M][T][S] RECFM=F[B][A|M][T|S]	RECFM=VS[B][A|M][T]

The characters U, V, F, B, and S represent

U - undefined records (records that do not conform to either the fixed-length or variable-length format)

V - variable-length records (records whose length can vary throughout the data set)

F - fixed-length records (records whose length is constant throughout the data set)

B - blocked records

S - for fixed-length records, the records are written as standard blocks, i.e., no truncated blocks or unfilled tracks within the data set, with the exception of the last block or track.

S - for variable-length records, a record may span more than one block.

The character A indicates the use of the extended American National Standard carriage control characters (see Appendix E); the character M indicates the use of machine code control characters.

Note: If A is not specified (or assumed), a carriage control character is treated as data and written. Single spacing is provided.

The character T specifies the use of the track overflow feature. Use of this feature results in more efficient utilization of track capacity and allows records to be written when the specified block size exceeds track size. RECFM subparameter specifications, and the type of processing each is associated with, follow:

RECFM=UT
Formatted Sequential I/O

RECFM=VT
Formatted Sequential I/O

RECFM=VST
Unformatted Sequential I/O

RECFM=FT
Direct Access I/O or Formatted Sequential I/O

Note that backspacing is not allowed when track overflow is specified. Therefore, a FORTRAN program using the track overflow feature may not contain the BACKSPACE statement.

RECORD LENGTH, BUFFER LENGTH, AND BLOCK LENGTH

For blocked records used by the compiler or linkage editor, the length of a block is specified by the buffer length which is specified by

BLKSIZE=xxxx

where xxxx is a multiple of the record length.

The record length (LRECL) is permanently specified by the compiler or linkage editor.

The SYSPRINT data set of the (G) compiler has a record length of 120 bytes (including the carriage control byte); the SYSPRINT data set of the (H) Compiler has a record length of 137 bytes. The SYSIN, SYSPUNCH, and SYSLIN data sets have a record length of 80 bytes.

For unblocked records used by the compiler or linkage editor, the programmer should set BLKSIZE equal to record length except for the FORTRAN IV (H) SYSPRINT data set, which has a record length of 141 bytes.

For unblocked fixed-length records or undefined records used during load module execution, the record length and the buffer length are specified by

```
BLKSIZE=xxxx
```

For unblocked variable-length records, the record length is specified by

```
LRECL=xxxx
```

buffer length is specified by

```
BLKSIZE=xxxx
```

For blocked variable-length or fixed-length records used by load modules, the record length is specified by

```
LRECL=xxxx
```

block length and buffer length are specified by

```
BLKSIZE=xxxx
```

Undefined records cannot be blocked.

Table 15 is a summary of the specifications made by the programmer for record types and blocking in FORTRAN processing.

FORTRAN RECORDS AND LOGICAL RECORDS

In FORTRAN, records for sequential data sets are defined by specifications in FORMAT statements and by READ/WRITE lists. A record defined by a specification in a FORMAT statement is a FORTRAN record (see the FORTRAN IV Language publication, Order No. GC28-6515). A record defined by a READ/WRITE list is a logical record. Within each category, there are three types

Table 15. Specifications Made by the FORTRAN Programmer for Record Types and Blocking

Step	Blocked or Unblocked	Record Type	RECFM Specification	Record Length	Buffer Length
Compiler or Linkage Editor	Unblocked	Fixed-Length	not specified	not specified[1]	BLKSIZE=record length
	Blocked	Fixed-Length	not specified[1]	not specified[1]	BLKSIZE=xxxx
Load Module Execution	Unblocked	Fixed-Length	RECFM=F[2]	BLKSIZE=xxxx[2]	BLKSIZE=xxxx
		Variable-Length	RECFM=V	LRECL=xxxx	
		Variable-Length Spanned	RECFM=VS	LRECL=xxxx	
		Undefined	RECFM=U	BLKSIZE=xxxx	
	Blocked	Fixed-Length	RECFM=FB	LRECL=xxxx	
		Variable-Length	RECFM=VB		
		Variable-Length Spanned	RECFM=VSB		
		Undefined	Blocked undefined records are not permitted		

[1]Permanently specified by the compiler and cannot be altered (see "DCB Assumptions for Load Module Execution").
[2]Not specified for direct-access data sets.

of records: fixed-length, variable-length, and undefined. In addition, fixed-length and variable-length records can be blocked.

For unformatted READ and WRITE statements the logical record, as defined by the I/O list, is placed into physical records and, if required, the logical record is spanned over physical records. When spanning occurs, FORTRAN library routines do not split-write an item over the span even though there is enough room in the buffer to accomodate part of the item. However, FORTRAN does provide the ability to read items split across segments.

FORMAT Control

The following discussion provides information on records written under control of a FORMAT statement.

UNBLOCKED RECORDS: For fixed-length and undefined records, the record length and buffer length are specified in the BLKSIZE subparameter. For variable-length records, the record length is specified in the LRECL subparameter; the buffer length in the BLKSIZE subparameter. The information coded in a FORMAT statement indicates the FORTRAN record length (in bytes).

Fixed-Length Records: For unblocked fixed-length records written under FORMAT control, the FORTRAN record length must not exceed BLKSIZE (see Figure 35).

Example: Assume BLKSIZE=44

```
10  FORMAT(F10.5,I6,2F12.5,'SUMS')
    WRITE(20,10)AB,NA,AC,AD
```

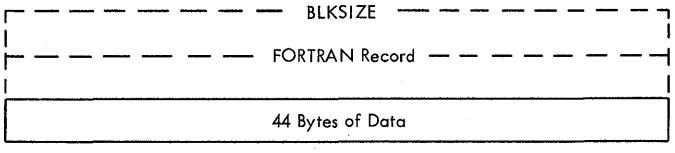


Figure 35. FORTRAN Record (FORMAT Control) Fixed-Length Specification

If the FORTRAN record length is less than BLKSIZE, the record is padded with blanks to fill the remainder of the buffer (see Figure 36). The entire buffer is written.

Example: Assume BLKSIZE=56

```
5  FORMAT(F10.5,I6,F12.5,'TOTAL')
   WRITE(15,5)BC,NB,BD
```

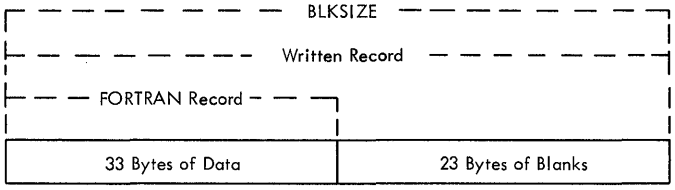


Figure 36. FORTRAN Record (FORMAT Control) Fixed-Length Specification and FORTRAN Record Length Less Than BLKSIZE

Variable-Length Records: For unblocked variable-length records written under FORMAT control, LRECL is specified as four greater than the maximum FORTRAN record length; and BLKSIZE as four greater than LRECL. These extra eight bytes are required for the 4-byte block descriptor word (BDW) and the 4-byte segment descriptor word (SDW), as shown in Figure 37. The BDW (see Figure 42) contains the length of the block; the SDW (see Figure 40) contains the length of the record segment, i.e., the data length plus four bytes for the SDW.

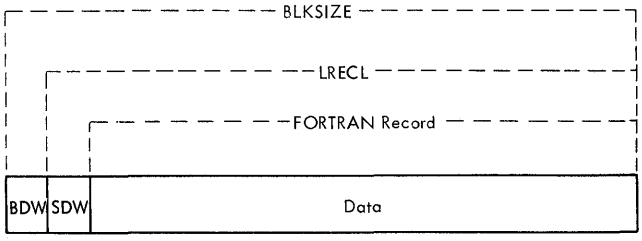


Figure 37. FORTRAN Record (FORMAT Control) Variable-Length Specification

If the data length is less than (LRECL-4), the unused portion of the buffer is not written (see Figure 38).

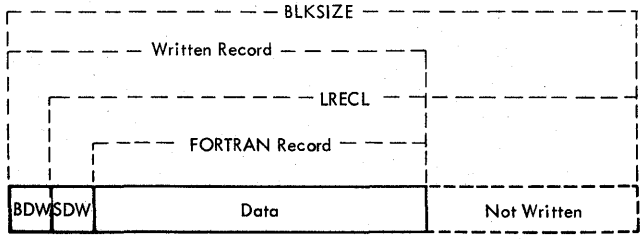


Figure 38. FORTRAN Record (FORMAT Control) With Variable-Length Specification and the FORTRAN Record Length Less Than (LRECL-4)

Undefined Records: For undefined records written under FORMAT control, BLKSIZE is specified as the maximum FORTRAN record length. If the FORTRAN record length is less than BLKSIZE, the unused portion of the buffer is not written (see Figure 39).

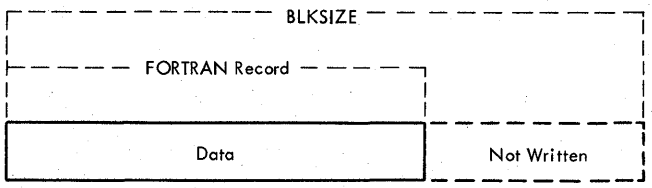


Figure 39. FORTRAN Record (FORMAT Control) With Undefined Specification and the FORTRAN Record Length Less Than BLKSIZE

BLOCKED RECORDS: For all blocked records, the record length is specified in the LRECL subparameter; the block length and buffer length in the BLKSIZE subparameter.

Fixed-Length Records: For blocked fixed-length records written under FORMAT control, LRECL is specified as maximum possible FORTRAN record length, and BLKSIZE must be an integral multiple of LRECL. If the FORTRAN record length is less than LRECL, the rightmost portion of the record is padded with blanks (see Figure 40).

Example: Assume BLKSIZE=48 and LRECL=24

```
10  FORMAT(I2,F4.1,F8.4,F10.5)
20  FORMAT(I3,F9.4)
      .
      .
      .
    WRITE(13,10)N,B,Q,S
      .
      .
      .
    WRITE(13,20)K,Z
```

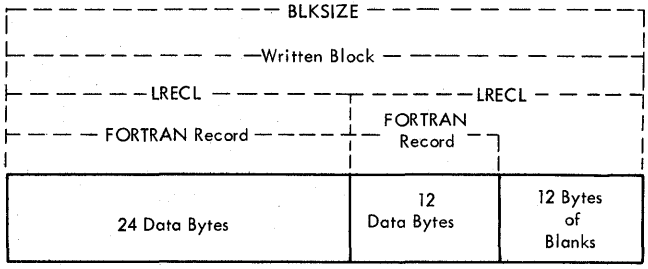


Figure 40. Fixed-Length Blocked Records Written Under FORMAT Control

Variable-Length Records: For blocked variable-length records written under FORMAT control, LRECL is specified as four greater than the maximum FORTRAN record length, and BLKSIZE must be 4 plus an integral multiple of LRECL. The four additional bytes allocated with BLKSIZE are required for the block descriptor word (BDW) that contains the block length. The four additional bytes allocated with LRECL are used for the segment descriptor word (SDW) that contains the record length indication.

If a WRITE is executed and the amount of space remaining in the present buffer is less than LRECL, only the filled portion of this buffer is written (see Figure 41); the new data goes into the next buffer. However, if the space remaining in a buffer is greater than LRECL, the buffer is not written, but held for the next WRITE (see Figure 41). If another WRITE is not executed before the job step is terminated, then the filled portion of the buffer is written.

If LRECL is omitted, its default value is set almost equal to the value of BLKSIZE. This results in having only one record written in any block.

Example: Assume BLKSIZE=28 and LRECL=12

```
30  FORMAT(I3,F5.2)
40  FORMAT(F4.1)
50  FORMAT(F7.3)
    WRITE(12,30)M,Z
    WRITE(12,40)V
    WRITE(12,50)Y
```

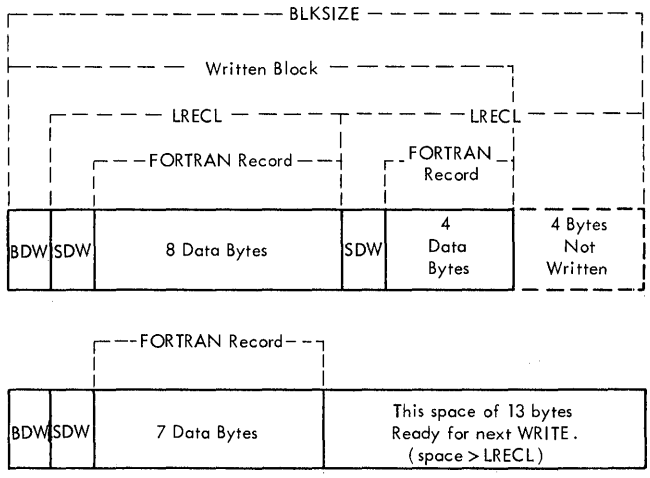


Figure 41. Variable-Length Blocked Records Written Under FORMAT Control

Unformatted Control

Only variable-length records can be written without format control, i.e., the RECFM subparameter must be VS or VBS. (If nothing is specified, VS is assumed.)

Records written with no FORMAT control have the following properties:

- The length of the logical record is controlled by the type and number of variables in the input/output list of its associated READ or WRITE statement.
- A logical record can be physically recorded on an external medium as one or more record segments. Not all segments of a logical record must fit into the same physical record (block).
- Two quantities control the manner in which records are placed on an external medium: the block size (as specified by the BLKSIZE parameter), and the logical record (as defined by the length of the I/O list). BLKSIZE is specified as part of the DCB parameter of the data definition (DD) statement. If not specified, FORTRAN provides default values.

Each block begins with a 4-byte block descriptor word (BDW); each segment begins with a 4-byte segment descriptor word (SDW). The SDWs and BDWs are provided by the system. Each buffer begins with a 4-byte block descriptor word (BDW). The SDWs and BDWs are provided by the system.

The format of a BDW is given in Figure 42.

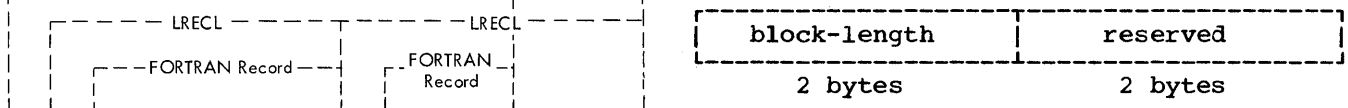


Figure 42. Format of a Block Descriptor Word (BDW)

where:

block-length
: is a binary count of the total number of bytes of information in the block. This includes four bytes for the BDW plus the sum of the segment lengths specified in each SDW in the block. (The permissible range is from 8 to 32,760 bytes.)

reserved
: is two bytes of zeros reserved for system use.

The format of an SDW is given in Figure 43.

Figure 43. Format of a Segment Descriptor Word (SDW)

where:

segment-length
: is a binary count of the number of bytes in the SDW (four bytes) plus the number of bytes in the data portion of the segment following the SDW. (The permissible range is from 4 to 32,756 bytes.)

code
: indicates the position of the segment with respect to the other segments (if any) of the record. Bits 0 through 5 are reserved for system use and are set to 0. Bits 6 and 7 contain the codes:

Code	Meaning
00	This segment is not followed or preceded by another segment of the record.
01	This segment is the first of a multisegment record.
10	This segment is the last of a multisegment record.
11	This segment is neither the first nor last of a multi-segment record.

reserved
is a byte of zeros reserved for system use.

UNBLOCKED RECORDS: For unblocked records, if the logical record length is less than or equal to the length of the block (allowing four bytes for the BDW and four bytes for the SDW), the block will contain the entire logical record. The remainder of the block is unused and is not transmitted. The next record is placed in the following block in the same manner (see Figure 44).

Example: Assume BLKSIZE=40

```
REAL*4 A,B,C,D,E,F,G,H
  .
  .
  .
WRITE(9) A,B,C,D
  .
  .
  .
WRITE(9) E,F,G,H
```

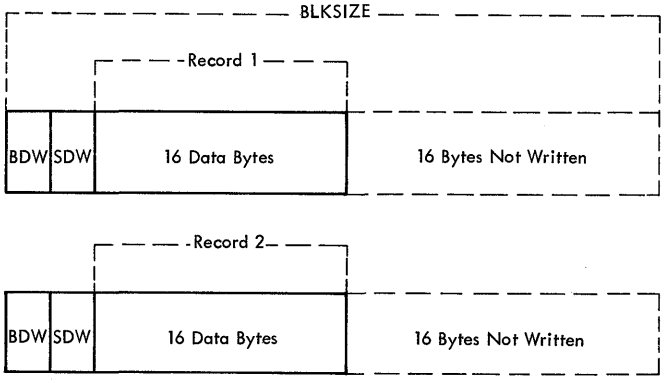


Figure 44. Unblocked Records Written Without FORMAT Control

If the logical record length is greater than the length of the block, the record is divided into record segments. The number of segments is determined from the READ/WRITE statement and the BLKSIZE specification. Again, only the filled portion of the block is transmitted and the next record is placed in the following block (see Figure 45).

Example: Assume BLKSIZE=32

```
REAL*8 A,B,C,D,E,F,G,H
  .
  .
  .
WRITE(9) A,B,C,D
  .
  .
  .
WRITE(9) E,F,G,H
```

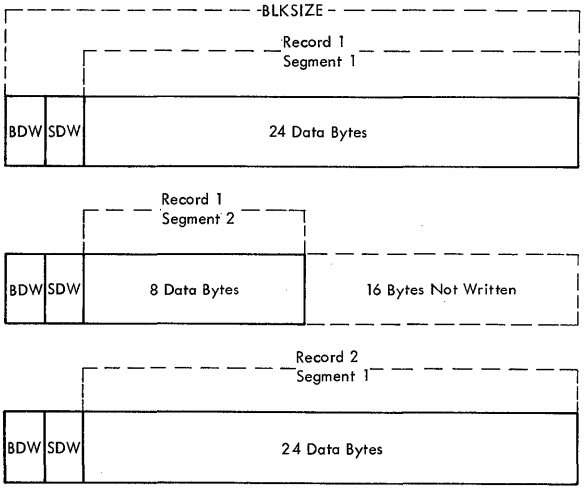


Figure 45. Unblocked Segmented Records Written Without FORMAT Control

BLOCKED RECORDS: For blocked records, if the logical record length is less than or equal to the length of the block (allowing four bytes for the BDW and four bytes for the SDW), the block will contain the entire logical record. The next record, preceded by its SDW, begins in the same block (see Figure 46).

Example: Assume BLKSIZE=44

```
REAL*8 A,B,C,D
  .
  .
  .
WRITE(9) A,B
  .
  .
  .
WRITE(9) C,D
```

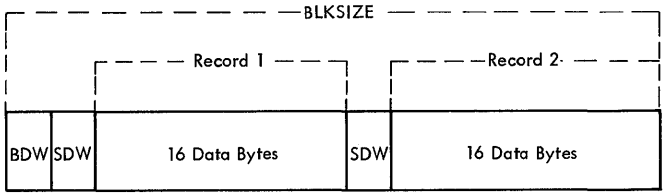


Figure 46. Blocked Records Written Without FORMAT Control

If the logical record is greater than the length of the block, the record is divided into record segments. The number of segments is determined from the READ/WRITE statement and the BLKSIZE specification. The next record, preceded by its SDW, begins in the same block. If the length of the second record exceeds the remainder of the block it too is segmented (see Figure 47).

Example: Assume BLKSIZE=32

```
REAL*8 A,B,C,D,E,F,G,H
  .
  .
  .
WRITE(9) A,B,C,D
  .
  .
  .
WRITE(9) E,F,G,H
```

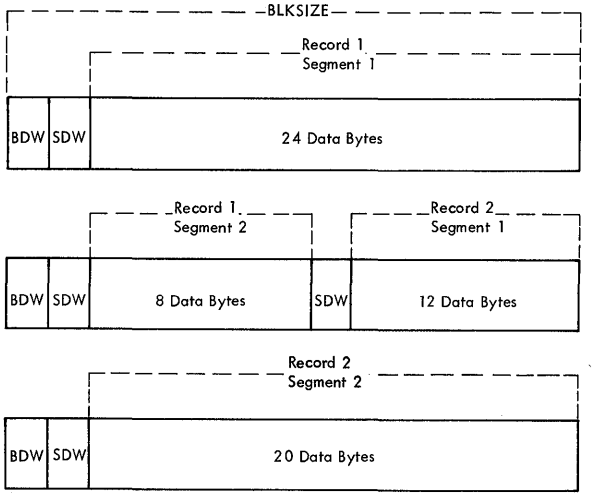


Figure 47. Blocked Segmented Records Written Without FORMAT Control

BACKSPACE Operations

Unblocked Records: For all unblocked records written with or without FORMAT control, the volume is positioned so that the last logical record read or written is transmitted next.

Blocked Records: Blocked records are backspaced on a logical record basis. Thus, a BACKSPACE may result in a deblocking operation rather than making available a new physical record.

Note: Logical records are usually synonymous with the amount of data specified in the I/O list for the READ or WRITE statement that processes the record. Thus, when there is no FORMAT control, the logical record may be spanned over one or more physical records on the volume; however, FORTRAN treats only the logical record as an entity. For records written under FORMAT control, a single READ/WRITE statement may refer to or create several logical records. This occurs when there is a "/" character in the FORMAT statement or when the I/O list exceeds the FORMAT specifications, causing the FORMAT statement to be used again from the first parenthesis.

Extending a Data Set: The execution of an ENDFILE followed by the execution of a BACKSPACE does not cause the FORTRAN sequence number to be incremented. The data set can be extended (written) using the same FORTRAN sequence number.

RECORD LENGTH, BUFFER LENGTH, AND NUMBER OF BUFFERS FOR DIRECT ACCESS DATA SETS

A direct-access data set can contain only fixed-length, unblocked records. Any attempt to read or write any other record format by specification in the DCB parameter is ignored. The record length and buffer length for a data set are specified by the programmer as the record size in the DEFINE FILE statement, and cannot be changed by specifying the BLKSIZE or LRECL subparameters in the DCB parameter. For example, the statement:

```
DEFINE FILE 8(1000,152,E,INDIC)
```

sets the record length and buffer length permanently at 152 bytes. The direct-access data set defined by this DEFINE FILE statement contains 1000 fixed-length, unblocked records; each record is 152 bytes long and is written under FORMAT control.

The programmer may specify the number of buffers for a direct-access data set as follows:

```
BUFNO=x
```

where: x is the number (1 or 2) of buffers used to read or write the data set.

For records written with FORMAT control, the record format is the same as for fixed-length unblocked records written with FORMAT control for sequential data sets. For records written with no FORMAT control, the records must be fixed-length and unblocked. These records do not contain a block control word or a segment control word.

SPANNING CONSIDERATIONS

For records written with no FORMAT control, the input/output list may exceed the logical record length (i.e., block size). In this case a new block is started on output, and the next block is processed on input. If it is shorter than the record length, the remaining portion of the record is padded with zeros (see Figure 48).

The DEFINE FILE field r (r=152 in the example shown above) specifies the maximum size of each record in a data set. It is only when this size is exceeded by the I/O list that spanning occurs.

Note that the spanning feature is an extension to FORTRAN language specifications in that it is applicable only for programs written in FORTRAN under the System/360 Operating System.

When spanning occurs, the FORTRAN library routines do not split an item over the span even if there is enough room in the buffer to accommodate part of the item. The same considerations apply to reading.

Example: A DEFINE FILE statement has specified the record length for a direct-access data set as 20. This statement is then executed:

```
WRITE(9'IX)DP1,DP2,R1,R2
```

where: DP1 and DP2 are real *8 variables.
R1 and R2 are real *4 variables.
IX is an integer variable that contains the record position.

BACKSPACE, END FILE, and REWIND operations are ignored for direct access data sets.

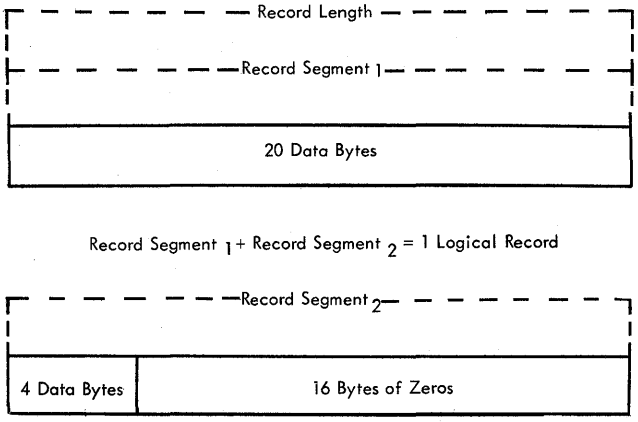


Figure 48. Logical Record (No FORMAT Control) for Direct Access

DCB ASSUMPTIONS FOR LOAD MODULE EXECUTION

For compilation, the LRECL value for the following data sets is fixed and cannot be altered by the programmer:

Data Set	LRECL Value
SYSPRINT	120(G), 137(H)
SYSIN	80
SYSPUNCH	80
SYSLIN	80

The SYSPRINT, SYSIN, and SYSPUNCH compiler data sets can contain blocked records. If the higher level linkage editor (program name: IEWLE440) is used, the SYSLIN data set can contain blocked records.

The BLKSIZE value must be an integral multiple of the corresponding LRECL value shown above. The maximum BLKSIZE value is limited only by the type of input/output device (see Table 16), except that for SYSLIN the maximum BLKSIZE value is 400 with linkage editor IEWLE440.

For load module execution, specifications depend on record type. For F type records, the BLKSIZE value must be an integral multiple of the LRECL value; for V type records, BLKSIZE must be specified as 4 + n x LRECL (where n is the number of records in the block); for U type records, no blocking is permitted. Note, too, that the BLKSIZE and LRECL range is limited only by the type of device used to directly write the data set. Load module DCB parameter default values are shown in Table 17.

Table 16. BLKSIZE Ranges: Device Considerations

	BLKSIZE Ranges	
Device Type	F and U Record Type	V Record Type
Card Reader	1≤x≤80	9≤x≤80
Card Punch	1≤x≤81	9≤x≤89
Printer:		
120 Spaces	1≤x≤121	9≤x≤129
132 Spaces	1≤x≤133	9≤x≤141
144 Spaces	1≤x≤145	9≤x≤153
Magnetic Tape	18≤x≤32,760	

Direct Access:	Without Track Overflow[1]	With Track Overflow[1]
2301	1≤x≤20,483	1≤x≤32,760
2302	1≤x≤4984	1≤x≤32,760
2303	1≤x≤4892	1≤x≤32,760
2311	1≤x≤3625	1≤x≤32,760
2314	1≤x≤7294	1≤x≤32,760

[1]If RECFM=V, the minimum block size is 9.

Table 17. Load Module DCB Parameter Default Values

		Sequential Data Sets		Direct-Access Data Sets	
Data Set Reference Number	ddname	Default BLKSIZE[1]	Default RECFM[2]	Default RECFM	Default LRECL or BLKSIZE
1	FT01Fyyy	800	U	F	The value specified as the maximum size of a record in the DEFINE FILE statement.
2	FT02Fyyy	800	U	F	
3	FT03Fyyy	800	U	FA[3]	
4	FT04Fyyy	800	U	F	
5	FT05Fyyy	80	F	F	
6	FT06Fyyy	133	UA[3]	F	
7	FT07Fyyy	80	F	F	
8	FT08Fyyy	800	U	F	
.	.	.	.	.	
.	.	.	.	.	
.	.	.	.	.	
99	FT99Fyyy	800	U	F	

[1]If the records have no FORMAT control, the default LRECL is 4 less than BLKSIZE, where the default BLKSIZE is as specified in this table. For direct-access data sets, blocksize is usually limited by track capacity, unless track overflow has been specified.
[2]If the records have no FORMAT control, the default RECFM is VS (F if it is direct access).
[3]The first character in the record is for carriage control.

For ease of reference this section, directed solely to the user of the FORTRAN IV (G) compiler, has been written as a self-contained, independent unit. For information on FORTRAN IV (H) cataloged procedures, see "FORTRAN IV (H) Cataloged Procedures."

This section contains figures illustrating the job control statements used in the FORTRAN IV (G) cataloged procedures and a brief description of each procedure. The statements used to override the statements and parameters in any cataloged procedure are also discussed in this section. (The use of cataloged procedures is described in "FORTRAN Job Processing.")

Compile

In each of the four cataloged procedures that include the compile step (Figures 49, 50, 52, and 53), the EXEC statement named FORT designates that the operating system is to execute the program IEYFORT (the FORTRAN IV (G) compiler).

The REGION parameter is ignored by sequential schedulers. For priority schedulers, it specifies a region size sufficient to compile approximately 400 statements.

MVT priority schedulers require that region size be specified, unless the user is willing to accept the default region size (as established in the input reader procedure).

The size of the region is directly related to the maximum number of source statements that can be compiled by the FORTRAN (G) compiler. A region size of 100K is estimated to be sufficient to compile approximately 400 statements assuming unblocked input and output and non-resident access methods. To adjust this region size for smaller or larger source programs, use 75 bytes per statement as a rule of thumb.

Note: If different region sizes are to be specified for each step in the job, the REGION parameter should be coded in the EXEC statement associated with each step instead of in the JOB statement.

The compiler options (shown in Figure 27) are not supplied with any procedure containing a compile step. Therefore, if the user wishes to have certain operations performed, he must specify those options in the job control statements. However, if the user does not specify any of the options, the system will assume certain default options which are noted by the underscores in Figure 27.

The control statements contained in the procedure, FORTGC (shown in Figure 49), designate the data sets to be used by the compiler during its operation. The source listing, compile-time information, and error messages are written on the data set designated by the SYSPRINT DD statement. The object module resulting from the operation of the FORTRAN compiler is written in the temporary data set &LOADSET, designated in the SYSLIN DD statement. This data set is sequential and is assigned to a sequential device such as a tape or direct-access device. However, if the direct-access device is assigned, a primary allocation of 200 records is requested with a secondary allocation of 100 records. Average record length is specified as 80 bytes. The data set is in PASS status, and records can be added to the data set. The SYSPUNCH DD statement defines the card punch to be used in obtaining an object deck.

The SYSOUT=B parameter on the SYSPUNCH DD statement is interpreted by sequential schedulers as indicating the system card punch unit. The priority scheduler will route the output to output class B.

The programmer can override any of the default options by using an EXEC statement which includes the options that are desired.

Compile and Linkage Edit

The cataloged procedure to compile the source module and linkage edit the resulting FORTRAN object module (FORTGCL) is shown in Figure 50. The control statements for compilation are the same as described above. However, output of the object module is defined by the SYSLIN DD statement.

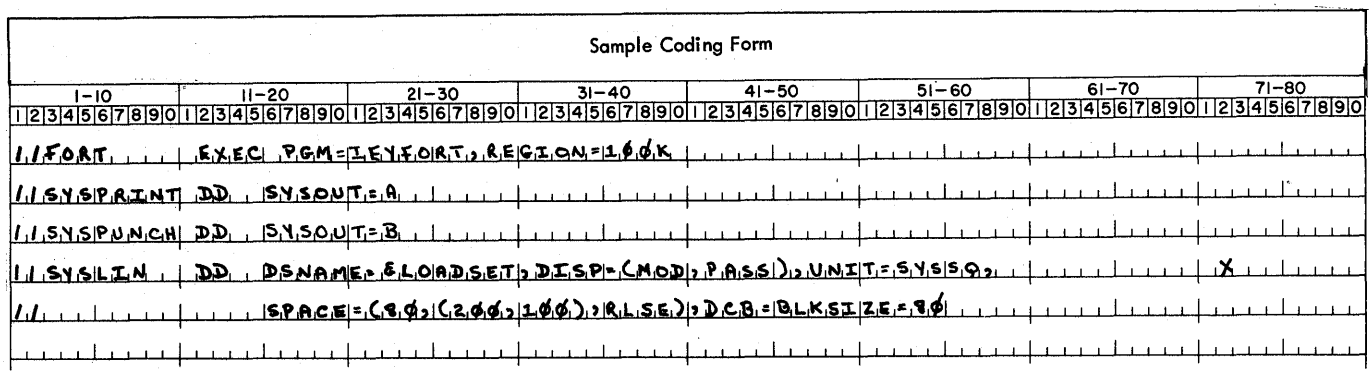


Figure 49. Compile Cataloged Procedure (FORTGC)

In each of the cataloged procedures that include a linkage edit step (Figures 50, 51 and 52), the EXEC statement named LKED specifies that the operating system is to execute the program IEWL (the linkage editor). However, the linkage editor step (or the remainder of the procedure) is not executed if a condition code greater than 4 was generated during the operation of the compile step in the same procedure.

Execution of the linkage editor step produces a list of the linkage editor control statements (in card image format), a map or cross-reference listing of the load module, and a list of linkage editor diagnostic messages on the data set specified by the SYSPRINT DD statement. The load module is marked executable even though error conditions are found during processing.

The primary input to the linkage editor may consist of concatenated data sets. The first, defined by the SYSLIN DD statement, is the output of the compiler; the second (may be omitted) is the data set defined by a LKED.SYSIN DD statement which is specified by the user and is external to the procedure.

External references made in a FORTRAN object module are resolved by the linkage editor. Some or all of these references can be resolved from the FORTRAN library (SYS1.FORTLIB) designated in the SYSLIB DD statement.

During processing, the linkage editor requires a work data set which is defined by the SYSUT1 DD statement. This data set is assigned to a direct-access device with primary allocation of 20 records and secondary allocation of 10 records. The load module produced by the linkage editor is written in the temporary PDS defined in the SYSLMOD DD statement. The data set is in the PASS status.

Linkage Edit and Execute

This cataloged procedure, FORTGLG, first linkage edits the FORTRAN object module and then executes the resulting load module. (The FORTGLG procedure is shown in Figure 51.) Since the linkage edit step is the first step in the procedure, the primary input is the data set defined by the LKED.SYSIN DD statement.

The execute step is included in two cataloged procedures (see Figures 51 and 52). In each of these procedures the execute step is invoked by the EXEC statement named GO. However, this step is bypassed if a condition code greater than 4 was generated during the operation of the linkage edit step in this procedure.

Input to the execute step is defined by a GO.SYSIN DD statement which is supplied by the user and is external to the procedure. The data set is read using data set reference number 5. In the linkage edit step, execution-time error messages are written in the data set defined by the SYSPRINT DD statement. In the execute step, error messages and information for traceback, DUMPs, and PDUMPs are written on the data set associated with the reference number 6. (Output from the load module can also be written in the same data set.) The card punch is associated with data set reference number 7.

In a multiprogramming environment with an MVT priority scheduler, main storage requirements for the execute step are determined by a number of factors. These include: the size of the object program produced by the compiler, the requirements of the data access method used, the blocking factors, the number and sizes of the data sets used, the number and sizes of library subprograms invoked, and the sizes of the execution time routines required by the program. If the default region size (established in the cataloged procedure for the input reader) is not large enough for

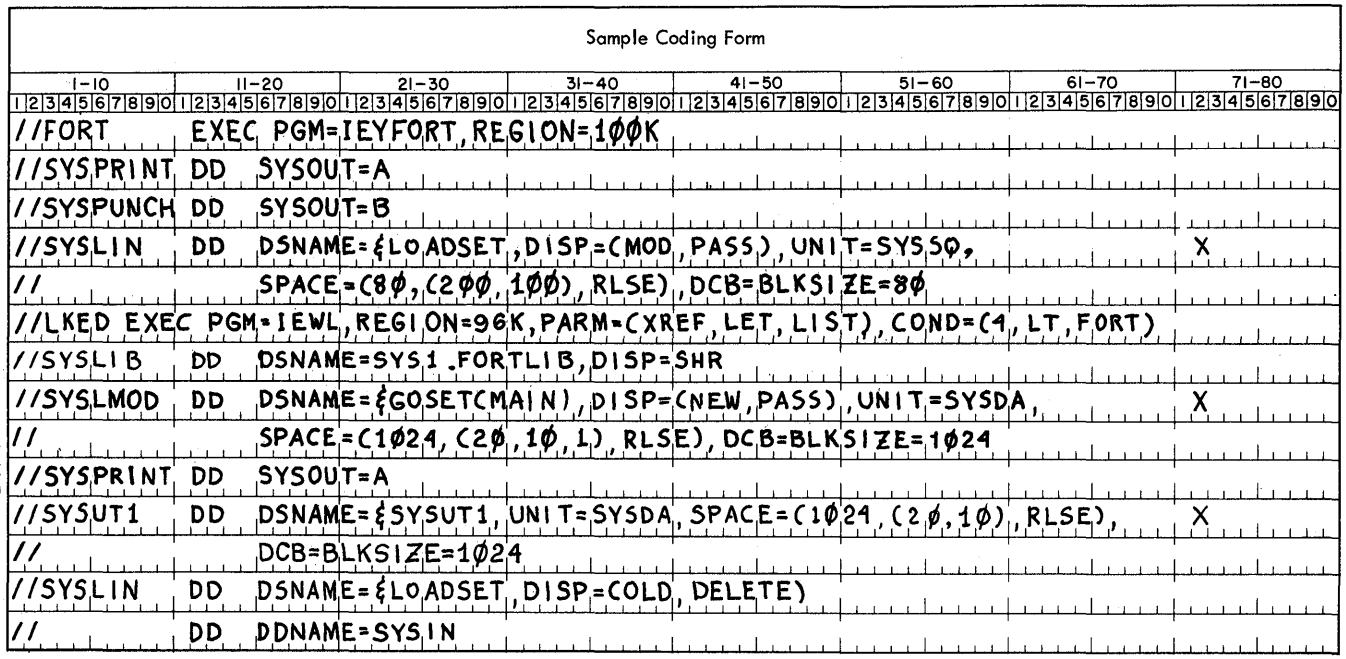


Figure 50. Compile and Linkage Edit Cataloged Procedure (FORTGCL)

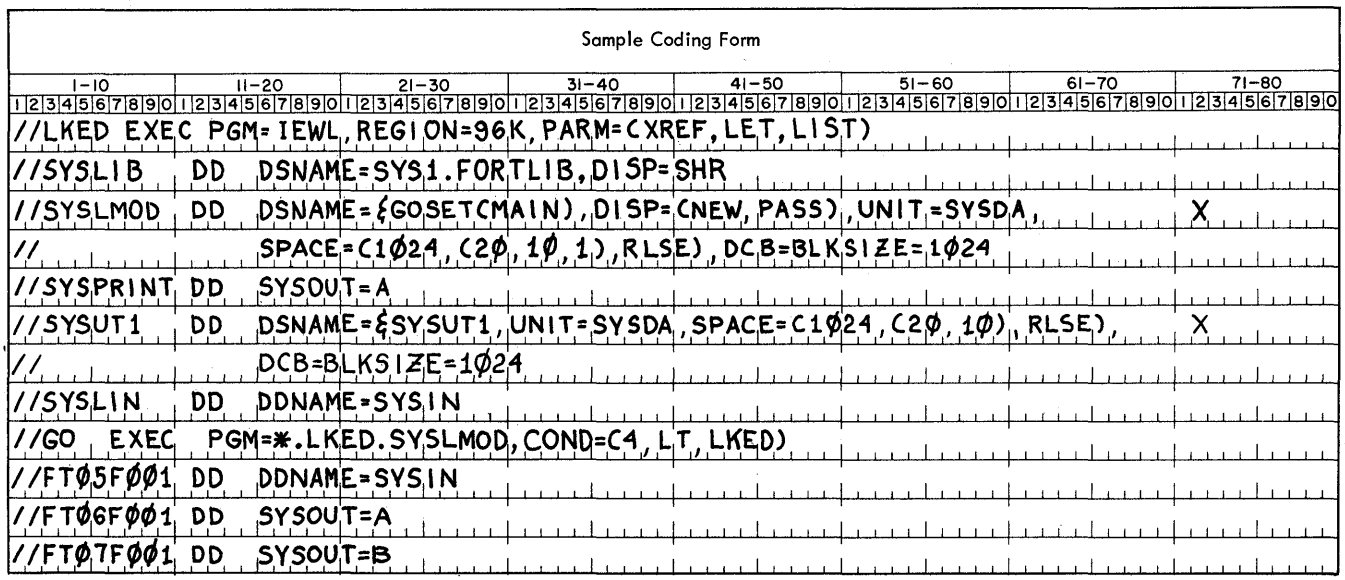


Figure 51. Linkage Edit and Execute Cataloged Procedure (FORTGLG)

the program, REGION.GO must be used to specify the region size for the execute step.

A listing of the execution time routines required for various input/output, interruption, and error procedures is contained in the FORTRAN IV Library Subprograms publication, Order No. GC28-6818. That publication also lists the sizes of both the execution-time routines and the mathematical subprograms.

An example of using a REGION.GO specification to indicate the main storage requirements for the execute step of a FORTRAN program follows.

```
//EXAMPLE1 JOB ACCOUNT1,'JOHNSMITH',      X
//               MSGLEVEL=1

// EXEC FORTGCLG,PARM.FORT=DECK,          X
                 REGION.GO=84K

//FORT.SYSIN DD *
                .
                .
                .
         FORTRAN SOURCE SYMBOLIC DECKS
                .
                .
                .
/*

//LKED.SYSIN DD *
                .
                .
                .
         PREVIOUSLY COMPILED OR ASSEMBLED
           OBJECT DECKS
                .
                .
                .
/*

//GO.SYSIN DD *
                .
                .
                .
         INPUT DATA
                .
                .
                .
/*
```

Compile, Linkage Edit, and Execute

The cataloged procedure (FORTGCLG) to compile, linkage edit, and execute FORTRAN source modules is shown in Figure 52. This cataloged procedure consists of the statements in the FORTGC and FORTGLG procedures, with the following exception: the SYSLIN DD statement defines the output of the compiler, and the same statement in the linkage edit step identifies this output as the primary input.

The programmer does not have to define the linkage editor input as was required for the FORTGLG procedure, but the input data set must be defined for the compiler so that the source module can be read. A data set containing primary input to the linkage editor may also be defined by using a LKED.SYSIN DD statement. This data set is concatenated with the data set containing the output of the compiler.

Compile and Load

The cataloged procedure (FORTGCLD) to compile and load FORTRAN source modules is shown in Figure 53. The control statements used in the compiler step are the same as those used in the cataloged procedure FORTGC (Figure 49).

The load step is invoked by the EXEC statement GO. This statement specifies that the loader (program name LOADER) is to be executed. The EXEC statement also specifies that a storage map and any diagnostic messages produced in the load step are to be placed in the data set specified in the SYSLOUT DD statement.

The load step will not be executed if a condition code greater than 4 was generated during the compile step.

Primary input to the load step is defined in the SYSLIN DD statement. This is the output data set produced by the compiler. Additional input may be defined by a GO.SYSIN DD statement which is supplied by the user and is external to the procedure. This data set is concatenated with the primary input data set.

Any external references made in the load step are resolved by the loader. Some or all of these references can be resolved from the FORTRAN library (SYS1.FORTLIB) designated in the SYSLIB DD statement. Private libraries may be concatenated with the FORTRAN library to help resolve external references. Figure 32 shows how this may be done.

USER AND MODIFIED CATALOGED PROCEDURES

The programmer can write his own cataloged procedures and tailor them to the facilities in his installation. He can also permanently modify the IBM-supplied cataloged procedures. For information

about permanently modifying cataloged procedures, see the Job Control Language publication.

The IBM-supplied cataloged procedures for FORTRAN IV (G) define logical unit 05 as SYSIN, 06 as SYSOUT, and 07 as SYSCP (see Figures 51, 52, and 53). If, during system generation, values other than 05 for the ONLNRD parameter, 06 for the OBJERR parameter, and 07 for the ONLNPCH parameter were specified in the FORTLIB macro instruction, one or more of the following DD cards must be added to the cataloged procedures, either to override them at execution time or to modify them permanently. (The System Generation publication, Order No. GC28-6554, describes the FORTLIB macro instruction.)

If a //GO.SYSIN DD * statement is used to define the input data set, DCB parameters should not be specified. However, if the data set defined as SYSIN resides somewhere other than on the system input device, the programmer should be aware that the default BLKSIZE is 800 and the default RECFM is U (see Table 17). Therefore, if he desires a BLKSIZE of 80 and a RECFM of F, he must specify them explicitly.

- For the unit specified as ONLNRD, use the DD card:

```
//GO.FTxxF001  DD  DDNAME=SYSIN
```

- For the unit specified as OBJERR, use the DD card:

```
//GO.FTxxF001 DD SYSOUT=A
```

- For the unit specified as ONLNPCH, use the DD card:

```
//GO.FTxxF001  DD  UNIT=SYSCP,                    X
//  DCB=(BLKSIZE=80,RECFM=F)
```

where:

xx (2 digits) is the unit specified

In addition, the DD card for FT05F001 must be deleted permanently from the cataloged procedure.

OVERRIDING CATALOGED PROCEDURES

Cataloged procedures are composed of EXEC and DD statements. A feature of the operating system is its ability to read control statements and modify a cataloged procedure for the duration of the current job. Overriding is only temporary; that is, the parameters added or modified are in effect only for the duration of the job. The following text discusses the techniques used to modify cataloged procedures.

Sample Coding Form

```
1-10      11-20      21-30      31-40      41-50      51-60      61-70      71-80
//FORT    EXEC PGM=JEYFORT,REGION=1ØØK
//SYSPRINT DD  SYSOUT=A
//SYSPUNCH DD  SYSOUT=B
//SYSLIN   DD  DSNAME=&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,                X
//           SPACE =(8Ø,(2ØØ,1ØØ),RLSE),DCB=BLKSIZE=8Ø
//LKED EXEC PGM=IEWL,REGION=96K,PARM=(XREF,LET,LIST),COND=(4,LT,FORT)
//SYSLIB   DD  DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSLMOD  DD  DSNAME=&GOSET(MAIN),DISP=(NEW,PASS),UNIT=SYSDA,             X
//           SPACE=(1Ø24,(2Ø,1Ø,1),RLSE),DCB=BLKSIZE=1Ø24
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  DSNAME=&SYSUT1,UNIT=SYSDA,SPACE=(1Ø24,(2Ø,1Ø),RLSE),       X
//           DCB=BLKSIZE=1Ø24
//SYSLIN   DD DSNAME=&LOADSET,DISP=(OLD,DELETE)
//           DD DDNAME=SYSIN
//GO EXEC  PGM=*.LKED.SYSLMOD,COND=((4,LT,FORT),(4,LT,LKED))
//FTØ5FØØ1 DD DDNAME=SYSIN
//FTØ6FØØ1 DD SYSOUT=A
//FTØ7FØØ1 DD SYSOUT=B
```

Figure 52. Compile, Linkage Edit, and Execute Cataloged Procedure (FORTGCLG)

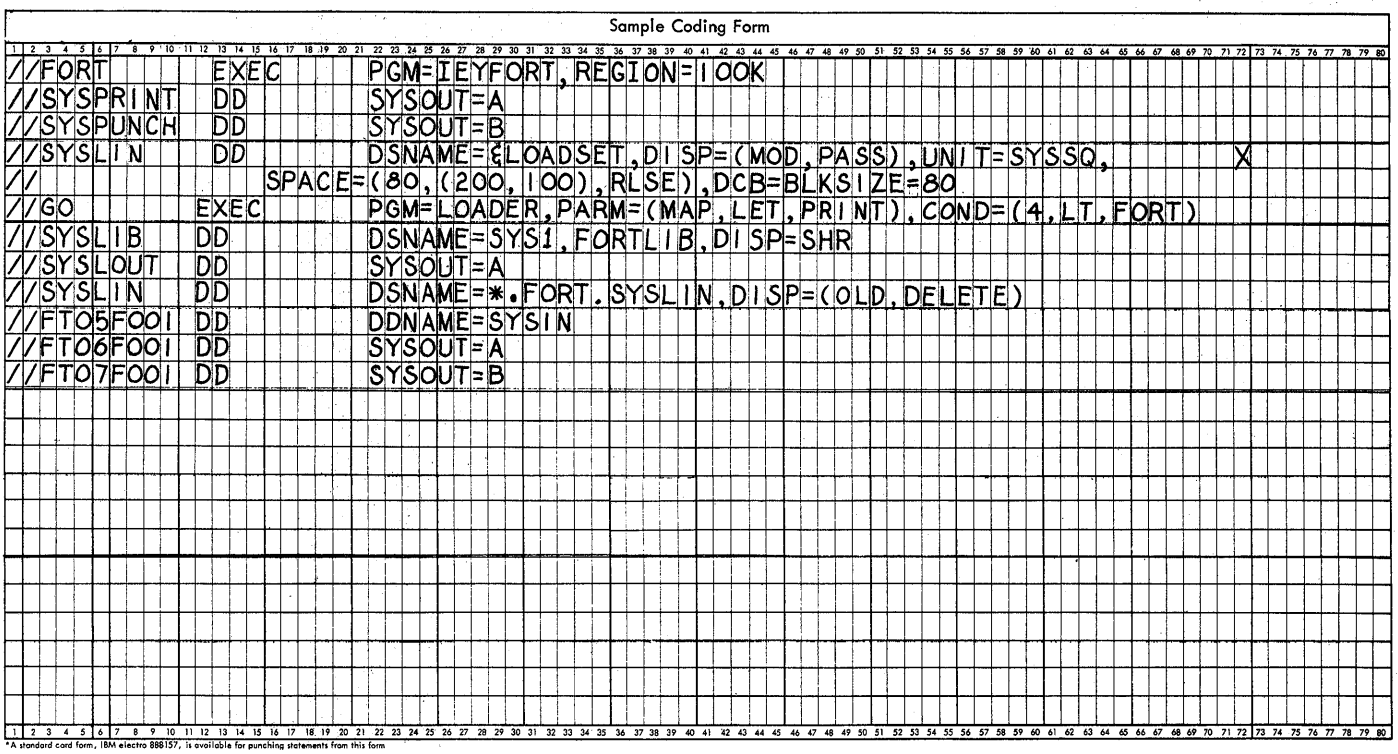


Figure 53. Compile and Load Cataloged Procedure (FORTGCLD)

Overriding Parameters in the EXEC Statement

Two forms of keyword parameters ("keyword" and "keyword.procstep") are discussed in "Job Control Language." The form "keyword.procstep" is used to add or override parameters in an EXEC statement in a cataloged procedure.

The FORTRAN programmer can, for example, add (or override) compiler, linkage editor, or loader options for an execution of a cataloged procedure, or he can state different conditions for bypassing a job step.

Note: When the PARM parameter is overridden, all compiler, linkage editor, or loader options stated in the EXEC statement in the procedure step are deleted and replaced by those in the overriding PARM parameter.

Example 1: Assume the cataloged procedure FORTGC is used to compile a program, and the programmer wants to specify the name of his program and the MAP option. The following statement can be used to invoke the procedure and to supply the compiler options.

```
//STEP1 EXEC FORTGC,                          X
//             PARM.FORT='MAP,NAME=MYPROG'
```

The PARM options apply to the procedure step FORT.

Example 2: Assume the cataloged procedure FORTGLG is used to linkage edit and execute a module. Furthermore, the MAP option overrides XREF, LET, and LIST in the linkage editor step and the COND parameter is changed for the execution of the load module. The following EXEC statement adds and overrides parameters in the procedure.

```
//DO EXEC FORTGLG,PARM.LKED=MAP,              X
//             COND.GO=(3,LT,DO.LKED)
```

The PARM parameter applies to the linkage editor procedure step LKED, and the COND parameter applied to the execution procedure step GO.

Example 3: Assume a source module is compiled and loaded using the cataloged procedure FORTGCLD. Furthermore, an external name is specified as the entry point of the loaded program. The following EXEC statement adds and overrides parameters in the procedure.

```
//STEP1  EXEC  FORTGCLD,PARM.GO='EP=FIRST'
```

Overriding and Adding DD Statements

A DD statement with the name "stepname.ddname" is used to override parameters in DD statements in cataloged procedures, or to add DD statements to cataloged procedures. The "stepname" identifies the step in the cataloged procedure. If "ddname" is the name of a DD statement present in the step, the parameters in the new DD statement override parameters in the DD statement in the procedure step. If "ddname" is the name of a DD statement not present in the step, the new DD statement is added to the step.

In any case, the modification is effective only for the current execution of the cataloged procedure.

When overriding, the original DD statement in the cataloged procedure is copied, and the parameters specified in it are replaced by the corresponding parameters in the new DD statement. Therefore, only parameters that must be changed are specified in the overriding DD statement.

If more than one DD statement is modified, the overriding DD statements must be in the same order as the DD statements appear in the cataloged procedure. Any DD statements that are added to the procedure must follow overriding DD statements.

When the procedures FORTGC, FORTGCL, FORTGCLG, and FORTGCLD are used, a DD statement must be added to define the SYSIN data set to the compile step in the procedures (see Figures 16, 17, 21, and 24). When the procedure FORTGLG is used, a DD statement must be added to define the SYSLIN data set (see Figure 19).

When the procedures FORTGCL, FORTGLG, and FORTGCLG are used, an overriding DD statement can be used to write the load module constructed in the linkage editor step in a particular PDS chosen by the programmer, and assign that member of the PDS a particular name.

During execution of procedure steps, the programmer can catalog data sets, assign names to data sets, supply DCB information for data sets, add data sets, or specify particular volumes for data sets by using overriding DD statements.

Example 1: Assume the data sets identified by ddnames FT04F001 and FT08F001 are named, cataloged, and assigned specific volumes. The following DD statements are used to add this information and indicate the location of the source module.

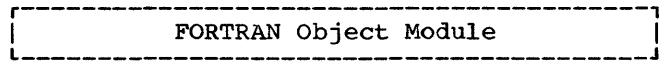
```
//JOB1 JOB MSGLEVEL=1
//STEP1 EXEC FORTGCLG
//FORT.SYSIN DD *
┌-----------------------------------------┐
|          FORTRAN Source Module          |
└-----------------------------------------┘
/*
//GO.FT04F001 DD DSNAME=MATRIX,            X
//   DISP=(NEW,CATLG),UNIT=TAPE,           X
//   VOLUME=SER=987K
//GO.FT08F001 DD DSNAME=INVERT,            X
//   DISP=(NEW,CATLG),UNIT=TAPE,           X
//   VOLUME=SER=1020
//GO.SYSIN DD *
┌-----------------------------------------┐
|          Input to Load Module           |
└-----------------------------------------┘
/*
```

Example 2: Assume DCB information is added to the DD statement identified by ddname FT08F001 and a data set for data set reference number 4 is created and cataloged.

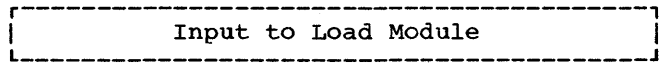
```
//JOB2 JOB
//STEP1 EXEC FORTGLG
//LKED.SYSIN DD *
┌-----------------------------------------┐
|          FORTRAN Object Module          |
└-----------------------------------------┘
/*
//GO.FT08F001 DD DCB=(RECFM=F,BLKSIZE=200)
//GO.FT04F001 DD DSNAME=FIRING,            X
//   UNIT=SYSDA,DISP=(NEW,CATLG),          X
//   SPACE=(100,(2000,200),,,ROUND),       X
//   VOLUME=(PRIVATE,SER=207H),            X
//   DCB=(RECFM=VB,LRECL=300,BLKSIZE=604)
//GO.SYSIN DD *
┌-----------------------------------------┐
|          Input to Load Module           |
└-----------------------------------------┘
/*
```

Example 3: Assume the linkage edit and execute cataloged procedure (FORTGLG) is used. The load module constructed in the linkage editor step is placed in the cataloged partitioned data set MATH and is assigned the member name DERIV.

```
//JOB3 JOB
//STEP1 EXEC FORTGLG
//LKED.SYSLMOD DD DSNAME=MATH(DERIV),      X
//    DISP=(OLD,PASS)
//LKED.SYSIN DD *
```




```
/*
//GO.SYSIN DD *
```




```
/*
```

Example 4: Assume the compile, linkage edit, and execute cataloged procedure (FORTGCLG) is used with three data sets in the input stream:

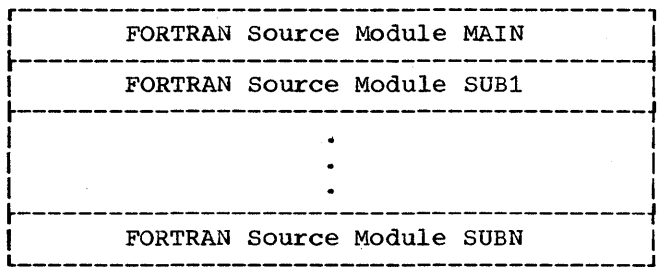
1. A FORTRAN main program MAIN with a series of subprograms, SUB1 through SUBN.

2. A linkage editor control statement that specifies an additional library, MYLIB. MYLIB is used to resolve external references for the symbols ALPHA, BETA, and GAMMA.

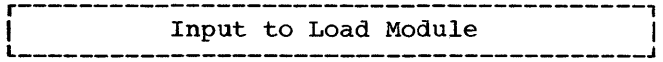
3. A data set used by the load module and identified by data set reference number 5 in the source module.

The following example shows the deck structure.

```
//JOBCLG JOB 00,FORTRANPROG,MSGLEVEL=1
//HXECCLGX EXEC FORTGCLG
//FORT.SYSIN DD *
```




```
/*
//LKED.ADDLIB DD DSNAME=MYLIB
//LKED.SYSIN DD *
  LIBRARY ADDLIB(ALPHA,BETA,GAMMA)
/*
//GO.SYSIN DD *
```




```
/*
```

The DD statement FORT.SYSIN indicates to the compiler that the source modules are in the input stream. The DD statement LKED.ADDLIB defines the additional library MYLIB to the linkage editor. The DD statement LKED.SYSIN defines a data set that is concatenated with the primary input to the linkage editor. The linkage editor control statements and the object modules appear as one data set to the linkage editor. The DD statement GO.SYSIN defines data in the input stream for the load module.

For ease of reference this section, directed solely to the user of the FORTRAN IV (H) compiler, has been written as a self-contained, independent unit. For information on FORTRAN IV (G) cataloged procedures, see "FORTRAN IV (G) Cataloged Procedures."

This section contains figures showing the job control statements used in the FORTRAN IV cataloged procedures and a brief description of each procedure. This section also describes statements used to override statements and parameters in any cataloged procedure. (The use of cataloged procedures is discussed in "FORTRAN Job Processing.")

Compile

In the four cataloged procedures that have a compile step (see Figures 54, 55, 57 and 58) the EXEC statement named FORT indicates that the operating system is to execute the program IEKAA00 (the FORTRAN IV H compiler).

The REGION parameter is ignored by sequential schedulers. MVT priority schedulers require that region size be specified, unless the user is willing to accept the default region size (as established in the input reader procedure).

The amount of main storage allocated for the FORTRAN H compiler depends on the region size in an MVT environment, the partition size in an MFT environment, or the machine size in a PCP environment. The compiler uses all available main storage except for 3K bytes which are left for non-resident system routines.

In certain instances, a programmer may wish to limit the amount of main storage used by the compiler. An example would be when the FORTRAN H compiler is executed as the original task in a multitasking environemnt. Unless the amount of main storage used by the compiler is limited, no subtasks could be created since no more storage would be available in the region.

The programmer may request the amount of main storage to be allocated for the compiler by specifying the SIZE option in the PARM parameter. Specific information concerning the SIZE option can be found in the section "Compiler Options."

Note: If different region sizes are to be specified for each step in the job, the REGION parameter should be coded in the EXEC statement associated with each step instead of the JOB statement.

Compiler options are not explicitly specified; default options are assumed -- in particular, SOURCE and LOAD. The source listing and compile-time information and error messages are written in the SYSOUT data set.

The object module is written in the temporary data set &LOADSET. The data set &LOADSET is a sequential data set and is in "pass" status; records can be added to the data set.

The SYSOUT=B parameter on the SYSPUNCH DD statement is interpreted by sequential schedulers as a specification for the system card punch unit. The priority schedulers route the output data set to system output class B. A programmer can get an object module card deck by overriding the default NODECK option with an explicit DECK option.

Several additional DD statements, external to the procedure, may be supplied. If the EDIT option is used, a work data set must be defined with a FORT.SYSUT1 DD statement. If the compiler XREF option is specified, a work data set must be defined with a FORT.SYSUT2 DD statement. Input to the compile step is defined by a FORT.SYSIN DD statement.

The data set SYSUT1 must be specified if the compiler option EDIT (produce structured source listing) was requested. SYSUT2 must be specified if the compiler option XREF (produce cross reference listing) was requested. Both data sets may reside on tape or direct access but must be defined in the sequential device class. The following is a typical DD statement for a utility data set:

```
//SYSUT1  DD  DSNAME=&UT1,UNIT=SYSSQ,      X
              SPACE=(TRK,(40))
```

&UT1
specifies a temporary data set.

UNIT=SYSSQ
specifies that the data set is to reside in a sequential device class.

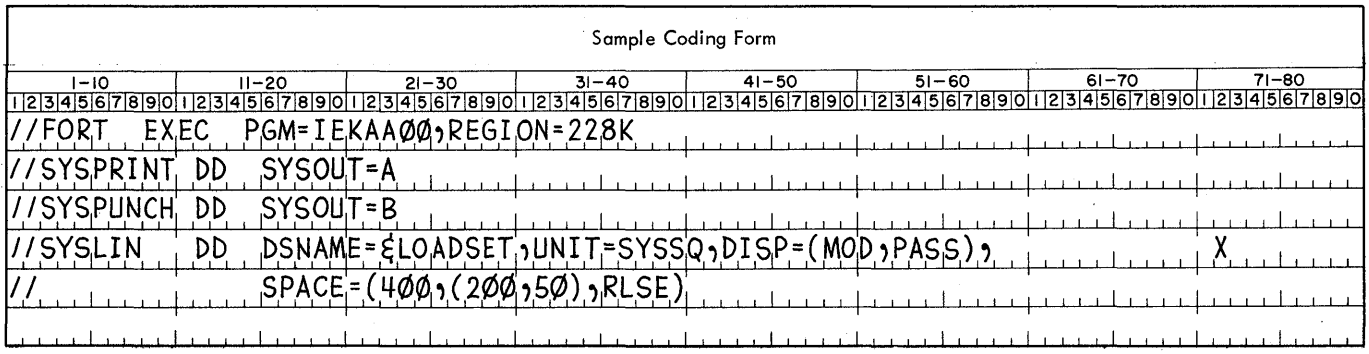


Figure 54. Compile Cataloged Procedure (FORTHC)

SPACE=(TRK,(40)
specifies that if the data set is assigned to a direct access device, 40 tracks are to be allocated to the data set.

Linkage Edit

In the three cataloged procedures that have a linkage edit step (see Figures 55, 56, and 57), the EXEC statement named LKED indicates that the operating system is to execute the program IEWL (the linkage editor). The linkage editor requires a region of 54K if used with MVT. The linkage editor step (or the remainder of a procedure) is not executed if a condition code greater than 4 was generated by a compile step in the same procedure.

If the linkage edit step is executed, a list of linkage editor control statements (in card image format), a map of the load module and a list of linkage editor diagnostic messages are written in the SYSOUT data set. The load module is marked executable even though error conditions are found during linkage editor processing.

If the linkage edit step is preceded by a compile step (see Figures 55 and 57), the primary input to the linkage editor may consist of concatenated data sets. The first, defined by the SYSLIN DD statement, is the output of the compiler (&LOADSET data set); the second (if present) is the data set defined by a LKED.SYSIN DD statement (external to the procedure). However, if the linkage edit step is the first step in a procedure (see Figure 56), the primary input is the data set defined by a LKFD.SYSIN DD statement.

External references made in a FORTRAN object module are resolved by the linkage editor. Some or all of these references can be resolved from the FORTRAN library (SYS1.FORTLIB) which is a system resident PDS.

During processing, the linkage editor requires a work data set which is defined by the SYSUT1 DD statement. This data set is assigned to a direct-access device.

The load module produced by the linkage editor is written in the temporary PDS &GOSET with a member name of MAIN. The data set is in "pass" status and is assigned to a direct-access device.

Execute

In the two cataloged procedures that have an execute step (see Figures 56 and 57), the EXEC statement named GO indicates that the operating system is to execute the load module (program) produced in a preceding linkage edit step in the same procedure. However, the execute step is bypassed if a condition code greater than 4 was generated by a compile or linkage edit step in the same procedure.

Input to the execute step is defined by a GO.SYSIN DD statement (external to the procedure) and is read using data set reference number 5. Execution-time error messages and information for traceback and FORTRAN dumps are written in the SYSOUT data set that is associated with data set reference number 6. (Output from the load module can also be written in the same data set.) The card punch is associated with data set reference number 7.

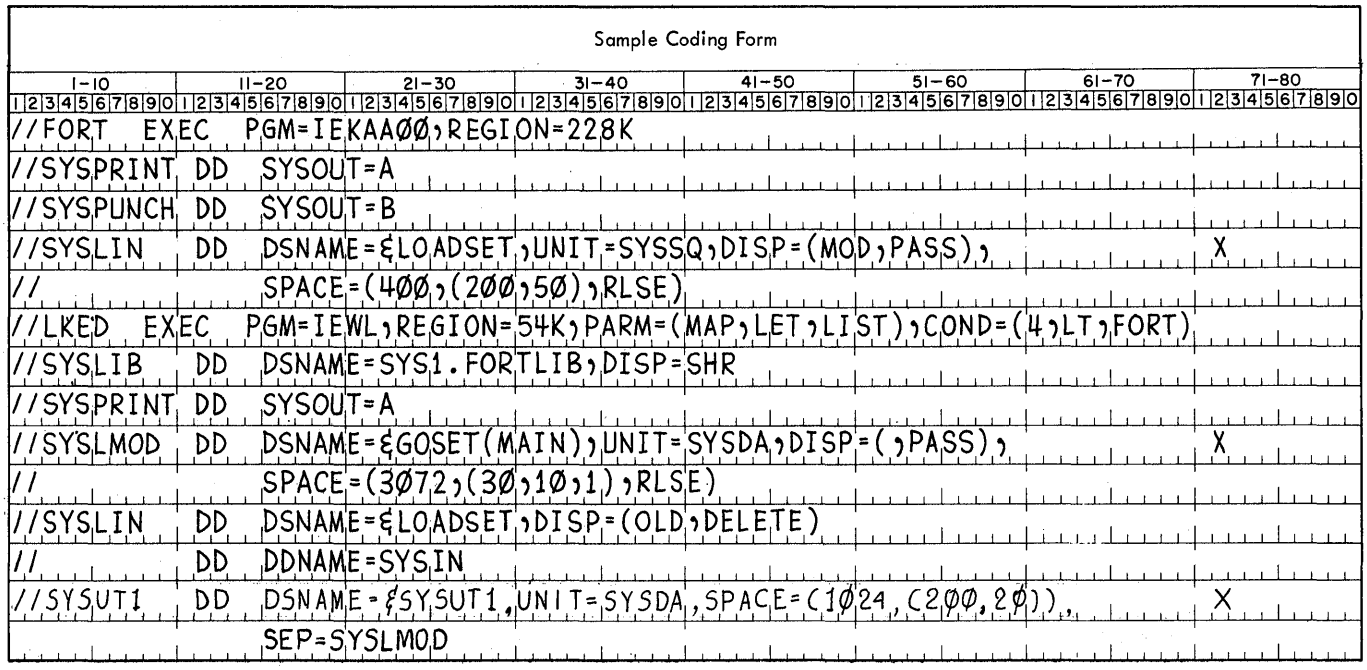


Figure 55. Compile and Linkage Edit Cataloged Procedure (FORTHCL)

In a multiprogramming environment with an MVT priority scheduler, main storage requirements for the execute step are determined by a number of factors. These include: the size of the object program produced by the compiler, the requirements of the data access method used, the blocking factors, the number and sizes of the data sets used, the number and sizes of library subprograms invoked, and the sizes of the execution time routines required by the program. If the default region size (established in the cataloged procedure for the input reader) is not large enough for the program, REGION.GO must be used to specify the region size for the execute step.

A list of the execution time routines required for various input/output, interruption, and error procedures is contained in the FORTRAN IV Library Subprograms publication, Order No. GC28-6818. That publication also lists the sizes of both the execution-time routines and the mathematical subprograms.

An example of using a REGION.GO specification to indicate the main storage requirements for the execute step of a FORTRAN program follows.

```
//EXAMPLE1 JOB ACCOUNT1,'JOHNSMITH',      X
//             MSGLEVEL=1
//  EXEC FORTHCLG,PARM.FORT=DECK,          X
//             REGION.GO=200K
//FORT.SYSIN DD  *
┌-----------------------------------------┐
|     FORTRAN Source Symbolic Decks       |
└-----------------------------------------┘
/*
//LKED.SYSIN DD  *
┌-----------------------------------------┐
|    Previously Compiled or Assembled     |
|              Object Decks               |
└-----------------------------------------┘
/*
//GO.SYSIN DD  *
┌-----------------------------------------┐
|               Input Data                |
└-----------------------------------------┘
/*
```

Load

In the one cataloged procedure that has a load step (see Figure 58), the EXEC statement named GO indicates that the operating system is to execute the program LOADER (the loader). The EXEC statement also specifies that a storage map of the loaded program is to be produced. This map is to be placed in the data set specified in the SYSLOUT DD statement.

The load step will not be executed if a condition code greater than 4 was generated during the compile step.

Primary input to the load step is defined in the SYSLIN DD statement. This is the output data set produced by the compiler. Additional input may be defined by a GO.SYSIN DD statement which is supplied by the user and is external to the procedure. This data set is concatenated with the primary input data set.

Any external references made in the load step are resolved by the loader. Some or all of these references can be resolved from the FORTRAN library (SYS1.FORTLIB) designated in the SYSLIB DD statement. Private libraries may be concatenated with the FORTRAN library to help resolve external references. Figure 32 shows how this may be done.

USER AND MODIFIED CATALOGED PROCEDURES

The programmer can write his own cataloged procedures and tailor them to the facilities in his installation. He can also permanently modify the IBM-supplied cataloged procedures. For information about permanently modifying cataloged procedures, see the Job Control Language Reference publication, Order No. GC28-6704.

The IBM-supplied cataloged procedures for FORTRAN IV (H) define logical unit 05 as SYSIN and 06 as SYSOUT (see Figures 56, 57, and 58). If, during system generation, values other than 05 for the ONLNRD parameter and 06 for the OBJERR parameter were specified in the FORTLIB macro instruction, one or both of the following DD cards must be added to the cataloged procedures, either at execution time or permanently.

- For the unit specified as ONLNRD, use the DD card:

```
//GO.FTxxF001 DD DDNAME=SYSIN,
  DCB=(BLKSIZE=80,RECFM=F)
```

- For the unit specified as OBJERR, use the DD card:

//GO.FTxxF001 DD SYSOUT=A where xx is the unit specified. (The System Generation publication, Order No. GC28-6554, describes the FORTLIB macro instruction.)

In addition, the DD card for FT05F001 must be deleted permanently from the procedure. The following section describes the general procedure for adding and deleting statements from cataloged procedures.

OVERRIDING CATALOGED PROCEDURES

Cataloged procedures are composed of EXEC and DD statements. A feature of the operating system is its ability to read control statements and modify a cataloged procedure for the duration of the current job. Overriding is only temporary; that is, the parameters added or modified are in effect only for the duration of the job. The following text discusses the techniques used to modify cataloged procedures.

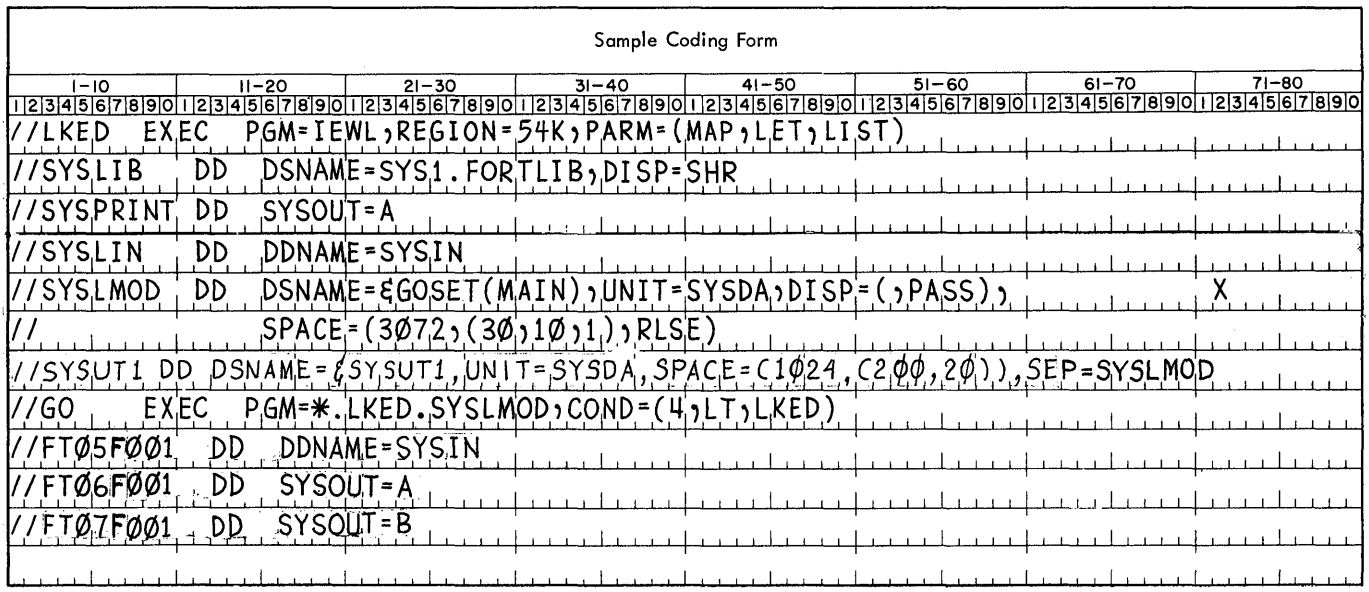


Figure 56. Linkage Edit and Execute Cataloged Procedure (FORTHLG)

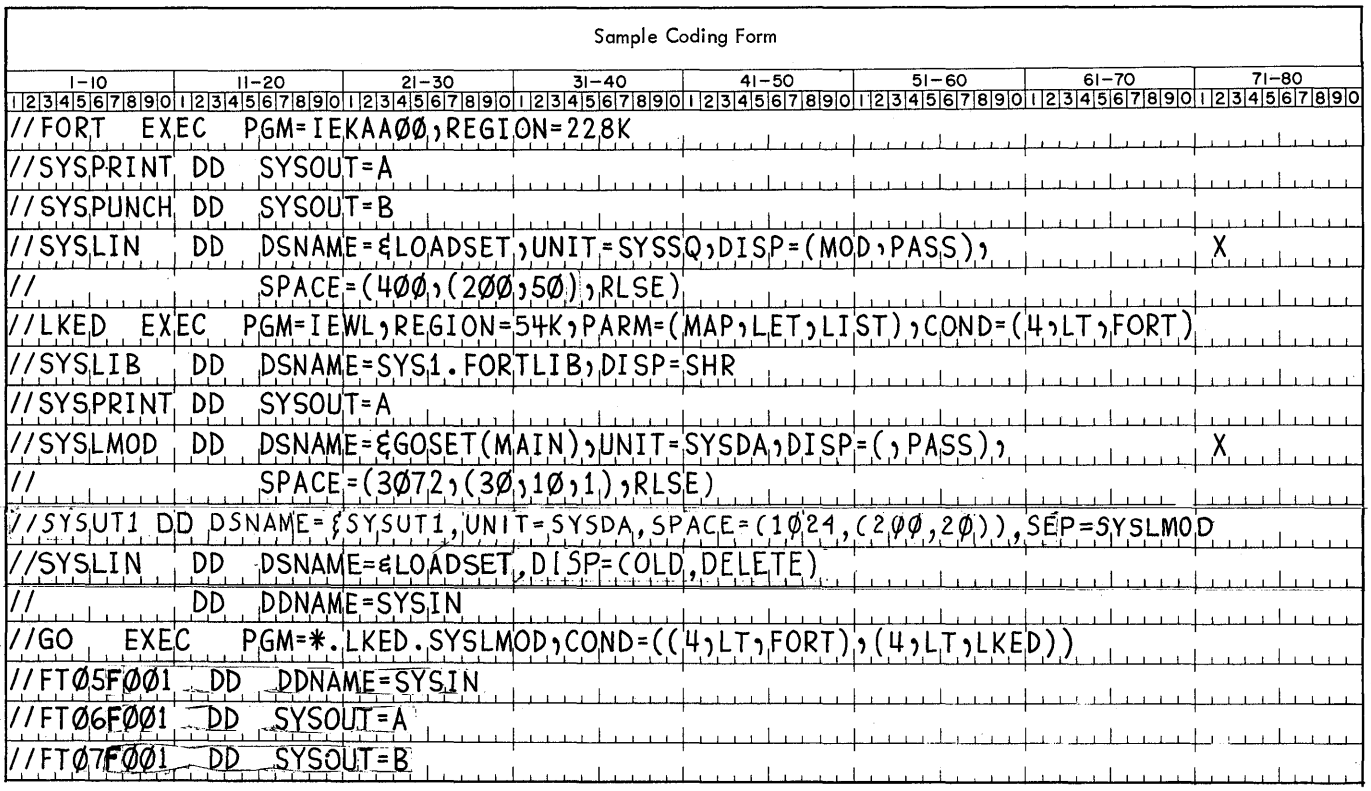


Figure 57. Compile, Linkage Edit, and Execute Cataloged Procedure (FORTHCLG)

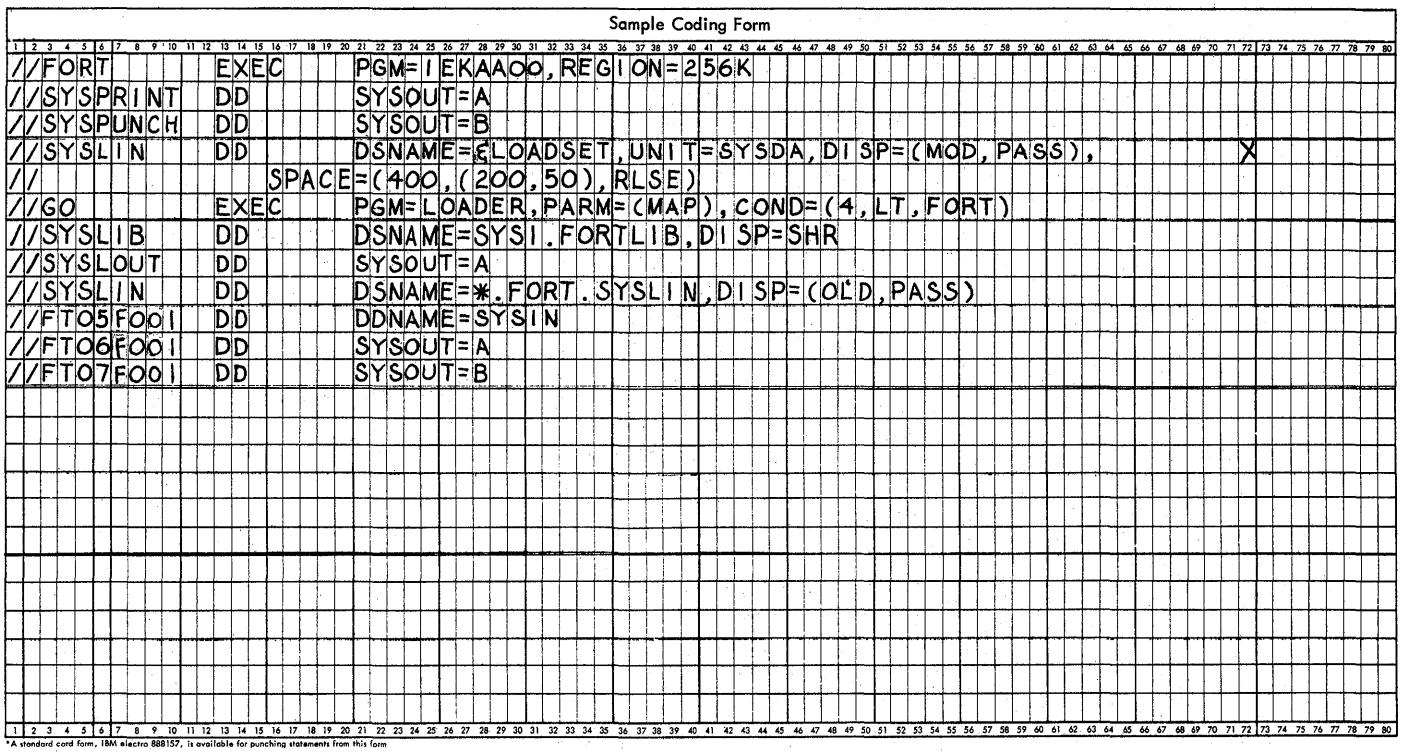


Figure 58. Compile and Load Cataloged Procedure (FORTHCLD)

Overriding Parameters in the EXEC Statement

Two forms of keyword parameters ("keyword" and "keyword.procstep") are discussed in "Job Control Language." The form "keyword.procstep" is used to add or override parameters in an EXEC statement in a cataloged procedure.

The FORTRAN programmer can, for example, add (or override) compiler or linkage editor options for an execution of a cataloged procedure, or he can state different conditions for bypassing a job step.

Note: When the PARM parameter is overridden, all compiler and/or linkage editor options stated in the EXEC statement in the procedure step are deleted and replaced by those in the overriding PARM parameter.

Example 1: Assume the cataloged procedure FORTHC is used to compile a program, and the programmer wants to specify the name of his program and the MAP option. The following statement can be used to invoke the procedure, and to supply the compiler options.

```
//STEP1 EXEC FORTHC,                                X
//     PARM.FORT='MAP,NAME=MYPROG'
```

The PARM options apply to the procedure step FORT.

Example 2: Assume the cataloged procedure FORTHLG is used to linkage edit and execute a module. Furthermore, the XREF option overrides MAP, LET, and LIST in the linkage editor step and the COND parameter is changed for the execution of the load module. The following EXEC statement adds and overrides parameters in the procedure.

```
//DO EXEC FORTHLG,PARM.LKED=XREF,                   X
//     COND.GO=(3,LT,DO.LKED)
```

The PARM parameter applies to the linkage editor procedure step LKED, and the COND parameter applies to the execution procedure step GO.

Example 3: Assume a source module is compiled, linkage edited, and executed using the cataloged procedure FORTHCLG. Furthermore, the compiler option OPT and the linkage editor option XREF are specified, and account number 506 is used for the execution procedure step. The following EXEC statement adds and overrides parameters in the procedure.

```
//STEP1 EXEC FORTHCLG,                    X
//    PARM.FORT='OPT=2',                  X
//    PARM.LKED=XREF,                     X
//    ACCT.GO=506
```

Example 4: Assume a source module is compiled and loaded using the cataloged procedure FORTHCLD. Furthermore an external name is specified as the entry point of the loaded program. The following EXEC statement adds and overrides parameters in the procedure.

```
//STEP1   EXEC   FORTHCLD,PARM.GO='EP=FIRST'
```

Overriding and Adding DD Statements

A DD statement with the name "stepname.ddname" is used to override parameters in DD statements in cataloged procedures, or to add DD statements to cataloged procedures. The "stepname" identifies the step in the cataloged procedure. If "ddname" is the name of a DD statement present in the step, the parameters in the new DD statement override parameters in the DD statement in the procedure step. If "ddname" is the name of a DD statement not present in the step, the new DD statement is added to the step.

In any case, the modification is only effective for the current execution of the cataloged procedure.

When overriding, the original DD statement in the cataloged procedure is copied, and the parameters specified in it are replaced by the corresponding parameters in the new DD statement. Therefore, only parameters that must be changed are specified in the overriding DD statement.

If more than one DD statement is modified, the overriding DD statements must be in the same order as the DD statements appearing in the cataloged procedure. Any DD statements that are added to the procedure must follow overriding DD statements.

Note: The following additional rules apply to overriding in cataloged procedures:

1. In the DCB parameter, individual subparameters can be overridden.
2. To nullify the use of any particular keyword parameter (except the DCB parameter), the overriding DD statement must specify

   keyword=,

3. A parameter can be overridden by specifying a mutually exclusive parameter in the overriding DD statement. For example, in the FORTHC procedure, the SPACE specification for SYSLIN may be overridden by using either the SPLIT or SUBALLOC parameter.

When the procedures FORTHC, FORTHCL, FORTHCLG, and FORTHCLD are used, a DD statement must be added to define the SYSIN data set to the compile step in the procedures (see Figures 16, 17, 21, and 24). With MVT, if SYSUT1 and SYSUT2 DD statements are added to the FORT step, the DSNAME=&SYSUT1 and DSNAME=&SYSUT2 DD parameters should be used in order to employ the dedicated workfile feature of the operating system. For information on dedicated workfiles, see the Job Control Language Reference publication. When the procedure FORTHLG is used, a DD statement must be added to define the SYSLIN data set (see Figures 18 and 19).

When the procedures FORTHCL, FORTHLG, and FORTHCLG are used, an overriding DD statement can be used to write the load module constructed in the linkage editor step in a particular PDS chosen by the programmer, and assign that member of the PDS a particular name.

In execution procedure steps, the programmer can catalog data sets, assign names to data sets, supply DCB information for data sets, add data sets, or specify particular volumes for data sets by using overriding and/or additional DD statements.

Example 1: Assume the data sets identified by ddnames FT04F001 and FT08F001 are named, cataloged, and assigned specific volumes. The following DD statements are used to add this information and indicate the location of the source module.

```
//JOB1 JOB MSGLEVEL=1
//STEP1 EXEC FORTHCLG
//FORT.SYSIN DD *
```

```
|          FORTRAN Source Module          |
```

```
/*
//GO.FT04F001 DD DSNAME=MATRIX,          X
//   DISP=(NEW,CATLG),UNIT=TAPE,         X
//   VOLUME=SER=987K
//GO.FT08F001 DD DSNAME=INVERT,          X
//   DISP=(NEW,CATLG),UNIT=TAPE,         X
//   VOLUME=SER=1020
//GO.SYSIN DD  *
```

```
|          Input to Load Module           |
```

```
/*
```

Example 2: Assume the linkage edit and execute cataloged procedure (FORTHLG) is used. The load module constructed in the linkage editor step is placed in the cataloged partitioned data set MATH and is assigned the member name DERIV.

```
//JOB3 JOB
//STEP1 EXEC FORTHLG
//LKED.SYSLMOD DD DSNAME=MATH(DERIV),    X
//   DISP=(MOD,PASS)
//LKED.SYSIN DD *
```

```
|          FORTRAN Object Module          |
```

```
/*
//GO.SYSIN DD *
```

```
|          Input to Load Module           |
```

```
/*
```

Example 3: Assume the compile, linkage edit, and execute cataloged procedure (FORTHCLG) is used with three data sets in the input stream:

1. A FORTRAN main program MAIN with a series of subprograms, SUB1 through SUBN.
2. A linkage editor control statement that specifies an additional library, MYLIB. MYLIB is used to resolve external references for the symbols ALPHA, BETA, and GAMMA.
3. A data set used by the load module and identified by data set reference number 5 in the source module.

The following example shows the deck structure.

```
//JOBCLG JOB 00,FORTRANPROG,MSGLEVEL=1
//HXECCLGX EXEC FORTHCLG
//FORT.SYSIN DD *
```

```
|       FORTRAN Source Module MAIN        |
|-----------------------------------------|
|       FORTRAN Source Module SUB1        |
|-----------------------------------------|
|                   .                     |
|                   .                     |
|                   .                     |
|-----------------------------------------|
|       FORTRAN Source Module SUBN        |
```

```
/*
//LKED.ADDLIB DD DSNAME=MYLIB
//LKED.SYSIN DD *
  LIBRARY ADDLIB(ALPHA,BETA,GAMMA)
/*
//GO.SYSIN DD *
```

```
|       Input to Load Module              |
```

```
/*
```

The DD statement FORT.SYSIN indicates to the compiler that the source modules are in the input stream. The DD statement LKED.ADDLIB defines the additional library MYLIB to the linkage editor. The DD statement LKED.SYSIN defines a data set that is concatenated with the primary input to the linkage editor. The linkage editor control statements and the object modules appear as one data set to the linkage editor. The DD statement GO.SYSIN defines data in the input stream for the load module.

This section discusses a variety of programming topics that should be considered in writing a FORTRAN program.

STORAGE LOCATIONS AND BYTES

Storage locations in System/360 are called bytes, words, and doublewords. One word is four bytes long; a doubleword is eight bytes long. When data is read into main storage, it is translated into internal format. See Table 18 for storage allocation according to the type and length of the constant or variable.

Table 18. Storage Allocation

Type	Length	Storage
Logical	1	1 byte
	4	4 bytes
Real	4	4 bytes
	8	8 bytes
Integer	2	2 bytes(variable only)
	4	4 bytes
Complex	8	8 bytes
	16	16 bytes
Character (BCD or EBCDIC)	--	1 character/byte
Hexadecimal	--	2 characters/byte

MINIMUM SYSTEM REQUIREMENTS FOR THE FORTRAN IV (G) AND (H) COMPILERS

The operating system is device independent. In particular, the FORTRAN IV (G) and (H) compilers can operate with any combination of devices (shown in Table 4); however, there are certain requirements.

- The FORTRAN IV (G) compiler requires at least a System/360, Model 40, with 128K bytes of storage and a standard instruction set with the floating-point option.
- The FORTRAN IV (H) compiler requires at least a System/360, Model 40, with 256K bytes of storage and the standard instruction set with the floating-point option.
- All programs require a device, such as the 1052 keyboard printer, for direct operator communication.
- At least one direct-access device must be used for residence of the operating system.
- For FORTRAN IV (G), the printer must have at least a 120-character print line; for FORTRAN IV (H), at least a 132-character print line.

BOUNDARY ADJUSTMENT OF VARIABLES IN COMMON BLOCKS AND EQUIVALENCE GROUPS

Variables in a COMMON block or EQUIVALENCE group may be in any order if the BOUNDRY=ALIGN option is specified in the FORTLIB macro instruction during system generation, because boundary alignment violations are corrected during execution. (The FORTLIB macro instruction is described in the System Generation publication.) If the BOUNDRY=NOALIGN option is specified and boundary violations are encountered during execution of the object program, the job terminates.

If the BOUNDRY=ALIGN option of the FORTLIB macro instruction is specified and a boundary violation occurs in a FORTRAN main program or in a FORTRAN or assembler language subprogram, each instruction that refers to the improperly aligned variable requires that (1) the specification exception resulting from this reference be proc- essed, and (2) the boundary alignment routine be invoked. Therefore, considerable programming efficiency is gained if the programmer ensures that all of the variables have proper boundary alignment. The FORTRAN IV Language publication contains information on boundary alignment.

When boundary alignment is performed, program interrupt message IHC210I is issued. (This message is described completely in the section "Program Interrupt Messages" in Appendix D). For boundary alignment, the letter A appears in the text of the message and the code 6 appears in the old PSW (program status word), which is included in the message. The number of warning messages printed is limited to 10. After 10 boundary alignment adjustments have been made, the message is suppressed, but boundary alignment violations continue to be corrected.

Note: Even if BOUNDRY=ALIGN is specified and a boundary error occurs in an EXECUTE, LM (load multiple), or STM (store multiple) instruction in a subprogram written in assembler language, boundary adjustment does not take place and the job terminates. Therefore, if these instructions refer to improperly aligned data, they should not be used in assembler language subprograms.

INDICATORS AND SENSE LIGHTS

At the start of program execution, the divide-check indicator, the overflow indicator, and the pseudo sense lights are not initialized. Therefore, if a programmer intends to use the indicators or sense lights, he should initialize them prior to use; otherwise, erroneous results may be obtained. (For additional information, see the FORTRAN IV Library publication.)

CONDITIONAL BRANCHING

A test for 0.0 in an IF statement is not recommended. Floating-point round-off errors may cause the low-order bit(s) to be set. Therefore, the test for 0.0 may not yield the expected result.

ARITHMETIC IF STATEMENT

A fixed-point overflow condition results in the following action:

- In FORTRAN (G), if the integer is positive, a negative branch is taken, i.e., the first branch. If the integer is negative, a positive branch is taken, i.e., the third branch.
- In FORTRAN (H), the zero (middle) branch is always taken.

USE OF STOP N STATEMENT

There are no checks made to determine if a value of n greater than 4095 is used in the STOP n statement. 4095 is the maximum value that can be used for n and still fit into the 3 digits used for the user condition-code. Any value of n greater than 4095 overflows into the system condition code.

REGISTER 15 AS A CONDITION CODE REGISTER

Register 15 is used by the compilers as a condition code register, a RETURN code register, and a STOP code register (STOP code = condition code). The particular values that Register 15 can contain and their explanations follow:

16 -- A terminal error has been detected during execution in a subprogram (an IHCxxxI message is generated).

4*i -- A RETURN i statement has been executed in a subprogram (i is a RETURN code).

n -- A STOP n statement has been executed (n is the condition code).

0 -- A RETURN or a STOP statement has been executed in either a main program or a subprogram (0 is a RETURN code or a condition code).

Note: Both FORTRAN (G) and (H) will generate a STOP (i.e., 0 is the condition code) for a RETURN or RETURN i issued in a main program.

USE OF EMBEDDED BLANKS IN FORTRAN PROGRAMS

To improve the readability of a source program, the programmer may use any number of blanks when writing FORTRAN statements. Except for literal data, in which blanks are retained as coded in the source statement, blanks are normally ignored by the compilers. Thus, the statements IJ = 10 is the equivalent of IJ=10. Both statements are syntactically correct assignment statements and are executed as such, i.e., a value of 10 is assigned to the variable IJ.

USE OF DUMP AND PDUMP

Under the operating system, a program may be loaded into different areas of storage for different executions of the same job. The following conventions should be observed when using the DUMP or PDUMP subroutine to insure that the appropriate areas of storage are dumped.

If an array and a variable are to be dumped at the same time, a separate set of arguments should be used for the array and for the variable. The specification of limits for the array should be from the first element in the array to the last

element. For example, if an array TABLE is dimensioned as:

```
DIMENSION TABLE (20)
```

the following statement could be used to dump TABLE and the real variable B in hexadecimal format and terminate execution after the dump is taken:

```
CALL DUMP (TABLE(1),TABLE(20),0,B,B,0)
```

If an area in COMMON is to be dumped at the same time as an area of storage not in COMMON, the arguments for the area in COMMON should be given separately. For example, if A is a variable in COMMON, the following statement could be used to dump the variables A and B in real format without terminating execution:

```
CALL PDUMP (A,A,5,B,B,5)
```

If variables not in COMMON are to be dumped, the programs should list each variable separately in the argument list. For example, if R, P, Q are defined implicitly in the program, the statement

```
CALL PDUMP(R,R,5,P,P,5,Q,Q,5)
```

should be used to dump the three variables. If

```
CALL PDUMP(R,Q,5)
```

is used, all main storage between R and Q is dumped.

If an array and a variable are passed as arguments to a subroutine, the arguments in the call to DUMP or PDUMP in the subroutine should specify the parameters used in the definition of the subroutine. For example, if the subroutine SUBI is defined as:

```
SUBROUTINE SUBI(X,Y)
DIMENSION X(10)
```

and the call to SUBI within the source module is:

```
DIMENSION A(10)
     .
     .
     .
CALL SUBI(A,B)
```

then the following statement in the subroutine should be used to dump the variables in hexadecimal format without terminating execution:

```
CALL PDUMP (X(1),X(10),0,Y,Y,0)
```

If the statement

```
CALL PDUMP (X(1),Y,0)
```

is used, all storage between A(1) and Y is dumped, due to the method of transmitting arguments. (Y does not occupy the same storage location as B.)

USE OF ERR PARAMETER IN READ STATEMENT

Use of the optional ERR parameter for a READ statement can indicate the source program statement to which transfer should be made if an error is encountered during data transfer. When transfer has been made to that statement, the first subsequent READ in the source program provides the record that was in error. If this is not the record desired, an additional READ should be issued.

If the ERR parameter is omitted from the READ statement, an input/output device error terminates program execution.

ARITHMETIC STATEMENT FUNCTIONS

The Arithmetic Statement Functions (ASF) can be used to cause selective "automatic" typing. For example, the ASF,

```
SQRT(X)=DSQRT(X)
```

causes the desired function name substitution, so that each use of SQRT(X) in an expression in the program will cause execution of DSQRT(X).

To accomplish the substitution, the argument type(s) in the ASF should agree with the type of argument(s) required for the desired subprogram. In the example above, X should be typed as REAL*8. Furthermore, the function name itself should be typed to agree with the function value type of the desired subprogram. In the example, SQRT should be typed as REAL*8.

FORTRAN (H) does not actually require that an ASF argument be typed as REAL*8, but because FORTRAN (G) does, it is recommended it be done for both compilers to ensure interchangeability. However, the actual argument should be REAL*8 or be typed to be REAL*8.

G ONLY | USE OF ASSIGN STATEMENT

The FORTRAN IV (G) compiler uses table entries produced in one of its phases to perform storage allocation for the

variables defined in the source module. This process may result in an unusual amount of compilation time if a large number of ASSIGN statements are specified in the source program. The use of a large number of ASSIGN statements should be avoided.

G ONLY | DO LOOP OPTIMIZATION

The following discussion applies to FORTRAN IV (G) only. For information on FORTRAN IV (H) DO loop optimization, see Appendix G, "FORTRAN IV (H) Optimization Facilities."

During the operation of the FORTRAN IV (G) compiler, one complete phase is included for the purpose of DO loop optimization.

Each loop is recorded internally as it is encountered in the source module. As each step of the optimization process progresses, the loops are further categorized for ease of reference in generating the corresponding object code.

If loops are nested, the end of each loop is denoted by a special reserve mark, which is placed at the end of the intermediate notation that is being produced. The level of nesting is also recorded for each group of nested loops. This minimizes execution time in determining at object time the depth to which calculation must be maintained to close the first loop of the nest.

A further categorization divides the loops into standard and non-standard. The compiler determines, based on frequency of use, which subscript expressions are to appear in general registers (standard) and which are to remain in storage (nonstandard). This method enables the compiler to make register assignments prior to the final generation of the object code. In this way, addresses are retrieved and inserted into the designated instruction without unnecessary repeated address calculation.

G ONLY | DUMMY ARGUMENT REFERENCES

In a subprogram, any dummy argument beyond the 128th should not be referred to within a DO loop.

H ONLY | SUPPORT OF AND, OR, AND COMPL

The functions listed in Table 19 are not part of the standard FORTRAN language, but are currently supported by the (H) compiler. Caution should be exercised in their use since continued support is not assured.

DATA INITIALIZATION STATEMENT

To initialize an array, the programmer should consider the following points:

1. He may initialize any element of an array by subscripting the array name. Only one element is initialized; if any excess characters are specified, they are truncated and not placed into the next element. (Overflow from one element to the next is known as spill.) A partially filled array element is padded on the right with blanks. The following example illustrates how individual array elements may be initialized:

```
DIMENSION A(10)
DATA A(1),A(2),A(4),A(5)/'ABCD',
 'QRSTUVW','123','6666'/
```

```
A(1) contains ABCD
A(2) contains QRST
A(3) is not initialized (note that
     spill does not occur for a
     subscripted array name)
A(4) contains 123b
A(5) contains 6666
A(6) through A(10) are not
     initialized.
```

2. Several consecutive elements of an array may be initialized with a single literal constant by specifying the array name without a subscript. Data spill occurs through as many elements as are necessary to insert the constant. If the last element initialized is only partially filled, it is padded on the right with blanks. (Any subsequent array elements are not initialized; that is, their contents are unchanged.) Truncation occurs if the constant exceeds the limit of the

array. The following example illustrates how several array elements may be initialized with one constant:

```
DIMENSION ARRAY(9)
DATA ARRAY/
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'/

ARRAY(1) contains ABCD
ARRAY(2) contains EFGH
ARRAY(3) contains IJKL
ARRAY(4) contains MNOP
ARRAY(5) contains QRST
ARRAY(6) contains UVWX
ARRAY(7) contains YZbb
ARRAY(8) and ARRAY(9) are not
    initialized.
```

Note that data spill occurs only at the beginning of an array. To begin data spill in the middle of an array, the EQUIVALENCE statement is used in the following manner:

```
DIMENSION ARRAYA(10),ARRAYB(5)
EQUIVALENCE (ARRAYA(6),ARRAYB(1))
DATA ARRAYB/'ABCDEFGHIJKLMNOPQRST'/

ARRAYA(1) through ARRAYA(5) are not
    initialized.
ARRAYA(6) contains ABCD
ARRAYA(7) contains EFGH
ARRAYA(8) contains IJKL
ARRAYA(9) contains MNOP
ARRAYA(10) contains QRST
```

3. The FORTRAN language requires that there must be a one-to-one correspondence between data elements and initializing constants. However, this correspondence may be violated when using data spill, both the (G) and (H) compilers will flag use of this technique but will not terminate processing. In this case, each constant should be specified immediately after the name of the array or element it is to initialize. In the following example, an array is initialized (using data spill); then a variable is initialized.

```
DIMENSION A(3)
DATA A/'ABCDEFGHIJKL'/,X/'MNOP'/

A(1) contains ABCD
A(2) contains EFGH
A(3) contains IJKL
X    contains MNOP
```

If each constant is not specified immediately after its associated array or variable name, overlay of spilled data may occur, as shown in the following example:

```
DIMENSION A(3)
DATA A,X/'ABCDEFGHIJKL',10.0/

A(1) contains ABCD
A(2) contains 10.0
A(3) contains IJKL
X    is not initialized
```

In this example, the second element of the array is overlaid by the second initializing constant.

OBJECT TIME INPUT/OUTPUT EFFICIENCY

FORTRAN processing time can be appreciably reduced by the use of programming techniques that result in greater data transfer efficiency. Such techniques are particularly important in executing programs that require substantial input/output operations. Discussed below are four programming areas in which the correct choice of programming method can increase FORTRAN processing speed.

READ/WRITE TYPE: The unformatted form of the READ and WRITE statement provides the fastest data transfer rate. For most efficient processing, therefore, the

Table 19. Additional Built-In Functions -- (H) Compiler

Function	Entry Name	In-Line I	No. of Arguments	Type of Arguments	Type of Function Value
Logical intersection of two arguments	AND	I	2	Real*4 or Integer*4	Real*4
Logical union of two arguments	OR	I	2	Real*4 or Integer*4	Real*4
Logical 1's complement of argument	COMPL	I	1	Real*4 or Integer*4	Real*4

unformatted form should be used to transfer information to or from an intermediate data set, a data set that is written out during a program, not examined by the programmer, and then read back for additional processing later in the program or in another program. Thus, for an intermediate data set, statement 11 in the following example is to be preferred to statement 9.

```
      COMMON A(10), B(10)
      DIMENSION D(20)
      EQUIVALENCE (A(1), D(1))

  9   WRITE (10,10)A, B
 10   FORMAT (10E13.3/)
 11   WRITE (9) D
```

IMPLIED DO: Array notation is far more efficient than the indexing capability of an implied DO in an I/O list. Thus, for efficiency, the statement WRITE (9) A (where A is an array name) is preferable to WRITE (9) (A(I),I=1,10).

EQUIVALENCE STATEMENT: In FORTRAN, on input, data is taken from a record and placed into storage locations that are not necessarily contiguous. On output, data is normally gathered from diverse storage locations. Input/output operations, however, can be made more efficient by storing and retrieving data from contiguous locations.

To construct an efficient READ or WRITE statement for an I/O list consisting of many variables, use a COMMON or named COMMON statement to force all the variables in the list to be allocated contiguous storage space. Next, use an EQUIVALENCE statement to define a single dimensioned variable that is the same length as the list of variables. Finally, use a WRITE on the single-dimensioned variable using array notation. The following example illustrates this technique:

```
      COMMON/LISTA/A(10),B(8),C,D,I,K,L(10)
      REAL*8 B
      COMPLEX*16 LIST(10)
      EQUIVALENCE(A(1),LIST(1))

      WRITE(9) LIST
```

BACKSPACE STATEMENT: Use of the BACKSPACE statement is not recommended if efficient processing is desired.

DATA DEFINITION CONSIDERATIONS

The DCB parameter of the DD statement allows for the redefinition of many data set characteristics at execution time. Those specifications that most concern the FORTRAN programmer are discussed below. For a full description of the DCB parameter, see the Supervisor and Data Management Macro Instructions publication.

BLKSIZE: The BLKSIZE subparameter specifies the buffer size to be used; the maximum is 32K. As a general rule, for tape, the larger the blocksize, the more efficient the processing. (Note that for tape, the user should not specify a blocksize of less than 18 bytes. Records of less than 18 bytes may be lost when read.) On disk, specifying the full track as a blocksize is more efficient than specifying a partial track. The blocksize specified should be large enough to hold the largest logical record produced. No spanning of a logical record into physical records will then occur.

BUFNO: The BUFNO subparameter specifies the number of buffers to be used. If a value of 1 is specified for BUFNO, single buffering is provided. If either no value or any value other than 1 is specified, double buffering, which offers an overlap advantage, is provided.

RECFM: The RECFM subparameter specifies both record format and the use of blocking. When records are blocked, fewer I/O requests are made to a device during the processing of logical records; I/O processing speeds are thereby increased. In general, large blocking factors improve performance. formance. (See "Record Format" for additional information.)

OPTCD: OPTCD=C requests the use of chained scheduling, a feature that results in the decrease of I/O transfer time. Chained scheduling is put into effect only when an I/O request is received before a previous I/O request has ended. For this reason it is difficult to predict when chained scheduling will be effective. However, the use of chained scheduling will provide a performance improvement in the formatting that is done with a new direct access data set. For sequential data sets the user may wish to measure the effect before selecting chained scheduling for production runs.

DIRECT-ACCESS PROGRAMMING

Using direct-access I/O rather than sequential I/O can decrease load module execution time: the direct access statements in the FORTRAN IV language enable the programmer to retrieve a record from any place on the volume without reading all the records preceding that record in the data set. For efficiency, direct data sets should be pre-formatted.

If, however, the NEW subparameter is specified in the DD statement for the data set, a FORTRAN-supplied load module will format the data set before the program begins processing.

Note: Direct-access I/O statements and sequential I/O statements may not be used to process the same direct data set within the same FORTRAN load module. However, sequential I/O statements may process under format control a direct data set in one load module, while direct access I/O statements process it in another.

Not all applications are suited to direct-access I/O, but an application that uses a large table that must be held in external storage can use direct-access I/O effectively. An even better example of a direct-access application is a data set that is updated frequently. Records in the data set that are updated frequently are called master records. Records in other data sets used to update the master records are called detail records.

Each of the master records should contain a unique identification that distinquishes this record from any other master record. Detail records used to update the masters should contain an identification field that identifies a detail record with a master record. For example, astronomers might have assigned unique numbers to some stars, and they wish to collect data for each star on a data set. The unique number for each star can be used as identification for each master record, and any detail record used to update a master record for a star would have to contain the same number as the star.

A FORTRAN program indicates which record to FIND, READ, or WRITE by its record position within the data set. The ideal situation would be to use the unique record identification as the record position. However, in most cases this is impractical. The solution to this problem is a randomizing technique. A randomizing technique is a function which operates on the identification field and converts it to a record position. For example, if 6-digit numbers are assigned to each star, the randomizing technique may truncate the last two digits of the number assigned to the star and use the remaining four digits as a record position. For example, star number 383320 would be assigned position 3833. Another example of a randomizing technique would be a mathematical operation performed on the identification number, such as squaring the identification number and truncating the first four digits and the last four digits of the result. Then the record for star number 383320 is assigned record position 3422. There is no general randomizing technique for all sets of identification numbers. The programmer must devise his own technique for a given set of identification numbers.

Two problems arise when randomizing techniques are used. The first problem is that there may be a lot of space wasted on the volume. The solution in this instance must be developed within the randomizing technique itself. For example, if the last two digits on the identification numbers for stars are truncated and no star numbers begin with zero, the first thousand record positions are blank. Then a step should be added to the randomizing technique to subtract 999 from the result of the truncation.

The second problem is that more than one identification may randomize to the same record location. For example, if the last two digits are truncated, the stars identified by numbers 383320, 383396, and 383352 randomize to the same record location - 3833. Records that randomize to the same record location are called synonyms. This problem can be solved by developing a different randomizing technique. However, in some situations this is difficult, and the problem must be solved by chaining.

Chaining is arranging records in a string by reserving an integer variable in each record to point to another record. This integer variable will contain eitner an indicator showing that there are no more records in this chain, or the record location of the next record in the chain. Records chained together are not adjacent to each other. Figure 59 shows the records for star numbers 383320, 383396, and 383352.

When records are chained, the first record encountered for a record position is written in the record position that resulted from randomizing the identification number. Any records that then randomize to that same record location must be written in record positions to which no other record identifications randomize. The space for these synonyms can be allocated either at the end or the beginning of the data set. However, this space must be allocated when the data set is first written. For example, if the randomizing technique assigns master records to record locations between 1 and 9999, the programmer may wish to reserve record locations 10000 to 12000 for master records that become synonyms.

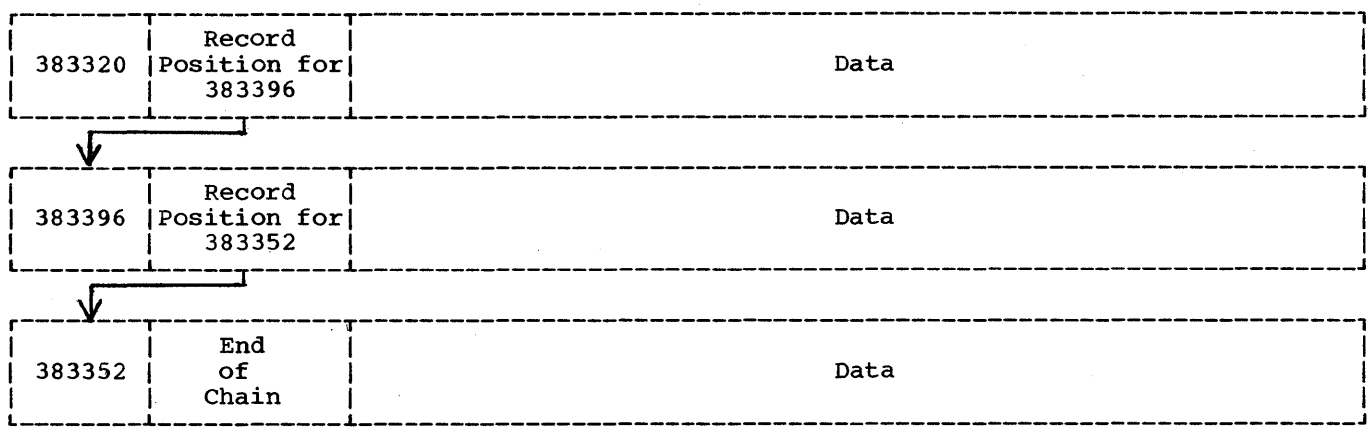


Figure 59. Record Chaining

The programmer must keep a record location counter to keep track of the space assigned for synonyms. When a synonym is inserted in this space, the record location counter must be incremented. The programmer should set up a dummy record in his data set to maintain this record location counter. When the direct-access data set is created, the record location counter should be set at the lower limit of the record positions available for synonyms (i.e., record location 10000 in the example used above).

Also an indicator should be reserved to indicate to the program that the end of a chain has been reached. Since no record position is designated as 0, 0 can be used to indicate the end of a chain.

Before a FORTRAN program writes a direct-access data set for the first time, the data set must be created by writing "skeleton records" in the space that is to be allocated for the direct-access data set. These skeleton records should be written by an installation-written program. After the skeleton records are written, the direct-access data set must be classified as OLD in the DISP parameter of the DD statement. However, if the skeleton records are not written before direct-access records are written by the FORTRAN program for the first time, a FORTRAN load module automatically creates the data set and writes the skeleton records. The programmer indicates that skeleton records have not been written by specifying NEW in the DISP parameter. When the data set is opened, records are initialized as blank records (hexadecimal 40). If unformatted WRITE statements are then encountered in the program, the buffer for the data set is initialized to binary zeroes before the data is placed in the buffer. If formatted WRITE statements are encountered, the buffer is initialized to blanks.

Figure 60 shows a block diagram of the logic that can be used to write a direct access data set for the first time. The block diagram does not show any attempt to write skeleton records.

Example 3 in Appendix B shows a program and job control statements used to update a direct-access data set.

DIRECT-ACCESS PROGRAMMING CONSIDERATIONS

In a job that creates a data set that will reside on a direct-access device and will be processed by some non-FORTRAN program, the DCB subparameter of the DD statement must specify DSORG = DA. This specification causes the creation of a label indicating a direct-access data set. (See "Creating a Direct Data Set" in the IBM Supervisor and Data Management Services publication.) If the direct-access data set will not be processed by a non-FORTRAN program, the DSORG parameter need not be specified since the default specification, DSORG=PS, is the one required.

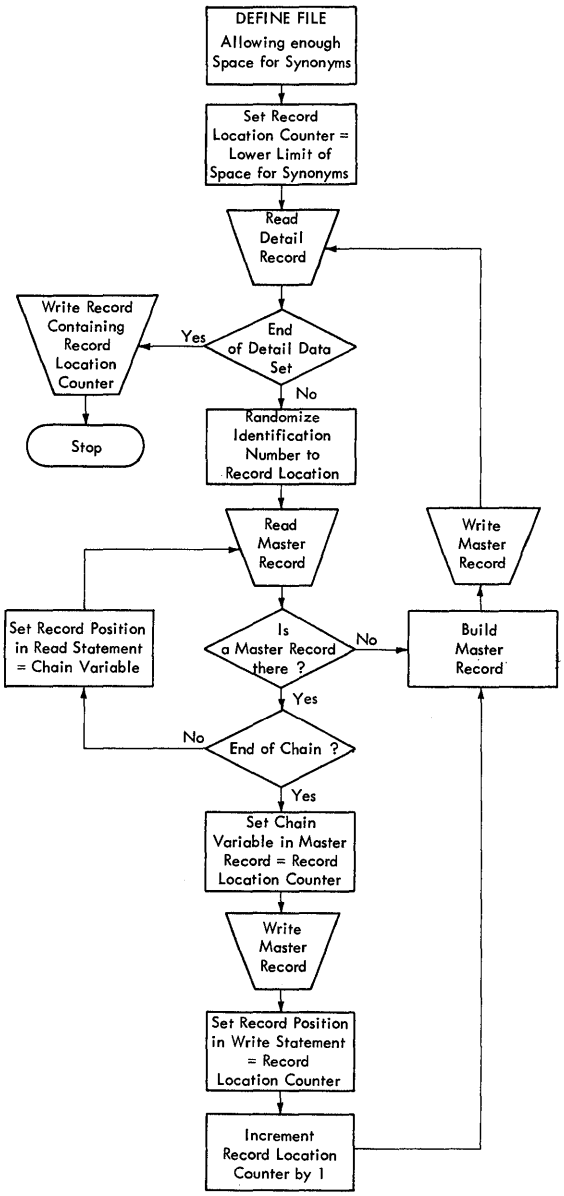


Figure 60. Writing a Direct-Access Data Set for the First Time

Space must be allocated in the SPACE parameter of the DD statement for a data set written on a direct-access volume. For direct-access data sets, the space allocated in the SPACE parameter should be consistent with the record length and number of records specified in the DEFINE FILE statement in the FORTRAN program. For example, in the DEFINE FILE statement

```
DEFINE FILE 8(1000,40,E,I)
```

the number of records is specified as 1000 and the record length is specified as 40. When this program is executed, the DD statement for this data set should contain the SPACE parameter

```
SPACE=(40,(1000))
```

indicating that space is allocated for 1000 records, with 40 bytes for each record.

The DEFINE FILE statement for a data set must be in a program unit that will not be overlaid, but does not have to be in the same program unit in which I/O operations occur. (If the DEFINE FILE is coded in a root segment, there is little chance of error.) For example, the DEFINE FILE statement can be given in a main program with a subprogram performing the I/O operations on the data set. However, if an associated variable defined in the main program is to be used by a subprogram, it must be passed to the subprogram in COMMON. Since an associated variable is updated by I/O operations, the subprogram cannot get to the updated value to make use of it in its operations unless the associated variable is in COMMON.

An associated variable should not be passed as a parameter between a main program and its subprograms because the associated variable is not passed in the same way that other variables are passed. Other variables reflect the result of any operations performed on them in the subprogram. An associated variable (if passed as a parameter) is not changed by operations performed on it in the subprogram.

The FIND statement permits record retrieval to occur concurrently with computation or I/O operations performed on different data sets. By using the FIND statement, load module execution time can be decreased. For example, the statements

```
10  A=SQRT(X)
    .
    .
    .
52  E=ALPHA+BETA*SIN(Y)
64  WRITE(9)A,B,C,D,E
76  READ(8'101)X,Y
```

are inefficient because computations are performed between statements 10 and 52 and an I/O operation is performed on another data set while record number 101 could be retrieved. If the following statements are substituted, the execution of this module becomes more efficient because record number 101 is retrieved during computation and I/O operations on other data sets.

```
 5  FIND(8'101)
10  A=SQRT(X)
    .
    .
    .
52  E=ALPHA+BETA*SIN(Y)
65  WRITE(9)A,B,C,D,E
76  READ(8'101)X,Y
```

G ONLY | COMPILER RESTRICTIONS

- The maximum level of nesting for DO loops and implied DOs is 25.
- The maximum number of expressions that can be nested is 100.
- The maximum level of nested references in an arithmetic statement function definition to another statement function or a function subprogram is 25.
- The maximum number of source cards for one compilation is dependent upon the amount of storage available to the compiler. A 400 card program requires approximately 90K bytes in PCP or MFT systems and 100K bytes in MVT systems. However, depending upon the complexity of a program, more storage may be required. For example, a program containing a high incidence of input/output statements generates more internal code, resulting in greater storage size requirements.
- The maximum number of comment cards between two statements is 30. The maximum number of continuation cards between two statements is 19. There is no restriction on the number of comment cards at the beginning of a deck.
- The repetition field (a) for format codes in a FORMAT statement, if present, must be an unsigned integer constant less than 256. This restriction also applies to execution-time formats.
- The FORMAT statement specification w, indicating the number of characters of data in the field, must be an unsigned integer constant less than 256. This restriction also applies to execution-time formats.
- The FORMAT statement specification r, specifying the position in the FORTRAN record where the transfer of data is to begin, must be an unsigned integer constant less than 256.
- In literal constants in the source program, any valid card code is permissible, except a 12-11-0-7-8-9 punch.

H ONLY | COMPILER RESTRICTIONS

- The maximum level of nesting for DO loops is 25.
- The maximum number of implied DOs per statement is 20.
- The maximum number of characters allowed in a literal constant is 255.
- The maximum number of characters allowed in a PAUSE message is 255.
- The maximum number of nested references to another statement function within a statement function definition is 50; the maximum number of times a statement function may be nested is 50.
- The repetition field (a) for FORMAT codes in a FORMAT statement, if present, must be an unsigned integer constant less than 256. This restriction also applies to execution-time formats.
- The FORMAT statement specification w, indicating the number of characters of data in the field, must be an unsigned integer constant less than 256. This restriction also applies to execution-time formats.
- The FORMAT statement specification r, specifying the position in the FORTRAN record where the transfer of data is to begin must be an unsigned integer constant less than 256.
- The debug facility is not supported.
- The maximum number of arguments in a CALL statement is 196. If an argument has a variable subscript, that argument is counted as two arguments.
- The maximum number of arguments in unique parameter lists in an entire program is dependent on the size of the compiler, with a maximum of 1024.
- The maximum number of arguments in a Statement Function Definition is 20.

(If any of the three preceding restrictions is violated, message IEK550I-"PUSHDOWN, ADCON, OR ASF ARGUMENT TABLE EXCEEDED" - is issued.)

- The maximum number of literal constants and arguments in unique parameter lists contained in an entire program is approximately 990. (If this restriction is violated, message IEK500I-"SOURCE PROGRAM IS TOO LARGE"- is issued. Either the program must be segmented or the number of literal constants and arguments must be reduced.)

Note: In this version of the compiler, Statement Functions are expanded inline.

H ONLY

COMPILER DATA SET CONCATENATION

The SIZE compiler option should be specified when using concatenated data sets with unlike attributes as input to the compiler. Failure to specify SIZE, may result in abnormal termination of the compile step due to unsatisfied buffer space requests. For a description of the SIZE compiler option refer to the heading "SIZE=nnnnK" in the section "FORTRAN Job Processing."

LIBRARY CONSIDERATIONS

The FORTRAN library is a group of subprograms residing in the partitioned data set SYS1.FORTLIB. For a detailed description of the FORTRAN library, see the FORTRAN IV Library Subprograms publication, Order No. GC28-6818. A programmer can change the subprograms in the library; he can add, delete, or substitute library subprograms; or he can create his own library. These topics are discussed in detail in the Utilities publication, Order No. GC28-6586.

When the FORTRAN library is changed for maintenance or to provide additional features, precompiled programs in a user library require special attention to benefit from the changed library modules. This can be accomplished by using the linkage edit facilities to include the current library modules, and storing the resultant load module back into the FORTRAN library. When the facilities of the linkage editor are used to provide an overlay structure or to replace a single control section, care should be taken not to mix FORTRAN library modules that are at diverse operating system levels.

This requirement to maintain all library routines at the same level also applies to IHCADJST. This module is dynamically loaded from the system link library when a specification exception occurs. Therefore, even though a FORTRAN program is in an executable form, if it is at a previous release level, it will still LOAD the current IHCADJST with which it may be incompatible.

DD STATEMENT CONSIDERATIONS

Several DD statement parameters and subparameters are provided for I/O optimization (see Figure 61). Other DD statement parameters are discussed in "Job Control Language" and "Creating Data Sets."

Channel Optimization

The SEP parameter indicates that I/O operations for specified data sets are to use separate channels (channel separation), if possible. The I/O operations for the data set, defined by the DD statement in which

SEP=(ddname[,ddname]...)

appears, are assigned to a channel different from those assigned to the I/O operations for data sets defined by the DD statements "ddname". Assigning data sets whose I/O operations occur at the same time to different channels increases the speed of I/O operations.

I/O Device Optimization

UNIT subparameters can be specified for device optimization.

VOLUME MOUNTING AND DEVICE SEPARATION:

UNIT=(name{,n / ,P}[,DEFER]

[,SEP=(ddname[,ddname]...)])

can be specified for volume mounting and device separation. The "name" and number of units are discussed in the section "Data Definition Statement".

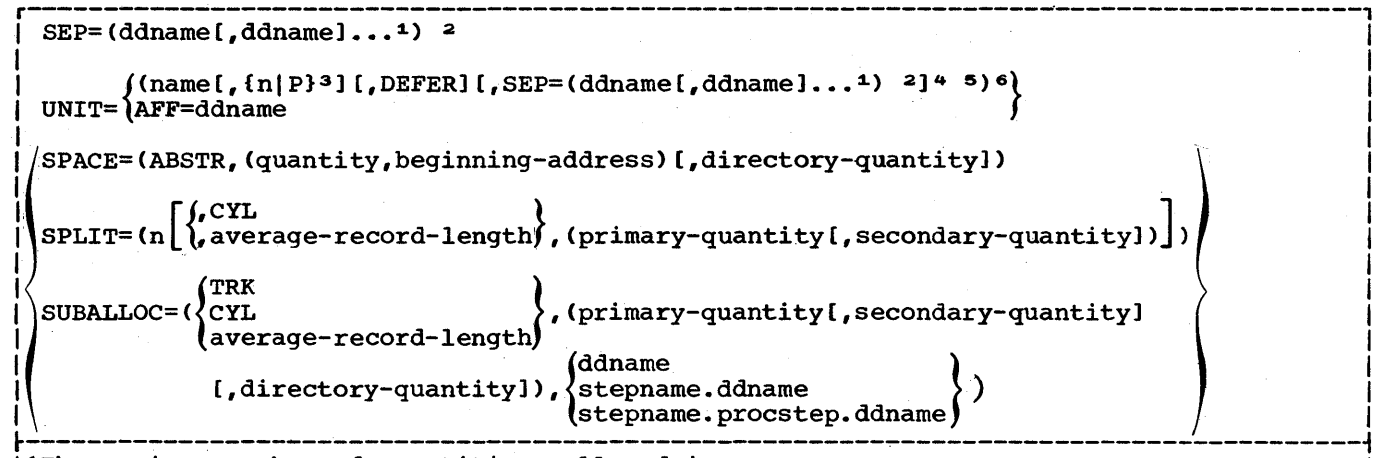


Figure 61. DD Statement Parameters for Optimization

DEFER
indicates that the volume(s) for the data set need not be mounted until needed. The control program notifies the operator when to mount the volume. Deferred mounting cannot be specified for a new output data set on a direct-access device.

SEP=(ddname[,ddname]...)
is used when a data set is not assigned to the same access arms on direct-access devices as the data sets identified by the list of ddnames. This subparameter is used to decrease access time for data sets and is meaningful only for direct-access devices. The operating system provides device separation if possible, but ignores the SEP subparameter if an insufficient number of access arms is available. The SEP subparameter in the UNIT parameter provides for device separation, while the SEP parameter provides for channel separation.

DEVICE AFFINITY: The use of the same device by data sets is specified by:

UNIT=AFF=ddname

The data set defined by the DD statement in which this UNIT parameter appears uses the same device as the data set defined by the DD statement "ddname" in the current job step.

Note: Channel separation and affinity requests are ignored if the system's automatic volume recognition feature is used.

Direct-Access Space Optimization

The SPACE parameter can be used to specify space beginning at a designated track address on a direct-access volume. The SPLIT or SUBALLOC parameters may be specified instead of SPACE to split the use of cylinders among data sets on a direct access volume or to use space specified for another data set which it did not use. (The other SPACE parameter is discussed in "Creating Data Sets.")

SPACE BEGINNING AT A SPECIFIED ADDRESS:

SPACE=(ABSTR,(quantity,beginning-address)
[,directory-quantity])
specifies space beginning at a particular track address on a direct access volume. The "quantity" is the number of tracks allocated to the data set. The "beginning address" is the relative track address on a direct access volume where the space begins. If the data set is a new partitioned data set (PDS) the "directory-quantity" specifies the number of 256-byte blocks that are allocated to the directory of the data set.

SPLITTING THE USE OF CYLINDERS AMONG DATA SETS: If several data sets use the same direct-access volume in a job step, processing time can be saved by splitting the use of cylinders among the data sets. Splitting cylinders eliminates seek operations on separate cylinders for different data sets. Seek operations are measured in milliseconds, while the data transfer is measured in microseconds.

```
SPLIT=(n[{,CYL                    }
         {,average-record-length}
         ,(primary-quantity
         [,secondary-quantity])])
```

is substituted for the SPACE parameter when the use of cylinders is split. If CYL is specified, "n" indicates the number of tracks per cylinder to be used for this data set. If "average record length" is specified, "n" indicates the percentage of tracks per cylinder used for this data set. The remaining subparameters are the same as those specified for SPACE in "Creating Data Sets."

More than one DD statement in a step will use the SPLIT parameter. However, only the first DD statement specifies all the subparameters; the remaining DD statements need only specify "n". For example,

```
//STEP4 EXEC PGM=TESTFI
//FT08F001 DD SPLIT=(45,800,(100,25))
            .
            .
            .
//FT17F001 DD SPLIT=(35)
            .
            .
            .
//FT23F001 DD SPLIT=(20)
```

ACCESSING UNUSED SPACE: Data sets in previous steps may not have used all the space allocated to them in a DD statement. The SUBALLOC parameter may be substituted for the SPACE parameter to permit a new data set to use this unused space.

```
SUBALLOC=({TRK                    }
          {CYL                    }
          {average-record-length}
          (,primary-quantity
          [,secondary-quantity]
          [,directory-quantity])
          {ddname                 }
          {stepname.ddname        }
          {stepname.procstep.ddname})
```

The data set from which unused space is taken is defined in the DD statement "ddname", which appears in the step "stepname." (The step must be in the current job.) The other subparameters specified in the SUBALLOC parameter are the same as the subparameters described for SPACE in "Creating Data Sets."

SYSTEM OUTPUT

The compilers, the linkage editor, and load modules produce aids which may be used to document and debug programs. This section describes the listings, maps, card decks, and error messages produced by these components of the operating system.

COMPILER OUTPUT

Both the FORTRAN IV (G) and the FORTRAN IV (H) compilers can generate, depending upon user-specified options, a listing of source statements, a table of source module names, an object module listing, and an object module card deck. Additionally, the (H) compiler can generate a structured listing of source statements, and a table of source module labels.

Source module diagnostic messages are also produced during compilation.

Source Listing

If the SOURCE option of the PARM parameter of the EXEC statement is specified, the source listing is written in the data set specified by the SYSPRINT DD statement. An example of a source program listing is shown in Figure 62. This printout is the source listing of the sample program shown in Figure 63. (This program will be used throughout the remainder of this publication for purposes of illustration.)

Storage Map

If the MAP option of the PARM parameter of the EXEC statement is specified, a table of names, which appear (or are implied) in the source module, is written in the data set specified by the SYSPRINT DD statement. The storage map produced differs according to the compiler used.

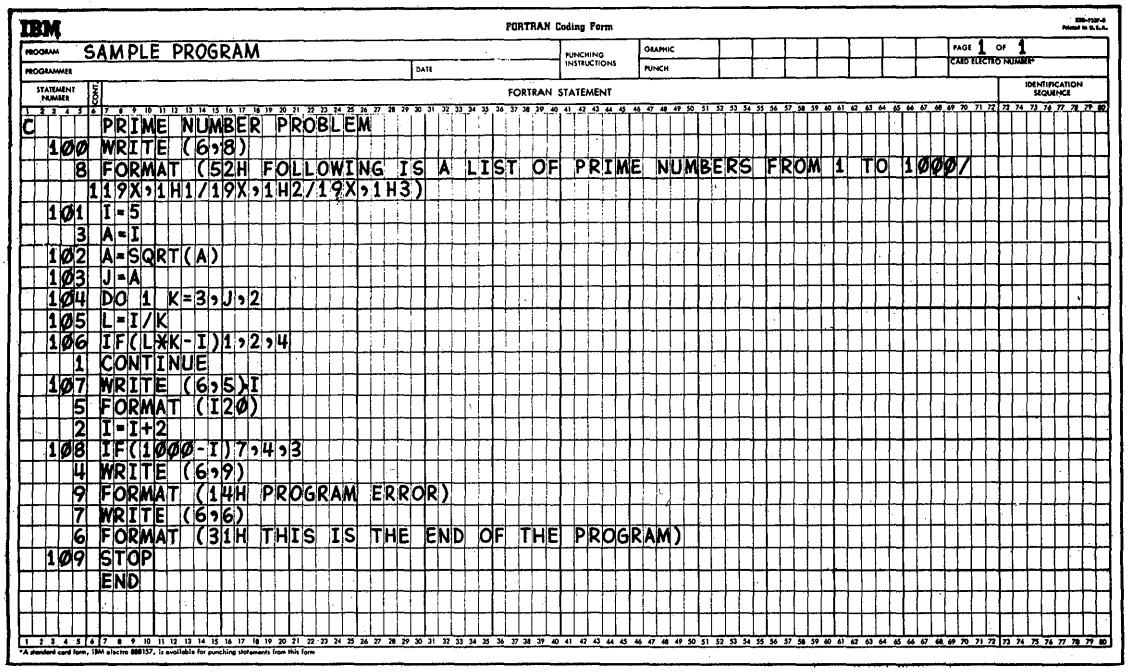
┌------┐
|G ONLY|
└------┘ A table is generated for each of seven classifications of variables used in the source module. Each table contains the names and locations of variables used in that particular context. The classifications are as follows:

- COMMON variables
- EQUIVALENCE variables
- Scalar variables
- Array variables
- Subprograms called
- NAMELIST variables
- FORMAT statements

Separate maps are produced for each classification, with the appropriate heading preceding the data. The variable names, statement labels or subprogram name

```
                      C     PRIME NUMBER PROBLEM
ISN 0002                100 WRITE (6,8)
ISN 0003                  8 FORMAT (52H FOLLOWING IS A LIST OF PRIME NUMBERS FROM 1 TO 1000/
                           119X,1H1/19X,1H2/19X,1H3)
ISN 0004                101 I=5
ISN 0005                  3 A=I
ISN 0006                102 A=SQRT(A)
ISN 0007                103 J=A
ISN 0008                104 DO 1 K=3,J,2
ISN 0009                105 L=I/K
ISN 0010                106 IF(L*K-I)1,2,4
ISN 0011                  1 CONTINUE
ISN 0012                107 WRITE (6,5)I
ISN 0013                  5 FORMAT (I20)
ISN 0014                  2 I=I+2
ISN 0015                108 IF(1000-I)7,4,3
ISN 0016                  4 WRITE (6,9)
ISN 0017                  9 FORMAT (14H PROGRAM ERROR)
ISN 0018                  7 WRITE (6,6)
ISN 0019                  6 FORMAT (31H THIS IS THE END OF THE PROGRAM)
ISN 0020                109 STOP
ISN 0021                    END
```

Figure 62. Source Module Listing



```
IBM                                  FORTRAN Coding Form
PROGRAM  SAMPLE PROGRAM                          PUNCHING INSTRUCTIONS   GRAPHIC
PROGRAMMER                       DATE            PUNCH                   PAGE 1 OF 1
STATEMENT NUMBER | CONT | FORTRAN STATEMENT                              IDENTIFICATION SEQUENCE
C       PRIME NUMBER PROBLEM
    100 WRITE (6,8)
        8 FORMAT (52H FOLLOWING IS A LIST OF PRIME NUMBERS FROM 1 TO 1000/
       119X,1H1/19X,1H2/19X,1H3)
    101 I=5
        3 A=I
    102 A=SQRT(A)
    103 J=A
    104 DO 1 K=3,J,2
    105 L=I/K
    106 IF(L*K-I)1,2,4
        1 CONTINUE
    107 WRITE (6,5)I
        5 FORMAT (I20)
        2 I=I+2
    108 IF(1000-I)7,4,3
        4 WRITE (6,9)
        9 FORMAT (14H PROGRAM ERROR)
        7 WRITE (6,6)
        6 FORMAT (31H THIS IS THE END OF THE PROGRAM)
    109 STOP
        END
```

Figure 63. Sample FORTRAN IV Program

SUBPROGRAMS CALLED

SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION
IBCOM#	BC	SQRT	CO						

SCALAR MAP

SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION
I	C8	A	CC	J	D0	K	D4	L	D8

FORMAT STATEMENT MAP

SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION
8	DC	5	126	9	12A	6	13C		

Figure 64. Storage Map -- (G) Compiler

are arranged across the page; five to a line. However, storage maps of variables not used in the source module are not produced.

Figure 64 is an example of a storage map produced by the (G) compiler for the sample program in Figure 63.

┌-------┐
|H ONLY|
└-------┘ Figure 65 is an example of a storage map produced by the (H) compiler for the program in Figure 63. The figure displays the following items:

1. The first line shows the name of the program (MAIN) and the program size in hexadecimal (00027C).

2. The second line shows headings which identify names used in the program. These headings and the information they describe are as follows;

   a. NAME lists names of all variables, statement functions, subprograms, and internal functions.

   b. TAG lists variable and name use codes. Variable use codes are as follows:

      A indicates that the variable was used as an argument; i.e., it appeared in a parameter list

      F indicates that the variable appeared on the right of the equal sign, was referenced in

an expression, or was specified in an output list, i.e., its value was used at some time.

S indicates that the variable appeared on the left of an equal sign, appeared in an input list, or was specified in a subroutine calling sequence; i.e., its value was stored at some time.

Name use codes are as follows:

C indicates variables in COMMON

E indicates variables that appear in an EQUIVALENCE statement

IF indicates an internal function.

NR indicates variables not referred to

SF indicates arithmetic statement functions

XF indicates subprograms

XR indicates variables, arrays, or subprograms that are referenced by name

Note that the name code SF should not be confused with the variable codes S and F. Codes S and F are positioned left of the heading TAG whereas the name identifier SF appears under the heading (actually, beginning under the letter G; see the placement of codes for the variable I and the subprogram SQRT in Figure 65).

c. Type identifies the type and length of each variable.

d. ADD identifies the relative address assigned to each name. (All functions and subroutines have a relative address of 00000.)

The FORTRAN (H) compiler also produces a map of each COMMON block, followed by a map of any equivalences made for the block. The map for each COMMON block contains the same type of information as for the main program. The map for the equivalences lists the name of each variable along with its displacement (offset) from the beginning of the common block.

H ONLY

Label Map

In addition to the storage map, specification of the MAP option for the (H) compiler produces a table of statement numbers known as a label map. The statement numbers that appear in the source module are written in the data set specified by the SYSPRINT DD statement. This table includes:

1. The statement number of each source label.
2. The relative address assigned to each label.
3. The symbol 'NR' next to each source label that is not referenced.

Figure 66 shows a portion of the label map produced by the (H) compiler for the program in Figure 63.

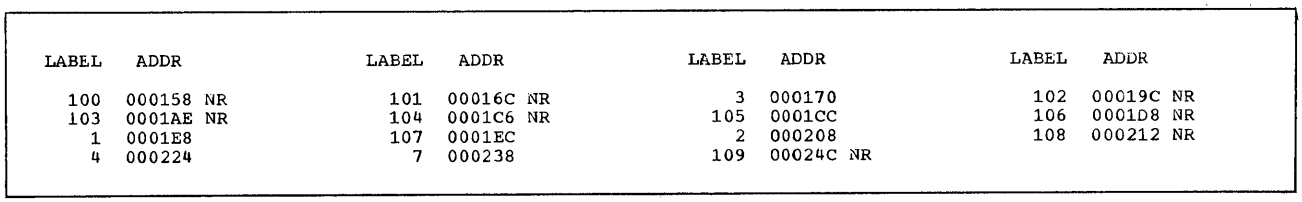
Object Module Listing

If the LIST option of the PARM parameter of the EXEC statement is specified, the object module listing is written in the data set specified by the SYSPRINT DD statement. The listing is in pseudo-assembler language format; i.e., all instructions are not valid assembler language instructions.

The listing produced differs according to the compiler used.

```
                                        /    MAIN /     SIZE OF PROGRAM 00027C HEXADECIMAL BYTES

NAME    TAG   TYPE  ADD.      NAME    TAG   TYPE  ADD.      NAME    TAG   TYPE  ADD.      NAME    TAG   TYPE  ADD.
   A SFA      R*4   000120       I SF       I*4   000124       J SF       I*4   000128       K SF       I*4   00012C
   L S        I*4   000130    SQRT     XF   R*4   000000  IBCOM=  F  XF   I*4   000000
```

Figure 65. Storage Map -- (H) Compiler



LABEL	ADDR	LABEL	ADDR	LABEL	ADDR	LABEL	ADDR
100	000158 NR	101	00016C NR	3	000170	102	00019C NR
103	0001AE NR	104	0001C6 NR	105	0001CC	106	0001D8 NR
1	0001E8	107	0001EC	2	000208	108	000212 NR
4	000224	7	000238	109	00024C NR		

Figure 66. Label Map -- (H) Compiler

G ONLY

The object module listing is arranged in column format, with headings, as follows:

Column 1: LOCATION--The address in hexadecimal of the instruction.

Column 2: STA NUM--The source program statement numbers of all FORTRAN executable statements.

Column 3: LABEL--Source labels and compiler generated labels.

Column 4 and Column 5: OP and OPERAND--The actual instruction.

Column 6: BCD OPERAND--Significant items referred to in the corresponding instruction, e.g., entry points, labels, variables, and constants.

Figure 67 shows an object module listing produced for the program in Figure 63.

H ONLY

The (H) compiler object module listing is arranged in column format as follows:

Column 1: The address (in hexadecimal) of the instruction.

Column 2: The assembly format (in hexadecimal) of the instruction.

Column 3: Source labels and compiler generated labels (compiler generated labels contain six digits).

Column 4: The actual instruction.

Column 5: Significant items referred to in the corresponding instruction, e.g., entry points, labels, variables, constants, and temporaries (.yxx where y is S, T, or Q and xx is two digits).

Figure 68 shows an object module listing produced for the program in Figure 63.

Object Module Card Deck

If the DECK option of the PARM parameter of the EXEC statement is specified, an object module card deck is produced. This deck is made up of four types of cards -- TXT, RLD, ESD, and END. A functional description of these cards is given in the following paragraphs.

```
LOCATION   STA NUM     LABEL    OP        OPERAND                BCD OPERAND
  000000                        BC        15,12(0,15)
  000004                        DC        06D4C1C9
  000008                        DC        D5404040
  00000C                        STM       14,12,12(13)
  000010                        LM        2,3,40(15)
  000014                        LR        4,13
  000016                        L         13,36(0,15)
  00001A                        ST        13,8(0,4)
  00001E                        STM       3,4,0(13)
  000022                        BCR       15,2
  000024                        DC        00000000               A4
  000028                        DC        00000000               A20
  00002C                        DC        00000000               A36
  0001A8              A36       L         13,4(0,13)
  0001AC                        L         14,12(0,13)
  0001B0                        LM        2,12,28(13)
  0001B4                        MVI       12(13),255
  0001B8                        BCR       15,14
  0001BA              A20       L         15,140(0,13)           IBCOM#
  0001BE                        LR        12,13
  0001C0                        LR        13,4
  0001C2                        BAL       14,64(0,15)
  0001C6                        LR        13,12
  0001C8        1     100       L         15,140(0,13)           IBCOM#
  0001CC                        BAL       14,4(0,15)
  0001D0                        DC        00000006
  0001D4                        DC        000000DC
  0001D8                        BAL       14,16(0,15)
  0001DC        3     101       L         0,360(0,13)
  0001E0                        ST        0,152(0,13)            I
  0001E4        4     3         L         0,152(0,13)            I
  0001E8                        LPR       1,0
  0001EA                        ST        1,324(0,13)
  0001EE                        LD        0,320(0,13)
  0001F2                        AD        0,304(0,13)
  0001F6                        LTR       0,0
  0001F8                        BALR      14,0
  0001FA                        BC        11,6(0,14)
  0001FE                        LCDR      0,0
  000200                        STE       0,156(0,13)            A
  000204        5     102       LA        1,148(0,13)
  000208                        L         15,144(0,13)           SQRT
  00020C                        BALR      14,15
  00020E                        STE       0,156(0,13)            A
  000212        6     103       SDR       0,0
  000214                        LE        0,156(0,13)            A
  000218                        AW        0,336(0,13)
  00021C                        STD       0,328(0,13)
  000220                        L         0,332(0,13)
  000224                        LTDR      0,0
  000226                        BALR      14,0
  000228                        BC        11,6(0,14)
  00022C                        LCR       0,0
  00022E                        ST        0,160(0,13)            J
  000232        7     104       L         0,364(0,13)
  000236              L44       ST        0,164(0,13)            K
  00023A        8     105       L         0,152(0,13)            I
  00023E                        SRDA      0,32(0)
  000242                        D         0,164(0,13)            K
  000246                        ST        1,168(0,13)            L
  00024A        9     106       L         1,168(0,13)            L
  00024E                        M         0,164(0,13)            K
  000252                        S         1,152(0,13)            I
  000256                        LTR       1,1
  000258                        L         14,104(0,13)           2
  00025C                        BCR       8,14
  00025E                        L         14,108(0,13)           4
  000262                        BCR       2,14
  000264       10     1         L         0,164(0,13)            K
  000268                        L         1,116(0,13)            L44
  00026C                        LA        2,2(0,0)
  000270                        L         3,160(0,13)            J
  000274                        BXLE      0,2,0(1)
```

Figure 67. Object Module Listing -- (G) Compiler (Part 1 of 2)

```
000278        11        107       L         15,140(0,13)              IBCOM#
00027C                            BAL       14,4(0,15)
000280                            DC        00000006
000284                            DC        00000126
000288                            BAL       14,8(0,15)
00028C                            DC        0450D098
000290                            BAL       14,16(0,15)
000294        13        2         L         0,152(0,13)               I
000298                            A         0,368(0,13)
00029C                            ST        0,152(0,13)               I
0002A0        14        108       L         0,372(0,13)
0002A4                            S         0,152(0,13)               I
0002A8                            LTR       0,0
0002AA                            L         14,112(0,13)              7
0002AE                            BCR       4,14
0002B0                            L         14,96(0,13)               3
0002B4                            BCR       2,14
0002B6        15        4         L         15,140(0,13)              IBCOM#
0002BA                            BCR       0,0
0002BC                            BAL       14,4(0,15)
0002C0                            DC        00000006
0002C4                            DC        0000012A
0002C8                            BAL       14,16(0,15)
0002CC        17        7         L         15,140(0,13)              IBCOM#
0002D0                            BAL       14,4(0,15)
0002D4                            DC        00000006
0002D8                            DC        0000013C
0002DC                            BAL       14,16(0,15)
0002E0        19        109       L         15,140(0,13)              IBCOM#
0002E4                            BAL       14,52(0,15)
0002E8                            DC        05404040
0002EC                            DC        40F0
                                  END
 *OPTIONS IN EFFECT*   NOID,EBCDIC,SOURCE,LIST,NODECK,LOAD,NOMAP
 *OPTIONS IN EFFECT*   NAME = MAIN    , LINECNT =       50
 *STATISTICS*     SOURCE STATEMENTS =       20,PROGRAM SIZE =      750
 *STATISTICS*  NO DIAGNOSTICS GENERATED
```

Figure 67. Object Module Listing -- (G) Compiler (Part 2 of 2)

```
                000000  47 F0 F 00C        MAIN  BC    15,12(0,15)

                000004  07404040                 DC    XL4'40404040'
                000008  40404040                 DC    XL4'40404040'
                00000C  90 EC D 00C              STM   14,12,12(13)
                000010  98 23 F 020              LM    2,3,32(15)
                000014  50 30 D 008              ST    3,8(13)
                000018  50 D0 3 004              ST    13,4(0,3)
                00001C  07 F2                    BCR   15,2

TEMPORARY FOR FLOAT/FIX
CONSTANTS
                000108  4E000000                 DC    XL4'4E000000'
                00010C  00000000                 DC    XL4'00000000'
                000110  00000002                 DC    XL4'00000002'
                000114  00000003                 DC    XL4'00000003'
                000118  00000005                 DC    XL4'00000005'
                00011C  000003E8                 DC    XL4'000003E8'
ADCONS FOR VARIABLES AND CONSTANTS
ADCONS FOR EXTERNAL REFERENCES
                000138  00000000                 DC    XL4'00000000'                    SQRT
                00013C  00000000                 DC    XL4'00000000'                    IBCOM=
                000148  41 40 0 002      100000  LA     4,   2( 0, 0)                      2
                00014C  41 90 0 003              LA     9,   3( 0, 0)                      3
                000150  41 A0 0 005              LA     10,  5( 0, 0)                      5
                000154  41 80 0 3E8              LA     8,1000( 0, 0)                      3E8
                000158  58 F0 D 08C          100 L      15, 140( 0,13)                   IBCOM=
                00015C  45 E0 F 064              BAL    14,  4( 0,15)
                000160  00000006                DC    XL4'00000006'                      F
                000164  00000028                DC    XL4'00000028'                    G+'
                000168  45 E0 F 010              BAL    14,  16( 0,15)
                00016C  50 A0 D 074          101 ST     10, 116( 0,13)                   I
                000170  58 00 D 074            3 L      0, 116( 0,13)                    I
                000174  68 00 D 058              LD     0,  88( 0,13)
                000178  60 00 D 050              STD    0,  80( 0,13)
                00017C  12 00                    LTR    0, 0
                00017E  47 40 D 0DE              BC     4, 222( 0,13)
                000182  50 00 D 054              ST     0,  84( 0,13)
                000186  6A 00 D 050              AD     0,  80( 0,13)
                00018A  47 F0 D 0E8              BC     15, 232( 0,13)
                00018E  10 00                    LPR    0, 0
                000190  50 00 D 054              ST     0,  84( 0,13)
                000194  6B 00 D 050              SD     0,  80( 0,13)
                000198  70 00 D 070              STE    0, 112( 0,13)                      A
                00019C  41 10 D 04C          102 LA     1,  76( 0,13)
                0001A0  58 F0 D 088              L      15, 136( 0,13)                   SQRT
                0001A4  05 EF                    BALR   14,15
                0001A6  70 00 D 090              STE    0, 144( 0,13)                    .T00
                0001AA  78 00 D 090              LE     0, 144( 0,13)                    .T00
                0001AE  70 00 D 070              STE    0, 112( 0,13)                      A
                0001B2  28 00                1O3 LDR    0, 0
                0001B4  78 00 D 070              LE     0, 112( 0,13)                      A
                0001B8  6E 00 D 058              AW     0,  88( 0,13)
                0001BC  60 00 D 050              STD    0,  80( 0,13)
                0001C0  58 50 D 054              L      5,  84( 0,13)
                0001C4  47 A0 D 11A              BC     10, 282( 0,13)
                0001C8  11 55                    LNR    5, 5
                0001CA  18 69                104 LR     6, 9
                0001CC  58 70 D 074              L      7, 116( 0,13)                    I
                0001D0  18 07                105 LR     0, 7
                0001D2  8E 00 0 020              SRDA   0,  32( 0, 0)
                0001D6  1D 06                    DR     0, 6
                0001D8  50 10 D 080              ST     1, 128( 0,13)                    L
                0001DC  18 36                106 LR     3, 6
                0001DE  5C 20 D 080              M      2, 128( 0,13)                    L
                0001E2  18 37                    SR     3, 7
                0001E4  47 80 D 15C              BC     8, 348( 0,13)                      2
                0001E8  47 20 D 178              BC     2, 376( 0,13)                      4
                0001EC  87 64 D 120            1 BXLE   6, 288( 4,13)                    105
                0001F0  58 F0 D 08C          107 L      15, 140( 0,13)                   IBCOM=
                0001F4  45 E0 F 004              BAL    14,  4( 0,15)
                0001F8  00000006                 DC    XL4'00000006'                      F
                0001FC  00000072                 DC    XL4'00000072'                    G -
                000200  45 E0 F 008              BAL    14,  8( 0,15)
                000204  0450D074                 DC    XL4'0450D074'                      I
                000208  45 E0 F 010              BAL    14,  16( 0,15)
                00020C  58 00 D 074            2 L      0, 116( 0,13)                    I
                000210  1A 04                    AR     0, 4
                000212  50 00 D 074              ST     0, 116( 0,13)                    I
                000216  18 28                108 LR     2, 8
                000218  5B 20 D 074              S      2, 116( 0,13)                    I
                00021C  47 40 D 18C              BC     4, 396( 0,13)                      7
                000220  47 80 D 178              BC     8, 376( 0,13)                      4
                000224  47 20 D 0C0              BC     2, 192( 0,13)                      3
                000228  58 F0 D 08C            4 L      15, 140( 0,13)                   IBCOM=
                00022C  45 E0 F 004              BAL    14,  4( 0,15)
                000230  00000006                 DC    XL4'00000006'                      F
                000234  00000076                 DC    XL4'00000076'                    G+8
                000238  45 E0 F 010              BAL    14,  16( 0,15)
                00023C  58 F0 D 08C            7 L      15, 140( 0,13)                   IBCOM=
                000240  45 E0 F 004              BAL    14,  4( 0,15)
                000244  00000006                 DC    XL4'00000006'
```

Figure 68. Object Module Listing -- (H) Compiler (Part 1 of 2)

```
                    000248  00000088                    DC    XL4'00000088'                    G+Q
                    00024C  45 E0 F 010                 BAL   14,  16( 0,15)
                    000250  58 F0 D 08C          109    L     15, 140( 0,13)                   IBCOM=
                    000254  45 E0 F 034                 BAL   14,  52( 0,15)
                    000258  05                          DC    XL1'00000005'
                    000259  40                          DC    XL1'00000040'
                    00025A  40                          DC    XL1'00000040'
                    00025B  40                          DC    XL1'00000040'
                    00025C  40                          DC    XL1'00000040'
                    00025D  F0                          DC    XL1'000000F0'
ADDRESS OF EPILOGUE
                    00025E  58 F0 D 08C                 L     15, 140( 0,13)
                    000262  45 E0 F 034                 BAL   14,  52( 0,15)                   IBCOM=
                    000266  0540                        DC    XL2'40400540'
                    000268  404040F0                    DC    XL4'404040F0'
ADDRESS OF PROLOGUE
                    00026E  58 F0 3 08C                 L     15, 140( 0,  3)
                    000272  45 E0 F 040                 BAL   14,  64( 0,15)                   IBCOM=
                    000276  18 D3                       LR    13,  3
                    000278  47 F0 D 098                 BC    15, 152( 0,13)
ADCON FOR PROLOGUE
                    000020  0000026E                    DC    XL4'0000026E'
ADCON FOR SAVE AREA
                    000024  00000080                    DC    XL4'00000080'
ADCON FOR EPILOGUE
                    0000B0  0000025E                    DC    XL4'0000025E'
ADCONS FOR PARAMETER LISTS
                    0000FC  80000120                    DC    XL4'80000120'                    A
ADCONS FOR TEMPORARIES
                    000140  00000000                    DC    XL4'00000000'
                    000144  00000000                    DC    XL4'00000000'
ADCONS FOR B BLOCK LABELS
```

Figure 68. Object Module Listing -- (H) Compiler (Part 2 of 2)

OBJECT MODULE CARDS: Every card in the object module deck contains a 12-2-9 punch in column 1 and an identifier in columns 2 through 4. The identifier consists of the characters ESD, RLD, TXT or END. The first four characters of the name of the program are placed in columns 73 through 76 with the sequence number of the card in columns 77 through 80.

ESD Card: Four types of ESD cards are generated as follows:

- ESD, type 0 - contains the name of the program and indicates the beginning of the object module.
- ESD, type 1 - contains the entry point name corresponding to an ENTRY statement in a subprogram.
- ESD, type 2 - contains the names of subprograms referred to in the source module by CALL statements, EXTERNAL statements, explicit function references, and implicit function references.
- ESD, type 5 - contains information about each COMMON block.

The number 0, 1, 2, or 5 is placed in card column 25.

RLD Card: An RLD card is generated for external references indicated in the ESD, type 2 cards. To complete external references, the linkage editor matches the addresses in the RLD card with external symbols in the ESD card. When external references are resolved, the storage at the address indicated in the RLD card contains the address assigned to the subprogram indicated in the ESD, type 2 card. RLD cards are also generated for a branch list produced for statement numbers.

TXT Card: The TXT card contains the constants and variables used by the programmer in his source module, any constants and variables generated by the compiler, coded information for FORMAT statements, and the machine instructions generated by the compiler from the source module.

END Card: One END card is generated for each compiled source module. This card indicates the end of the object module to the linkage editor, the relative location of the main entry point, and the length (in bytes) of the object module.

OBJECT MODULE DECK STRUCTURE: Because of implementation, object module deck structures differ by compiler. Figure 69 shows the deck structure for the FORTRAN IV (G) compiler; Figure 70 shows the deck structure for the (H) compiler.

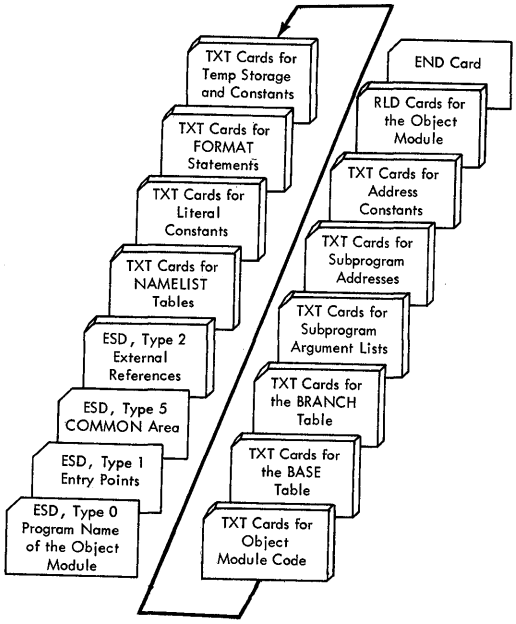


Figure 69. Object Module Deck Structure -- (G) Compiler

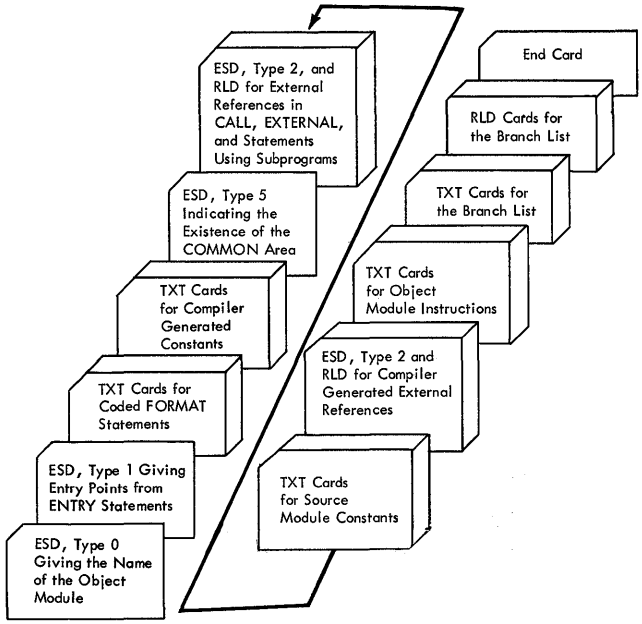


Figure 70. Object Module Deck Structure -- (H) Compiler

H ONLY

Cross Reference Listing

If the compiler XREF option is specified, a cross reference listing of variables and labels is written in the data set specified by the SYSPRINT DD statement. The variable names are listed in alphabetical order, according to length. (Variable names of one character appear first in the listing.) The labels are listed in ascending sequence along with the internal statement number of the statement in which the label is defined.

For both variable names and labels, the listing also contains the internal statement number of each statement in which the variable or label is used. Figure 71 shows a compiler cross reference listing produced for the program in Figure 63.

H ONLY

Structured Source Listing

If the EDIT option is specified, a structured source listing is written in the data set specified by the SYSPRINT DD statement. This listing is independent of the usual source listing and indicates the loop structure and logical continuity of the source program.

Each loop is assigned a unique 3-digit number. Entrance to the loop is indicated by a left parenthesis followed by a 3-digit loop number -- (xxx -- before the internal statement number of the first statement in the loop; exit from the loop is indicated by the 3-digit loop number followed by a right parenthesis -- xxx) -- on a separate line before the next non-comment line.

```
SYMBOL   INTERNAL STATEMENT NUMBERS
A        0005  0006  0006  0007
I        0004  0005  0009  0010  0012  0014  0014  0015
J        0007  0008
K        0008  0009  0010
L        0009  0010
SQRT     0006

LABEL    DEFINED    REFERENCES
   1     0011       0008  0010
   2     0014       0010
   3     0005       0015
   4     0016       0010  0015
   5     0013       0012
   6     0019       0018
   7     0018       0015
   8     0003       0002
   9     0017       0016
 100     0002
 101     0004
 102     0006
 103     0007
 104     0008
 105     0009
 106     0010
 107     0012
 108     0015
 109     0020
```

Figure 71. Compiler Cross Reference Listing -- (H) Compiler

Indentations are used to show dominance relationships among executable source statements. Statement A dominates statement B if A is the last statement common to all logical paths from which B receives control. Statement A is called a dominator, statement B is called a dominee. By this definition, a statement can have only one dominator, but a dominator may have several dominees. For example, a computed GO TO statement is the last statement through which control passes before reaching three other statements. The GO TO statement is a dominator with three dominees.

A dominee is indented from its dominator unless it is either the only dominee or the last dominee of that dominator. The line of sight between a dominator and its dominee(s) may be obscured by intervening statements. This is a dominance discontinuity and is indicated by C--- on a separate line above the dominee.

Comments and non-executable statements are not involved in dominance relationships; their presence never causes a dominance discontinuity. Comments line up with the last preceding non-comment line; nonexecutable statements line up either with the last preceding executable statement or with the first one following.

Figure 72 shows a structured source listing produced for the program in Figure 63.

Source Module Diagnostics

FORTRAN IV (G) and (H) compiler messages are described in Appendix D.

LINKAGE EDITOR OUTPUT

The linkage editor produces a map of a load module if the MAP option of the PARM parameter of the EXEC statement is specified, or a cross reference list and a map if the XREF option is specified. The linkage editor also produces diagnostic messages, which are discussed in the Linkage Editor and Loader publication.

Module Map

The module map is written in the data set specified in the SYSPRINT DD statement for the linkage editor. To the linkage editor, each program (main or subprogram) and each COMMON (blank or named) block is a control section.

Each control section name is written along with origin and length of the control section. For a program and named COMMON, the name is listed; for blank COMMON, the name $BLANKCOM is listed. The origin and length of a control section is written in hexadecimal numbers. A segment number is also listed for overlay structures (see the Linkage Editor and Loader publication).

For each control section, any entry points and their locations are also written; any functions called from the data set specified by the SYSLIB DD statement are listed and marked by asterisks.

The total length and entry point of the load module are listed. Figure 73 shows, for the (G) compiler, a load module map produced for the program in Figure 63; the map produced for the (H) compiler is shown in Figure 74.

```
                              C     PRIME NUMBER PROBLEM
          ISN 0002       100 WRITE (6,8)
          ISN 0003         8 FORMAT (52H FOLLOWING IS A LIST OF PRIME NUMBERS FROM 1 TO 1000/
                           119X,1H1/19X,1H2/19X,1H3)
          ISN 0004       101 I=5
       (002ISN 0005        3 A=I
          ISN 0006       102 A=SQRT(A)
          ISN 0007       103 J=A
          ISN 0008       104 DO 1 K=3,J,2
       (001ISN 0009       105 L=I/K
          ISN 0010       106 IF(L*K-I)1,2,4
          ISN 0011         1    CONTINUE
           001)                  C
          ISN 0012       107    WRITE (6,5)I
          ISN 0013         5    FORMAT (I20)
          ISN 0014         2    I=I+2
          ISN 0015       108    IF(1000-I)7,4,3
           002)                  C
          ISN 0016         4    WRITE (6,9)
          ISN 0017         9    FORMAT (14H PROGRAM ERROR)
          ISN 0018         7 WRITE (6,6)
          ISN 0019         6 FORMAT (31H THIS IS THE END OF THE PROGRAM)
          ISN 0020       109 STOP
          ISN 0021           END
```

Figure 72. Structured Source Listing -- (H) Compiler

```
CONTROL SECTION                                  ENTRY
  NAME     ORIGIN  LENGTH                        NAME     LOCATION     NAME     LOCATION     NAME     LOCATION     NAME     LOCATION
MAIN          00     2E6                        MAIN          00
IHCFCOMH*    2E8     FB3                        IBCOM=       2E8     FDIOCS=      3A4
IHCSSQRT*   12A0      AC                        SQRT        12A0
IHCFCVTH*   1350     FEB                        ADCON=      1350     FCVZO       149C     FCVAO       1542     FCVLO       15CA
                                                FCVIO       18D8     FCVEO       1D72     FCVCO       1F6C
IHCFIOSH*   2340     C30                        FIOCS=      2340
IHCUATBL*   2F70     108
```

Figure 73. Load Module Map -- (G) Compiler

```
CONTROL SECTION                                  ENTRY
  NAME     ORIGIN  LENGTH                        NAME     LOCATION     NAME     LOCATION     NAME     LOCATION     NAME     LOCATION
MAIN          00     27C
IHCSSQRT*    280      AC                        SQRT        280
IHCFCOMH*    330     FFD                        IBCOM=      330     FDIOCS=      3EC
IHCUOPT *   1330       8
IHCTRCH *   1338     258
IHCFCVTH*   1590     FF3                        ADCON=      1590     FCVZO       16DC     FCVAO       1782     FCVLO       180A
                                                FCVIO       1818     FCVEO       1FBC     FCVCO       21B6
IHCFIOSH*   2588     CF2                        FIOCS=      2588
IHCUATBL*   3280     638
```

Figure 74. Load Module Map -- (H) Compiler

Cross-Reference List

If the linkage editor XREF option of the PARM parameter of the EXEC statement is specified, a cross reference list is written with the module map. This cross reference list gives the location from which an external reference is made, the symbol externally referred to in this control section, the control section in which the symbol appears, and the segment number of the control section in which the symbol appears. Unless the linkage editor is building an overlay structure, the cross reference list appears after the module map for all control sections. Figure 75 shows, for the (G) compiler, the cross reference list produced for the program in Figure 63; the list produced for the (H) compiler is shown in Figure 76.

LOAD MODULE OUTPUT

At execution time, FORTRAN load module diagnostics are generated in three forms - error code diagnostics, program interrupt messages, and operator messages. An error code indicates an input/output error or a misuse of a FORTRAN library function. A program interrupt message indicates a condition that is beyond the capacity of System/360 to correct. An operator message is generated when a STOP or PAUSE is executed.

LOCATION	REFERS TO SYMBOL	CONTROL SECTION
D0	IBCOM=	IHCFCOMH
D4	SQRT	IHCSSQRT
1134	ADCON=	IHCFCVTH
112C	FIOCS=	IHCFIOSH
1138	FCVEO	IHCFCVTH
113C	FCVLO	IHCFCVTH
1140	FCVIO	IHCFCVTH
1144	FCVCO	IHCFCVTH
1148	FCVAO	IHCFCVTH
114C	FCVZO	IHCFCVTH
1324	IBCOM=	IHCFCOMH
21FC	IBCOM=	IHCFCOMH
2464	IHCUATBL	IHCUATBL
2470	IBCOM=	IHCFCOMH
ENTRY ADDRESS	00	
TOTAL LENGTH	3078	

Figure 75. Linkage Editor Cross Reference List -- (G) Compiler

LOCATION	REFERS TO SYMBOL	IN CONTROL SECTION
138	SQRT	IHCSSQRT
13C	IBCOM=	IHCFCOMH
304	IBCOM=	IHCFCOMH
11D8	ADCON=	IHCFCVTH
11D0	FIOCS=	IHCFIOSH
10E0	IHCUOPT	IHCUOPT
11DC	FCVEO	IHCFCVTH
11E0	FCVLO	IHCFCVTH
11E4	FCVIO	IHCFCVTH
11E8	FCVCO	IHCFCVTH
11EC	FCVAO	IHCFCVTH
11F0	FCVZO	IHCFCVTH
11C8	IHCTRCH	IHCTRCH
1490	IBCOM=	IHCFCOMH
1494	ADCON=	IHCFCVTH
1498	FIOCS=	IHCFIOSH
2444	IBCOM=	IHCFCOMH
2680	IHCUATBL	IHCUATBL
268C	IBCOM=	IHCFCOMH
ENTRY ADDRESS	00	
TOTAL LENGTH	3888	

Figure 76. Linkage Editor Cross Reference List -- (H) Compiler

Error Code Diagnostics and Traceback without Extended Error Handling Message Facility

If an error is detected during execution of a FORTRAN load module, a message and a diagnostic traceback are written in the error message data set (see "FORTRAN Job Processing"). The message is of the form:

```
     message text
TRACEBACK FOLLOWS-ROUTINE ISN REG.  14,
REG.  15, REG.  0, REG.  1
```

These error messages are described in Appendix D. For the error conditions numbered 211 through 214, 217, 219, 220, and 231 through 237, the message will consist of only IHCxxxI where xxx is a 3-digit error code. The errors detected by the FORTRAN mathematical functions will provide message text describing the error condition. The traceback, which follows the error message, is a list of routines in the direct line of call to the routine in error, in reverse order of use. After the traceback is completed, for error message IHC218I, control is passed to the statement designated in the ERR parameter of the FORTRAN READ statement if that parameter was specified. In all other cases, execution of the job step is terminated and a condition code of 16 is returned to the operating system.

Each entry in the traceback contains the name of the called routine, an internal statement number (ISN) from the calling routine (if one was generated for that call), and the contents, in hexadecimal, of register 14 (which indicate the point of return to the calling routine).

The first routine listed in the traceback is the one that called the library subprogram in which the error occurred, except when the name given is IBCOM. Then, the error could have occurred in IHCFCOMH or one of the routines that it calls: IHCFCVTH, or IHCFIOSH. The error code in the message indicates the actual origin of the error.

Note: For an assembler language program or subprogram, the routine name field in the traceback contains the identifier specified in the SAVE macro instruction or equivalent coding. (If the identifier specified is longer than eight characters, only the first eight appear.) If no identifier is specified, the traceback routine name field is either blank or its contents are meaningless in the traceback.

Internal statement number identifiers are generated for function references and calls when the ID option is specified on the EXEC statement for the compile step. These identifiers appear in the traceback, except for FORTRAN calls to IBCOM for which no identifiers are generated. If NOID is specified, no identifiers are generated and the internal statement number field will be blank.

```
IHC219I
TRACEBACK FOLLOWS    ROUTINE     ISN     REG. 14      REG. 15     REG. 0      REG. 1
                     IBCOM               820068FC     XXXXXXXX    XXXXXXXX    XXXXXXXX
                     MASTR       010     00005378
                     PAYROLL             00003148     XXXXXXXX    XXXXXXXX    XXXXXXXX
ENTRY POINT = 00005000
```

Figure 77. Sample Traceback for Execution-Time Errors

Note: For an assembler language program or subprogram, the internal statement number field contains the value of the binary calling sequence identifier specified in the CALL macro instruction or equivalent coding. If no identifier was specified, the field is either blank or its contents are meaningless in the traceback.

If the traceback cannot be completed, the message TRACEBACK TERMINATED is issued and the job step is terminated. This message appears only if either 13 names of subprograms appear in the traceback or a calling loop has been detected (e.g., subprogram A calling B calling A).

At the end of the traceback, whether it was completed or not, the entry point of the main FORTRAN program is given in hexadecimal.

Figure 77 shows the traceback information placed in the error message data set for the following example.

Example: A FORTRAN program PAYROLL calls the subroutine MASTR, which contains a READ statement. The IHCFIOSH routine is called to perform the input operation, but an error condition arises because there is no DD statement for the data set.

Explanation: PAYROLL was entered at location 5000 and called MASTR at internal statement number (ISN) 10 in PAYROLL. IBCOM (in this case, the error occurred in the IHCFIOSH routine) would have returned to location 68FC in MASTR; MASTR would have returned to location 5378 in PAYROLL and PAYROLL would have returned to location 3148 in the supervisor. Execution terminates and a condition code of 16 is returned to the operating system.

Program Interrupt Messages

Program interrupt messages containing the old Program Status Word (PSW) are produced when one of the following occurs:

- Protection Exception (4)
- Addressing Exception (5)
- Specification Exception (6)
- Data Exception (7)
- Fixed-Point Divide Exception (9)
- Exponent-Overflow Exception (C)
- Exponent-Underflow Exception (D)
- Floating-Point Divide Exception (F)

The characters in parentheses following the exceptions are PSW codes that appear in the program interrupt message to indicate the type of exception. Appendix D contains a complete description of the message and its format.

The program interrupt messages are written on a data set specified by the programmer. (See "FORTRAN Job Processing.") Operator intervention is not required for any of these interruptions.

ABEND Dump

If a program interrupt occurs that causes abnormal termination of a load module, an indicative dump is given (i.e., only the contents of significant registers, indicators, etc., are dumped). However, if a programmer adds the statement

```
//GO.SYSABEND DD SYSOUT=A
```

to the execute step of a cataloged procedure, main storage and significant registers, indicators, etc., are dumped. (For information about interpreting an ABEND dump, see the Guide to Debugging publication.)

Operator Messages

A message is transmitted to the operator when a STOP or PAUSE is encountered during load module execution. Operator messages are written on the device specified for operator communication. For a description of these messages, see Appendix D.

The value associated with the STOP statement (0 if a value is not coded) is passed to the next job step and can be tested as a condition code by the COND

parameter in the EXEC statement. This condition code will have a value of 16 if FORTRAN terminates the step because of a detected execution error. The STOP statement, like CALL EXIT, terminates a FORTRAN program and causes automatic closing and positioning of FORTRAN data sets, i.e., the writing out of the last buffer on output, the releasing of dynamic storage, the closing of data sets, etc.

LOADER OUTPUT

The loader produces error and diagnostic messages and a storage map of the loaded program if the PRINT and MAP options, respectively, are specified in in the PARM field of the EXEC statement. The storage map and diagnostic messages are produced on the data set specified in the SYSLOUT DD statement. The diagnostic messages are fully described in the Messages and Codes publication.

The storage map includes the name and absolute address of each control section and entry point defined in the program. It is written on SYSLOUT concurrently with input (SYSLIN) processing, as its entries appear in the same order as the input ESD items. The total size and storage extent of the loaded program are also included. Figure 78 shows the storage map produced for the program in Figure 63.

```
                                        OS/360 LOADER

OPTIONS USED - PRINT,MAP,LET,CALL,RES,SIZE=102400

    NAME  TYPE  ADDR       NAME  TYPE  ADDR       NAME  TYPE  ADDR       NAME  TYPE  ADDR       NAME  TYPE  ADDR

  MAIN      SD   63360    IHCSSQRT* SD   63650    SQRT    * LR   63650    IHCECOMH* SD   63798    IBCOM#  * LR   63798
  FDIOCS# * LR   63854    INTSWTCH* LR   646B6    IHCCOMH2* SD   646D0    SEQDASD * LR   64958    IHCERRM * SD   64C40
  ERRMON  * LR   64C40    IHCERRE * LR   64C58    IHCUOPT * SD   65200    IHCEFNTH* SD   65500    ARITH#  * LR   65500
  ADJSWTCH* LR   6586C    IHCEFIOS* SD   65A18    FIOCS#  * LR   65A18    FIOCSBEP* LR   65A1E    IHCFCVTH* SD   66C98
  ADCON#  * LR   66C98    FCVAOUTP* LR   66D42    FCVLOUTP* LR   66DD2    FCVZOUTP* LR   66F22    FCVIOUTP* LR   672C8
  FCVEOUTP* LR   677CA    FCVCOUTP* LR   679E4    INT6SWCH* LR   67CCB    IHCUATBL* SD   67E30    IHCETRCH* SD   68038
  IHCTRCH * LR   68038    ERRTRA  * LR   68040

  TOTAL LENGTH     4F68
  ENTRY ADDRESS   63360
```

Figure 78. Storage Map Produced by the Loader

Overlay is a feature of linkage editor processing that allows the FORTRAN programmer to reduce the main storage requirements of his program by breaking it up into two or more segments that need not be in main storage at the same time. These segments can be assigned the same relative storage addresses and can be loaded at different times during execution of the program. The programmer uses linkage editor control statements to specify the relationship of segments within the overlay structure.

DESIGNING A PROGRAM FOR OVERLAY

Programs are placed in an overlay structure according to the size, frequency of use, and logical relationships between the program units that they comprise. The basic principle of overlay is illustrated by the simple example in Figure 79. It shows a FORTRAN program consisting of a main program and two very large subprograms named SUBA and SUB. Normally, all three program units would be loaded into main storage at the same time and would remain there throughout execution of the entire program. However, if there was not enough main storage space available to accommodate all three program units at once, and if SUBA and SUBB did not have to be in main storage at the same time, the programmer could design an overlay structure in which the MAIN routine stayed in main storage at all times, while subprograms SUBA and SUBB made use of the remaining space as they were needed.

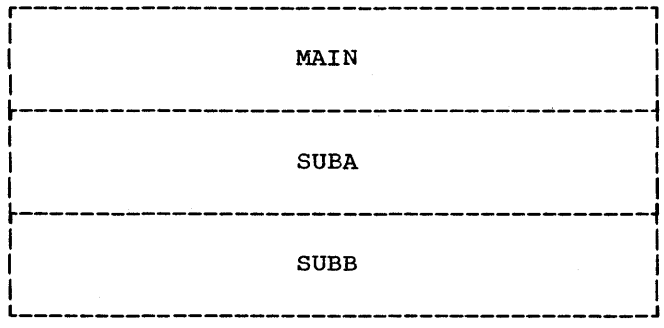


Figure 79. A FORTRAN Program Consisting of Three Program Units

Figure 80 shows what happens at execution time to the program in Figure 79. The MAIN routine is loaded and processing begins. When the MAIN routine calls SUBA, SUBA is loaded and processing continues until SUBB is called. SUBB then overlays SUBA in main storage and remains there until SUBA is called again. The main storage requirements of the program are thus reduced from the total number of bytes in all three program units to the total number of bytes in the MAIN routine plus the larger of the two subprograms.

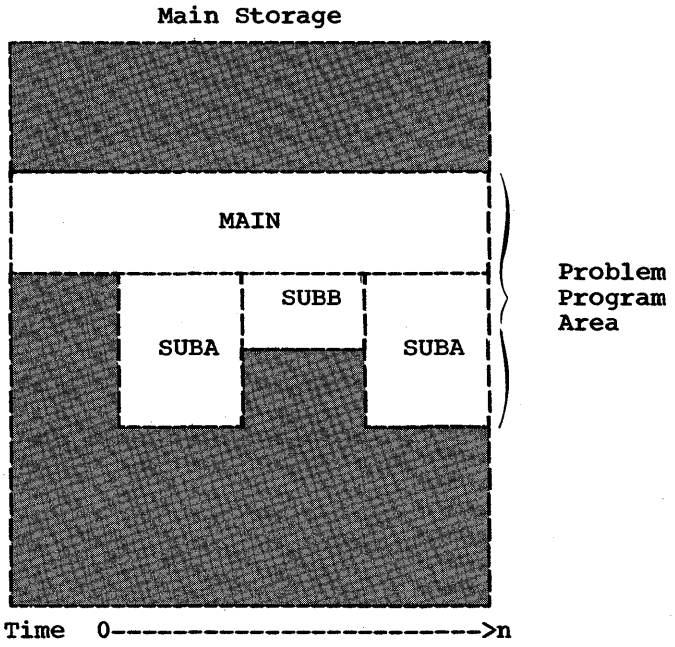


Figure 80. Time/Storage Map of a Three Segment Overlay Structure

SEGMENTS

The relationships among the program units in the overlay program described above can be graphically represented by an overlay "tree" structure, as shown in Figure 81. Each "branch" of the overlay tree consists of a separately loadable unit of the program to which the linkage editor assigns a number. These overlay segments may contain one or more subprograms totaling 512K bytes (524,288 bytes).

The first segment in any overlay program is called the root segment. The root segment remains in main storage at all times during execution of the program. It must contain:

- The program unit which receives control at the start of processing. Usually this is the main routine in which processing begins at the entry point named MAIN.
- Any program units which should remain in main storage throughout processing. For greater efficiency, subprograms which are frequently called should also be placed in the root segment if possible.
- Any program units containing DEFINE FILE statements.
- Certain information which is needed by the operating system to control the overlay operation. This information is automatically included in the root segment by the linkage editor.
- Any automatically called FORTRAN library subprograms that have not been explicitly positioned in the overlay structure.

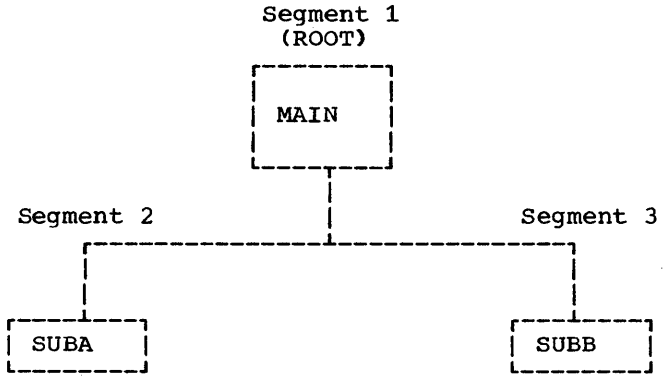


Figure 81. Overlay Tree Structure of Three Program Units

PATHS

The relationships among the segments of an overlay program are expressed in terms of "paths". A path consists of a given segment and any segments between it and the root segment. The root segment is thus a part of every path, and when a given segment is in main storage, all segments in its path are also in main storage. The simple program in Figure 81 is made up of only two paths, as shown in Figure 82.

The paths of an overlay program are determined by the dependencies between the program units which it comprises. A program unit is considered to be dependent on any program unit which it calls or whose data it must process.

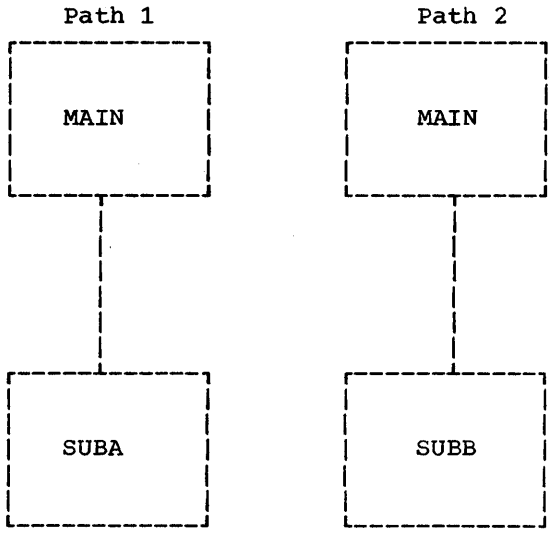


Figure 82. The Paths in the Overlay Tree in Figure 81

Figure 83 shows a FORTRAN program in an overlay tree structure. The paths implied by that structure are illustrated in Figure 84. The MAIN routine and subprograms SUB1

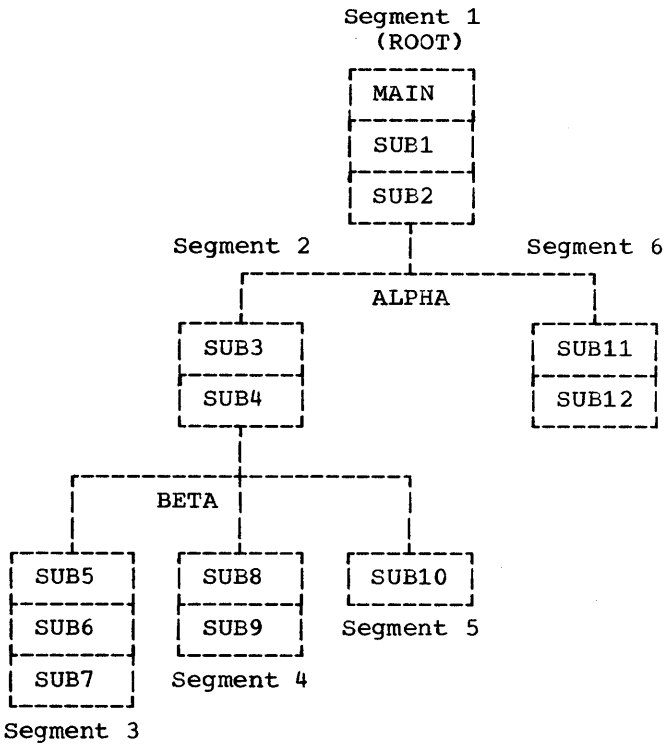


Figure 83. Overlay Tree Structure Having Six Segments

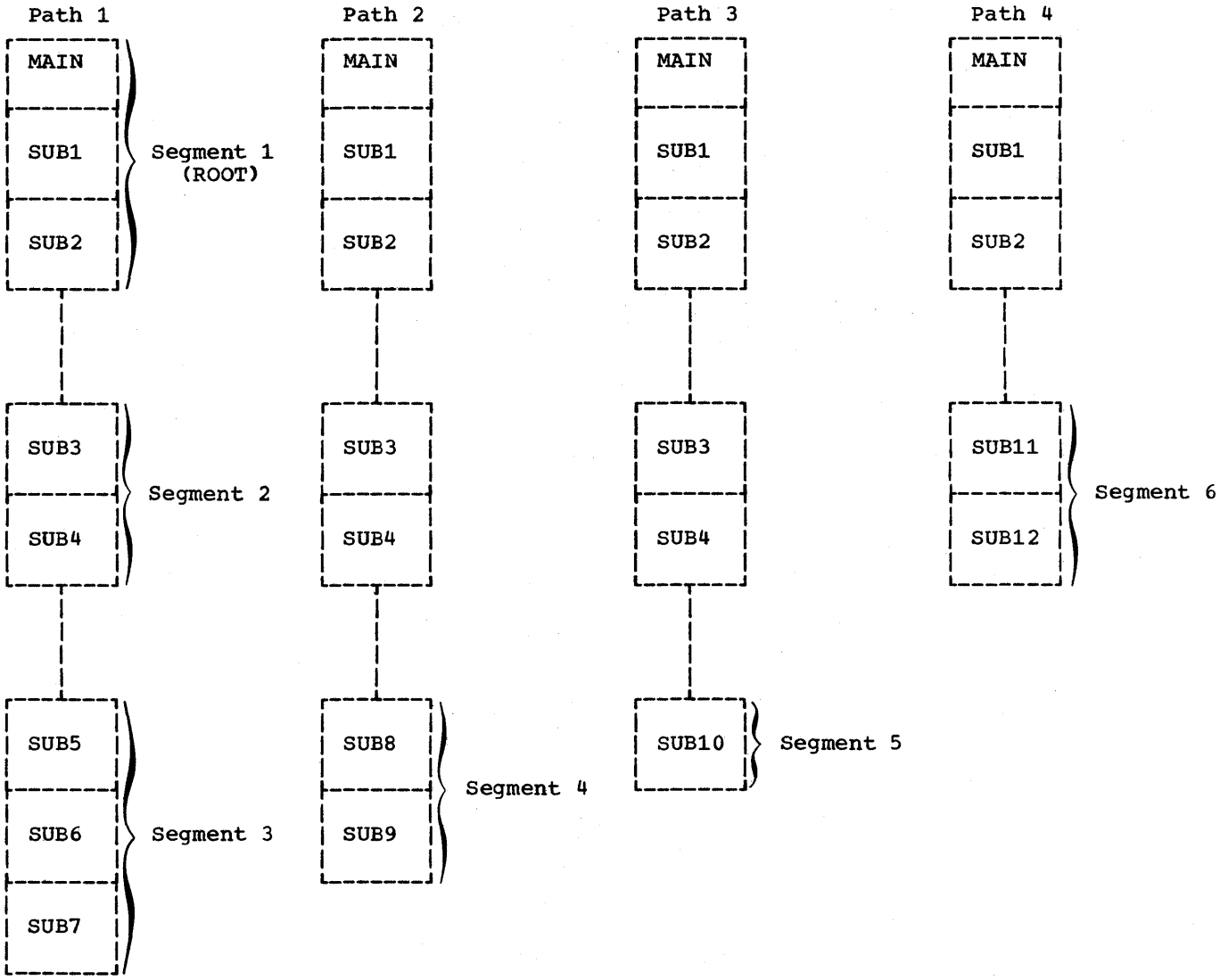


Figure 84. Paths Implied by Tree Structure in Figure 83

and SUB2 will remain in main storage for the duration of execution time. They will occupy the root segment. The segment containing subprograms SUB3 and SUB4 will use the same area of main storage as the segment containing subprograms SUB11 and SUB12. Likewise, the main storage area used by the segment containing SUB5, SUB6, and SUB7 will be used by the segment containing SUB8 and SUB9, as well as by the segment containing SUB10. Figure 85 is a time/storage map of the program shown in Figures 83 and 84.

The structure in Figures 83 and 84 consists of segments numbered 1 through 6, with segment 1 being the root segment. Segments 2 and 6 have the same relative origin; that is, they will start at the same location when in main storage. This origin has been given the symbolic name ALPHA by the programmer (on an OVERLAY control card). The relative origin of segments 3, 4, and 5 has been given the symbolic name BETA.

The relative origin of the root segment, also called the relocatable origin, is assigned at 0. The relative origin of any segment other than the root segment is determined by adding the lengths of all segments in its path, including the root segment. When the program is loaded for execution, the first location of the root segment (the relocatable origin of the program) is assigned to an absolute storage address. All other origins are automatically increased by the value of that storage address.

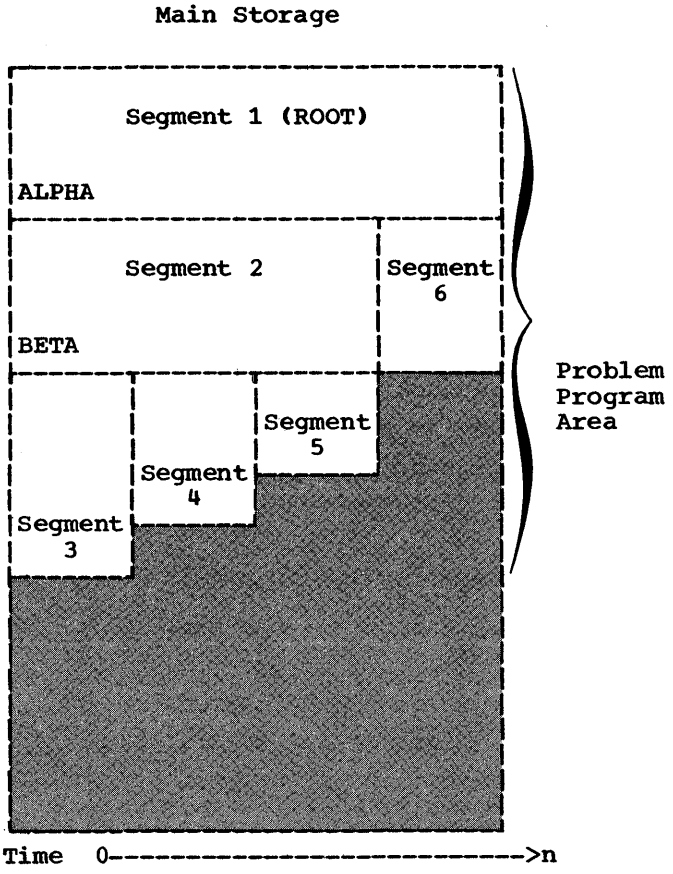


Figure 85. Time/Storage Map of Six Segment Structure

COMMUNICATION BETWEEN SEGMENTS

Overlay segments can be related to one another by being either inclusive or exclusive. Inclusive segments are those which can be in main storage simultaneously, in other words, those which lie in the same path. Exclusive segments are those which lie in different paths. Thus, in the program shown in Figures 83 and 84, segments 2 and 5 are inclusive, while segments 2 and 6 are exclusive.

INCLUSIVE REFERENCES

An inclusive reference is a reference from a segment in main storage to a subprogram that will not overlay the calling segment. When a CALL is made from a program unit in one segment to a program unit in an inclusive segment, control may be returned to the calling segment by means of a RETURN statement.

When a CALL is issued in any segment to a subprogram which is higher (closer to the root segment) on the overlay tree, the called subprogram must return control to the calling segment by a RETURN statement before any exclusive overlay segments may be loaded.

EXCLUSIVE REFERENCES

Exclusive references are those made in one segment to another segment that will overlay it.

An exclusive reference is considered valid only if the called routine is also referred to in a segment common to both the segment to be loaded and the segment to be overlaid. Assume, for example, in Figure 86 that the main program (common segment) contains a reference to segment A but not to segment B. A reference in segment B to a routine in segment A is valid because there is also an inclusive reference between the common segment and segment A. (A table in the common segment, supplied by the linkage editor, contains the address of segment A. The overlay does not destroy this table.) An exclusive reference in segment A to a routine in segment B is invalid because there is no reference to segment B in the common segment.

Both valid and invalid exclusive references are considered errors by the linkage editor; however, by use of the LET or XCAL processing options described later in this section, the programmer can allow a program containing a valid exclusive reference to be executed. Programs containing invalid exclusive references are never executable.

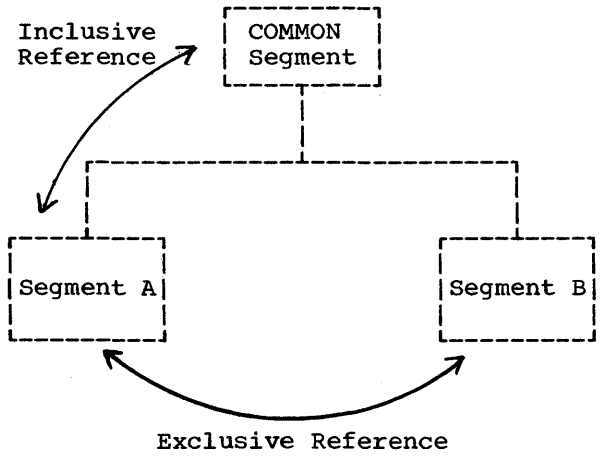


Figure 86. Communication Between Overlay Segments

OVERLAY PROCESSING

Overlay is initiated at execution time in response to a reference to a subprogram which is not already in main storage. The subprogram reference may be either a FUNCTION name or a CALL statement reference to a SUBROUTINE name. When the reference is executed, the overlay segment containing the required subprogram, as well as any segments in its path not currently in main storage, are loaded.

When a segment is loaded, it overlays any segment in storage with the same relative origin. It also overlays any segments that are lower in the path of the overlaid segment (i.e., farther from the root segment). For example, if segments 1, 2, and 3 in Figures 83 and 84 are in main storage when the main program executes a call to subprogram SUB11, segments 2 and 3 will not be available for as long as segment 6 is in main storage.

Whenever a segment is loaded, it contains a fresh copy of the program units that it comprises; any data values that may have been established or altered during previous processing are returned to their initial values each time the segment is loaded. Thus, data values that are to be retained for longer than a single load phase should be placed in the root segment.

Overlay is not initiated when a return is made from a subprogram, or when a segment in main storage executes a reference to a subprogram that is already in main storage.

COMMON AREAS

The linkage editor treats all FORTRAN COMMON areas as separate subprograms. When modules containing COMMON areas are processed by the linkage editor, the COMMON areas are collected. That is, when two or more blank (unnamed) COMMON areas are encountered in the input to the linkage editor, only the largest of them is retained in the output module. Likewise, when two or more named COMMON areas of the same name are encountered, only the largest of them is retained in the output module.

In an overlay program, the ultimate location of blank and named COMMON areas in the output module depends upon which linkage editor control statements are used in the building of the overlay structure (see the section "Construction of the Overlay Program"). Overlay structures built without the use of INSERT statements (those in which the program units for each segment are included between OVERLAY statements) produce an output module in which the linkage editor "promotes" the COMMON areas automatically. The promotion process places each COMMON area in the lowest

possible segment on the overlay tree. The lowest possible segment is one that will always be in main storage with every segment containing a reference to it.

Figures 87 and 88 show an overlay program as it appears before and after the automatic promotion of COMMON areas. The exact position of a promoted COMMON area within the segment to which it is promoted is unpredictable.

If INSERT statements are used to structure the overlay program, a blank COMMON area should appear physically in the input stream in the segment to which it belongs. A named COMMON area either should appear physically in the segment to which it belongs, or should be placed there with an INSERT statement.

COMMON areas encountered in modules from automatic call libraries are automatically promoted to the root segment. If such COMMON areas are named, they can be positioned by the use of an INSERT statement.

Named COMMON areas in BLOCK DATA subprograms must be at least as large as any identically named COMMON areas in FORTRAN programs that are to be link edited with the BLOCK DATA subprograms.

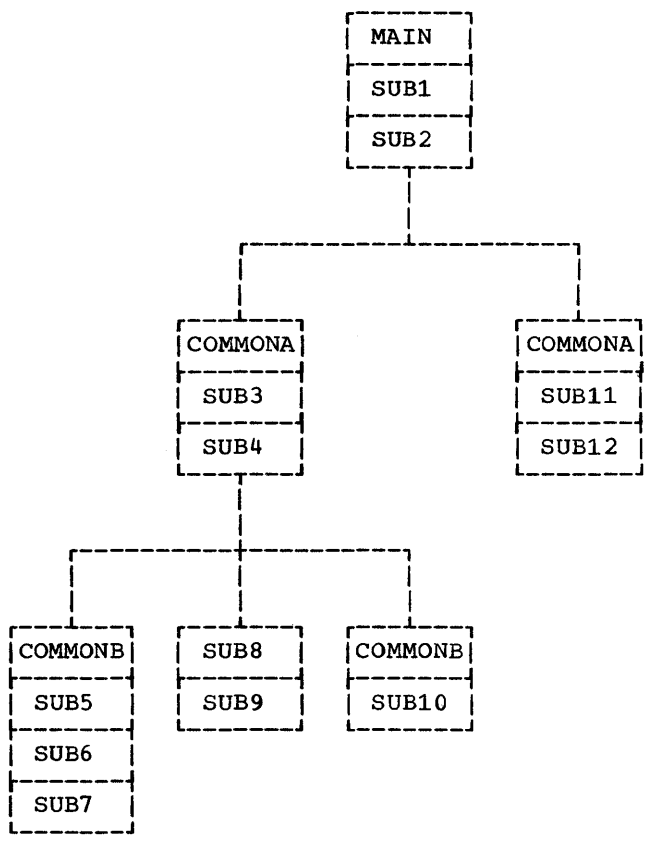


Figure 87. Overlay Program Before Automatic Promotion of Common Areas

CONSTRUCTION OF THE OVERLAY PROGRAM

The programmer communicates his overlay strategy to the operating system in two ways: through the use of special processing options which he specifies in the PARM parameter of the EXEC statement which invokes the linkage editor, and through the use of linkage editor control statements. The general functions of these options and statements are described in the section "FORTRAN Job Processing." Those which are of particular interest to the programmer constructing an overlay program are discussed below.

LINKAGE EDITOR CONTROL STATEMENTS

Once the programmer has designed an overlay tree structure for his program, he places the program in that structure by indicating to the linkage editor the relative positions of the segments which make up the tree. The control statements which accomplish this are placed in the input stream following the //SYSIN DD* card, or after the //LKED.SYSIN DD* card if a cataloged procedure is used. Linkage editor control statements have the following form:

Operation	Operand
verb	operand(s)

where "verb" is the name of the operation to be performed. The first column of all linkage editor control statements must be blank, and the operation field, which begins in column 2, must contain a verb. The operand field, which must be separated from the operation field by at least one blank, must contain one or more symbols separated by commas. No embedded blanks may appear in the operand field. Linkage editor control statements are placed before, between, or after modules in the input stream. They may be grouped, but they may not be placed, within a module.

The most important control statements for implementing an overlay program are the OVERLAY, INSERT, INCLUDE and ENTRY statements. The OVERLAY statement

indicates the beginning of an overlay segment. The INSERT statement is used to

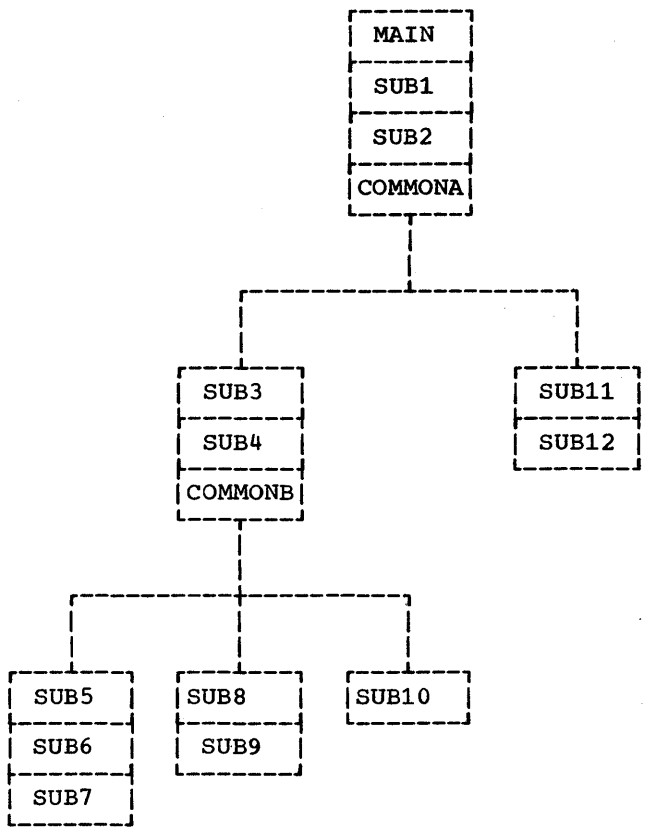


Figure 88. Overlay Program After Automatic Promotion of Common Areas

rearrange the sequence of object modules in the resulting load module(s). The INCLUDE statement is used to incorporate input from secondary sources into the load module. The ENTRY statement specifies the first instruction to be executed.

The OVERLAY Statement

The OVERLAY statement indicates the beginning of an overlay segment. Its general form is:

Operation	Operand
OVERLAY	symbol

where the operand "symbol" is the programmer's identification of the beginning of the segment, that is, the symbolic name of the relative origin. Such symbols may be any group of from one through eight alphameric characters beginning with an alphabetic character.

The OVERLAY statement for a segment is placed in one of three places: directly before the object module deck for the first program unit of the new segment, or before an INSERT statement specifying the program units to be placed in the segment, or before an INCLUDE statement specifying the program units to be placed in the segment. Assuming that object module decks are available, the input deck to the linkage editor for the program in Figures 83 and 84 could be arranged as follows:

```
MAIN  }
SUB1  } object module deck
SUB2  }
OVERLAY ALPHA
SUB3  }
      } object module deck
SUB4  }
OVERLAY BETA
SUB5  }
SUB6  } object module deck
SUB7  }
OVERLAY BETA
SUB8  }
      } object module deck
SUB9  }
OVERLAY BETA
SUB10 } object module deck
OVERLAY ALPHA
SUB11 }
      } object module deck
SUB12 }
ENTRY MAIN
```

The order in which the overlay segments are specified has nothing to do with the order of execution, which is determined by subprogram references; however, once a symbolic name has been specified for a point of origin, it may not be used again in the deck after specifications have been made for a point higher in the overlay tree. Thus, in the example above, no further segments could be specified for load point BETA after the second specification for load point ALPHA.

An OVERLAY statement must never be placed before the root segment.

The INSERT Statement

There are many instances in which it is inconvenient or impossible for the programmer to position object module decks physically in the input stream. Library routines, which are normally placed in the root segment, and routines compiled in an earlier step in the same job, are examples of program units for which the object module decks are not available for positioning at the time the job is set up.

The INSERT statement is used to position control sections from such program units in an overlay structure. A control section, or CSECT, is the operating system designation for the smallest separately relocatable unit of a program. Examples of FORTRAN control sections are: main programs, subprograms, blank and named COMMON blocks.

The INSERT statement has the form:

Operation	Operand
INSERT	csectname[,csectname ...]

where "csectname" is the name of the control section to be positioned. Multiple operands, separated by commas (not blanks), may be specified.

The INSERT statement is placed directly after the OVERLAY statement for the segment containing the control section. If the control section is to be positioned in the root segment, the INSERT statement is placed before the first OVERLAY statement.

Using INSERT statements and a FORTRAN object module deck, the overlay structure specified in Figures 83 and 84 could be implemented as follows:

```
FORTRAN object
module deck con-
taining units MAIN
through SUB12
```

```
ENTRY MAIN
INSERT MAIN, SUB1,SUB2
OVERLAY ALPHA
INSERT SUB3,SUB4
OVERLAY BETA
INSERT SUB5,SUB6,SUB7
OVERLAY BETA
INSERT SUB8,SUB9
OVERLAY BETA
INSERT SUB10
OVERLAY ALPHA
INSERT SUB11,SUB12
```

If INSERT statements are used more than once in the same program for a control section of the same name, the CSECT will be positioned in the segment specified by the first occurrence of the CSECT name in the input stream. Any additional INSERT statements referring to the CSECT will be ignored and, at execution time, all references to the CSECT will resolve to the first one positioned. Thus, if a subprogram is required in more than one path, it must be either inserted in the root segment or renamed before being used with an INSERT statement.

The INCLUDE Statement

The INCLUDE statement is described in the section "FORTRAN Job Processing." When used in an overlay program, the INCLUDE statement is generally placed in the segment in which the material to be included is required. It is possible to manipulate the control sections which were added by an INCLUDE statement through the use of the INSERT statement. Assuming that the control sections of the overlay program from the previous examples resided in a partitioned data set named LIBA and a sequential data set LIBB as follows:

LIBA	
BOOK1	BOOK2
MAIN	SUB3
SUB1	SUB4
SUB2	

LIBB
SUB5
SUB6
SUB7
SUB8
SUB9
SUB10
SUB11
SUB12

Then the overlay structure could be implemented by the use of the following control statements:

```
ENTRY MAIN
INCLUDE LIBA(BOOK1)
INCLUDE LIBB
OVERLAY ALPHA
INCLUDE LIBA(BOOK2)
OVERLAY BETA
INSERT SUB5,SUB6,SUB7
OVERLAY BETA
INSERT SUB8,SUB9
OVERLAY BETA
INSERT SUB10
OVERLAY ALPHA
INSERT SUB11,SUB12
```

The ENTRY Statement

The ENTRY statement specifies the first instruction of the program to be executed. It has the form:

Operation	Operand
ENTRY	External-name

where the operand "external-name" must be the name of an instruction in the root segment. Usually it will be the name MAIN.

The ENTRY statement may be placed before, between, or after the program units or other control statements in the input stream. An ENTRY statement is necessary in all overlay programs because, after linkage editor processing, the first part of the root segment contains special overlay control information rather than executable code. The previous examples of overlay implementation show the use and placement of the ENTRY statement.

PROCESSING OPTIONS

Along with the necessary linkage editor control statements, the programmer implementing an overlay structure must provide certain information to the operating system by means of the PARM parameter of the EXEC statement which invokes the linkage editor. This information is in the form of keyword parameters such as OVLY, LIST, XCAL, etc. Thus, the EXEC statement invoking the linkage editor might have the form:

```
//LKED EXEC PGN=IEWL,PARM='OVLY,LIST,...'
```

When the linkage edit is one step (stepname LKED) of a compile, linkage edit, and execute procedure such as FORTGCLG, the PARM information is supplied in the EXEC statement for the cataloged procedure as follows:

```
//STEP EXEC FORTGCLG,PARM.LKED='OVLY,
  LIST,...'
```

When PARM is specified for a cataloged procedure, any processing options which were originally part of the procedure are nullified. It is therefore good practice to list all desired options when PARM is used for the linkage editor step of such procedures.

Those linkage editor processing options which are of special interest to the overlay programmer are discussed below.

- The OVLY option indicates that the load module produced will be an overlay structure, as directed by subsequent linkage editor control statements. OVLY must be specified for all overlay processing.
- When LIST is specified, all linkage editor control statements will be listed in card image format on the diagnostic output data set, SYSPRINT.
- The MAP option instructs the linkage editor to produce a map of the output module. The map of the output module of an overlay structure shows the control sections grouped by segment. Within each segment, the control sections are listed in ascending order according to their assigned origins. The number of the segment in which each appears is also printed.
- When the XREF option is specified, the linkage editor produces a cross-reference table of the output module. The cross-reference table includes a module map and a list of all address constants that refer to other control sections. Since the cross-reference table includes a module map, XREF and MAP cannot both be specified for one linkage editor job step.
- When XCAL is specified to the linkage editor, a valid exclusive call is not considered an error, and the load module is to be marked executable, even though improper branches were made between control sections.
- When LET is specified, any exclusive call (valid or invalid) is accepted. The output module will be marked "ready for execution" even though certain error or abnormal conditions were found during linkage editing. At execution time, a valid exclusive call may or may not be executed correctly. An invalid call will usually cause unpredictable results; the requested segment will not be loaded.

This section describes the error diagnostic facilities available during program execution when the extended error handling facility has been requested at system generation time.[1]

The extended error handling facility provides the user with information about data-dependent or program errors detected in a FORTRAN program during execution.[2] (These errors are not syntactical or semantic in nature.) When a data-dependent or program error occurs, the user is given:

- Messages more informative than those issued with standard diagnostic facilities.
- Traceback information more extensive than that provided with standard diagnostic facilities.
- Either standard FORTRAN corrective action with continued execution or, optionally, the opportunity to examine and alter erroneous data.

When an error occurs with extended error handling in effect, a short message text is printed along with an error identification number. The data in error (or some other associated information) is printed as part of the message text. A summary error count, printed when a job is completed, informs the user how many times each error occurred.

A traceback map, tracing the subroutine flow back to the main program, is printed after each error occurrence; execution then continues. (If the extended error handling facility is not specified, a traceback map is printed only for errors causing program termination and -- if the ERR= option has been specified in a READ statement -- for error IHC218I.)

For each error condition detected, the user has both dynamic and default control over:

- The number of times the error is allowed to occur before program termination.
- The maximum number of times each message may be printed.
- Whether or not the traceback map is to be printed with the message.
- Whether or not a user-written error-exit routine is to be called.

The action that takes place is governed by information stored in an area of main storage called the option table. (A permanent copy of the option table is maintained in the FORTRAN library.)

FUNCTIONAL CHARACTERISTICS

When an error is detected, the FORTRAN error monitor (ERRMON) receives control. The error monitor is passed the following information:

- An error identification number.
- The text of the appropriate message to be printed on the object error unit.
- A pointer to the data in the error.
- The address of an area for a return code.

The error monitor prints the necessary diagnostic and informative messages and then takes one of the following actions:

- Terminates the job.
- Returns control to the calling routine, which takes a standard corrective action and then continues execution.
- Calls a user-written closed subroutine to correct the data in error, and then returns to the routine that detected the error, which then continues execution.

The actions of the error monitor are controlled by settings in the option table. The option table consists of a doubleword preface, followed by a doubleword entry for each error condition. (If the extended error handling facility is not specified, the option table is reduced to the preface

---

[1]This facility is requested by means of the OPTERR parameter of the FORTLIB macro instruction. For details, see the System Generation publication.

[2]The errors detected by the extended error handling facility are listed in Appendix D under the heading "Extended Error Messages for Execution Errors."

alone.) IBM provides a default of 95 entries; the programmer can provide additional entries during system generation. Figures 89 and 90 describe the fields of the option table and list the system generation default values for the contents of these fields. Table 20 shows the system generation default values for each error condition. Note that default values can be overridden only; they cannot be permanently changed.

SUBPROGRAM FOR THE EXTENDED ERROR HANDLING FACILITY

To make full use of the extended error handling facility, the programmer may call four IBM-supplied subroutines in his FORTRAN source program: ERRSAV, ERRSTR, ERRSET, and ERRTRA. These subroutines allow access to the option table to alter it dynamically.[1] Changes made dynamically are in effect for the duration of the program that made the change. Only the current copy of the option table in main storage is affected; the copy in the FORTRAN library remains unchanged. All passed parameters, unless otherwise indicated, are 4-byte (fullword) integers.

Accessing and Altering the Option Table Dynamically

1. The CALL ERRSAV statement, described below, can be used in modifying an entry temporarily to save the original entry for later restoration. The statement causes an option table entry to be copied into an 8-byte storage area accessible to the FORTRAN programmer.

   CALL ERRSAV (ierno,tabent)

   ierno
   is an integer equal to the error number to be referenced in the option table. Should any number not within the range of the option table be used, an error message will be printed.

   tabent
   is the address of an 8-byte storage area where the option table entry is to be stored.

2. To store an entry in the option table, the following statement is used:

   CALL ERRSTR (ierno,tabent)

   ierno
   is an integer equal to the error number for which the entry is to be stored in the option table. Should any number not within the range of the option table be used, an error message will be printed.

   tabent
   is the address of an 8-byte storage area containing the table entry data.

3. The CALL ERRSET statement, described below, permits the user to change up to five different options in an option table entry. A procedure for altering only one option without altering others is explained in the definition of the parameters. Another procedure is to omit the final parameter (or the last two or three parameters) from the calling sequence, or to give the value of zero to a parameter to indicate no change.

   CALL ERRSET (ierno,inoal,inomes,
   itrace,iusadr,irange)

   ierno
   is an integer equal to the error number to be referenced in the option table. Should any number not within the range of the option table be used, an error message will be printed. (Note that if ierno is specified as 212, there is a special relationship between the ierno and irange parameters. See the explanation for irange.)

   inoal
   is an integer specifying that execution be terminated when this number of errors has occurred. For example, setting inoal equal to 10 will permit transfer of control to a user-supplied error routine for the first nine error occurrences. On the tenth error occurrence execution will be terminated. If inoal is specified as either zero or a negative number, the specification is ignored, and the number-of-errors option is not altered. If a value of more than 255 is specified, an unlimited number of errors is permitted.

---

[1] Certain option table entries may be protected against alteration when the option table is set up. If a request is made by means of CALL ERRSTR or CALL ERRSET to alter such an entry, the request is ignored. (See Table 20 for which IBM-supplied option table entries cannot be altered.)

inomes
is an integer indicating the number of messages to be printed. A negative value specified for inomes causes all messages to be suppressed while setting inomes to 256 will cause unlimited message printing. A specification of zero indicates that the number-of-messages option is not to be altered.

itrace
is an integer whose value may be 0, 1, or 2. A specification of 0 indicates the option is not to be changed; a specification of 1 requests that no traceback be printed after an error occurrence; a specification of 2 requests the printing of a traceback after each error occurrence. (If a value other than 1 or 2 is specified, the option remains unchanged.)

iusadr
is an optional parameter that may contain either:

a. the value 1, as a 4-byte integer, indicating that the option table is to be set to show there is no user-exit routine (i.e., standard corrective action is to be used when continuing execution).

b. the name of a closed subroutine that is to be executed after the occurrence of the error identified by ierno. The name must appear in an EXTERNAL statement in the source program, and the routine to which control is to be passed must be available at linkage editing time.

c. the value 0, indicating that the table entry is not to be altered.

irange
is an optional parameter specified as an integer that performs a double function and indicates that the inoal, inomes, itrace, and iusadr options values are to be applied to the range of error numbers ierno to irange. If irange is smaller than ierno, irange is ignored (unless ierno has been specified as 212).

If ierno has been specified as 212, irange functions as a control carriage parameter. Thus, if ierno is specified as 212, and irange as 1, single spacing is provided on an overflow line (standard fixup for WRITE). If a value other than 1 is specified, no carriage control is provided. (Note that if ierno has been specified as 212 and the carriage control option is not to be changed, irange must be omitted from the call to ERRSET.)

4. Under the extended error handling facility, a user may dynamically request a traceback and continued execution. To obtain subroutine trace, the following statement is used:

```
CALL ERRTRA
```

The call has no parameters.

USER-SUPPLIED ERROR HANDLING

The user has the ability of calling, in his own program, the FORTRAN error monitor (ERRMON) routine, the same routine used by FORTRAN itself when it detects an error. ERRMON examines the option table for the appropriate error number and its associated entry and takes the actions specified. If a user-exit address has been specified, ERRMON transfers control to the user-written routine indicated by that address. Thus, the user has the option of handling errors in one of two ways: (1) simply by calling ERRMON -- without supplying a user-written exit routine; or (2) by calling ERRMON and providing a user-written exit routine.

In either case, certain planning is required at the installation level. For example, error numbers must be assigned to error conditions to be detected by the user, and additional option table entries must be made available for these conditions. The routine that uses the error monitor for error service should have the status of an installation general-purpose function similar to the IBM-supplied mathematical functions. The number of installation error conditions must be known when the FORTRAN library is created at system generation, so that entries will be provided in the option table by the ADDNTRY parameter of the FORTLIB macro instruction. The error numbers chosen for user subprograms are restricted in range. IBM-designated error conditions have reserved error codes from 000 to 301. Error codes for installation-designated error situations must be assigned in the range 302 to 899. The error code is used by FORTRAN to find the proper entry in the option table.

To call the ERRMON routine, the following statement is used:

```
CALL ERRMON (imes,iretcd,ierno
  [,data1,data2,...] )
```

imes
: is the address of an array aligned on a fullword boundary, that contains, in EBCDIC characters, the text of the message to be printed. The number of the error condition should be included as part of the text, because the error monitor prints only the text passed to it. The first item of the array contains an integer whose value is the length of the message. Thus, the first four bytes of the array will not be printed. If the message length is greater than 133 characters, it will be printed on two or more lines of printed output.

iretcd
: is an integer variable made available to the error monitor for the setting of a return code. A code of 0 or 1 can be set. An interpretation of these codes follows:

  0 - The option table or user-exit routine indicates that standard correction is required.

  1 - The option table indicates that a user exit to a corrective routine has been executed. The function is to be reevaluated using arguments supplied in the parameters data1,data2.... For input/output type errors, the value 1 indicates that standard correction is not wanted.

ierno
: is an integer representation of the error condition. The value assigned identifies an error condition for which there is a unique entry in the option table. Should any number not within the range of the option table be specified, an error message will be printed.

data1,data2...
: are variable names in an error-detecting routine for the passing of arguments found to be in error. One variable must be specified for each argument. Upon return to the error-detecting routine, results obtained from corrective action are in these variables. Because the content of the variables can be altered, the locations in which they are placed should be used only in the CALL statement to the error monitor; otherwise, the user of the function may have literals or variables destroyed.

  Since data1 and data2 are the parameters which the error monitor will pass to a user-written routine to correct the detected error, care must be taken to make sure that these parameters agree in type and number in the call to ERRMON and in a user-written corrective routine, if one exists.

User-Supplied Exit Routine

When a user-exit address is supplied in the option table entry for a given error number, the error monitor calls the specified subroutine for corrective action. The subroutine may be user-written and is called by the equivalent of the following FORTRAN statement:

```
CALL x (iretcd,ierno,data1,data2...)
```

x
: is the name of the routine whose address was placed into the option table by the iusadr parameter of the CALL ERRSET statement. (Interpretations of the other parameters -- iretcd, ierno, data1, data2 -- are the same as those for the CALL ERRMON statement.) If an input/output error is detected (i.e., an error for codes 211 to 237), subroutine "x" must not execute any FORTRAN I/O statements, i.e., READ, WRITE, BACKSPACE, END FILE, REWIND, DEBUG, or any calls to PDUMP or ERRTRA. Similarly, if errors for codes 216 or 241-301 occur, the subroutine "x" must not call the library routine that detected the error or any routine which uses that library routine. For example, a statement such as

  ```
  R = A ** B
  ```

  cannot be used in the exit routine for error 252, because FRXPR# uses EXP, which detects error 252.

Note that although a user-written corrective routine may change the setting of the return code (iretcd), such a change is subject to the following restrictions:

1. If iretcd is set to 0, then data1 and data2 must not be altered by the corrective routine, since standard corrective action is requested. If data1 and data2 are altered when iretcd is set to 0, the operations that follow will have unpredictable results.

2. Only the values 0 and 1 are valid for iretcd. A user-exit routine must ensure that one of these values is used if it changes the return code setting.

Note, too, that the user-written exit routine can be written in FORTRAN or in assembler language. In either case, it must be able to accept the call to it as shown above. The user-exit routine must be a closed subroutine that returns control to the caller.

If the user-written exit routine is written in assembler language, the end of the parameter list can be checked. The high-order byte of the last parameter will have the hexadecimal value 80. If the routine is written in FORTRAN, the parameter list must match in length the parameter list passed in the CALL statement issued to the error monitor.

Actions the user may take if he wishes to correct an error are described in Tables 21, 22 (parts 1, 2, 3), and 23.

OPTION TABLE CONSIDERATIONS

When a user-written exit subroutine is to be executed for a given error condition, the programmer must enter the address of the routine into the option table entry associated with that error condition.

Addresses for user-exit subroutines cannot be entered into option table entries during system generation. An installation may, however, construct an option table containing user-exit addresses and placed that option table into the FORTRAN library. (Each address must be specified as a V-type address constant.) Use of this procedure, though, results in the inclusion, in the load module, of all such user-exit subroutines by the linkage editor.

If the user-exit address is not specified in advance through the use of V-type address constants, the programmer must issue a CALL ERRSET statement at execution time to insert an address into the option table that was created during system generation.

The programmer should be warned that altering an option table entry to allow "unlimited" error occurrence (specifying a number greater than 255) may cause a program to loop indefinitely.

Option Table Default Values

Table 20 shows the default values for the option table. If an option recorded in a table entry does not apply to a particular error condition, it is shown as not applicable (NA).

The field that is defined as the user-exit address also serves as a means of specifying standard corrective action. When the table entry contains an address, the user exit is specified; when it contains the integer 1, standard correction is specified. It is not possible for the system generation process to create an option table entry with the user-exit address specified. The user exit must be specified by altering the option table at execution time. To specify that no corrective action -- either standard or user-written -- is to be taken, the table entry must specify that only one error is to be allowed before termination of execution.

HOW TO CREATE OR ALTER AN OPTION TABLE

As previously explained, the option table supplied during system generation may be altered dynamically for any particular FORTRAN job by the use of the subprograms ERRSET and ERRSTR. However, to provide a new set of options for the entire installation, the option table must be reassembled and linkage edited into the FORTRAN library -- after system generation and before the system is used. A procedure for accomplishing this is described in the following text.

An assembler language macro definition can be used to generate an option table. The macro definition and use of the macro for each option table entry are supplied as input to the assembler procedure ASMFCL to replace the system-generated option table with the new one. An example of an assembler language macro definition used to generate an option table is shown in Figure 91. This example may be used as a guide by the user.

In the example, the macro parameters are as follows:

PREFACE a,b,c

a
: is the number of user entries to be created.

b
: is the boundary alignment desired. A

value of 0 is used for no alignment; a value of 1 for alignment.

c

is the number of times the SETENT macro instruction is to be issued. (SETENT is described below.)

SETENT (a,b,c,d)

a

is the error entry to be altered.

b

is the count of errors to be allowed. (A specification of 0 indicates unlimited error occurrence.)

c

is the count of the number of times the message should be printed before suppression.

d

is two hexadecimal digits that specify the option bits field. This field is described in Figure 90.

The macro instructions are used as follows:

1. Only one PREFACE macro instruction is allowed.

2. As many SETENT macro instructions as are desired may be used. From 1 to 200 error entries can be specified in the use of a single SETENT macro instruction by using continuation cards.

3. Only error entries that differ from the default options need be specified. The default options will be the same as those listed in Table 20.

4. Error codes must be placed in ascending order in the SETENT macro instruction. For IBM-supplied entries, error codes are in the range 207 to 301. User entries are in the range 302 to 899.

5. Changing one option for any error entry requires that all four parameters be specified. If default values are desired for an entry, they must be respecified. For example:

   SETENT (241,50,5,42)

   indicates that for error 241, the number of errors to be allowed is 50; the other two parameters, which must be specified, are simply the default values shown in Table 20.

ERRORS IN USE OF FACILITY

When the extended error handling facility encounters a condition or a request that requires user notification, an informational message is printed.

The error monitor is not recursive: If it has already been called for an error, it cannot be re-entered if the user-written corrective routine causes any of the error conditions that are listed in the option table. Boundary misalignment is therefore not allowed in a user-exit routine.

PROGRAMMING EXAMPLE

The programming example in Figure 92 shows how features of the error handling facility may be used.

In the example, a FORTRAN job utilizes a user-supplied library subprogram that makes use of the error handling facility to handle a divide-by-zero situation. A user-written routine is supplied to take corrective action after the detection of the error. Comments in the FORTRAN program describe what is being done.

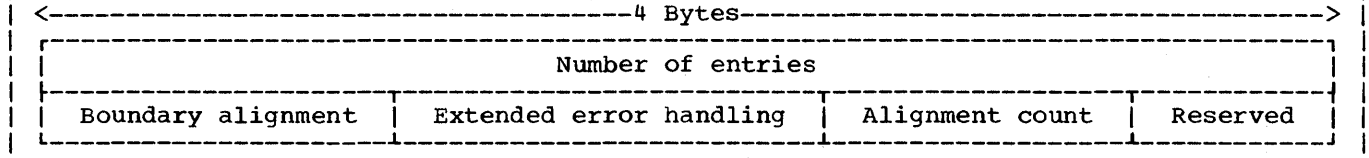
CONSIDERATIONS FOR THE LIBRARY WITHOUT EXTENDED ERROR HANDLING FACILITY

When the extended error handling facility is not chosen at system generation, execution terminates after the first occurrence of an error, unless it is one caused by boundary misalignment, divide check, exponent underflow, or exponent overflow. The messages for errors 215, 216, 218, 221-225, and 241-301 are the same as those with the extended error handling facility. The other error messages are of the form "IHCxxxI" with no text.

Without the facility, ERRMON becomes an entry point to the traceback routine. User programs that call the error monitor do not have to be altered. The error message will be printed with a traceback map and execution will terminate.

Note, too, that if the facility is not selected at system generation, the ERRTRA, ERRSET, ERRSAV, and ERRSTR subprograms are assumed to be user supplied if they are called in a FORTRAN program.

Format



Description

Field Contents	Length in Bytes	Field Description
Number of entries	4	Number of entries in the Option Table. The default setting is 95.
Boundary alignment	1	Bit 1 of this byte indicates whether boundary alignment was chosen at system generation. (Bits 0 and 2 through 7 are reserved for future use.) Bit 1: 0 = NOALIGN 1 = ALIGN The default setting is 1 (ALIGN).
Extended error handling	1	Indicates whether extended error handling facility was chosen at system generation. FF(hexadecimal) = EXCLUDE 00(hexadecimal) = INCLUDE
Alignment count	1	Maximum number of boundary alignment messages when extended error handling facility is not chosen. The default setting is 10.
Reserved	1	Reserved for future use.

Figure 89. Option Table Preface

Format

<------1 Byte------>	<------1 Byte------>	<------1 Byte------>	<------1 Byte------>
Number of error occurrences allowed	Number of messages to print	Error count	Option bits
User exit address			

Description

Field Contents	Default[1]	Field Description
Number of error occurrences allowed	10[2]	Number of times this error condition should be allowed to occur. The maximum setting is 255. When the value of the error count field (below) matches the value in this field, the job is terminated. A value of 0 means an unlimited number of occurrences.[3] An attempt to set this field at a value greater than 255 results in the field being set to 0.
Number messages to print	5[4]	The number of times to print the error message when this error occurs. Message printing is suppressed thereafter. A value of 0 means no message is to be printed.
Error count	0	A count of the number of times this error has occurred. A value of 0 means none have occurred.
Option bits	42 (hexa-decimal)	Eight option bits defined as follows: Bit 0: 0 = no control character is supplied for overflow lines. 1 = control character supplied, when needed, to provide single spacing for overflow lines. The default setting is 0. Bit 1: 0 = table entry not modifiable.[5] 1 = table entry modifable. The default setting is 1. Bit 2: 0 = fewer than 256 errors have occurred. 1 = more than 256 errors have occurred. To determine how many, add 256 to the value contained in the error count field (above). The default setting is 0. Bit 3:[6] 0 = buffer content with message for this error is not to be printed. 1 = buffer content is to be printed. The default setting is 0. Bit 4: Reserved for future use. Bit 5: 0 = unlimited printing of error message was not requested. The error message is to be printed only the number of times shown in the number-of-messages-to-print field (above). 1 = message is to be printed with every occurrence of this error. The default setting is 0. Bit 6: 0 = traceback is not to be printed. 1 = traceback is to be printed. The default setting is 1. Bit 7: Reserved for future use.
User exit	1	A value of 1 in this field indicates that no user-supplied exit is to be taken. If a value other than 1 appears in this field, it is the address of the user-supplied exit routine to be taken when this error occurs.

[1]The default values shown apply to all error numbers (including additional user entries) unless excepted by a footnote.
[2]Errors 208, 210, and 215 are set as unlimited, and errors 217 and 230 are set to 1.
[3]When the user sets the count of allowed errors as unlimited, the FORTRAN job may loop endlessly unless the operator intervenes.
[4]Error 210 is set to 10, and errors 217 and 230 are set to 1.
[5]The entry for error 230 and 240 is not modifiable.
[6]This entry is set to 0 except for error numbers 212, 215, 218, 221, 222, 223, 224, and 225.

Figure 90. Option Table Entry

Table 20. Option Table Default Values

Error Code	Number of Errors Allowed	Number of Messages Allowed	Print Control	Modifiable Entry	Print Buffer Content	Traceback Allowed	Standard Corrective Action	User Exit
207	10	5	NA	Yes	NA	Yes	Yes	No
208	Unlimited	5	NA	Yes	NA	Yes	Yes	No
209	10	5	NA	Yes	NA	Yes	Yes[1]	No[1]
210	Unlimited	10	NA	Yes	NA	Yes	Yes[1]	No
211	10	5	NA	Yes	NA	Yes	Yes	No
212	10	5	No character supplied[2]	Yes	Yes	Yes	Yes	No
213	10	5	NA	Yes	NA	Yes	Yes	No
214	10	5	NA	Yes	NA	Yes	Yes	No
215	Unlimited	5	NA	Yes	Yes	Yes	Yes	No
216	10	5	NA	Yes	NA	Yes	Yes[3]	No
217	1[4]	1	NA	Yes	NA	Yes	Yes	No
218	10[5]	5	NA	Yes	Yes[5]	Yes	Yes	No
219	10[6]	5	NA	Yes	NA	Yes	Yes	No
220	10	5	NA	Yes	NA	Yes	Yes	No
221	10	5	NA	Yes	Yes	Yes	Yes	No
222	10	5	NA	Yes	Yes	Yes	Yes	No
223	10	5	NA	Yes	Yes	Yes	Yes	No
224	10	5	NA	Yes	Yes	Yes	Yes	No
225	10	5	NA	Yes	Yes	Yes	Yes	No
230	1	1	NA	No	NA	Yes	No	No
231	10	5	NA	Yes	NA	Yes	Yes	No
232	10	5	NA	Yes	NA	Yes	Yes	No
233-237	10	5	NA	Yes	NA	Yes	Yes	No
238	10	5	NA	Yes	Yes	Yes	Yes	No
239	10	5	NA	Yes	NA	Yes	Yes	No
240	1	1	NA	NO	NA	Yes	No	No
241-301	10	5	NA	Yes	NA	Yes	Yes	No

[1]No corrective action is taken except to return to execution. For boundary alignment, the corrective action is part of the support for misalignment. For divide check, the contents of the result register are not altered.

[2]If a print control character is not supplied, the overflow line is not shifted to incorporate the print control character. Thus, if the device is tape, the data is intact, but if the device is a printer, the first character of the overflow line is not printed and is treated as the print control. Unless the user has planned the overflow, the first character would be random and thus the overflow print line control can be any of the possible ones. It is suggested that when the device is a printer, the option be changed to single space supplied.

[3]Corrective action consists of return to execution for SLITE.

[4]It is not considered an error if the END parameter is present in a READ statement. No message or traceback is printed and the error count is not altered.

[5]For an I/O error, the buffer may have been partially filled or not filled at all when the error was detected. Thus, the buffer contents could be blank when printed. When an ERR parameter is specified in a READ statement, it is honored even though the error occurrence is greater than the amount allowed.

[6]The count field does not necessarily mean that up to 10 missing DD cards will be detected in a single debugging run, since a single WRITE performed in a loop could cause 10 occurrences of the message for the same missing DD card.

Table 21. Corrective Action After Error Occurrence

Error Code	Parameters Passed to User	Standard Corrective Action	User-Supplied Corrective Action
211	A,B,C	Treat format field containing C as end of FORMAT statement	(a) If compiled FORMAT statement, put hexadecimal equivalent of character in C (see Note 1). (b) If variable format, move EBCDIC character into C (See Note 1)
212	A,B,D	Input: Ignore remainder of I/O list. Output: Continue by starting new output record. Supply carriage control character if required by Option Table.	See Note 2
213	A,B,D	Ignore remainder of I/O list	See Note 2
214	A,B,D	Input: Ignore remainder of I/O list. Output: If unformatted write initially requested changed record format to VS (or VBS). If formatted write initially requested, ignore I/O.	If user correction is requested, the remainder of the I/O list is ignored.
215	A,B,E	Substitute zero for the invalid character.	The character placed in E will be substituted for the invalid character I/O operations may not be performed. (see Note 1)
217	A,B,D	Increment FORTRAN sequence number and read next file	See Note 2
218†	A,B,D,F	Ignore remainder of I/O list	See Note 2
219-224	A,B,D	Ignore remainder of I/O list	See Note 2
225	A,B,E	Substitute zero for the invalid character	The character placed in E will be substituted for the invalid character (see Note 1)
231	A,B,D	Ignore remainder of I/O list	See Note 2
232	A,B,D,G	Ignore remainder of I/O list	See Note 2
233	A,B,D	Change record number to list maximum allowed (32,000)	See Note 2
234-236	A,B,D	Ignore remainder of I/O list	See Note 2
237	A,B,D,F	Ignore remainder of I/O list	See Note 2

Meanings:

A - Address of return code field (Integer*4)
B - Address of error number (Integer*4)
C - Address of invalid format character (Logical*1)
D - Address of data set reference number (Integer*4)
E - Address of invalid character (Logical*1)
F - Address of DECB
G - Address of record number requested (Integer *4)

Notes:

1. Alternatively, the user can set the return code to 0, thus requesting a standard corrective action.

2. The user can do anything he wishes except perform another I/O operation - i.e., issue a READ, WRITE, BACKSPACE, END FILE, REWIND, PAUSE, PDUMP, DBUG, or ERRTRA. On return to the Library, the remainder of the I/O request will be ignored.

†If error condition 218 (I/O error detected) occurs while error messages are being written on the object error data set, the message is written on the console and the job is terminated.

If no DD card has been supplied for the object error unit, error message IHC219I is written on the SYSOUT data set and the job is terminated.

Table 22. Corrective Action After Mathematical Subroutines Error Occurrence (Part 1 of 3)

Error Code	FORTRAN Reference	Invalid Argument Range	Options: Standard Corrective Action	Options: User-Supplied Corrective Action (See Note 1)
216	CALL SLITE (I)	I>4	The call is treated as a no operation	I
216	CALL SLITET (I,J)	I>4	J=2	I
241	K=I**J	I=0, J≤0	K=0	I,J
242	Y=X**I	X=0, I≤0	Y=0	X,I
243	DA=D**I	D=0, I≤0	DA=0	D,I
244	XA=X**Y	X=0, Y≤0	XA=0	X,Y
245	DA=D**DB	D=0, DB≤0	DA=0	D,DB
246	CA=C**I	C=0+0i, I≤0	CA=0+0i	C,I
247	CDA=CD*I	C=0+0i,I≤0	CA=0+0i	CD,I
251	Y=SQRT (X)	X<0	$Y=\lvert X\rvert^{1/2}$	X
252	Y=EXP (X)	X>174.673	Y=*	X
253	Y=ALOG (X)	X=0 X<0	Y=-* $Y=\log_{10}\lvert X\rvert$	X X
	Y=ALOG10 (X)	X=0 x<0	Y=-* $y=\log_{10}\lvert x\rvert$	X x
254	Y=COS (X) Y=SIN (X)	$\lvert X\rvert \geq 2^{18}*\pi$	$Y=\sqrt{2}/2$	X
255	Y=ATAN2 (X,XA)	X=0, XA=0	Y=0	X,XA
256	Y=SINH (X) Y=COSH (X)	$\lvert X\rvert \geq 174.673$	Y=*	X
257	Y=ARSIN (X) Y=ARCOS (X)	$\lvert X\rvert >1$	Y=* IF X>1.0; $ARSIN(X)=\frac{\pi}{2}$ IF X<-1.0; $ARSIN(X)=-\frac{\pi}{2}$ IF X>1.0; ARCOS(X)=0 IF X<-1.0; $ARCOS(X)=\pi$	X
258	Y=TAN (X) Y=COTAN (X)	$\lvert X\rvert \geq (2^{18})*\pi$	Y=1	X
259	Y=TAN (X)	X is too close to an odd multiple of $\frac{\pi}{2}$	y=*	X

Variable	Type
I,J	Variables of INTEGER*4
X,XA,Y	Variables of REAL*4
D,DA,DB	Variables of REAL*8
C,CA	Variables of COMPLEX*8
Z, X_1, X_2	Complex variables to be given the length of the functioned argument when they appear
CD	Variables of COMPLEX*16

Notes:
1. The user-supplied answer is obtained by recomputation of the function using the value set by the user routine for the parameters listed.
2. The largest number that can be represented in floating point is indicated above by *.

Table 22. Corrective Action After Mathematical Subroutines Error Occurrence (Part 2 of 3)

Error Code	FORTRAN Reference	Invalid Argument Range	Options: Standard Corrective Action	Options: User-Supplied Corrective Action (See Note 1)
	Y=COTAN (X)	X is too close to a multiple of π	Y=*	X
261	DA=DSQRT (D)	D<0	DA=$\|D\|^{1/2}$	D
262	DA=DEXP (D)	D>174.673	DA=*	D
263	DA=DLOG (D)	D=0 D<0	DA=-* DA=log|X|	D
	DA=DLOG10 (D)	D=0 D<0	DA=-* DA=$\log_{10}\|X\|$	D
264	DA=DSIN (D) DA=DCOS (D)	$\|D\| \geq 2^{50} * \pi$	DA=$\sqrt{2}/2$	D
265	DA=DATAN2(D,DB)	D=0,DB=0	DA=0	D,DB
266	DA=DSINH (D) DA=DCOSH (D)	$\|D\| \geq 174.673$	DA=*	D
267	DA=DARSIN (D) DA=DARCOS (D)	$\|D\| > 1$	DA=0	D
268	DA=DTAN (D) DA=DCOTAN (D)	$\|X\| \geq 2^{50} * \pi$	DA=1	D
269	DA=DTAN (D)	D is too close to an odd $\frac{\pi}{2}$ multiple of	DA=*	D
	DA=DCOTAN (D)	D is too close to a multiple of π	DA=*	D
For errors 271 through 275, $C=X_1+iX_2$				
271	Z=CEXP (C)	$X_1 > 174.673$	Z=*(COS X_2+ SIN X_2)	C
272	Z=CEXP (C)	$\|X_2\| \geq 2^{18} * \pi$	Z=0+0i	C
273	Z=CLOG (C)	C=0+0i	z=-*+0i	C

Variable	Type
I,J	Variables of INTEGER*4
X,XA,Y	Variables of REAL*4
D,DA,DB	Variables of REAL*8
C,CA	Variables of COMPLEX*8
Z, X_1, X_2	Complex variables to be given the length of the functioned argument when they appear
CD	Variables of COMPLEX*16

Notes:
1. The user-supplied answer is obtained by recomputation of the function using the value set by the user routine for the parameters listed.
2. The largest number that can be represented in floating point is indicated above by *.

ıble 22. Corrective Action After Mathematical Subroutines Error Occurrence (Part 3 of 3)

Error Code	FORTRAN Reference	Invalid Argument Range	Options: Standard Corrective Action	Options: User-Supplied Corrective Action (See Note 1)
274	Z=CSIN (C) Z=CCOS (C)	$\|X_1\| \geq 2^{18} * \pi$	Z=0+0i	C
275	Z=CSIN (C)	$X_2 > 174.673$	$Z = \frac{*}{2}(\text{SIN } X_1 + i\text{COS } X_1)$	C
	Z=CCOS (C)		$Z = \frac{*}{2}(\text{COS } X_1 - i\text{SIN } X_1)$	C
	Z=CSIN (C)	$X_2 < -174.673$	$Z = \frac{*}{2}(\text{SIN } X_1 - i\text{COS } X_1)$	C
	Z=CCOS (C)		$Z = \frac{*}{2}(\text{COS } X_1 + i\text{SIN } X_1)$	C
For errors 281 through 285, $CD = X_1 + iX_2$				
281	Z=CDEXP (CD)	$X_1 > 174.673$	$Z = *(\text{COS } X_2 + i\text{SIN } X_2)$	CD
282	Z=CDEXP (CD)	$\|X_2\| \geq 2^{50} * \pi$	Z=0+0i	CD
283	Z=CDLOG (CD)	CD=0+0i	Z=-*+0i	CD
284	Z=CDSIN (CD) Z=CDCOS (CD)	$\|X_1\| \geq 2^{50} * \pi$	Z=0+0i	CD
285	Z=CDSIN (CD)	$X_2 > 174.673$	$Z = \frac{*}{2}(\text{SIN } X_1 + i\text{COS } X_1)$	CD
	Z=CDCOS (CD)		$Z = \frac{*}{2}(\text{COS } X_1 - i\text{SIN } X_1)$	CD
	Z=CDSIN (CD)	$X_2 < -174.673$	$Z = \frac{*}{2}(\text{SIN } X_1 - i\text{COS } X_1)$	CD
	Z=CDCOS (CD)		$Z = \frac{*}{2}(\text{COS } X_1 + i\text{SIN } X_1)$	CD
290	Y=GAMMA (X)	$X \leq 2^{-252}$ or $X \geq 57.5744$	Y=*	X
291	Y=ALGAMA (X)	$X \leq 0$ or $X \geq 4.2937 * 10^{73}$	Y=*	X
300	DA=DGAMMA (D)	$D \leq 2^{-252}$ or $D \geq 57.5774$	DA=*	D
301	DA=DLGAMA (D)	$D \leq 0$ or $D \geq 4.2937 * 10^{73}$	DA=*	D

Variable	Type
I,J	Variables of INTEGER*4
X,XA,Y	Variables of REAL*4
D,DA,DB	Variables of REAL*8
C,CA	Variables of COMPLEX*8
Z, X_1, X_2	Complex variables to be given the length of the functioned argument when they appear
CD	Variables of COMPLEX*16

Notes:
1. The user-supplied answer is obtained by recomputation of the function using the value set by the user routine for the parameters listed.
2. The largest number that can be represented in floating point is indicated above by *.

Table 23. Corrective Action After Program Interrupt Occurrence

Program Interrupt Messages			Options	
Error Code	Parameters Passed to User Exit	Reason for Interrupt[1]	Standard Corrective Action	User-Supplied Corrective Action
207	D,I	Exponent overflow (Interrupt Code 12)	Result register set to the largest possible floating point number. The sign of the result register is not altered.	User may alter D.[2]
208	D,I	Exponent underflow (Interrupt Code 13)	The result register is set to zero.	User may alter D.[2]
209	D,I[4]	Divide check, Integer divide (interrupt Code 9), Decimal divide (Interrupt Code 11), Floating point divide (Interrupt Code 15).[3]	There is no standard fixup. Result registers are not touched.	See Note 5.
210	None	Specification interrupt (Interrupt Code 6) occurs for boundary misalignment. Other interrupts occur during boundary alignment adjustment. They will be shown with this error code and the PSW portion of the message will identify the interrupt.	No special corrective action other than correcting boundary misalignments.	See Note 5.

Variable	Type	Description
D	A variable REAL*8	This variable contains the contents of the result register after the interrupt.
I	A variable INTEGER*4	This variable contains the "exponent" as an integer value for the number in D. It may be used to determine the amount of the underflow or overflow. The value in I is not the true exponent, but what was left in the exponent field of a floating point number after the interrupt.

[1]A program interrupt occurs asynchronously. Interrupts are described in IBM System/360 Operating System: Principles of Operation, Order No. GA22-6821.

[2]The user exit routine may supply an alternate answer for the setting of the result register. This is accomplished by placing a value for D in the user-exit routine. Although the interrupt may be caused by a long or short floating-point operation, the user-exit routine need not be concerned with this. The user-exit routine should always set a REAL*8 variable and the FORTRAN library will load short or long depending upon the floating-point operation that caused the interrupt.

[3]For floating-point divide check, the contents of the result register is shown in the message.

[4]For integer and decimal divide checks, no parameters are passed to the user exit routines.

[5]The user-exit routine does not have the ability to change result registers after a divide check. The boundary alignment adjustments are informational messages and there is nothing to alter before execution continues.

```
//OPTAB  JOB   1,'SAMPLE MACRO',MSGLEVEL=1   CREATE IHCUOPT
//VER1   EXEC  ASMFC,PARM.ASM=NODECK
//ASM.SYSIN  DD  *
         MACRO
         PREFACE  &ADENT,&ADJST,&SETENT
.*   THIS MACRO GENERATES THE PREFACE TO THE OPTION TABLE AND SETS
.*   GLOBALS FOR SUBSEQUENT CALLS TO THE SETENT MACRO
.*   THE USE OF THIS MACRO GENERATES AN OPTION TABLE AS DEFINED BY IBM
.*   AND ALLOWS CHANGES TO INDIVIDUAL ERROR NUMBERS AS DESIRED, BY USE
.*   OF SETENT
         GBLA  &COUNT,&TOTAL,&SETNR
         LCLA  &A
IHCUOPT  CSECT ,
&SETNR   SETA  &SETENT
&COUNT   SETA  207            ERROR NUMBER OF FIRST ENTRY IN TABLE
&TOTAL   SETA  &ADENT+301     NUMBER OF LAST ENTRY IN TABLE
&A       SETA  &ADENT+95
         DC    F'&A'          TOTAL NUMBER OF ENTRIES IN TABLE
         DC    B'0&ADJST.000000'
         DC    AL3(0)
         MEND
         MACRO
         SETENT  &E
         GBLA  &COUNT,&TOTAL,&SETNR
         LCLA  &B
&B       SETA  1
&SETNR   SETA  &SETNR-1
.AGAIN   ANOP  START OF LOOP TO GEN ONE ENTRY IN TABLE FOR ERROR NUMBER
         AIF   (&COUNT GT &TOTAL).MEND  HAVE ALL ENTRIES BEEN CREATED
         AIF   (&B LE N'&SYSLIST).TEST
         AIF   (&SETNR EQ 0).DEFAULT
         MEXIT
.TEST    ANOP
.* IF THERE IS NO USER SUPPLIED INFO FOR THIS ERROR NO TAKE DEFAULT
         AIF   (&SYSLIST(&B,1) NE &COUNT).DEFAULT
ERR&COUNT DC   AL1(&SYSLIST(&B,2)) NUMBER OF ERRORS TO ALLOW FR SETENT
         DC    AL1(&SYSLIST(&B,3)) NO OF MSGS TO PRINT FROM SETENT
         DC    X'00'
         DC    X'&SYSLIST(&B,4)'  OPTION BITS SUPPLIED BY SETENT
         DC    F'1'
&COUNT   SETA  &COUNT+1
&B       SETA  &B+1
         AGO   .AGAIN  RETURN TO LOOP
.DEFAULT ANOP IBM DEFAULTS FOR ERRORS NOT INDICATED BY SETENT
.*   IBM SPECIAL CASES FOR MESSAGE COUNT
         AIF   (&COUNT EQ 208).UNLIM
         AIF   (&COUNT EQ 210).UNLIM
         AIF   (&COUNT EQ 215).UNLIM
         AIF   (&COUNT EQ 217).ONE
         AIF   (&COUNT EQ 230).ONE
ERR&COUNT DC   AL1(10)
.BACK1   ANOP
         DC    AL1(5)
.BACK2   ANOP
         DC    X'00'
.*   IBM SPECIAL CASES FOR OPTION BITS
         AIF   (&COUNT EQ 212).SPBITS
         AIF   (&COUNT EQ 215).SPBITS
         AIF   (&COUNT EQ 218).SPBITS
         AIF   (&COUNT EQ 221).SPBITS
         AIF   (&COUNT EQ 222).SPBITS
         AIF   (&COUNT EQ 223).SPBITS
         AIF   (&COUNT EQ 224).SPBITS
         AIF   (&COUNT EQ 225).SPBITS
         DC    X'42'
         AGO   .CONT
.SPBITS  DC    X'52'
.CONT    ANOP
         DC    F'1'
&COUNT   SETA  &COUNT+1
         AGO   .AGAIN RETURN TO LOOP
.UNLIM   ANOP
ERR&COUNT DC   AL1(0)
         AIF   (&COUNT NE 210).BACK1
         DC    AL1(10)
         AGO   .BACK2
.ONE     ANOP
ERR&COUNT DC   AL1(1)
         DC    AL1(1)
         AIF   (&COUNT EQ 217).BACK2
         DC    X'00'
         DC    X'02'
         AGO   .CONT
.MEND    ANOP
         MEND
*        END OF MACRO DEFINITION
*
*        EXAMPLE OF THE USE OF THE MACRO
*
 PREFACE  50,1,2
 SETENT (220,5,2,21),(235,10,5,42),(255,2,0,4)
 SETENT  (300,56,65,3)
         END
/*
 END OF DATA
```

Figure 91. Example of Assembler Language Macro Definition Used To Generate Option Table

```
//SAMPLE JOB     1,SAMPLE, MSGLEVEL=1
//STEP1  EXEC    FORTHCLG
//FORT.SYSIN DD *
C       MAIN PROGRAM THAT USES THE SUBROUTINE DIVIDE
        COMMON E
        EXTERNAL FIXDIV
C       SET UP OPTION TABLE WITH ADDRESS OF USER EXIT
        CALL ERRSET(302,30,5,1,FIXDIV)
        E=0
C       GET VALUES TO CALL DIVIDE WITH
        READ(5,9)A,B
        IF(B) 1,2,1
2       E=1.0
1       CALL DIVIDE(A,B,C)
        WRITE(6,10)C
9       FORMAT(2E20.8)
10      FORMAT('1',E20.8)
        STOP
        END
        SUBROUTINE DIVIDE(A,B,C)
C       ROUTINE TO PERFORM THE CALCULATION C=A/B
C       IF B=0 THEN USE ERROR MESSAGE FACILITY TO SERVICE ERROR
C       PROVIDE MESSAGE TO BE PRINTED
        DIMENSION MES(4)
        DATA MES(1)/12/,MES(2)/' DIV'/,MES(3)/'302I'/,MES(4)/' B=0'/
        DATA RMAX/Z7FFFFFFF/
C       MESSAGE TO BE PRINTED IS
C       DIV302I B=0
C       ERROR CODE 302 IS AVAILABLE AND ASSIGNED TO THIS ROUTINE
C       STEP1 SAVE A,B IN LOCAL STORAGE
        D=A
        E=B
C       STEP2   TEST FOR ERROR CONDITION
100     IF(E) 1,2,1
C       NORMAL CASE -- COMPUTE FUNCTION
1       C=D/E
        RETURN
C       STEP3  ERROR DETECTED CALL ERROR MONITOR
2       CALL ERRMON(MES,IRETCD,302,D,E)
C       STEP 4   BE READY TO ACCEPT A RETURN FROM THE ERROR MONITOR
        IF(IRETCD) 5,6,5
C        IF IRETCD=0 STANDARD RESULT IS DESIRED
C       STANDARD RESULT WILL BE C=LARGEST NUMBER IF D IS NOT ZERO
C       CR C=0 IF E=0 AND D=0
6       IF(D) 7,8,7
8       C=0.0
        GO TO 9
7       C=RMAX
9       RETURN
C       USER FIX UP INDICATED. RECOMPUTE WITH NEW VALUE PLACED IN E
5       GO TO 100
        END
        SUBROUTINE FIXDIV(IRETCD,INO,A,B)
C       THIS IS A USER EXIT TO SERVE THE SUBROUTINE DIVIDE
C       THE PARAMETERS IN THE CALL MATCH THOSE USE IN THE CALL TO
C       ERRMON MADE BY SUBROUTINE DIVIDE
C     STEP1  IS ALTERNATE VALUE FOR B AVAILABLE -- MAIN PROGRAM
C       HAS SUPPLIED A NEW VALUE IN E.  IF E=0 NO NEW VALUE IS AVAILABLE
        COMMON E
        IF(E) 1,2,1
C       NEW VALUE AVAILABLE TAKE USER CORRECTION EXIT
1       B=E
        RETURN
C       NEW VALUE NOT AVAILABLE USE STANDARD FIX UP
2       IRETCD=0
        RETURN
        END
/*
//GO.SYSIN  DD *
              0.1E00           0.0E00
/*
```

Figure 92. Sample Program Using Extended Error Handling Facility

FORTRAN can be invoked by a problem program through the use of the CALL, ATTACH, or LINK macro instructions.

The program must supply to the FORTRAN compiler:

- The information usually specified in the PARM parameter of the EXEC statement.
- The ddnames of the data sets to be used during processing by the FORTRAN compiler.

Name	Operation	Operand
[name]	{LINK / ATTACH}	EP=compiler-name, PARAM=(optionaddr [,ddnameaddr]),VL=1
[name]	CALL	IEKAA00, (optionaddr [,ddnameaddr]),VL

compiler-name
specifies the program name of the compiler to be invoked. IEYFORT is spec- ified for FORTRAN IV (G); IEKAA00, for FORTRAN IV (H).

optionaddr
specifies the address of a variable length list containing information usually specified in the PARM parameter of the EXEC statement.

The option list must begin on a half-word boundary. The two high-order bytes contain a count of the number of bytes in the remainder of the list. If there are no parameters, the count must be zero. The option list is free form with each field separated by a comma. No blanks should appear in the list.

ddnameaddr
specifies the address of a variable length list containing alternate ddnames for the data sets used during FORTRAN compiler processing. This address is supplied by the invoking program. If standard ddnames are used, this operand may be omitted.

The ddname list must begin on a halfword boundary. The two high-order bytes contain a count of the number of bytes in the remainder of the list. Each name of less than eight bytes must be left justified and padded with blanks. If an alternate ddname is omitted from the list, the standard name is assumed. If the name is omitted within the list, the 8-byte entry must contain binary zeros. Names can be completely omitted only from the end of the list.

The sequence of the 8-byte entries in the ddname list is as follows:

Entry	Alternate Name
1	SYSLIN
2	00000000
3	00000000
4	00000000
5	SYSIN
6	SYSPRINT
7	SYSPUNCH
8	SYSUT1
9	SYSUT2

VL=1 or VL
specifies that the sign bit of the last fullword of the address parameter list is to be set to 1.

APPENDIX B: EXAMPLES OF JOB PROCESSING

The following examples show several methods to process load modules.

Example 1

Problem Statement: A previously created data set SCIENCE.MATH.MATRICES contains a set of 80 matrices. Each matrix is an array containing real*4 variables. The size of the matrices varies from 2x2 to 25x25; the average size is 10x10. The matrices are inverted by a load module MATINV in the library MATPROGS. Each inverted matrix is written (assume FORMAT control) as a single record on the data set SCIENCE.MATH.INVMATRS. The first variable in each record denotes the size of the matrix.

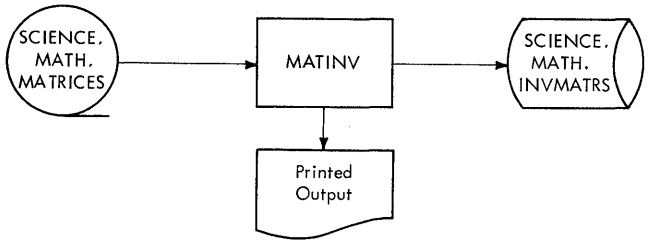


Figure 93. Input/Output Flow for Example 1

The I/O flow for the example is shown in Figure 93. The job control statements used to define this job are shown in Figure 94.

Explanation: The JOB statement identifies the programmer as JOHN SMITH and supplies the account number 537. Control statements and control statement error messages are written in the SYSOUT data set.

The JOBLIB DD statement indicates that the private library MATPROGS is concatenated with the system library.

The EXEC statement indicates that the load module MATINV is executed.

DD statement FT08F001 identifies the input data set, SCIENCE.MATH.MATRICES. (Data set reference number 8 is used to read the input data set.) Because this data set has been previously created and cataloged, no information other than the data set name and disposition has to be supplied.

DD statement FT10F001 identifies the printed output. (Data set reference number 10 is used for printed output.)

DD statement FT04F001 defines the output data set. (Data set reference number 4 is used to write the data set containing the inverted matrices.) Because the data set is created and cataloged in this job step, a complete data set specification is supplied.

The DSNAME parameter indicates that the data set is named SCIENCE.MATH.INVMATRS. The DISP parameter indicates that the data

Sample Coding Form

```
1-10      11-20     21-30     31-40     41-50     51-60     61-70     71-80
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901
//INVERT JOB 537,JOHNSMITH,MSGLEVEL=1
//JOBLIB DD DSNAME=MATPROGS,DISP=OLD
//INVERT EXEC PGM=MATINV
//FT08F001 DD DSNAME=SCIENCE.MATH.MATRICES,DISP=OLD
//FT10F001 DD SYSOUT=A
//FT04F001 DD DSNAME=SCIENCE.MATH.INVMATRS,                              1
//          DISP=(NEW,CATLG),UNIT=DACLASS,VOLUME=SER=1089W,              2
//          SPACE=(408,(80,9),RLSE,CONTIG,ROUND),SEP=FT08F001,           3
//          DCB=(RECFM=VB,LRECL=2504,BLKSIZE=2508)
```

Figure 94. Job Control Statements for Example 1

set is new and is to be cataloged. The SPACE parameter indicates that space is reserved for 80 records, 408 characters long (80 matrices of average size). When space is exhausted, space for 9 more records is allocated. The space is contiguous; any unused space is released, and allocation begins and ends on cylinder boundaries.

The DCB parameter indicates variable-length records, because the size of matrices vary. The record length is specified as 2504, the maximum size of a variable-length record. (The maximum size of a record in this data set is the maximum number of elements (625) in any matrix multiplied by the number of bytes (4) allocated for an element, plus 4 for the segment control word (SCW) that indicates the count of the number of data bytes contained in the record.) The buffer length is specified as 2508 (the 4 bytes are for the block control word (BCW) that contains the length of the block).

The SEP parameter indicates that read and write operations should take place on different channels.

Example 2

Problem Statement: A previously created data set RAWDATA contains raw data from a test firing. A load module PROGRD refines data by comparing the data set RAWDATA against a forecasted result, PROJDATA. The output of PROGRD is a data set &REFDATA, which contains the refined data.

The refined data is used to develop values from which graphs and reports can be generated. The load module ANALYZ contains a series of equations and uses a previously created and cataloged data set PARAMS which contains the parameters for these equations. ANALYZ creates a data set &VALUES, which contains intermediate values.

These values are used as input to the load module REPORT, which prints graphs and reports of the data gathered from the test firing. Figure 1 in the "Introduction" shows the I/O flow for the example. Figure 95 shows the job control statements used to process this job.

The load modules PROGRD, ANALYZ, and REPORT are contained in the private library FIRING.

Explanation: The JOB statement indicates the programmer's name, JOHN SMITH, and specifies that control statements and control statement error messages are written in the SYSOUT data set.

The JOBLIB DD statement indicates that the private library FIRING is concatenated with the system library.

The EXEC statement STEP1 defines the first job step in the job and indicates that the load module PROGRD is executed.

The DD statements FT10F001 and FT11F001 identify the data sets containing raw data (RAWDATA) and the forecasted result (PROJDATA), respectively.

DD statement FT12F001 defines a temporary data set, &REFDATA, created for input to the second step. (In the load module, data set reference number 12 is used to write &REFDATA.) The DISP parameter indicates that a data set is new and is passed. The data set is written using the device class TAPECLS. The VOLUME parameter indicates that the volume identified by serial number 2107 is used for this data set. The DCB parameter indicates that the volume is written using high density; the records are fixed-length with FORMAT control and the buffer length is 400.

The EXEC statement STEP2 defines the second job step in the job and indicates that the load module ANALYZ is executed.

DD statement FT17F001 identifies the data set which contains refined data. The DISP parameter indicates that the data set is deleted after execution of this job step. The DD statement FT18F001 identifies the previously created and cataloged data set PARAMS.

DD statement FT20F001 defines the temporary data set &VALUES containing the intermediate values. The DISP parameter indicates that the data set is created in this step, and that it is passed to the next job step. The data set is written on volume 2108 using one of the devices assigned to the class TAPECLS. The DCB parameter indicates high density and fixed-length blocked records (written under FORMAT control). Each record is 204 characters long.

The EXEC statement STEP3 defines the third job step and indicates that the load module REPORT is executed. DD statement FT08F001 identifies the data set containing intermediate values.

DD statement FT06F001 indicates that the data set reference number 06 is used to print the reports and graphs for job step three.

Sample Coding Form

```
1-10      11-20     21-30     31-40     41-50     51-60     61-70     71-80
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
//TESTFIRE JOB ,JOHNSMITH,MSGLEVEL=1
//JOBLIB DD DSNAME=FIRING,DISP=(OLD,PASS)
//STEP1 EXEC PGM=PROGRD
//FT10F001 DD DSNAME=RAWDATA,DISP=OLD
//FT11F001 DD DSNAME=PROJDATA,DISP=OLD
//FT12F001 DD DSNAME=&REFDATA,DISP=(NEW,PASS),UNIT=TAPECLS,                1
//              VOLUME=(,RETAIN,SER=2107),                                 2
//              DCB=(DEN=2,RECFM=F,BLKSIZE=400)
//STEP2 EXEC PGM=ANALYZ
//FT17F001 DD DSNAME=*.STEP1.FT12F001,DISP=OLD
//FT18F001 DD DSNAME=PARAMS,DISP=OLD
//FT20F001 DD DSNAME=&VALUES,DISP=(NEW,PASS),UNIT=TAPECLS,                  1
//              DCB=(DEN=2,RECFM=F,BLKSIZE=204),VOLUME=SER=2108
//STEP3 EXEC PGM=REPORT
//FT08F001 DD DSNAME=*.STEP2.FT20F001,DISP=OLD
//FT06F001 DD UNIT=PRINTER
```

Figure 95. Job Control Statements for Example 2

Example 3

A data set has been created that contains master records for an index of stars. Each star is identified by a unique 6-digit star identification number. Each star is assigned a record position in the data set by truncating the last two digits in the star identification number. Because synonyms arise, records are chained.

Problem Statement: Figure 96 shows a block diagram illustrating the logic for this problem.

A card data set read from the input stream is used to update the star master data set. Each record (detail record) in this data set contains:

1. The star identification field of the star master record that the detail record is used to update.
2. Six variables that are to be used to update the star master.

The following conventions must be observed in processing this data set:

1. The star master record that contains the record location counter pointing to space reserved for chained records is assigned to record location 1.
2. A zero in the chain variable indicates that the end of a chain has been reached.
3. The first variable in each star master record is the star identification field; the second variable in each star master is the chain variable.
4. Each record contains six other variables that contain information about that star.

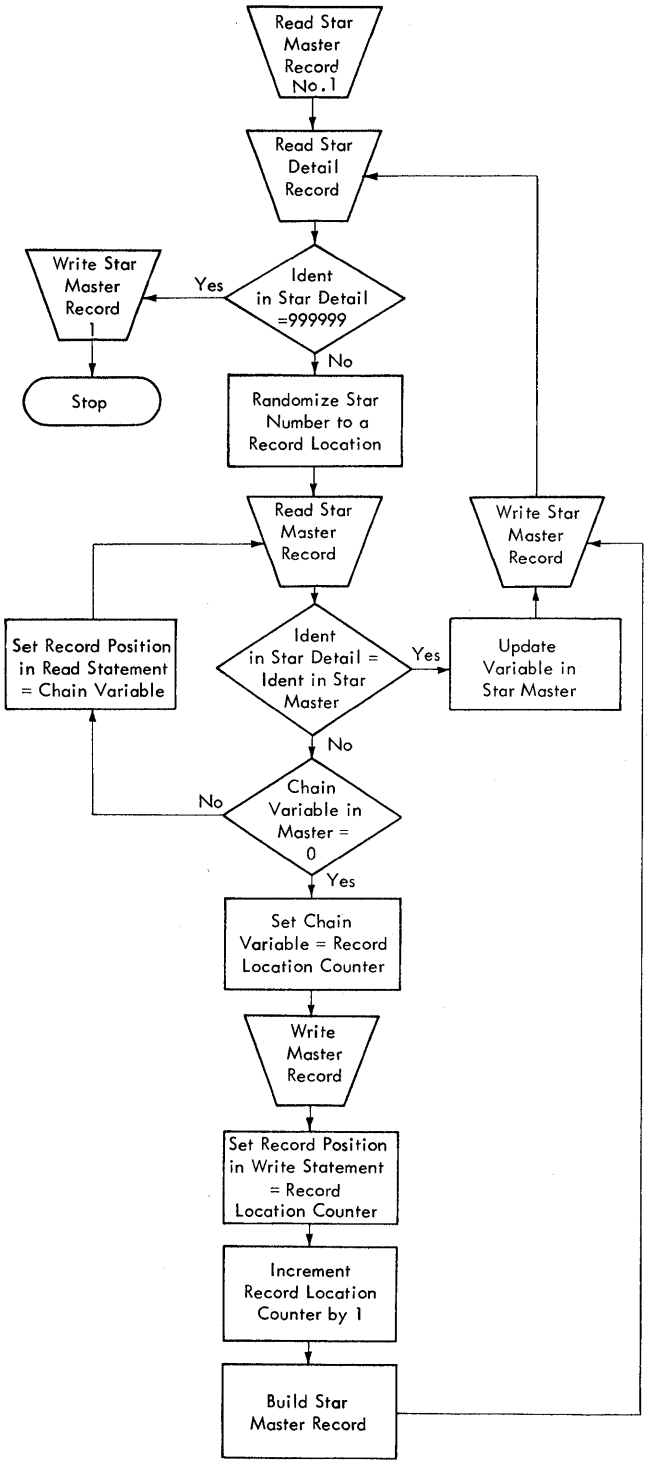


Figure 96. Block Diagram for Example 3

When a star detail record is read, its identification field is randomized, and the appropriate star master record is read. If the correct star master record is found, the record is to be updated. If a star master is not found, then a star master record is to be created for that star.

The last record in the star detail data set contains a star identification number 999999 which indicates that processing the star detail data set is completed.

Explanation: Figure 96 is similar to the diagram shown in Figure 60 except Figure 96 includes blocks that describe updating variables in master records already present in the data set. (Figure 60 includes blocks describing certain operations that must be performed when a direct access data set is first written.) Also, Figure 96 is adapted to Example 3, while Figure 60 is more general. Figure 98 shows the FORTRAN coding for this program.

The star master record that contains the record counter is read, placing the record location counter in LOCREC. Whenever a detail record is read, the identification variable is checked to determine if the end of the detail data set has been reached. The star detail records contain the variables A, B, C, D, E, and F.

The identification number in the detail record is randomized and the result is placed in the variable NOREC, which is used to read a master record. The master record contains the star identification number (IDSTRM), a chain record location (ICHAIN), and six variables (T, U, V, X, Y, and Z) which are to be updated by the variables in the star detail records. IDSTRM and IDSTRD are compared to see if the correct star master is found. If it is not found, then the variables containing the chain record numbers are followed until the correct star master is found or a new star master is created.

Job Control Statements: The program shown in Figure 98 is compiled and link edited, placing the load module in the PDS STARPGMS and assigning the load module the name UPDATE. The data set that contains the star master records was cataloged and assigned the name STARMSTR when it was created. Figure 97 shows the job control statements needed to execute the module UPDATE.

Sample Coding Form

```
1-10      11-20     21-30     31-40     41-50     51-60     61-70     71-80
1234567890123456789012345678901234567890123456789012345678901234567890
//STARDAUP JOB 323,'J.ASTRONOMER',MSGLEVEL=1
//JOBLIB DD DSNAME=STARPGMS,DISP=OLD
// EXEC PGM=UPDATE
//FT07F001 DD DSNAME=STARMSTR,DISP=OLD
//FT01F001 DD *              STAR DETAILS FOLLOW
        Star Detail Data Set
/*                 END OF STAR DETAILS
```

Figure 97. Job Control Statements for Example 3

```
      DEFINE FILE 7(12000,130,E,NEXT)
C READ RECORD CONTAINING RECORD LOCATION COUNTER
      READ(7'1,101)IDSTRM,LOCREC
C READ STAR DATA AND CHECK FOR LAST STAR DATA RECORD
26    READ(1,102)IDSTRD,A,B,C,D,E,F
      IF(IDSTRD-999999)20,99,99
C RANDOMIZE IDENTIFICATION FIELD IN STAR DATA AND READ STAR MASTER
20    NOREC=IDSTRD/100
27    READ(7'NOREC,103)IDSTRM,ICHAIN,T,U,V,X,Y,Z
C IS THIS CORRECT STAR MASTER
      IF(IDSTRD-IDSTRM)21,22,21
C IS THERE A CHAIN VARIABLE
21    IF(ICHAIN)24,24,23
C NO.BEGIN CONSTRUCTING NEW MASTER AND CHAIN
C UPDATE CHAIN VARIABLE IN LAST STAR MASTER RECORD AND WRITE LAST RECORD
24    ICHAIN=LOCREC
      WRITE(7'NOREC,103)IDSTRM,ICHAIN,T,U,V,X,Y,Z
C SET RECORD NUMBER TO BEGIN CONSTRUCTION OF NEW STAR MASTER. UPDATE
C RECORD LOCATION COUNTER. BUILD NEW STAR MASTER RECORD
      NOREC=LOCREC
      LOCREC=LOCREC+1
        .
        .
        .
C GO TO WRITE STAR MASTER RECORD
      GO TO 25
C IF RECORD IS FOUND, UPDATE AND WRITE STAR MASTER
22    Z=A/B
        .
        .
        .
25    WRITE(7'NOREC,103)IDSTRM,ICHAIN,T,U,V,X,Y,Z
C GO TO READ NEXT STAR DATA RECORD
      GO TO 26
C IF CHAIN VARIABLE IN RECORD READ THE NEXT STAR MASTER IN THE CHAIN
23    NOREC=ICHAIN
      GO TO 27
C IF END OF STAR DATA,WRITE STAR MASTER CONTAING RECORD LOCATION COUNTER
99    IDSTRM=0
      WRITE(7'1,101)IDSTRM,LOCREC
      STOP 99999
101   FORMAT(I6,I4)
102   FORMAT(I6,6F10.3)
103   FORMAT(I6,I4,6F20.3)
      END
```

Figure 98. FORTRAN Coding for Example 3

APPENDIX C: ASSEMBLER LANGUAGE SUBPROGRAMS

A FORTRAN programmer can use assembler language subprograms with his FORTRAN main program. This section describes the linkage conventions that must be used by the assembler language subprogram to communicate with the FORTRAN main program. To understand this appendix, the reader must be familiar with the Assembler Language publication, Order No. GC28-6514 and the Assembler Programmer's Guide.

SUBROUTINE REFERENCES

The FORTRAN programmer can refer to a subprogram in two ways: by a CALL statement or a function reference within an arithmetic expression. For example, the statements

```
CALL MYSUB(X,Y,Z)
I=J+K+MYFUNC(L,M,N)
```

refer to a subroutine subprogram MYSUB and a function subprogram MYFUNC, respectively.

For subprogram reference, the compiler generates:

1. A contiguous argument list; the addresses of the arguments are placed in this list to make the arguments accessible to the subprogram.
2. A save area in which the subprogram can save information related to the calling program.
3. A calling sequence to pass control to the subprogram.

Argument List

The argument list contains addresses of variables, arrays, and subprogram names used as arguments. Each entry in the argument list is four bytes and is aligned on a fullword boundary. The last three bytes of each entry contain the 24-bit address of an argument. The first byte of each entry contains zeros, unless it is the last entry in the argument list. If this is the last entry, the sign bit in the entry is set to 1.

The address of the argument list is placed in general register 1 by the calling program.

Save Area

The calling program contains a save area in which the subprogram places information, such as the entry point for this program, an address to which the subprogram returns, general register contents, and addresses of save areas used by programs other than the subprogram. The amount of storage reserved by the calling program is 18 words. Figure 99 shows the layout of the save area and the contents of each word. The address of the save area is placed in general register 13.

The called subprogram does not have to save and restore floating-point registers.

Calling Sequence

A calling sequence is generated to transfer control to the subprogram. The address of the save area in the calling program is placed in general register 13. The address of the argument list is placed in general register 1, and the entry address is placed in general register 15. If there is no argument list, then general register 1 will contain zero. A branch is made to the address in register 15 and the return address is saved in general register 14. Table 24 illustrates the use of the linkage registers.

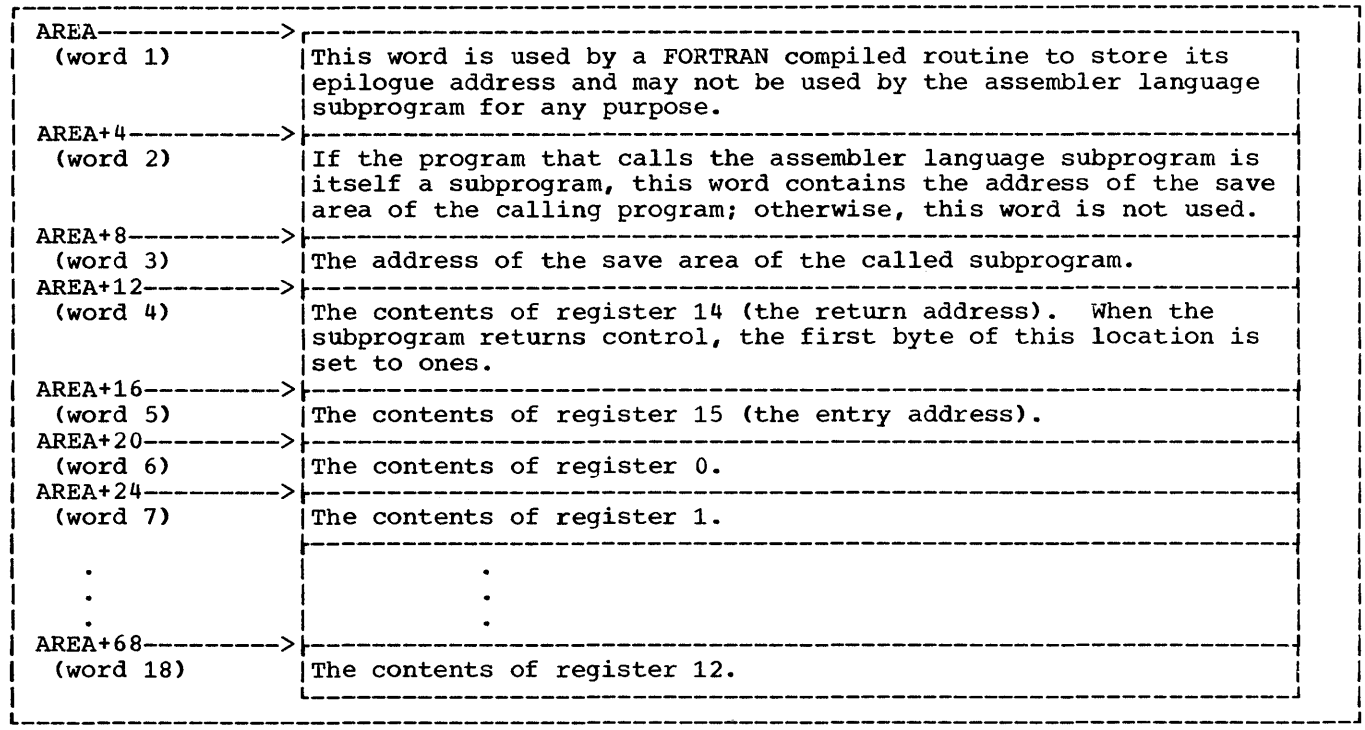


Figure 99. Save Area Layout and Word Contents

Table 24. Linkage Registers

Register Number	Register Name	Function
0	Result Register	Used for function subprograms only. The result is returned in general or floating-point register 0. However, if the result is a complex number, it is returned in floating-point registers 0 (real part) and 2 (imaginary part). Note: For subroutine subprograms, the result(s) is returned in a variable(s) passed by the programmer.
1	Argument List Register	Address of the argument list passed to the called subprogram.
2	Result Register	See Function of Register 0.
13	Save Area Register	Address of the area reserved by the calling program in which the contents of certain registers are stored by the called program.
14	Return Register	Address of the location in the calling program to which control is returned after execution of the called program.
15	Entry Point Register	Address of the entry point in the called subprogram. Note: Register 15 is also used as a condition code register, a RETURN code register, and a STOP code register. The particular values that can be contained in the register are 16 - a terminal error was detected during execution of a subprogram (an IHCxxxI message is generated) 4*i - a RETURN i statement was executed n - a STOP n statement was executed 0 - a RETURN or a STOP statement was executed

CODING THE ASSEMBLER LANGUAGE SUBPROGRAM

Two types of assembler language subprograms are possible: the first type (lowest level) assembler subprogram does not call another subprogram; the second type (higher level) subprogram does call another subprogram.

Coding a Lowest Level Assembler Language Subprogram

For the lowest level assembler language subprogram, the linkage instructions must include:

1. An assembler instruction that names an entry point for the subprogram. The entry point name must conform to FORTRAN naming conventions.
2. An instruction(s) to save any general registers used by the subprogram in the save area reserved by the calling program. (The contents of linkage registers 0 and 1 need not be saved.)
3. An instruction(s) to restore the "saved" registers before returning control to the calling program.
4. An instruction that sets the first byte in the fourth word of the save area to ones, indicating that control is returned to the calling program.
5. An instruction that returns control to the calling program.

Figure 100 shows the linkage conventions for an assembler language subprogram that does not call another subprogram. In addition to these conventions, the assembler program must provide a method to transfer arguments from the calling program and return the arguments to the calling program.

Higher Level Assembler Language Subprogram

A higher level assembler subprogram must include the same linkage instructions as the lowest level subprogram, but because the higher level subprogram calls another subprogram, it must simulate a FORTRAN subprogram reference statement and include:

1. A save area and additional instructions to insert entries into its save area.
2. A calling sequence and a parameter list for the subprogram that the higher level subprogram calls.
3. An assembler instruction that indicates an external reference to the subprogram called by the higher level subprogram.
4. Additional instructions in the return routine to retrieve entries in the save area.

Note: If an assembler language main program calls one or more FORTRAN subprograms, the following instructions must be executed once (and only once) in

```
Name      Oper.  Operand        Comments
deckname  START  0
          BC     15,m+1+4(15)   BRANCH AROUND CONSTANTS IN CALLING SEQUENCE
          DC     X'm'           name IS THE SUBPGM NAME OF LENGTH m.  m MUST BE AN ODD
                                INTEGER TO INSURE THAT THE PROGRAM
          DC     CLm'name'      STARTS ON A HALF-WORD BOUNDARY.  THE NAME CAN BE PADDED
*                                WITH BLANKS.
          STM    14,R,12(13)    THE CONTENTS OF REGISTERS 14, 15, AND 0 THROUGH R ARE
*                                STORED IN THE SAVE AREA OF THE CALLING PROGRAM. R IS ANY
*                                NUMBER FROM 2 THROUGH 12.
          BALR   B,0            ESTABLISH BASE REGISTER (2≤B≤12)
          USING  *,B
          user   written source statements
                   .
                   .
                   .
          LM     2,R,28(13)     RESTORE REGISTERS
          MVI    12(13),X'FF'   INDICATE CONTROL RETURNED TO CALLING PROGRAM
          BCR    15,14          RETURN TO CALLING PROGRAM
```

Figure 100. Linkage Conventions for Lowest Level Subprogram

the assembler language program before the first FORTRAN subprogram is called:

```
L    15,=V(IBCOM#)
BAL  14,64(15)
```

These instructions cause initialization of return coding interruption exceptions, and opening of the error message data set. If this is not done and the FORTRAN subprogram terminates either with a STOP statement or because of an execution-time error, the data sets opened by FORTRAN are not closed and the result of the termination cannot be predicted. Register 13 must contain the address of the save area that contains the registers to be restored upon termination of the FORTRAN subprogram. Specifically, the fifth word of the save area must contain the address which error-handling routines are to consider the program entry point. If control is to return to the assembler language subprogram, then register 13 contains the address of its save area. If control is to return to the operating system, then register 13 contains the address of its save area.

Figure 101 shows the linkage conventions for an assembler subprogram that calls another assembler subprogram.

```
|Name      |Oper. |Operand        Comments
|----------|------|-------------------------------------------------------------
|deckname  |START |0
|          |EXTRN |name2          NAME OF THE SUBPROGRAM CALLED BY THIS SUBPROGRAM
|          |BC    |15,m+1+4(15)   m IS THE NUMBER OF CHARACTERS IN THE SUBPROGRAM NAME.
|          |DC    |X'm'
|          |DC    |CLm'name1      NAME OF THIS SUBPROGRAM.
|*         |      |SAVE ROUTINE
|          |STM   |14,R,12(13)    THE CONTENTS OF REGISTERS 14, 15, AND 0 THROUGH R ARE
|*         |      |               STORED IN THE SAVE AREA OF THE CALLING PROGRAM. R IS
|*         |      |               ANY NUMBER FROM 2 THROUGH 12.
|          |BALR  |B,0            ESTABLISH BASE REGISTER
|          |USING |*,B
|          |LR    |Q,13           LOADS REGISTER 13, WHICH POINTS TO THE SAVE AREA OF THE
|*         |      |               CALLING PROGRAM, INTO ANY GENERAL REGISTER, Q, EXCEPT 0,
|*         |      |               1, 13, AND B.
|          |LA    |13,AREA        LOADS THE ADDRESS OF THIS PROGRAM'S SAVE AREA INTO
|*         |      |               REGISTER 13.
|          |ST    |13,8(0,Q)      STORES THE ADDRESS OF THIS PROGRAM'S SAVE AREA INTO THE
|*         |      |               CALLING PROGRAM'S SAVE AREA
|          |ST    |Q,4(0,13)      STORES THE ADDRESS OF THE PREVIOUS SAVE AREA (THE SAVE
|*         |      |               AREA OF THE CALLING PROGRAM) INTO WORD 2 OF THIS PRO-
|*         |      |               GRAM'S SAVE AREA
|          |BC    |15,prob1
|AREA      |DS    |18F            RESERVES 18 WORDS FOR THE SAVE AREA
|*         |      |END OF SAVE ROUTINE
|          |      |
|prob1     |user  |written program statements
|          |      |
|*         |      |CALLING SEQUENCE
|          |LA    |1,ARGLIST      LOAD ADDRESS OF ARGUMENT LIST
|          |L     |15,ADCON
|          |BALR  |14,15
|          |more  |user written program statements
|*         |      |RETURN ROUTINE
|          |L     |13,AREA+4      LOADS THE ADDRESS OF THE PREVIOUS SAVE AREA BACK INTO
|*         |      |               REGISTER 13
|          |LM    |2,R,28(13)
|          |L     |14,12(13)      LOADS THE RETURN ADDRESS INTO REGISTER 14.
|          |MVI   |12(13),X'FF'
|          |BCR   |15,14          RETURN TO CALLING PROGRAM
|*         |      |END OF RETURN ROUTINE
|ADCON     |DC    |A(name2)
|*         |      |ARGUMENT LIST
|ARGLIST   |DC    |AL4(arg1)      ADDRESS OF FIRST ARGUMENT
|          |  .   |
|          |  .   |
|          |  .   |
|          |DC    |X'80'          INDICATE LAST ARGUMENT IN ARGUMENT LIST
|          |DC    |AL3(argn)      ADDRESS OF LAST ARGUMENT
```

Figure 101. Linkage Conventions for Higher Level Subprogram

Inline Argument List

The assembler programmer may establish an inline argument list instead of out-of-line list. In this case, he may substitute the calling sequence and argument list shown in Figure 102 for that shown in Figure 101. The 'isn' entry following the inline argument list represents a user-supplied internal statement number corresponding to the return point in the calling routine.

```
ADCON    DC     A(prob1)
           .
           .
           .
         LA     14, RETURN
         L      15, ADCON
         CNOP   2,4
         BALR   1,15
         DC     AL4(arg1)
         DC     AL4(arg2)
           .
           .
           .
         DC     X'80'
         DC     AL3(argn)
RETURN   BC     0,X'isn'
```

Figure 102. Inline Argument List

Sharing Data in COMMON

Both named and blank COMMON in a FORTRAN IV program can be referred to by an assembler language subprogram. To refer to named COMMON, the V-type address constant

```
name DC V(name of COMMON)
```

is used.

If a FORTRAN program has a blank COMMON area and blank COMMON is also defined (by the COM instruction) in an assembler language subprogram, only one blank COMMON area is generated for the output load module. Data in this blank COMMON is accessible to both programs.

To refer to blank COMMON, the following linkage may be specified:

```
     COM
name DS  0F
     .                    .
     .                    .
     .                    .
-------------->   cname   CSECT
                          .
     L   11,=A(name)      .
     USING name,11        .
```

RETRIEVING ARGUMENTS FROM THE ARGUMENT LIST

The argument list contains addresses for the arguments passed to a subprogram. The order of these addresses is the same as the order specified for the arguments in the calling statement in the main program. The address for the argument list is placed in register 1. For example, when the statement

```
CALL MYSUB(A,B,C)
```

is compiled, the following argument list is generated.

00000000	address for A
00000000	address for B
10000000	address for C

For purposes of discussion, A is a real*8 variable, B is a subprogram name, and C is an array.

The address of a variable in the calling program is placed in the argument list. The following instructions in an assembler language subprogram can be used to move the real*8 variable A to location VAR in the subprogram.

```
L      Q,0(1)
MVC    VAR(8),0(Q)
```

where
Q is any general register except 0.

For a subprogram reference, an address of a storage location is placed in the argument list. The address at this storage location is the entry point to the subprogram. The following instructions can be used to enter subprogram B from the subprogram to which B is passed as an argument.

```
L      Q,4(1)
L      15,0(Q)
BALR   14,15
```

where
Q is any general register except 0.

For an array, the address of the first variable in the array is placed in the argument list. An array [for example, a three-dimensional array C(3,2,2)] appears in this format in main storage.

```
  C(1,1,1)  C(2,1,1)  C(3,1,1)  C(1,2,1)--┐
┌-------------------------------------------┘
└-C(2,2,1)  C(3,2,1)  C(1,1,2)  C(2,1,2)--┐
┌-------------------------------------------┘
└-C(3,1,2)  C(1,2,2)  C(2,2,2)  C(3,2,2)
```

Table 25 shows the general subscript format for arrays of 1, 2, and 3 dimensions.

Table 25. Dimension and Subscript Format

Array A	Subscript Format
A(D1)	A(S1)
A(D1,D2)	A(S1,S2)
A(D1,D2,D3)	A(S1,S2,S3)
D1, D2, and D3 are integer constants used in the DIMENSION statement. S1, S2, and S3 are subscripts used with subscripted variables.	

The address of the first variable in the array is placed in the argument list. To retrieve any other variables in the array, the displacement of the variable, that is, the distance of a variable from the first variable in the array, must be calculated. The formulas for computing the displacement (DISPLC) of a variable for one, two, and three dimensional arrays are

```
DISPLC=(S1-1)*L
DISPLC=(S1-1)*L+(S2-1)*D1*L
DISPLC=(S1-1)*L+(S2-1)*D1*L+(S3-1)*D2*D1*L
```

where
L is the length of each variable in this array.

For example, the variable C(2,1,2) in the main program is to be moved to a location ARVAR in the subprogram. Using the formula for displacement of integer variables in a three-dimensional array, the displacement (DISP) is calculated to be 28. The following instructions can be used to move the variable,

```
L    Q,8(1)
L    R,DISP
L    S,0(Q,R)
ST   S,ARVAR
```

where
Q and R are any general register except 0.
S is any general register. Q and R cannot be general register 0.

Example: An assembler language subprogram is to be named ADDARR, and a real variable, an array, and an integer variable are to be passed as arguments to the subprogram. The statement

```
CALL ADDARR (X,Y,J)
```

is used to call the subprogram. Figure 103 shows the linkage used in the assembler subprogram.

RETURN i in an Assembler Language Subprogram

When a statement number is an argument in a CALL to an assembler language subprogram, the subprogram cannot access the statement number argument.

To accomplish the same thing as the FORTRAN statement RETURN i (used in FORTRAN subprograms to return to a point other than that immediately following the CALL), the assembler subprogram must place 4*i in register 15 before returning to the calling program.

For example, when the statement

```
CALL SUB (A,B,&10,&20)
```

is used to call an assembler language subprogram, the following instructions would cause the subprogram to return to the proper point in the calling program:

```
    .
    .
    .
LA 15,4 (to return to 10)

BCR 15,14
    .
    .
    .
LA 15,8 (to return to 20)

BCR 15,14
```

Object-Time Representation of FORTRAN Variables

The programmer who uses FORTRAN in connection with assembler language may need to know how the various FORTRAN data types appear in the computer. The following examples illustrate the object-time representation of FORTRAN variables as they appear in System/360.

```
|Name    |Oper.|Operand
|--------|-----|-------------------------------------------------------------
|ADDARR  |START|0
|B       |EQU  |8
|        |BC   |15,12(15)
|        |DC   |X'7'
|        |DC   |CL7'ADDARR'
|        |STM  |14,12,12(13)
|        |BALR |B,0
|        |USING|*,B
|        |L    |2,8(1)         MOVE 3RD ARGUMENT TO  LOCATION CALLED
|        |MVC  |INDEX(4),0(2) INDEX IN ASSEMBLER LANGUAGE SUBPROGRAM.
|        |L    |3,0(1)         MOVE 1ST ARGUMENT TO  LOCATION CALLED VAR
|        |MVC  |VAR(4),0(3)    IN ASSEMBLER LANGUAGE SUBPROGRAM.
|        |L    |4,4(1)         LOAD  ADDRESS OF  ARRAY INTO REGISTER 4.
|        |user |written statements
|        |     |   .
|        |     |   .
|        |     |   .
|        |LM   |2,12,28(13)
|        |MVI  |12(13),X'FF'
|        |BCR  |15,14
|        |DS   |0F
|INDEX   |DS   |1F
|VAR     |DS   |1F
```

Figure 103. Assembler Subprogram Example

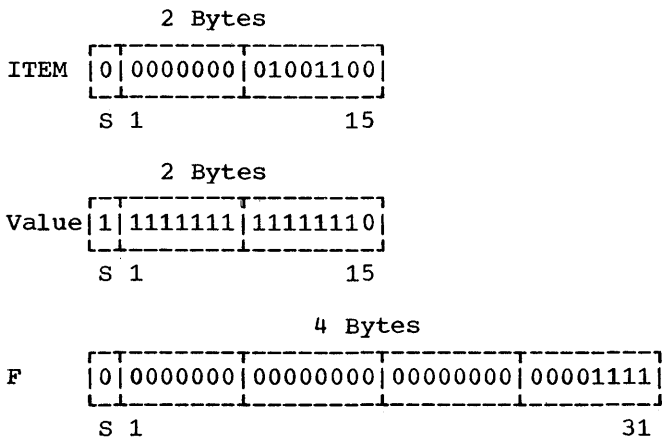
INTEGER Type

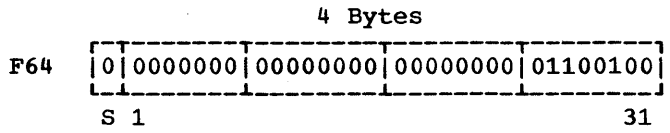
INTEGER variables are treated as fixed-point operands by the FORTRAN IV (G) and (H) compilers, and are governed by the principles of System/360 fixed-point arithmetic. INTEGER variables are converted into either fullword (32 bit) or halfword (16 bit) signed integers.

The statements:

```
INTEGER*2 ITEM/76/,VALUE
INTEGER*4 F,F64/100/
F = 15
VALUE = -2
```

would cause the variables ITEM, VALUE, F, F64 to appear in the computer as:





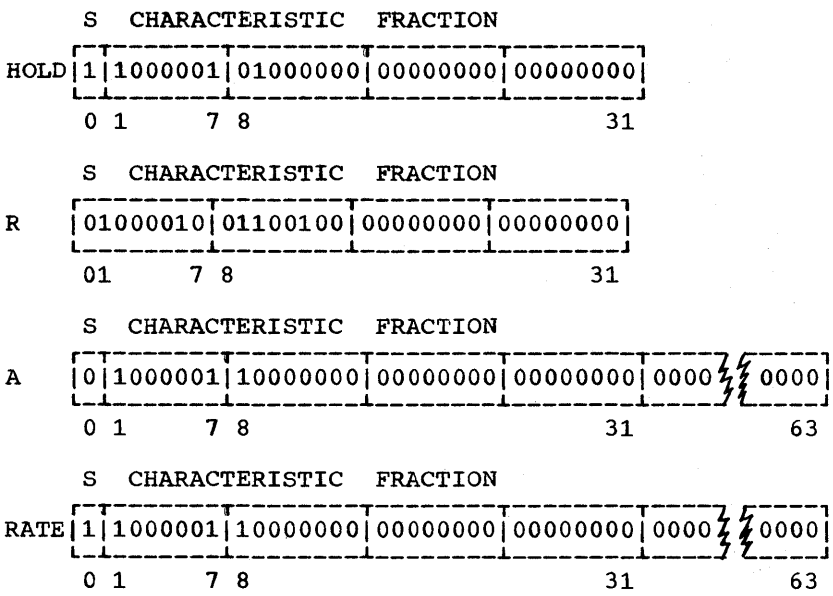
where S in bit position 0 represents the sign bit. All negative numbers are represented in two's complement notation with a one in the sign-bit position. (For a more complete explanation of fixed-point arithmetic, refer to the publication IBM System/360 Principles of Operation, Order No. GA22-6821.)

REAL Type

All REAL variables are converted into short (32 bit) or long (64 bit) floating-point numbers by the compiler. The length of the numbers is determined by FORTRAN IV specification conventions. For example, the statements:

```
REAL*4 HOLD,R/100./
REAL*8 A,RATE/-8./
HOLD = -4.
A = 8.
```

would cause the variables to appear internally as:



where:

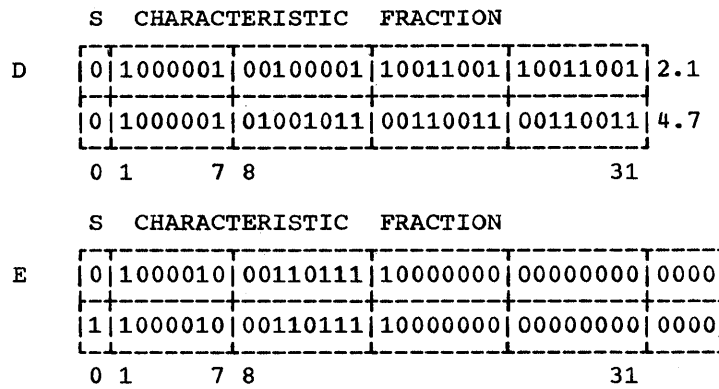
- The sign bit (S) occupies bit position 0.
- The characteristic occupies bit positions 1-7.
- The fraction occupies either bit positions 8-31 for a short floating-point number or bit positions 8-63 for a long floating-point number.

COMPLEX Type

A COMPLEX variable has two parts (real and imaginary) and is treated as a pair of REAL numbers. The COMPLEX parts are converted into two contiguous, short or long, floating-point numbers. For example:

```
COMPLEX D/(2.1,4.7)/,E*16
E = (55.5 -55.5)
```

will cause the variables D and E to appear in the computer as:



Note: Floating-point operations in System/360 may sometimes produce a negative zero, i.e., the sign bit of a floating-point zero will contain a one. FORTRAN IV compilers consider all floating-point numbers having a fraction of zero as equivalent. The setting of the sign bit is unpredictable in floating-point zeros computed by a FORTRAN G or H object program. (A detailed explanation of floating-point operations can be found in the publication IBM System/360: Principles of Operation, Order No. GA22-6821.)

LOGICAL Type

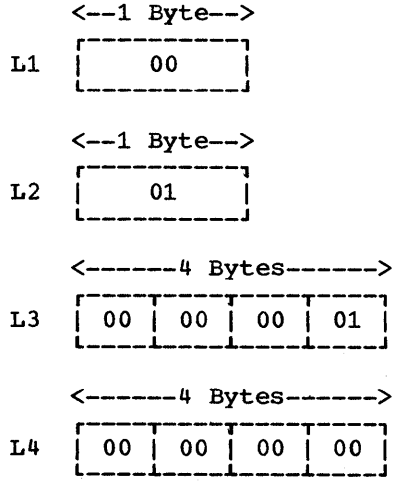
FORTRAN IV LOGICAL variables may specify only 2 values:

.TRUE. or .FALSE.

These logical values are assigned numerical values of '1' and '0', for .TRUE. and .FALSE., respectively, by both the G and H compilers. The statements:

```
LOGICAL*1 L1,L2/.TRUE./
LOGICAL*4 L3,L4/.FALSE./
L1 = .FALSE.
L3 = .TRUE.
```

would cause the variables L1, L2, L3, L4 to be assigned the following values (using hexadecimal notation):



Note: The values shown above for LOGICAL variables are those assigned for the current implementation of the FORTRAN IV (G) and (H) compilers. The assembler language programmer should not assume these values for future versions of either the (G) or (H) compilers, since both compilers are subject to change.

The DUMP or PDUMP subroutine can also be used as an additional tool for understanding the object-time representation of FORTRAN data. Refer to the "Use of DUMP and PDUMP" section in the "Programming Considerations" chapter of this publication or consult the FORTRAN IV Library Subprograms publication.

This appendix contains a detailed description of the diagnostic messages produced during compilation and load module execution. The description of each message includes a suggested operator or programmer response.

FORTRAN IV (G) COMPILER DIAGNOSTIC MESSAGES

Two types of compiler diagnostic messages are generated -- error/warning and status.

The error/warning messages produced by the compiler are noted on the source listing immediately after the statement in which they occur. A maximum of four messages appears on each line. Figure 104 illustrates the format of the messages as they are written in the data set specified by the SYSPRINT DD statement.

There are two types of error/warning messages: serious error messages, and warning messages. The serious error messages have a condition code of 8 and the warning messages a code of 4 or 0.

Status messages are produced during the operation of the compiler. Most indicate termination of compilation resulting from internal compiler errors.

```
XX = A+B+-C/(X**3-A**-75)
         $             $

n) y message, n) y message

Where:  n is an integer noting the ordi-
          nal occurrence of the error on
          each card.

        y is a 1- to 3-digit message num-
          ber in IEYxxxI format.

        $ is the symbol used by the
          compiler for flagging the
          particular error in the
          statement. (This symbol is
          usually noted on the line
          following the source statement
          and may appear at various
          points beneath the error
          statement depending on the type
          of error.)

        message is the actual message
          printed
```

Figure 104. Format of Diagnostic Messages

Error/Warning Messages

The following text contains a description of error/warning messages produced by the compiler. The message is shown with an explanation.

IEY001I ILLEGAL TYPE

Explanation: The type of a constant, a variable, or an expression is not correct for its usage. For example, the variable in an Assigned GO TO statement is not an integer variable; or the variable in an assignment statement on the left of the equal sign is of logical type and the expression on the right side does not correspond; or an argument in a reference to an IBM-supplied subprogram is not the type required by the subprogram. (Condition code - 8)

Programmer Response: Probable user error. Make sure that the variable in an Assigned GO TO statement is an integer variable. Verify that any variable in an assignment statement on the left of the equal sign is not of logical type with the expression on the right side not in correspondence. Make sure that an argument in a reference to an IBM-supplied subprogram is the type required by the subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY002I LABEL

Explanation: The statement in question is unlabeled and follows a transfer of control; the statement therefore cannot be executed. (Condition code - 0)

Programmer Response: Probable user error. Correct an unlabeled statement following a transfer of control, as it cannot be executed. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY003I NAME LENGTH

Explanation: The name of a variable, COMMON block, NAMELIST or subprogram exceeds six characters in length; or two variable names appear in an expression without a separating operation symbol.
(Condition code - 0)

Programmer Response: Probable user error. Make sure that the name of a variable, COMMON block, NAMELIST, or subprogram does not exceed six characters in length. Check that two variable names do not appear in an expression without a separating operation symbol. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY004I COMMA

Explanation: The delimiter required in the statement has been omitted.
(Condition code - 0)

Programmer Response: Probable user error. Correct or delete invalid delimiters and insert the required delimiter that has been omitted. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY005I ILLEGAL LABEL

Explanation: Illegal usage of a statement label; for example, an attempt is made to branch to the label of a FORMAT statement.
(Condition code - 8)

Programmer Response: Probable user error. Correct the illegal usage of a statement label. Example: No branch to the label of a FORMAT statement should be coded. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY006I DUPLICATE LABEL

Explanation: The label appearing in the label field of a statement has previously been defined for another statement.
(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the previously defined label and adjust any code referencing the label. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY007I ID CONFLICT

Explanation: The name of a variable or subprogram has been used in conflict with the type that was defined for the variable or subprogram in a previous statement. (Condition code - 8)

Programmer Response: Probable user error. Correct any variable or subprogram name used in conflict with the type defined for the variable or subprogram in a previous statement. Examples: The name listed in a CALL statement is the name of a variable; a single name appears more than once in the dummy list of a statement function; a name listed in an EXTERNAL statement has been defined in another context. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY008I ALLOCATION

Explanation: The storage allocation specified by a source module statement cannot be performed because of an

inconsistency between the present usage of a variable name and some prior usage of that name. (Condition code - 8)

Programmer Response: Probable user error. Correct the statement since the storage allocation specified by a source module statement cannot be performed. Make sure that an inconsistency between present usage of a variable name and some prior usage of that name does not occur. Examples: A name listed in a COMMON block has been listed in another COMMON block; a variable listed in an EQUIVALENCE statement is followed by more than seven subscripts. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY009I ORDER

Explanation: The statements contained in the source module are used in an improper sequence. (Condition code - 8)

Programmer Response: Probable user error. Make sure that statements contained in the source module are used in proper sequence. Examples: An IMPLICIT statement does not appear as the first or second statement of the source module; an ENTRY statement appears within a DO loop. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY010I SIZE

Explanation: A number used in the source module does not conform to the legal values for its use. (Condition code - 8)

Programmer Response: Probable user error. Make sure that a number used in the source module conforms to the legal values for its use. Examples: A label used in a statement exceeds the legal size for a statement label; the size specification in an EXPLICIT statement is not acceptable; an integer constant is not within the allowable magnitude. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY011I UNDIMENSIONED

Explanation: A variable name is used as an array (i.e., subscripts follow the name), and the variable has not been dimensioned. (Condition code - 8)

Programmer Response: Probable user error. Make sure that a variable name is not used as an array (i.e., subscripts must not follow the name). Include a DIMENSION statement if one is missing. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY012I SUBSCRIPT

Explanation: The number of subscripts used in an array reference is either too large or too small for the array. (Condition code - 8)

Programmer Response: Probable user error. Make sure that the number of subscripts used in an array reference corresponds to the number appearing in the DIMENSION statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY013I SYNTAX

Explanation: The statement or part of a statement to which this message refers does not conform to the FORTRAN IV syntax. (Condition code - 8)

Programmer Response: Probable user error. Make sure that all source code conforms to the FORTRAN IV syntax. Examples: The statement cannot be identified; a non-digit appears in the label field; fewer than three labels follow the expression in an Arithmetic IF statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY014I CONVERT

Explanation: The mode of the constant used in a DATA or in an Explicit Specification statement is different from the mode of the variable with which it is associated. The constant is then converted to the correct mode. (Condition code - 0)

Programmer Response: Probable user error. Make sure that the mode of the constant used in a DATA or in an EXPLICIT specification statement is identical to the mode of the variable with which it is associated.

IEY015I NO END CARD

Explanation: The source module does not contain an END statement. (Condition code - 0)

Programmer Response: Probable user error. Include the necessary END statement for the source module. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY016I ILLEGAL STA.

Explanation: The context in which the statement in question has been used is illegal. (Condition code - 8)

Programmer Response: Probable user error. Correct or delete illegal context. Examples: A specification or a DO statement appears in a Logical IF; an ENTRY statement appears outside a subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY017I ILLEGAL STA. WRN.

Explanation: The message is produced as a result of any of the following: a RETURN statement appears and the source module is not a subprogram; a RETURN i statement appears in a FUNCTION subprogram. (Condition code - 0)

Programmer Response: Probable user error. Correct or delete a RETURN statement appearing outside a subprogram or a RETURN I statement appearing in a FUNCTION subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY018I NUMBER ARG

Explanation: The reference to a library subprogram specifies an incorrect number of arguments. (Condition code - 4)

Programmer Response: Probable user error. Correct or delete an invalid reference to a library subprogram specifying an incorrect number of arguments. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY019I FUNCTION ENTRIES UNDEFINED

Explanation: In a FUNCTION subprogram, there was no statement to assign a value to the function name or an entry name. If the FUNCTION subprogram contains no ENTRY statements, the error must be corrected. When ENTRY statements are present, this message is a warning provided that at least one function or entry name is assigned a value in the FUNCTION subprogram. However if no function or entry name is assigned a value, that error must be corrected. (Condition code - 4)

Programmer Response: Probable user error. Make sure that the function name and each entry name is assigned a value in a FUNCTION subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY020I COMMON BLOCK name ERRORS

Explanation: This message pertains to errors that exist in the definitions of EQUIVALENCE sets which refer to the COMMON area. The message is produced when there is a contradiction in the allocation specified, a designation to extend the beginning of the COMMON area, or if the assignment of COMMON storage attempts to allocate a variable to a location which does not fall on the appropriate boundary; "name" is the name of the COMMON block in error.
(Condition code - 4)

Programmer Response: Probable user error. Verify that definitions of EQUIVALENCE sets which refer to a COMMON area are correct. Make sure that none of the following occurs: a contradicition in the allocation specified, a designation to extend the beginning of the COMMON area, or an assignment of COMMON storage attempts to allocate a variable to a location which does not fall on the appropriate boundary. Use the MAP option to determine offsets in the COMMON block designated in error ("name"). If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Have source and associated listing available.

IEY021I UNCLOSED DO LOOPS

Explanation: The message is produced if DO loops are initiated in the source module, but their terminal statements do not exist. A list of the labels which appeared in the DO statements but were not defined follows the printing of the message.
(Condition code - 8)

Programmer Response: Probable user error. Correct or insert statements where DO loops are initiated and their terminal statements do not exist. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY022I UNDEFINED LABELS

Explanation: If any labels are used in the source module but are not defined, this message is produced. A list of the undefined labels appears on the lines following the message.
(Condition code - 8)

Programmer Response: Probable user error. Correct or insert necessary references to labels which require definition. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY023I EQUIVALENCE ALLOCATION ERRORS

Explanation: This message is produced when there is a conflict between two EQUIVALENCE sets, or if there is an incompatible boundary alignment in the EQUIVALENCE set. The message is followed by a list of the variables which could not be allocated according to source module specifications.
(Condition code - 4)

Programmer Response: Probable user error. Correct or delete conflicts between two EQUIVALENCE sets or incompatible boundary alignments in the EQUIVALENCE set. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Have source and associated listing available.

IEY024I EQUIVALENCE DEFINITION ERRORS

Explanation: This message denotes an error in an EQUIVALENCE set when an array element is outside the array. (Condition code - 4)

Programmer Response: Probable user error. Correct or delete the invalid reference in the EQUIVALENCE set. If the problem

recurs, do the following before calling IBM for programming support:

- Make sure that MAP has been specified as a parameter on the EXEC statement.
- Have source and associated listing available.

IEY025I DUMMY DIMENSION ERRORS

Explanation: If variables specified as dummy array dimensions are not in COMMON and are not global dummy variables, the above error message is produced. A list of the dummy variables which are found in error is printed on the lines following the message. (Condition code - 4)

Programmer Response: Probable user error. Make sure that variables assigned to a program block are defined previously as in COMMON. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY026I BLOCK DATA PROGRAM ERRORS

Explanation: This message is produced if variables in the source module have been assigned to a program block but have not been defined previously as COMMON. A list of these variables is printed on the lines following the message. (Condition code - 4)

Programmer Response: Probable user error. Make sure that variables assigned to a program block are defined previously as in COMMON. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY027I CONTINUATION CARDS DELETED

Explanation: More than 19 continuation lines were read for one statement. All subsequent lines are skipped until the beginning of the next statement is encountered.
(Condition code - 8)

Programmer Response: Probable user error. Delete the continuation cards in error and begin a new source statement to correct the source. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY032I NULL PROGRAM

Explanation: This message is produced when an end of file mark precedes any true FORTRAN statements in the source module. (Condition code - 0)

Programmer Response: Probable user error. Correct or delete an end-of-file mark preceding any two FORTRAN statements in the source module. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY033I COMMENTS DELETED

Explanation: More than 30 comment lines were read between the initial lines of two consecutive statements. The 31st comment line and all subsequent comment lines are skipped until the beginning of the next statement is encountered. (There is no restriction on the number of comment lines preceding the first statement.)
(Condition code - 0)

Programmer Response: Probable user error. Make sure that more than 30 comment lines do not appear between the initial lines of two consecutive statements. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY036I ILLEGAL LABEL WRN

Explanation: The label on this nonexecutable statement has no valid use beyond visual identification, and may produce errors in the object module if the same label is the target of a branch type statement. (Only branches to executable statements are valid.) This message is

produced, for example, when an END statement is labeled. The message is issued as a warning only. (Condition code - 4)

Programmer Response: Probable user error. Correct or delete the occurrence of a label on a nonexecutable statement having no valid use beyond visual identification. Make sure that only branches to executable statements are indicated. Example: An END statement is labeled. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY037I PREVIOUSLY DIMENSIONED WRN

Explanation: This message is produced if arrays are redimensioned. The dimensions first to be given are used. An example of a situation that would cause this message to be issued follows: Dimension information for an array is given in a type statement and subsequent COMMON and/or DIMENSION statements redefine the dimensions. (Condition code - 4)

Programmer Response: Probable user error. Make sure that arrays have not been redimensioned. Example: Dimension information for an array is given in a type statement and subsequent COMMON and/or DIMENSION statements redefine the dimensions. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY038I SIZE WRN

Explanation: A variable has data initializing values that exceed the size of the scalar or the array or array element. Examples of situations that would cause this message to be issued follow: (1) Five bytes of initializing data are given for a scalar variable, as in REAL A/'ABCDE'/ (2) Excessive bytes are given for an element of an array, as in DATA A (1)/'ABCDEFG'/. This message is also produced when data spill is employed to initialize an array. For example:

```
DIMENSION ARRAY(3)
ARRAY/'ABCDEFGHIJKL'/
```

(Condition code - 4)

Programmer Response: Probable user error. Make sure that data initializing values for a variable do not exceed the size of the scalar or the array or array element. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY039I RETURN

Explanation: A RETURN statement is needed. (Condition code - 0)

Programmer Response: Probable user error. Insert missing RETURN statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY040I COMMON ERROR IN BLOCK DATA

Explanation: An error has ocurred in the use of a BLOCK DATA subprogram. There must be at least one named COMMON statement within the BLOCK DATA subprogram. The BLOCK DATA subprogram cannot contain any references to blank COMMON. (Condition code - 8)

Programmer Response: Probable user error. Make sure that there is at least one named COMMON statement within the BLOCK DATA subprogram. The BLOCK DATA subprogram cannot contain any references to blank COMMON. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY041I COMMON INITIALIZATION ERRORS

Explanation: An attempt has been made to initialize a variable in unlabeled COMMON or an attempt has been made, outside of a BLOCK DATA subprogram, to initialize a variable in labeled COMMON. A list of the variables in error

follows the message.
(Condition code - 8).

Programmer Response: Delete the invalidly referenced variables. Make sure that variables in COMMON (by declaration or equivalence) are initialized in a BLOCK DATA subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY045I SP CONSTANT

Explanation: The constant flagged is typed single precision (REAL*4) regardless of the number of digits coded.
(Condition code - 0).

Programmer Response: Reduce the number of digits specified for the constant to seven or fewer. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEY046I DP CONSTANT

Explanation: The constant flagged is typed double precision (REAL*8) because it contains more than seven digits; the letter D was not specified.
(Condition code - 0)

Programmer Response: Specify the letter D for all double precision constants. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

Status Messages

The following paragraphs describe the messages that are produced during the operation of the (G) compiler which denote the progress of the compilation. Most of the messages discussed in this section pertain to the conditions that result in the termination of the compilation.

IEY028I NO CORE AVAILABLE-COMPILATION TERMINATED

Explanation: This message is produced when the system is unable to supply the compiler with an additional 4K byte block of roll (or table) storage.
(Condition code - 16)

Programmer Response: Probable user error. Either segment the program unit into subroutines or specify a larger REGION size on the JOB or EXEC card. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Have source and associated listing available.

IEY029I DECK OUTPUT DELETED

Explanation: If the DECK option has been specified, and an error occurs during the process of punching the designated output, this message is produced. No condition code is generated for this error.

Programmer Response: If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) has been specified on the JOB statement.
- Have source and associated listing available.

IEY030I LINK EDIT OUTPUT DELETED

Explanation: If the LOAD option has been specified, and an error occurs during the process of generating the load module, this message is produced.
(Condition code - 16)

Programmer Response: If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) has been specified on the JOB statement.
- Have source and associated listing available.

IEY031I ROLL SIZE EXCEEDED

Explanation: This message is produced when the WORK or EXIT roll (table) has exceeded the storage capacity to which it has been assigned, or some other roll used by the compiler has exceeded 64K bytes of storage. (Condition code - 16)

Programmer Response: Restructure the program unit and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Have source and associated listing available.

IEY034I I/O ERROR [COMPILATION TERMINATED] XXX...XXX

Explanation: This message is produced when an input/output error is detected during compilation. If the error occurred on SYSPUNCH, compilation is continued and the "COMPILATION TERMINATED" portion of the message is not printed. (Condition code - 8). If the error occurred on SYSIN, SYSPRINT or SYSLIN, compilation is terminated. (Condition code - 16). xxx...xxx is the character string formatted by the SYNADAF macro instruction. For an interpretation of this information, see the publication IBM System/360 Operating System: Supervisor and Data Management Macro Instructions, Order No. GC28-6647.

Programmer Response: Check all DD statements. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) has been specified on the JOB statement.
- Have source and associated listing available.

IEY035I UNABLE TO OPEN ddname

Explanation: This message is produced when the required ddname data definition card is missing or the ddname is misspelled.

Programmer Response: Probable user error. Include the required ddname data definition statement or correct a misspelled ddname. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) has been specified on the JOB statement.
- Have source and associated listing available.

Informative Messages

Informative messages are generated by the compiler to inform the programmer of the result of the compilation. The messages are shown with any compiler action taken.

```
*OPTIONS IN EFFECT* option {,option}...
*OPTIONS IN EFFECT* option {,option}...
*STATISTICS* SOURCE STATEMENTS=nnnnnnnn1
  PROGRAM SIZE=nnnnnnnn2
          and
*STATISTICS* NO DIAGNOSTICS GENERATED
          or
*STATISTICS* nnn DIAGNOSTICS GENERATED,
  HIGHEST SEVERITY CODE IS n
```

where

$nnnnnnnn_1$
: is the number of source statements, expressed as a decimal integer.

$nnnnnnnn_2$
: is the size, in bytes, of the object module expressed as a decimal integer.

nnn
: is the number of diagnostics generated, expressed as a decimal integer.

n
: is the condition code.

The first statistics message (giving source statements and program size) is suppressed whenever a BLOCK DATA subprogram is compiled; however, the two options-in-effect messages and one of the other statistics messages will still appear.

If there was more than one compilation (i.e., there was a multiple compilation), one final message is printed. This message is suppressed when there is only one main or subprogram. The message appears as either:

STATISTICS NO DIAGNOSTICS THIS STEP

or

STATISTICS nnn DIAGNOSTICS THIS STEP

Explanation: If there were no errors in any of the compilations the first message is printed. The second message is printed when there are errors in one or more of the compilations; the cumulative number of errors is indicated by the value nnn.

FORTRAN IV (H) COMPILER DIAGNOSTIC MESSAGES

Two types of compiler diagnostic messages are generated - informative and error/warning.

Informative Messages

Six unnumbered messages are generated by the compiler to provide the programmer with compilation information.

LEVEL- nn.n(date) OS/360 FORTRAN H
DATE- yy.ddd/HH.mm.ss

Explanation: This message is generated at the beginning of every compilation. The level of the compiler corresponds to the release number and release date of the operating system. The number of the day (ddd) in the year (yy) that the compilation takes place is given by "yy.ddd"; the time of day in hours (HH), minutes (mm), and seconds (ss) (based on a 24-hour clock) is given by "HH.mm.ss". The time is also punched into the END card of the object deck.

COMPILER OPTIONS - option{option}...

Explanation: This message is printed on the first page of every source listing. Options explicitly specified in the PARM parameter and any default options appear in the message.

OPTIONS IN EFFECT NAME=xxxxxx,OPT=0n,
LINECNT=xx, SIZE=nnnnK
OPTIONS IN EFFECT option {,option}...
STATISTICS SOURCE STATEMENTS=nnnnnnnn$_1$,
PROGRAM SIZE=nnnnnnnn$_2$
STATISTICS NO DIAGNOSTICS GENERATED

or

STATISTICS nnn DIAGNOSTICS GENERATED,
HIGHEST SEVERITY CODE IS n

Explanation: This message appears immediately before the ****** END OF COMPILATION ****** message. It indicates the size of both the source module and the object module as well as the FORTRAN (H) environment in which they were processed.

nnnn
is the value specified for the SIZE option expressed as a decimal integer. (IF this parameter was not specified the value listed in the message is 0000.)

nnnnnnnn$_1$
is the number of source statements (comments cards are not included) expressed as a decimal integer.

nnnnnnnn$_2$
is the size of the object module, in bytes, expressed as a decimal integer.

nnn
is the number of diagnostics for each compilation expressed as a decimal integer.

The options indicated are the default options and those explicitly specified in the PARM parameter of the EXEC statement. For a full explanation of these options, see "Compiler Options" in the "FORTRAN Job Processing" section of this publication. The severity code, n, is the completion code.

nnnnK BYTES OF CORE NOT USED

Explanation: This message is produced if more than 10K bytes of available work area remain unused during compilation. This message appears immediately after the ****** END OF COMPILATION ****** message. The term nnnn indicates how much smaller the region size could have been specified for optimal storage use during compilation. If the SIZE option was indicated for the compilation step, the term nnnn indicates how much smaller the specified SIZE value could have been.

******END OF COMPILATION******

Explanation: This message, which indicates that all processing of the source program has been completed, is generated at the end of every compilation except when an abnormal termination causes the generation of the message COMPILATION DELETED. n

If there is more than one compilation in a job step, one of the following messages will be printed after the last compilation:

STATISTICS NO DIAGNOSTICS THIS STEP

or

STATISTICS nnn DIAGNOSTICS THIS STEP,
HIGHEST SEVERITY CODE IS n

Explanation: If there were no errors in any of the compilations, the first message is printed. The second message is printed when there are errors in one or more of the compilations; the cumulative number of errors is indicated by the value nnn. The severity code, n, is the completion code.

Error/Warning Messages

The following text contains a description of error/warning messages produced by the compiler. The message is shown with an explanation, and any compiler action or user action that is required.

All error/warning messages produced are written in a group following the source module listing and object module name table. Figure 105 shows the format of each message as it is written in the data set specified by the SYSPRINT DD statement.

There are three types of messages: (1) a terminal error message, (2) serious error messages, and (3) warning messages. The terminal error message returns a condition code of 16; the serious error messages a code of 8; and the warning messages a code of 4.

ISN a LABEL b NAME c	ERROR NO. IEKxxxI	ERROR MESSAGE message
a	is the internal statement number of either the statement in error or the statement following the last previous executable statement.	
b	is a source label (statement number)	
c	is a variable name	
xxx	is a 3-digit message number	
message	is the actual message printed	

Figure 105. Format of Diagnostic Messages

In addition, following the statement in which a serious error is detected, the following appears in the source listing:

ERROR DETECTED - SCAN POINTER = x; x represents the position of the character pointed to by the compiler's scan pointer at the time the error is detected. Any FORTRAN keywords and/or meaningless blanks are ignored in determining the position of the character. (If the statement is found to be invalid during the classification process, the value of x always equals one.)

In the case of compilation deletion, the following message appears:

COMPILATION DELETED. n

where n can be 1, 2, 3, 4, 5, 6, or 7

Explanation: The message is generated by the FORTRAN System Director. The compilation is deleted because of the reason indicated by the value of n. (Condition code - 16)

n=1 Phase 10 Program too large to compile or main storage allocation is too small for compiler size.

n=2 A program interrupt occurred during execution of the compiler. A program interrupt message and the contents of all registers are written preceding the message.

Figure 106 shows the format of the compile-time program interrupt message when the extended error message facility has not been specified at system generation time. In the old PSW, c is a hexadecimal number that indicates the cause of the interruption; c may be one of the following values:

c	Cause
1	Operation
2	Privileged operation
3	Execute
4	Protection
5	Addressing
6	Specification
7	Data
8	Fixed-point overflow
9	Fixed-point divide
A	Decimal overflow
B	Decimal divide
C	Exponent overflow
D	Exponent underflow
E	Significance
F	Floating-point divide

```
IHC210I  PROGRAM INTERRUPT - OLD PSW IS xxxxxxxcxxxxxxxx - PHASE SWITCH m
```

Figure 106. Compile-Time Program Interrupt Message

Following PHASE SWITCH, m is a hexadecimal number that indicates which phase of the compiler was executing when the interrupt occurred; m may be one of the following values:

m	Phase
1	Phase 10
2	Phase 10 (STALL routine)
4	Phase 15 (PHAZ15 routine)
8	Phase 15 (CORAL routine)
10	Phase 20
20	Phase 25
40	Phase 30

n=3 Phase 15 Program too large to compile
n=4 Phase 20 Program too large to compile
n=5 Phase in control requested System Director to terminate compilation immediately. (Any error messages generated by the calling phase will also be written.)
n=6 Error detected by IHCFCOMH (IBCOM) I/O error detected during compilation - an IHCxxxI message may also be generated
n=7 End of file, no END statement in source module

IEK001I THE NUMBER OF ENTRIES IN THE ERROR TABLE HAS EXCEEDED THE MAXIMUM.

(Condition code - 8)

Programmer Response: Probable user error. Correct the statements in error. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that all indicated statements in error are corrected.
- Have source and associated listing available.

IEK002I THE DO LOOPS ARE INCORRECTLY NESTED.

(Condition code - 8)

Programmer Response: Probable user error. Resubmit the job with all statements in the range of the inner DO also in the range of the outer DO. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that the extended range of a DO statement does not contain another DO statement that has an extended range if the second DO is within the same program unit as the first.
- Have source and associated listing available.

IEK003I THE EXPRESSION HAS AN INVALID LOGICAL OPERATOR.

(Condition code - 8)

Programmer Response: Probable user error. Correct the invalid logical operator. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK005I THE STATEMENT HAS AN INVALID USE OF PARENTHESES.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the parenthesis in question. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK006I THE STATEMENT HAS AN INVALID LABEL.

(Condition code - 8)

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK007I THE EXPRESSION HAS AN INVALID DOUBLE DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK008I THE EXPRESSION HAS A CONSTANT WHICH IS GREATER THAN THE ALLOWABLE MAGNITUDE.

(Condition code - 8)

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK009I THE EXPRESSION HAS A NON-NUMERIC CHARACTER IN A NUMERIC CONSTANT.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the constant contains only numeric characters. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK010I THE EXPRESSION HAS A CONSTANT WITH AN INVALID EXPONENT.

(Condition code - 8)

Programmer Response: Probable user error. Delete or correct the invalid exponent. Make sure that the base and exponent are valid combinations of operand types. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK011I THE ARITHMETIC OR LOGICAL EXPRESSION USES AN EXTERNAL FUNCTION NAME AS A VARIABLE NAME.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the external function name is not employed as a variable. Resubmit the job using the MAP option to obtain indication of the use of each name. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK012I THE EXPRESSION HAS A COMPLEX CONSTANT WHICH IS NOT COMPOSED OF REAL CONSTANTS.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that both the real and imaginary parts of the complex number are valid real constants. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK013I AN INVALID CHARACTER IS USED AS A DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid character. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK014I THE STATEMENT HAS AN INVALID NON-INTEGER CONSTANT.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid non-integer constant. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK015I THE ARITHMETIC OR LOGICAL EXPRESSION USES A VARIABLE NAME AS AN EXTERNAL FUNCTION NAME.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the variable is not employed as an external function. Resubmit the job using the MAP option to obtain indications of the use of each name. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK016I THE GO TO STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the statement contains valid delimiters. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK017I THE ASSIGNED OR COMPUTED GO TO HAS AN INVALID ELEMENT IN ITS STATEMENT NUMBER LIST.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the statement number list contains executable statement numbers. If an assigned GO TO is in question, make sure that the ASSIGN statement is correct. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK019I THE ASSIGNED GO TO HAS THE OPENING PARENTHESIS MISPLACED OR MISSING.

(Condition code - 8)

Programmer Response: Probable user error. Correct or insert parenthesis. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK020I THE ASSIGNED GO TO HAS AN INVALID DELIMITER FOLLOWING THE ASSIGNED VARIABLE.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that a comma follows the assigned variable. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK021I THE COMPUTED GO TO HAS AN INVALID COMPUTED VARIABLE.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the referenced variable is integer and non-subscripted. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK022I THE VARIABLE IN THE ASSIGNED GO TO STATEMENT IS NOT INTEGRAL.

(Condition code - 8)

Programmer Response: Probable user error. Correct the non-integral variable. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK023I THE DEFINE FILE STATEMENT HAS AN INVALID DATA SET REFERENCE NUMBER.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the data set reference number or size is an integer constant. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK024I THE DEFINE FILE STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Correct the invalid delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK025I THE DEFINE FILE STATEMENT HAS AN INVALID INTEGER CONSTANT AS THE RECORD NUMBER OR SIZE.

(Condition code - 8)

Programmer Response: Probable user error. Correct the invalid integer. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK026I THE DEFINE FILE STATEMENT HAS AN INVALID FORMAT CONTROL CHARACTER.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the format control character is one of L, E, or U. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK027I THE ASSIGN STATEMENT HAS AN INVALID INTEGER VARIABLE.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid integer variable. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK028I THE ASSIGN STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Determine the invalid delimiter and correct the statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK030I THE DO STATEMENT HAS AN INVALID END OF RANGE STATEMENT NUMBER.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the end of range statement number is an executable statement number appearing after the DO statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK031I THE DO STATEMENT OR IMPLIED DO HAS AN INVALID INITIAL VALUE.

(Condition code - 8)

Programmer Response: Probable user error. Check that the initial value is either an unsigned integer constant greater than zero, or an unsigned non-subscripted integer variable greater than zero. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK034I THE ASSIGNMENT STATEMENT BEGINS WITH A NON-VARIABLE.

(Condition code - 8)

Programmer Response: Probable user error. Correct the statement. Resubmit the job with the MAP option to determine the nature of the non-variable in question, if possible. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK035I THE NUMBER OF CONTINUATION CARDS EXCEEDS THE COMPILER LIMIT.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the statement does not extend over more than 19 continuation cards. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK036I THE STATEMENT CONTAINS INVALID SYNTAX. THE STATEMENT CANNOT BE CLASSIFIED.

(Condition code - 8)

Programmer Response: Probable user error. Correct the statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK039I THE DEFINE FILE STATEMENT HAS AN INVALID ASSOCIATED VARIABLE.

(Condition code - 8)

Programmer Response: Probable user error. Check that the associated variable is non-subscripted and integral. Make sure that the associated variable does not appear in the I/O list of a READ or WRITE for a data set associated with the DEFINE FILE statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK040I IT IS ILLEGAL TO HAVE A & STATEMENT NUMBER PARAMETER OUTSIDE A CALL STATEMENT.

(Condition code - 8)

Programmer Response: Probable user error. Delete the & statement number. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK044I ONLY THE CALL, FORMAT, OR DATA STATEMENTS MAY HAVE LITERAL FIELDS.

(Condition code - 8)

Programmer Response: Probable user error. Delete or correct the misplaced literal field. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK045I THE EXPRESSION HAS A LITERAL WHICH IS MISSING A DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Insert the missing delimiter, or correct the erroneous delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK047I THE LITERAL HAS MORE THAN 255 CHARACTERS IN IT.

(Condition code - 8)

Programmer Response: Probable user error. Delete the excessive characters, or make sure that the constant is properly delimited. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK050I THE ARITHMETIC IF HAS THE SYNTAX OF THE BRANCH LABELS INCORRECT.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that there are three executable statement numbers with commas following the first two. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK052I THE EXPRESSION HAS AN INCORRECT PAIRING OF PARENTHESES OR QUOTES.

(Condition code - 8)

Programmer Response: Probable user error. Check that there are as many left parentheses as there are right. If a FORMAT statement contains an H Format Code, check that w is large enough to accommodate the data and does not encompass the closing parenthesis. Make sure that there is an even number of single quotes, and that single quotes within data, if any, are represented by two successive single quotes. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK053I THE STATEMENT HAS A MISPLACED EQUAL SIGN.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the equal sign indicated. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK056I THE FUNCTION STATEMENT MUST HAVE AT LEAST ONE ARGUMENT.

(Condition code - 8)

Programmer Response: Probable user error. Correct the function statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK057I THE STATEMENT HAS A NON-VARIABLE SPECIFIED AS A SUBPROGRAM NAME.

(Condition code - 8)

Programmer Response: Probable user error. Correct the non-variable subprogram name. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK058I THE SUBPROGRAM STATEMENT HAS AN INVALID ARGUMENT.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid argument. Use the MAP option to determine its nature, if necessary. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK059I THE FUNCTION STATEMENT HAS AN INVALID LENGTH SPECIFICATION.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the length specification is permissible for the associated type. Check that a type has been specified and that DOUBLE PRECISION has not been included. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK061I THE EQUIVALENCE STATEMENT CONTAINS A NON-SUBSCRIPTED ARRAY ITEM. INCORRECT ADCONS MAY BE GENERATED.

(Condition code - 4)

Programmer Response: Probable user error. Include the necessary subscripts. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK062I THE EQUIVALENCE STATEMENT HAS AN ARRAY WITH AN INVALID NUMBER OF SUBSCRIPTS.

(Condition code - 8)

Programmer Response: Probable user error. Delete the invalid subscripts or include those necessary for agreement with the associated specification statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK064I THE NAMELIST STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Delete or correct the invalid delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK065I THE NAMELIST STATEMENT HAS A NAMELIST NAME NOT BEGINNING WITH AN ALPHABETIC CHARACTER.

(Condition code - 8)

Programmer Response: Probable user error. Correct the invalid character. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK066I THE NAMELIST STATEMENT HAS A NON-UNIQUE NAMELIST NAME.

(Condition code - 8)

Programmer Response: Probable user error. Correct the NAMELIST name. Invoke the MAP option for indications of the use of each name, if necessary. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK067I THE NAMELIST STATEMENT HAS AN INVALID LIST ITEM.

(Condition code - 8)

Programmer Response: Probable user error. Check that the list item is a variable or array name. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK069I THE COMMON STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Delete or correct the invalid delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK070I THE EQUIVALENCE STATEMENT HAS A MISSING OR MISPLACED DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that there are as many left parentheses as there are right parentheses. Check all commas. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK071I THE EQUIVALENCE STATEMENT DOES NOT SPECIFY AT LEAST TWO VARIABLES TO BE EQUIVALENCED.

(Condition code - 8)

Programmer Response: Probable user error. Check delimiters and correct the invalid source. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK072I THE EQUIVALENCE STATEMENT HAS AN INVALID VARIABLE NAME.

(Condition code - 8)

Programmer Response: Probable user error. Delete or correct the invalid variable name. Make sure that the variable in question is not a dummy argument. If necessary, invoke the MAP option for indications of the use of variable names. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK073I THE EQUIVALENCE STATEMENT HAS A SUBSCRIPT WHICH IS NOT AN INTEGER CONSTANT.

(Condition code - 8)

Programmer Response: Probable user error. Correct the invalid subscript. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK074I THE STATEMENT HAS A VARIABLE WITH MORE THAN SEVEN SUBSCRIPTS.

(Condition code - 8)

Programmer Response: Probable user error. Check that all commas are in correct position. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK075I THE COMMON STATEMENT HAS A VARIABLE THAT HAS BEEN REFERENCED IN A PREVIOUS COMMON STATEMENT.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the redundant entry. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK076I THE IMPLICIT STATEMENT IS NOT THE FIRST STATEMENT IN A MAIN PROGRAM OR THE SECOND STATEMENT IN A SUBPROGRAM.

(Condition Code - 8)

Programmer Response: Probable user error. Place the IMPLICIT statement in correct sequence. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK077I THE IMPLICIT STATEMENT HAS A MISPLACED DELIMITER IN THE TYPE SPECIFICATION FIELD.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK078I THE IMPLICIT STATEMENT HAS AN INVALID TYPE.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that for any type there is a corresponding valid standard or optional length specification. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK079I THE IMPLICIT STATEMENT HAS A MISSING LETTER SPECIFICATION.

(Condition code - 8)

Programmer Response: Probable user error. Insert the omitted specification. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK080I THE IMPLICIT STATEMENT HAS AN INVALID LETTER SPECIFICATION.

(Condition code - 8)

Programmer Response: Probable user error. Correct the invalid letter specification. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK081I THE IMPLICIT STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Delete or correct the invalid delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK082I THE IMPLICIT STATEMENT DOES NOT END WITH A RIGHT PARENTHESIS.

(Condition code - 8)

Programmer Response: Probable user error. Insert the omitted parenthesis. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK083I THE IMPLICIT STATEMENT HAS A MISPLACED DELIMITER IN ITS PARAMETER FIELD.

(Condition code - 8)

Programmer Response: Probable user error. Delete or correct the misplaced delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK084I THE IMPLICIT STATEMENT CONTAINS A LITERAL FIELD.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that there are no apostrophes or wH specifications in the IMPLICIT statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK086I THE COMMON STATEMENT SPECIFIES A NON-VARIABLE TO BE ENTERED.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that delimiters are in correct position. Check that only variable or array names are specified for entrance into a common block. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK087I THE COMMON STATEMENT SPECIFIES A NON-VARIABLE COMMON BLOCK NAME.

(Condition code - 8)

Programmer Response: Probable user error. Correct the invalid name. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK088I A DUMMY ARGUMENT IN A SUBPROGRAM STATEMENT MAY NOT BE IN COMMON

(Condition code - 8)

Programmer Response: Probable user error. Make sure that only variable or array names appear in the COMMON statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK090I THE EXTERNAL STATEMENT HAS A NON-VARIABLE DECLARED AS EXTERNAL.

(Condition code - 8)

Programmer Response: Probable user error. Correct the non-variable name. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK091I THE EXTERNAL STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK092I THE TYPE STATEMENT MULTIPLY DEFINES THE VARIABLE.

Condition code - 8)

Programmer Response: Probable user error. Correct or delete the variable in question. If the

problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK093I THE TYPE STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid delimiter. If the probelm recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK094I THE TYPE STATEMENT HAS A NON-VARIABLE TO BE TYPED.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the non-variable. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK095I THE TYPE STATEMENT HAS THE WRONG LENGTH FOR THE GIVEN TYPE.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the specified length is permissible for the associated type. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK096I THE TYPE STATEMENT HAS A MISSING DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Correct, delete, or insert the delimiter in question. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK101I THE DO STATEMENT OR IMPLIED DO HAS AN INVALID DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK102I THE BACKSPACE/REWIND/END FILE STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK104I THE BACKSPACE/REWIND/END FILE STATEMENT HAS A DATA SET REFERENCE NUMBER THAT IS EITHER A NON-INTEGER OR AN ARRAY NAME.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the specified data set reference number is either an unsigned integer constant or an integer variable of length 4. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK109I THE PAUSE STATEMENT HAS A MISPLACED DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the PAUSE statement contains either no delimiter or a literal constant enclosed in single quotes. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK110I THE PAUSE STATEMENT SPECIFIES A VALUE WHICH IS NEITHER A LITERAL NOR AN INTEGER CONSTANT.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the value. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK111I THE PAUSE STATEMENT HAS MORE THAN 255 CHARACTERS IN ITS LITERAL FIELD.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid characters. Make sure that continuation cards are correctly indicated. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK112I THE DICTIONARY HAS OVERFLOWED.

(Condition code - 16)

Programmer Response: Probable user error. If the SIZE parameter is specified incorrectly, (IEK410I) correct it and resubmit the job. If SIZE is coded correctly, then increase the SIZE value. Make sure that the region or partition in which the compiler is running is at least 10K larger than the specified SIZE value. If SIZE is not specified, increase the REGION size on the JOB or EXEC card. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the JOB statement.
- Have source and associated listing available.
- Have available the value of the SIZE parameter assigned at system generation time.

IEK115I THE VARIABLE RETURN STATEMENT HAS NEITHER AN INTEGER CONSTANT NOR VARIABLE FOLLOWING THE KEYWORD.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid constant or variable. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK116I THE DO STATEMENT OR IMPLIED DO HAS AN INVALID PARAMETER.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the DO variable is a non-subscripted integer variable. Check that the initial value, test value, and increment (if specified) are either unsigned integer constants greater than zero, or unsigned non-subscripted integer variables whose value is greater than zero. Verify that the test value does not exceed the allowable magnitude. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK117I THE BLOCK DATA STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK120I THE BLOCK DATA STATEMENT WAS NOT THE FIRST STATEMENT OF THE SUBPROGRAM.

(Condition code - 8)

Programmer Response: Probable user error. Delete the invalid delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK121I THE DATA STATEMENT HAS A VARIABLE WHICH HAS A NON-ALPHABETIC FIRST CHARACTER.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid variable, or correct any erroneous delimiters. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK122I THE DATA STATEMENT CONTAINS A SUBSCRIPTED VARIABLE WHICH HAS NOT BEEN DEFINED AS AN ARRAY.

(Condition code - 8)

Programmer Response: Probable user error. Either define the variable as an array, or correct the variable in the DATA statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK123I THE DATA STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK124I THE DATA STATEMENT HAS A VARIABLE WITH AN INVALID INTEGER SUBSCRIPT.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the subscript quantity contains only integer constants separated by commas. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK125I THE DATA STATEMENT HAS A VARIABLE WITH A SUBSCRIPT THAT CONTAINS AN INVALID DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK129I THE STATEMENT CONTAINS AN INVALID DATA CONSTANT.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the constant is valid for its designated class and/or type. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK132I THE DATA STATEMENT HAS AN INVALID DELIMITER IN ITS INITIALIZATION VALUES.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK133I THE DO STATEMENT CANNOT FOLLOW A LOGICAL IF STATEMENT.

(Condition code - 8)

Programmer Response: Probable user error. Change the DO to a GO TO n, where n is the statement label of the DO located elsewhere in the program unit. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK134I THE DO STATEMENT HAS AN INVALID INTEGER DO VARIABLE.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the integer DO variable is a non-subscripted integer variable. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK135I THE DO STATEMENT OR IMPLIED DO HAS AN INVALID TEST VALUE.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the test value is either an unsigned integer constant greater than zero, or an unsigned nonsubscripted integer variable greater than zero. Verify that the test value does not exceed $2^{31}-1$. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK136I THE NUMBER OF NESTED DO'S EXCEEDS THE COMPILER LIMIT.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the maximum level of nesting for DO loops, 25, is not exceeded. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK137I THE DO STATEMENT OR IMPLIED DO HAS AN INVALID INCREMENT VALUE.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the increment value is either an unsigned integer constant greater than zero, or an unsigned non-subscripted integer variable greater than zero. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK138I THE DO STATEMENT HAS A PREVIOUSLY DEFINED STATEMENT NUMBER SPECIFIED TO END THE DO RANGE.

(Condition code - 8)

Programmer Response: Probable user error. Verify that the statement number specified to end the DO range is an executable statement number appearing after the DO statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK139I A LOGICAL IF IS FOLLOWED BY ANOTHER LOGICAL IF OR A SPECIFICATION STATEMENT.

(Condition code - 8)

Programmer Response: Probable user error. Change the latter logical IF to a GO TO n, where n is the statement label of the logical IF located elsewhere in the program unit. Include the specification statement in the prescribed order prior to executable statements and statement function definitions, if any. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK140I THE IF STATEMENT BEGINS WITH AN INVALID CHARACTER.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid character. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK141I THE FORMAT STATEMENT DOES NOT END WITH A RIGHT PARENTHESIS.

(Condition code - 8)

Programmer Response: Probable user error. Correct any invalid delimiters and include the right

parenthesis. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK143I THE STATEMENT FUNCTION HAS AN ARGUMENT WHICH IS NOT A VARIABLE.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the arguments are non-subscripted variables. If the problem recurs, do the following defore calling IBM for programming support:

- Have source and associated listing available.

IEK144I THE STATEMENT FUNCTION HAS MORE THAN 20 ARGUMENTS.

(Condition code - 8)

Programmer Response: Probable user error. Redefine the arguments so that the 20 argument limit is not exceeded. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK145I THE STATEMENT FUNCTION HAS AN INVALID DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK146I THE STATEMENT FUNCTION HAS A MISPLACED EQUAL SIGN.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the misplaced equal sign. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK147I A STATEMENT FUNCTION DEFINITION MUST PRECEDE THE FIRST EXECUTABLE STATEMENT.

(Condition code - 8)

Programmer Response: Probable user error. Resequence the definition so that it follows only SUBPROGRAM, IMPLICIT, or other specification statements. Include a DIMENSION statement if it has been omitted. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK148I THE DIMENSIONED ITEM HAS A NON-INTEGER SUBSCRIPT.

(Condition code - 8)

Programmer Response: Probable user error. Verify that the subscript quantity is validly constructed. If the problem recurs, do the following before calling IBM for programmer support:

- Have source and associated listing available.

IEK149I A VARIABLE TO BE DIMENSIONED USING ADJUSTABLE DIMENSIONS MUST HAVE BEEN PASSED AS AN ARGUMENT AND MUST NOT APPEAR IN COMMON.

(Condition code - 8)

Programmer Response: Probable user error. Remove the variable name from the COMMON statement, if it has been entered in a COMMON statement. Place the variable in the argument list, if not already there. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK150I THE DIMENSIONED ITEM HAS AN INVALID DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK151I THE STATEMENT SPECIFIES A NON-VARIABLE TO BE DIMENSIONED.

(Condition code - 8)

Programmer Response: Probable user error. Correct the non-variable name. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK152I THE SUBPROGRAM STATEMENT HAS AN INVALID DELIMITER IN THE ARGUMENT LIST.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK153I THE STATEMENT HAS AN INVALID NAME SPECIFIED AS A FUNCTION REFERENCE.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the function has been correctly defined. Verify that the type of the name used for the reference agrees with the type of name used in the definition. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK156I THE I/O STATEMENT HAS AN INVALID NAME PRECEDING THE EQUAL SIGN.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid name. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK157I THE I/O STATEMENT HAS A NON-VARIABLE SPECIFIED AS A LIST ITEM.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the I/O list contains variable names, subscripted or unsubscripted array names, or array names accompanied by indexing specifications in the form of an implied DO. Verify that no function references or arithmetic expressions appear in the I/O list. Use the MAP option to determine the use of names in the program unit. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK158I THE I/O STATEMENT HAS AN IMPROPER PAIRING OF PARENTHESES IN AN IMPLIED DO, OR A NON-INTEGRAL INDEX.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that there are as many left parentheses as there are right parentheses. Correct any invalid index. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK159I THE FORMAT STATEMENT DOES NOT HAVE A STATEMENT NUMBER.

(Condition code - 4)

Programmer Response: Probable user error. Insert the required statement number. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK160I THE I/O STATEMENT HAS AN INVALID DELIMITER IN THE PARAMETERS.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid delimiter. If the problem recurs, do the following before

calling IBM for programming support:

- Have source and associated listing available.

IEK161I THE I/O STATEMENT HAS A DUPLICATE PARAMETER.

(Condition code - 8)

Programmer Response: Probable user error. Delete the duplicate parameter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK163I THE I/O STATEMENT HAS AN ARRAY WHICH IS NOT DIMENSIONED.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the subscripted array name or array name in the form of an IMPLIED DO has been previously declared in a DIMENSION statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK164I THE I/O STATEMENT HAS AN ARITHMETIC EXPRESSION OR A FUNCTION NAME SPECIFIED AS A LIST ITEM.

(Condition code - 8)

Programmer Response: Probable user error. Verify that no function references or arithmetic expressions are contained in the I/O list. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK165I THE I/O STATEMENT HAS A PARAMETER WHICH IS NOT AN ARRAY AND NOT A NAMELIST NAME.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid parameter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK166I THE I/O STATEMENT HAS A NON-INTEGER CONSTANT OR VARIABLE REPRESENTING THE DATA SET REFERENCE NUMBER.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the data set reference number is either an unsigned integer constant or an integer variable of length 4. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK167I THE STATEMENT HAS AN INVALID USE OF A STATEMENT FUNCTION NAME.

(Condition code - 8)

Programmer Response: Probable user error. Verify that the statement function name has been previously defined. Correct any invalid references. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK168I THE STATEMENT SPECIFIES AS A SUBPROGRAM NAME A VARIABLE WHICH HAS BEEN PREVIOUSLY USED AS A NON-SUBPROGRAM NAME.

(Condition code - 8)

Programmer Response: Probable user error. If the desired subprogram name duplicates a variable name, change the variable name and all references to it. Use the XREF option to determine where the variable occurs if the program unit contains many statements. If the subroutine name is in error, correct it. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement, and that the necessary DD statement is included.

IEK169I THE DIRECT ACCESS I/O STATEMENT MAY NOT SPECIFY A NAMELIST NAME.

(Condition code - 8)

Programmer Response: Probable user error. Change the data set reference number so that it refers to a valid sequential device. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK170I THE DIRECT ACCESS I/O STATEMENT HAS A NON-INTEGER SPECIFYING THE RECORD'S RELATIVE POSITION.

(Condition code - 8)

Programmer Response: Probable user error. Correct the invalid non-integer reference. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK171I THE NAME SPECIFIED FOR AN ENTRY POINT HAS ALREADY BEEN MULTIPLY DEFINED.

(Condition code - 8)

Programmer Response: Probable user error. Change the entry point name and all references to it so that duplication is eliminated. Determine if the name was erroneously used previously and correct it. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement, and that the necessary DD statement is included.

IEK176I THE I/O STATEMENT CONTAINS INVALID SYNTAX IN ITS IMPLIED DO.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that indexing parameters are correctly specified. Verify that there are no more than 20 implied DO's per statement. Correct any erroneous delimiters. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK192I THE STATEMENT HAS A LABEL WHICH IS SPECIFIED AS BOTH THE LABEL OF A FORMAT STATEMENT AND THE OBJECT OF A BRANCH.

(Condition code - 8)

Programmer Response: Probable user error. If the branch has been specified erroneously to a FORMAT statement, correct it. Correct or delete any misplaced labels. Use the XREF option for listings of the internal statement number of the statements in which the label is defined and referenced. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement, and that the necessary DD statement is included.

IEK193I THE STATEMENT NUMBER HAS BEEN PREVIOUSLY DEFINED.

(Condition code - 8)

Programmer Response: Probable user error. Change the statement number. Use the XREF option where many labels occur to determine which are unused. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement, and that the necessary DD statement is included.

IEK194I THE TYPE STATEMENT HAS A MISSING DELIMITER IN THE INITIALIZATION VALUES.

(Condition code - 8)

Programmer Response: Probable user error. Correct any invalid delimiters. Supply the missing delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK197I THE STOP STATEMENT HAS A NON-INTEGER CONSTANT AFTER THE KEYWORD.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the constant following the keyword is a string of 1 to 5 decimal digits, inclusive. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK199I THE SUBROUTINE OR FUNCTION STATEMENT WAS NOT THE FIRST STATEMENT.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that no statements except comments occur prior to the SUBROUTINE or FUNCTION statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK200I QUOTE LITERALS MAY APPEAR ONLY IN CALL, DATA INITIALIZATION, FUNCTION AND FORMAT STATEMENTS.

(Condition code - 8)

Programmer Response: Probable user error. Delete the invalid reference to the quote literals. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK202I THE STATEMENT HAS A VARIABLE WHICH HAS BEEN PREVIOUSLY DIMENSIONED. THE INITIAL DIMENSION FACTORS ARE USED.

(Condition code - 4)

Programmer Response: Probable user error. Delete the unnecessary or erroneous dimension specification. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK203I AN ENTRY STATEMENT MUST NOT APPEAR IN A MAIN PROGRAM. THE STATEMENT IS IGNORED.

(Condition code - 8)

Programmer Response: Probable user error. Delete the ENTRY statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK204I THE STOP STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 4)

Programmer Response: Probable user error. Correct the invalid delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK205I THE ASSIGNED OR COMPUTED GO TO HAS AN INVALID ELEMENT FOLLOWING THE CLOSING PARENTHESIS.

(Condition code - 4)

Programmer Response: Probable user error. Correct or delete the invalid element. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK206I THE STATEMENT HAS A NON-SUBSCRIPTED ARRAY ITEM.

(Condition code - 4)

Programmer Response: Probable user error. Correct the invalid array reference. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK207I THE CONTINUE STATEMENT DOES NOT END AFTER THE KEY WORD CONTINUE.

(Condition code - 4)

Programmer Response: Probable user error. Delete any code following the CONTINUE keyword. Make sure that a continuation has not been indicated on the statement immediately following the CONTINUE. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK208I THE CONTINUE STATEMENT DOES NOT HAVE A STATEMENT NUMBER.

(Condition code - 4)

Programmer Response: Probable user error. Delete the CONTINUE statement if no related diagnostic appears with respect to undefined statement numbers. If there is an undefined statement number related to a label omission on the CONTINUE, then correct the CONTINUE statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK209I THE STATEMENT HAS AN OCTAL CONSTANT SPECIFIED AS AN INITIAL VALUE. THE VALUE IS REPLACED BY ZERO.

(Condition code - 4)

Programmer Response: Probable user error. If the value of the octal constant is necessary, convert it to the appropriate hexadecimal equivalent. Verify that a leading "0" has not been inadvertently specified for a leading "0" in an initialization statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK211I THE STATEMENT HAS A COMPLEX CONSTANT WHOSE REAL CONSTANTS DIFFER IN LENGTH.

(Condition code - 4)

Programmer Response: Probable user error. Correct the constant so that both parts are either REAL*4 or REAL*8. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK212I THE BLOCK DATA SUBPROGRAM CONTAINS EXECUTABLE STATEMENT(S). THE EXECUTABLE STATEMENT(S) IS IGNORED.

(Condition code - 4)

Programmer Response: Probable user error. Delete the executable statements. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK222I THE EXPRESSION HAS A LITERAL WITH A MISSING DELIMITER.

(Condition code - 4)

Programmer Response: Probable user error. Correct invalid delimiters or insert the missing delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK224I THE STATEMENT AFTER AN ARITHMETIC IF, GO TO, OR RETURN HAS NO LABEL.

(Condition code - 4)

Programmer Response: Probable user error. Insert any necessary labels. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK225I A LABEL APPEARS ON A NON-EXECUTABLE STATEMENT. THE LABEL IS IGNORED.

(Condition code - 4)

Programmer Response: Probable user error. Delete the label. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK226I THE STATEMENT HAS A VARIABLE WITH MORE THAN SIX CHARACTERS. THE RIGHTMOST CHARACTERS ARE TRUNCATED.

(Condition code - 4)

Programmer Response: Probable user error. Delete extraneous characters, or insert any missing delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK229I ALL THE ARGUMENTS OF AN ARITHMETIC STATEMENT FUNCTION ARE NOT USED IN THE DEFINITION.

(Condition code - 4)

Programmer Response: Probable user error. If the definition is correct, then delete extraneous arguments. If arguments were omitted in the definition, then include them. Verify that the expression on the right contains as many distinct variables as there are distinct dummy arguments. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK302I THE EQUIVALENCE STATEMENT HAS EXTENDED COMMON BACKWARDS.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that any implicit equivalencing or assignment statements involving arrays do not create an extension such that elements are added before the beginning of an established COMMON block. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK303I THE EQUIVALENCE STATEMENT CONTAINS AN ARRAY WHICH IS NOT DIMENSIONED.

(Condition code - 8)

Programmer Response: Probable user error. Include the necessary subscript quantity for the array name. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK304I THE EQUIVALENCE STATEMENT HAS LINKED BLOCKS OF COMMON TOGETHER.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that implicit equivalencing does not link COMMON blocks together. Use the MAP option to determine locations of variable names in the COMMON blocks in question. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.
- Make sure that the MAP option is specified in the PARM field of the EXEC statement.

IEK305I THE EQUIVALENCE STATEMENT CONTAINS AN ARRAY WITH A SUBSCRIPT WHICH IS OUT OF RANGE.

(Condition code - 4)

Programmer Response: Probable user error. Correct the invalid subscript. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.
- Make sure that the XREF option is specified in the PARM parameter of the EXEC statement, and that the associated DD statement is included in the job stream.

IEK306I THE EQUIVALENCE STATEMENT HAS AN INCONSISTENCY.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the EQUIVALENCE statement does not contradict itself or any previously established equivalences. Verify that implicit equivalencing, if it occurs, does not create inconsistencies. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK307I THE DATA STATEMENT CONTAINS A VARIABLE THAT IS NOT REFERENCED.

(Condition code - 4)

Programmer Response: Probable user error. Make sure that the indicated variable has not inadvertently been omitted from a program unit. If not, delete the variable from the DATA statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK308I THE EQUIVALENCE STATEMENT HAS EQUIVALENCED TWO VARIABLES IN THE SAME COMMON BLOCK.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that source is correct. If necessary and if possible, replace one of the invalid variables with a variable not in COMMON. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.
- Make sure that the MAP option is specified in the PARM field of the EXEC statement.

IEK312I THE EQUIVALENCE STATEMENT CONTAINS AN EXTERNAL REFERENCE.

(Condition code - 8)

Programmer Response: Probable user error. Delete or correct the invalid externally referenced name in the EQUIVALENCE group. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK314I THE EQUIVALENCE STATEMENT MAY CAUSE WORD BOUNDARY ERRORS.

(Condition code - 4)

Programmer Response: Probable user error. Arrange variables in fixed descending order according to length, or force proper alignment with dummy variables. Construct the group so that the displacement of each variable in the group can be evenly divided by the reference number associated with the variable. Use the MAP option for information on variables and relative addresses. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.
- Make sure that the MAP option is specified in the PARM field of the EXEC statement.

IEK315I THE EQUIVALENCE STATEMENT WILL CAUSE WORD BOUNDARY ERRORS.

(Condition code - 4)

Programmer Response: Probable user error. Arrange variable in fixed descending order according to length or force proper alignment with dummy variables. Construct the group so that the displacement of each variable in the group can be evenly divided by the reference number associated with the variable. Use the MAP option for information on variables and relative addresses. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK317I THE BLOCK DATA PROGRAM DOES NOT CONTAIN A COMMON STATEMENT.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that all elements of a COMMON block in any main program or subprogram are listed in a COMMON statement in

the BLOCK DATA subprogram. If the problem recurs, do the following before calling IBM for progamming support:

- Have source and associated listing available.

IEK318I THE DATA STATEMENT IS USED TO ENTER DATA INTO COMMON OUTSIDE A BLOCK DATA SUBPROGRAM.

(Condition code - 8)

Programmer Response: Probable user error. Delete the invalid reference to the variable in COMMON. Include the reference in a BLOCK DATA subprogram or in an assignment statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK319I DATA IS ENTERED INTO A LOCAL VARIABLE IN A BLOCK DATA PROGRAM.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that the variable appears in COMMON. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK320I DATA MAY NOT BE ENTERED INTO A VARIABLE WHICH HAS BEEN PASSED AS AN ARGUMENT.

(Condition code - 8)

Programmer Response: Probable user error. Delete any dummy arguments that appear in the data initialization list. Make sure source is correct. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK322I THE COMMON STATEMENT MAY CAUSE WORD BOUNDARY ERRORS.

(Condition code - 4)

Programmer Response: Probable user error. Arrange variables in fixed descending order according to length, or force proper alignment with dummy variables. Construct the block so that the displacement of each variable can be evenly divided by the reference number associated with the variable. Use the MAP option for information on the relative address of each variable in the block. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.
- Make sure that the MAP option is specified in the PARM field of the EXEC statement.

IEK323I THE COMMON STATEMENT WILL CAUSE A WORD BOUNDARY ERROR.

(Condition code - 4)

Programmer Response: Probable user error. Arrange variable in fixed descending order according to length, or force proper alignment with dummy variables. Construct the block so that the displacement of each variable can be evenly divided by the reference number associated with the variable. Use the MAP option for information on the relative address of each variable in the block. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.
- Make sure that the MAP option is specified in the PARM field of the EXEC statement.

IEK332I THE STATEMENT NUMBER IS UNDEFINED.

(Condition code - 8)

Programmer Response: Probable user error. Correct the source so that a valid statement number is referenced. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK334I THE COMMON STATEMENT HAS A VARIABLE WITH A VARIABLE DIMENSION.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that a subscript quantity contains only 1 through 7 unsigned integer constants separated by commas. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK350I THE DATA STATEMENT HAS A MISSING PARENTHESIS.

(Condition code - 8)

Programmer Response: Probable user error. Correct any invalid delimiters and insert the appropriate parenthesis. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK351I THE DATA INITIALIZATION VALUE IS LARGER THAN THE VARIABLE OR ARRAY ELEMENT - TRUNCATION OR SPILL WILL OCCUR.

(Condition code - 4)

Explanation: An array or variable was initialized with a constant whose length was greater than the length of an array element. If the constant was specified as the first element in a non-subscripted array, part of the constant will spill over into the succeeding array element(s). If the constant was specified as other than the first element in a non-subscripted array, or if it was specified as any element in a subscripted array, the constant will be truncated.

Programmer Response: Probable user error. If spill is not desired, make sure that the length of a constant specified does not exceed the length of the element. If truncation is not desired, make sure that the length of any constant specified as a subsequent element in the array does not exceed the element length. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK352I THE DATA STATEMENT HAS TOO MANY INITIALIZATION VALUES.

(Condition code - 4)

Programmer Response: Probable user error. Make sure that a one-to-one correspondence exists between the total number of elements specified or implied in the data list and the total number of constants specified by the corresponding list embedded in slashes. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK353I THE DIMENSION STATEMENT HAS A VARIABLE WHICH HAS A SUBSCRIPT OF REAL MODE.

(Condition code - 8)

Programmer Response: Probable user error. Verify that all subscripts are integer. Make sure that only a combination of integer and real mixed mode expressions occurs, if mixed mode is present. Check that no subscript rules are violated. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK354I A VARIABLE TO BE DIMENSIONED USING ADJUSTABLE DIMENSIONS MUST HAVE BEEN PASSED AS AN ARGUMENT AND MUST NOT APPEAR IN COMMON.

(Condition code - 8)

Programmer Response: Probable user error. If the variable has not been entered in a COMMON statement, remove it. If the variable is not already in the argument list, place it there. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK355I ADCON TABLE EXCEEDED.

(Condition code - 16)

Programmer Response: Restructure any complex statement into less involved statements. If the problem recurs, do the following

before calling IBM for programming support:

- Have source and associated listing available.

IEK356I A PARAMETER CANNOT ALSO BE IN COMMON.

(Condition code - 8)

Programmer Response: Probable user error. Correct the source and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK357I THE ARRAY HAS AN INCORRECT ADJUSTABLE DIMENSION.

(Condiiton code - 8)

Programmer Response: Probable user error. Correct invalid adjustable dimension. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK358I THE ADJUSTABLE DIMENSION IS NOT PASSED AS AN ARGUMENT OR IN COMMON.

(Condition code - 8)

Programmer Response: Probable user error. Either include the adjustable dimension in an argument list, or place it in COMMON. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.
- Make sure that the XREF option is specified in the PARM field and that the associated DD statement is included in the job stream.

IEK402I OPEN ERROR ON ddname

Explanation: The data set corresponding to the ddname cannot be opened. If either the EDIT or the XREF option has been requested, the corresponding SYSUT1 or SYSUT2 DD card has not been found.

(Condition code - 16)

Programmer Response: Probable user error. Make sure that appropriate DD cards are included in correct sequence with necessary keyword and/or positional parameters. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the JOB statement.
- Have source and associated listing available.

IEK403I OPEN ERROR ON SYSPRINT

(Condition code - 16)

Programmer Response: Probable user error. Check the SYSPRINT DD statement and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the JOB statement.
- Have source and associated listing available.

IEK404I SYNCHRONOUS ERROR ON SYSPRINT

(Condition code - 16)

Programmer Response: Probable user error. Check the SYSPRINT DD statement and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the JOB statement.
- Have source and associated listing available.

IEK410I AN INVALID SIZE PARAMETER HAS BEEN SPECIFIED. IT WILL BE IGNORED.

(Condition code - 4)

Programmer Response: Probable user error. Correct the SIZE parameter. Make sure that it is specified within the allowable range, and that the size of the region or partition in which the compiler is running is at least 10K larger than the specified SIZE value. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified in the JOB statement.
- Have source and associated listing available.

IEK500I AN ARGUMENT TO A FORTRAN SUPPLIED FUNCTION IS OF THE WRONG TYPE. THE FUNCTION IS ASSUMED TO BE USER DEFINED.

(Condition code - 4)

Programmer Response: Probable user error. If the function is user-suppied, make sure the function name appears in an EXTERNAL statement. Make sure that an argument mode is correct. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK501I THE EXPRESSION HAS A COMPLEX EXPONENT.

(Condition code - 8)

Programmer Response: Probable user error. Correct the exponent so that it is integral with a complex base, and otherwise integral or real. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK502I THE EXPRESSION HAS A BASE WHICH IS COMPLEX BUT THE EXPONENT IS NON-INTEGER.

(Condition code - 8)

Programmer Response: Probable user error. Correct the expression so that a complex base has an integer exponent. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK503I A NON-SUBSCRIPTED ARRAY ITEM APPEARS IMPROPERLY WITHIN A FUNCTION REFERENCE OR A CALL.

(Condition code - 8)

Programmer Response: Probable user error. Correct the invalid array item following before calling IBM for programming support:

- Have source and associated listing available.

IEK504I THE BASE AND/OR EXPONENT IS A LOGICAL VARIABLE.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that base and/or exponent are only of type real, integer, or complex. Check placement of parentheses. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK505I THE INPUT/OUTPUT STATEMENT REFERS TO THE STATEMENT NUMBER OF A NON-FORMAT STATEMENT.

(Condition code - 8)

Programmer Response: Probable user error. Correct the invalid statement number. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK506I THERE IS A MISSING OPERAND PRECEEDING A RIGHT PARENTHESIS.

(Condition code - 8)

Programmer Response: Probable user error. Correct the statement problem recurs, do the following

before calling IBM for programming support:

- Have source and associated listing available.

IEK507I A NON-SUBSCRIPTED ARRAY ITEM IS USED AS AN ARGUMENT TO AN IN-LINE FUNCTION.

(Condition code - 8)

Programmer Response: Probable user error. Correct the array item. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK508I THE NUMBER OF ARGUMENTS TO AN IN-LINE FUNCTION IS INCORRECT.

(Condition code - 8)

Programmer Response: Probable user error. Make sure that necessary delimiters are indicated. Correct or delete items in the argument list. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK509I THE PROGRAM DOES NOT END WITH A STOP, RETURN, OR GO TO.

(Condition code - 4)

Programmer Response: Probable user error. Insert the necessary terminal statement. If the problem recurs, do the following before calling IBM for programmer support:

- Have source and associated listing available.

IEK510I THE EXPRESSION HAS A LOGICAL OPERATOR WITH A NON-LOGICAL OPERAND.

(Condition code - 8)

Programmer Response: Probable user error. Correct the operand. Make sure that a logical primary or logical expressions have correct form. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK512I THE LOGICAL IF DOES NOT CONTAIN A LOGICAL EXPRESSION.

(Condition code - 8)

Programmer Response: Probable user error. Correct the statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK515I THE EXPRESSION HAS A RELATIONAL OPERATOR WITH A COMPLEX OPERAND.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the complex operand. Equivalence a real array of two elements to a complex variable to permit use of the relational operator, if necessary. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK516I THE ARITHMETIC IF CONTAINS A COMPLEX EXPRESSION.

(Condition code - 8)

Programmer Response: Probable user error. Correct the expression. Convert the expression, if possible, to a permissible type. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK520I THERE IS A COMMA IN AN INVALID POSITION.

(Condition code - 8)

Programmer Response: Probable user error. Delete or reposition the comma. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK521I THE EXPRESSION HAS AT LEAST ONE EXTRA RIGHT PARENTHESIS.

(Condition code - 8)

Programmer Response: Probable user error. Delete or correct extraneous and invalid parentheses. Make sure parentheses are balanced. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK522I THE EXPRESSION HAS AT LEAST ONE TOO FEW RIGHT PARENTHESES.

(Condition code - 8)

Programmer Response: Probable user error. Correct or add needed parentheses. Make sure parentheses are balanced. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK523I THE EQUAL SIGN IS IMPROPERLY USED.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid equal sign. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK524I THE EXPRESSION HAS AN OPERATOR MISSING AFTER A RIGHT PARENTHESIS.

(Condition code - 8)

Programmer Response: Probable user error. Insert the missing operator or delete the erroneous parenthesis. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK525I THE EXPRESSION USES A LOGICAL OR RELATIONAL OPERATOR INCORRECTLY.

(Condition code - 8)

Programmer Response: Probable user error. Either correct the logical or relational operator, or change invalid operand expressions. Make sure operators are preceded and followed by a period. Verify that expressions precede operators where required. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK529I A FUNCTION NAME APPEARING AS AN ARGUMENT HAS NOT BEEN DECLARED EXTERNAL.

(Condition code - 8)

Programmer Response: Probable user error. Insert the required EXTERNAL statement, or delete or correct the invalid function reference. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK530I THE EXPRESSION HAS A VARIABLE WITH AN IMPROPER NUMBER OF SUBSCRIPTS.

(Condition code - 8)

Programmer Response: Probable user error. Check for all necessary delimiters. Make sure that there are as many subscripts as are declared in the associated specification statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK531I THE EXPRESSION HAS A STATEMENT FUNCTION REFERENCE WITH AN IMPROPER NUMBER OF ARGUMENTS.

(Condition code - 8)

Programmer Response: Probable user error. Correct the invalid function reference. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK541I AN ARGUMENT TO A LIBRARY FUNCTION HAS AN INVALID TYPE.

(Condition code - 8)

Programmer Response: Probable user error. Corrrect the invalid argument. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK542I A LOGICAL EXPRESSION APPEARS IN INVALID CONTEXT.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the source statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK550I PUSHDOWN, ADCON, OR ASF ARGUMENT TABLE EXCEEDED.

(Condition code - 16)

Programmer Response: Change the program structure. If there are many subroutine references in a program unit, subdivide the program unit. Restructure deeply nested expressions or eliminate some ASF arguments where many occur, if possible. Where parentheses are deeply nested, restructure the source statement, if possible, to eliminate some of the nesting. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK552I SOURCE PROGRAM IS TOO LARGE.

(Condition code - 16)

Programming Response: Subdivide the program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK570I TABLE EXCEEDED. OPTIMIZATION DOWNGRADED.

(Condition code - 0)

Explanation: Probable user error. The program is too large to permit optimization. This is a warning message and appears in the source listing at the point where the table (RMAJOR) overflows. The compiler performs OPT=1 register allocation only; no other optimization is performed.

Programmer Response: Either the program should be segmented or the size of the table RMAJOR should be increased. RMAJOR may be increased by increasing the SIZE option of the FORTRAN macro at system generation time. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK580I COMPILER ERROR.

(Condition code - 16)

Explanation: One of the following four conditions occurred: an invalid adjective code was detected; an illegal element length was detected; no equivalence group was found; an unusual primary adjective code was detected.

Programmer Response: Probable user error. Make sure that the source program is correct. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK600I INTERNAL COMPILER ERROR. LOGICALLY IMPOSSIBLE BRANCH TAKEN IN A COMPILER SUBROUTINE.

(Condition code - 16)

Programmer Response: Make sure that the source code is correct. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK610I THE STATEMENT NUMBER OR GENERATED LABEL IS UNREACHABLE.

(Condition code - 4)

Note: This message is generated only if OPT=2 is specified in the EXEC statement.

Programmer Response: Probable user error. Make sure that control statements indicate correct branch targets. Verify that an unlabeled STOP, RETURN, or GO TO does not immediately follow any one of these same three source statements. Make sure that the statement following an arithmetic IF is labeled. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK620I THE STATEMENT NUMBER OR GENERATED LABEL IS A MEMBER OF AN UNREACHABLE LOOP.

(Condition code - 4)

Note: This message is generated only if OPT=2 is specified in the EXEC statement.

Programmer Response: Probable user error. Make sure that control statements indicate correct branch targets. Correct labels so that the loop may be the target of a branch. Delete invalid terminal source statements. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK630I INTERNAL TOPOLOGICAL ANALYSIS TABLE EXCEEDED.

(Condition code - 16)

Programmer Response: Insert statement numbers where a large span of source code exists without statement numbers. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK640I COVERAGE BY BASE REGISTER 12 IN OBJECT MODULE EXCEEDED.

(Condition code - 16)

Programmer Response: Probable user error. Segment the program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK650I INTERNAL ADCON TABLE EXCEEDED.

(Condition code - 16)

Programmer Response: Segment the program and recompile. If the default value for the FORTRAN macro SIZE option was used, regenerate using a larger SIZE value. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK660I INTERNAL COMPILER ERROR. TEMPORARY FETCHED BUT NEVER STORED.

(Condition code - 16)

Programmer Response: Make sure that source is correct. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK661I INTERNAL COMPILER ERROR. UNABLE TO FREE A REGISTER.

(Condition code - 16)

Programmer Response: Segment large spans of unlabeled source code into smaller extents delimited by statement numbers. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK662I INTERNAL COMPILER ERROR. TEMPORARY NOT ENTERED IN ASSIGNMENT TABLE.

(Condition code - 16)

Programmer Response: Make sure that source is correct. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK670I LOGICALLY IMPOSSIBLE BRANCH TAKEN IN A COMPILER SUBROUTINE.

(Condition code - 16)

Programmer Response: Make sure that source code is correct. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK671I LOGICALLY IMPOSSIBLE BRANCH TAKEN IN A COMPILER SUBROUTINE.

(Condition code - 16)

Programmer Response: Make sure that source code is correct. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK710I THE FORMAT STATEMENT SPECIFIES A FIELD WIDTH OF ZERO.

(Condition code - 8)

Programmer Response: Probable user error. Correct the invalid field width. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK720I THE FORMAT STATEMENT CONTAINS AN INVALID CHARACTER.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid character. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK730I THE FORMAT STATEMENT HAS UNBALANCED PARENTHESES.

(Condition code - 8)

Programmer Response: Probable user error. Correct the statement. Delete any unnecessary or insert missing parentheses. Make sure wH specifications are correct. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK740I THE FORMAT STATEMENT HAS NO BEGINNING LEFT PARENTHESIS.

(Condition code - 8)

Programmer Response: Probable user error. Correct the source statement. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK750I THE FORMAT STATEMENT SPECIFIES A COUNT OF ZERO FOR A LITERAL FIELD.

(Condition code - 8)

Programmer Response: Probable user error. Correct the incorrectly specified count. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK760I THE FORMAT STATEMENT CONTAINS A MEANINGLESS NUMBER.

(Condition code - 8)

Programmer Response: Probable user error. Correct or delete the invalid number. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK770I THE FORMAT STATEMENT HAS A MISSING DELIMITER.

(Condition code - 8)

Programmer Response: Probable user error. Either correct or delete invalid delimiters, or insert the missing delimiter. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK780I THE FORMAT STATEMENT CONTAINS A NUMERIC SPECIFICATION GREATER THAN 255.

(Condition code - 8)

Programmer Response: Probable user error. Correct the invalid numeric specification. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK790I THE FORMAT STATEMENT CONTAINS GROUP FORMAT SPECIFICATIONS NESTED TO A LEVEL GREATER THAN TWO

Explanation: The compiler detected more than two left parentheses without intervening right parentheses, thereby indicating a nesting level exceeding two. The nesting is encoded as written.

(Condition code - 8).

Programmer Response: Probable user error. Correct the invalid specification. If the problem recurs do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK800I SOURCE PROGRAM IS TOO LARGE.

(Condition code - 16)

Programmer Response: Segment the program and recompile. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IEK1000I INTERNAL COMPILER ERROR

(Condition code - 4)

Explanation: An erroneous error number has been placed in the error table.

Programmer Response: Probable user error. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

LOAD MODULE EXECUTION DIAGNOSTIC MESSAGES

The load module produces three types of diagnostic messages:

- Program interrupt messages
- Execution error messages
- Operator message

Program Interrupt Messages

Program interrupt messages containing the old Program Status Word (PSW) are written when an exception occurs. The format of the program interrupt message when the extended error message facility has not been specified at system generation time is given in Figure 107. Program interrupt messages IHC207I, IHC208I, and IHC209I are produced only when the extended error message facility has been specified. The format of these messages can be found in the section "Execution Error Messages."

Note: Codes 4, 5, 6, and 7 are associated with the execution-time adjustment of boundary alignment errors and appear only when the system is generated to provide boundary alignment adjustment; i.e., when BOUNDRY=ALIGN is specified in the FORTLIB macro instruction during system generation (see the System Generation publication).

The letter A in the message indicates that boundary adjustment has taken place.The letter P in the message indicates that the interruption was precise. This will always be the case for non-specification interrupt messages in FORTRAN except when using machines with special hardware on which imprecise interruptions may occur. The eighth character in the PSW (i.e., 4, 5, 6, 7, 9, C, D, or F) represents the code number (in hexadecimal) associated with the type of interruption. The following text describes these interruptions.

Protection Exception: The protection exception (code 4), is recognized when the key of an operand in storage does not match the protection key in the PSW. A message is issued only if a specification exception (code 6) has already been recognized in the same instruction. Otherwise, the job terminates abnormally.

If the extended error message facility is specified, the following information is provided.

```
IBCOM - PROGRAM INTERRUPT - ALIGNMENT OLD
 PSW IS xxxxxxx4xxxxxxxx
```

Supplemental Data: None.

Standard Corrective Action: Continue execution at point of interrupt.

Programmer Response: Probable user error. If the job has been terminated with a completion code of SYSTEM=0C6 (specification interrupt), correct the source causing boundary misalignment. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP, LIST, and DUMP have been specified as parameters on the EXEC statement and provide the necessary GO.SYSUDUMP or GO.SYSABEND DD statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

Addressing Exception: The addressing exception (code 5) is recognized when the address of the data is outside of the available storage for the particular installation. A message is issued only if a specification exception (code 6) has already been recognized in the same instruction. Otherwise, the job terminates abnormally.

If the extended error message facility is specified, the following information is provided.

```
IBCOM - PROGRAM INTERRUPT - ALIGNMENT OLD
 PSW IS xxxxxxx5xxxxxxxx
```

Supplemental Data: None.

Standard Corrective Action: Continue execution at point of interrupt.

Programmer Response: Probable user error. If the job has been terminated with a completion code of SYSTEM=0C6 (specification interrupt), correct the source causing boundary misalignment. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP, LIST, and DUMP have been specified as parameters on the EXEC statement and provide the necessary GO.SYSUDUMP or GO.SYSABEND DD statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

Specification Exception: The specification exception (code 6) is recognized when a data address does not specify an integral boundary for that unit of information. For example, a specification error would occur during execution of the following instructions.

```
REAL*8 D, E
COMMON A, B, C
EQUIVALENCE (B, D)
D = 3.0D02
```

Note: If an instruction contains a boundary violation, a specification interrupt occurs and the message is issued with code 6. The boundary adjustment routine is invoked if the BOUNDRY=ALIGN option was specified in the FORTLIB macro instruction during system generation. If an instruction which has been processed for a boundary violation also contains a protection, addressing, or data error, the interrupt message is reissued with the appropriate code (4, 5, or 7). The job then terminates because both a

```
IHC210I  PROGRAM INTERRUPT (A) -- OLD PSW IS xxxxxxx { 4 } xxxxxxxx
                           (P)                       { 5 }
                                                     { 6 }
                                                     { 7 }
                                                     { 9 }
                                                     { C }
                                                     { D }
                                                     { F }
```

Figure 107. Program Interrupt Message Format Without Extended Error Message Facility

specification error and a protection, addressing, or data error have been detected. The completion code in the dump indicates that the job terminated because of the specification error.

If the extended error message facility is specified, the following information is provided.

```
IBCOM - PROGRAM INTERRUPT - ALIGNMENT OLD
 PSW IS xxxxxxx6xxxxxxxx
```

Supplemental Data: None.

Standard Corrective Action: Continue execution at point of interrupt.

Programmer Response: Probable user user error. Make sure that proper alignment of variables is guaranteed. Arrange variables in fixed descending order according to length, or force proper alignment with dummy variables. Construct COMMON blocks so that the displacement of each variable can be evenly divided by the reference number associated with the variable. Use the MAP option for information on the relative address of each variable in the block. Make sure that EQUIVALENCE statements do not cause misalignment. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP, LIST, and DUMP have been specified as parameters on the EXEC statement and provide the necessary GO.SYSUDUMP or GO.SYSABEND DD statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

Data Exception: The data exception (code 7), is recognized when the sign and digit codes for a CONVERT TO BINARY instruction are incorrect. A message is issued only if a specification exception (code 6) has already been recognized in the same instruction. Otherwise, the job terminates abnormally.

If the extended error message facility is specified, the following information is provided.

```
IBCOM - PROGRAM INTERRUPT - ALIGNMENT OLD
 PSW IS xxxxxxx7xxxxxxxx
```

Supplemental Data: None.

Standard Corrective Action: Continue execution at point of interrupt.

Programmer Response: Probable user error. If the job has been terminated with a completion code of SYSTEM=0C6 (specification interrupt), correct the source causing boundary misalignment. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP, LIST, and DUMP have been specified as parameters on the EXEC statement and provide the necessary GO.SYSUDUMP or GO.SYSABEND DD statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

Fixed-Point-Divide Exception: The fixed-point-divide exception, assigned code number 9, is recognized when division of a fixed-point number by zero is attempted. A fixed-point divide exception will occur during execution of the following statement:

```
K=I/J
```

where: I=7 and J=0

Note: When dealing with large numbers, the programmer should be aware that fixed-point overflow does not cause an interrupt and any overflow causes incorrect results. No error message is issued.

Exponent-Overflow Exception: The exponent-overflow exception, assigned code number C, is recognized when the result of a floating-point addition, subtraction, multiplication, or division is greater than or equal to 16^{63} (approximately 7.2 x 10^{75}). For example, an exponent-overflow will occur during execution of the statement:

```
A = 1.0E+75 + 7.2E+75
```

When the interrupt occurs, the result register contains a floating-point number whose fraction is normalized and whose sign is correct. However, the number is not usable for further computation since its characteristic field no longer reflects the true exponent. The content of the result register as it existed when the interrupt occurred is printed following the program interrupt message with the format:

```
REGISTER CONTAINED hhhhhhhhhhhhhhhh
```

where: hhhhhhhhhhhhhhhh is the floating-point number in hexadecimal notation.

If the improved floating-point engineering change is not in effect, the register content cannot be used to calculate the true value.

If the improved floating-point engineering change is in effect, exponent overflow causes "exponent wraparound" - i.e., the characteristic field represents an exponent that is 128 smaller than the correct one. Treating bits 1 to 7 (the exponent characteristic field) of the floating-point number as a binary integer, the true exponent may be computed as follows:

TE = (Bits 1 to 7) + 128 - 64

Before program execution continues, the FORTRAN library sets the result register to the largest possible floating-point number that can be represented in short precision ($16^{63}*(1-16^{-6})$) or in long precision ($16^{63}*(1-16^{-14})$), but the sign of the result is not changed. The condition code is not altered.

Exponent-Underflow Exception: The exponent-underflow exception, assigned code number D, is recognized when the result of a floating-point addition, subtraction, multiplication, or division is less than 16^{-65} (approximately $5.4x10^{-79}$). For example, an exponent-underflow exception will occur during execution of the statement:

A = 1.0E-50 * 1.0E-50

Although exponent underflows are maskable, FORTRAN jobs are executed without the mask so that the library will handle such interrupts.

When the interrupt occurs, the result register contains a floating-point number whose fraction is normalized and whose sign is correct. However, the number is not usable for further computation since its characteristic field no longer reflects the true exponent. The content of the result register as it existed when the interrupt occurred is printed following the program interrupt message with the format:

REGISTER CONTAINED hhhhhhhhhhhhhhhh

where: hhhhhhhhhhhhhhhh is the floating-point number in hexadecimal notation.

If the improved floating-point engineering change is not in effect, the exponent underflow always leaves a zero in the result register.

If the improved floating-point engineering change is in effect, exponent underflow causes "exponent wraparound" - i.e., the characteristic field represents an exponent that is 128 larger than the correct one. Treating bits 1 to 7 (the exponent characteristic field) of the floating-point number as a binary integer, the true exponent may be computed as follows:

TE = (Bits 1 to 7) - 128 - 64

Before program execution continues, the library sets the result register to a true zero of correct precision. If the interrupt resulted from a floating-point addition or subtraction operation, the condition code is set to zero to reflect the setting of the result register.

Note: The System/360 Operating System FORTRAN programmer who wishes to take advantage of the "exponent wraparound" feature and handle the interrupt in his own program must call an assembly language subroutine to issue a SPIE macro instruction that will override the FORTRAN interruption routine.

Floating-Point-Divide Exception: The floating-point-divide exception, assigned code number F, is recognized when division of a floating-point number by zero is attempted. For example, a floating-point divide exception will occur during execution of the following statements:

C=A/B

where: A=1.0 and B=0.0

Execution Error Messages

Execution error messages have the form:

```
IHCxxxI [message text]
   TRACEBACK FOLLOWS-ROUTINE ISN REG.  14,
     REG.  15, REG.  0, REG.  1
```

The facility for error detection and diagnostic messages is controlled by a system generation option. When the parameter OPTERR=INCLUDE is specified in the FORTLIB macro instruction at system generation time, the extended error handling facility is made available during program execution. This facility is not made available if OPTERR=EXCLUDE is specified or if no parameter is specified at system generation time.

The description of each diagnostic message contains the error code, the abbreviated name for the origin of the error, and an explanation describing the type of error. In addition, supplemental

data is provided and standard corrective action to be taken to correct the error is described. Supplementary data and standard corrective action are applicable only if OPTERR=INCLUDE was specified.

Variable information in the message is shown as X, and in the corrective action descriptions, * denotes the largest possible number that can be represented in floating point. For all load module execution error messages except IHC210I, a condition code of 16 is generated and the job step is terminated unless the OPTERR=INCLUDE parameter was specified.

The abbreviated name for the origin of the error is:

IBC - IHCFCOMH routine (performs interruption, and error procedures).

FIOCS - IHCFIOSH routine (performs I/O operations for FORTRAN load module execution).

NAMEL - IHCNAMEL routine (performs namelist processing).

DIOCS - IHCDIOSE routine (performs direct access I/O operations for FORTRAN load module execution).

IBERR - IHCIBERH routine (performs the processing of errors detected during execution of the load modules.)

LIB - SYS1.FORTLIB. In the explanation of the messages, the module name is given followed by the entry point name(s) enclosed in parentheses.

FCVTH - IHCFCVTH routine (performs conversions).

IHC207I IBCOM - PROGRAM INTERRUPT-OVERFLOW OLD PSW IS xxxxxxxCxxxxxxxx REGISTER CONTAINED X

This message is produced only when the extended error message facility is specified.

Supplemental Data: The floating-point number before alteration.

Standard Corrective Action: Continue execution at point of interrupt with result register set to the largest possible floating-point number that can be represented in short precision ($16^{63}*(1-16^{-6})$) or in long precision ($16^{63}*(1-16^{-14})$).

Programmer Response: Probable user error. Make sure that a variable or variable expression does not exceed the allowable magnitude. Verify that all variables have been initialized correctly in previous source statements, and have not been inadvertently modified in intermediate source. If the problem recurs, do the following before calling IBM for programming support.

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and the necessary DD statement is included.
- Have source and associated listing available.

IHC208I IBCOM - PROGRAM INTERRUPT-UNDERFLOW UNDERFLOW OLD PSW IS xxxxxxxDxxxxxxxx REGISTER CONTAINED X

This message is produced only when the extended error message facility is specified.

Supplemental Data: The floating-point number before alteration.

Standard Corrective Action: Continue execution at point of interrupt with result register set to a true zero of correct precision.

Programmer Response: Probable user error. Make sure that a variable or variable expression is not smaller than the allowable magnitude. Verify that all variables have been initialized correctly in previous source statements and have not been inadvertently modified in intermediate source. If the problem recurs, do the following before calling IBM for programming support.

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC209I IBCOM - PROGRAM INTERRUPT-DIVIDE CHECK OLD PSW IS xxxxxxx{9/F}xxxxxxxx

This message is produced only when the extended error message facility is specified.

Supplemental Data: None.

Standard Corrective Action: Leave register unmodified.

Programmer Response: Probable user error. Either correct the source where division by zero is occurring, or modify previous source statements to test for the possibilities or bypass the illegal division. If the problem recurs, do the following before calling IBM for programming support.

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC210I Message IHC210I is a program interrupt message. For a description, see Figure 107 and the section "Program Interrupt Messages."

IHC211I

Explanation: IBC -- An invalid character has been detected in a FORMAT statement.

If the extended error message facility is specified, the following information is provided:

IBCOM - ILLEGAL COMPILED FORMAT CHARACTER SPECIFIED
or
IBCOM - ILLEGAL VARIABLE FORMAT CHARACTER SPECIFIED X

Supplemental Data: Character in error.

Standard Corrective Action: Format field treated as an end of format.

Programmer Response: Probable user error. Make sure that all format specifications read in at object time are valid. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Have source and associated listing available.

IHC212I

Explanation: IBC -- An attempt has been made to read or write a record, under FORMAT control, that exceeds the buffer length.

If the extended error message facility is specified, the following information is provided:

IBCOM - FORMATTED I/O, END OF RECORD ON UNIT X

Supplemental Data: Unit number.

Standard Corrective Action: For a read, ignore remainder of I/O list; for a write, start new record with no control character.

Programmer Response: Probable user error. If the error occurs on input, verify that a FORMAT statement does not define a FORTRAN record longer than the record referred to in the data set. If reading in data, either keep a counter to avoid exceeding end of record or file, or insert an END= parameter or on the READ statement for appropriate transfer of control on end of data set. No record to be punched should be specified as longer than 80 characters. For printed output make sure that no specification is longer than the printer's line length. Check all DD statements. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified on the JOB statement.
- Have source and associated listing available.
- Make sure that LIST has been specified as a parameter on the EXEC statement.

IHC213I

Explanation: IBC -- The input list in an I/O statement without a FORMAT specification is larger than the logical record.

If the extended error message facility is specified, the following information is provided:

IBCOM - UNFORMATTED READ, END OF RECORD ON UNIT X

Supplemental Data: Unit number.

Standard Corrective Action: Ignore remainder of I/O list.

Programmer Response: Probable user error. Either keep a counter to avoid exceeding end of record or file, or insert an END= parameter on the READ statement for appropriate transfer of control on end of data set. Check all DD statements. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified on the JOB statement.
- Have source and associated listing available.
- Make sure that LIST has been specified as a parameter on the EXEC statement.

IHC214I

Explanation: FIOCS -- For unformatted records read or written in sequentially organized data sets, the record format (RECFM) specification must include the characters VS (variable spanned); any of the optional characters (B, A, M, or T) may be specified with the characters VS. This message appears if the programmer has coded RECFM=V, RECFM=U, or RECFM=F.

If the extended error message facility is specified, the following information is provided:

FIOCS - UNFORMATTED I/O, RECORD FORMAT SPECIFIED AS F, U, OR V ON UNIT X

Supplemental Data: Unit number.

Standard Corrective Action: For read, ignore I/O request; for write, change record form to VS.

Programmer Response: Probable user error. Correct the RECFM subparameter. Change a V (variable) or U (undefined) or F (fixed) specification to VS. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Have source and associated listing and the associated job stream available.
- Make sure that MSGLEVEL=(1,1) was specified on the JOB statement.

IHC215I CONVERT - ILLEGAL DECIMAL CHARACTER X

Explanation: An invalid character exists for the decimal input corresponding to an I, E, F, or D format code.

Supplemental Data: Display the record in which character appeared.

Standard Corrective Action: Zero replaces the character encountered.

Note: If the standard or corrective user action results in a null format, no output will result. If the FORMAT statement is terminated in such a way that no conversion type is called for, an alpha-
numeric literal may be repeated for each list item.

Programmer Response: Probable user error. If an IHC214I message has occurred previously, correct the source causing the error. Otherwise, make sure that all decimal input is valid. Correct any FORMAT statements specifying decimal input where character should be indicated. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Have source and associated listing available.

IHC216I SLITE-SLITET X IS AN ILLEGAL VALUE

Explanation: LIB -- An invalid sense light number was detected in the argument list in a call to the SLITE or SLITET subroutine.

Supplemental Data: The sense light value supplied.

Standard Corrective Action: For SLITE, no action; for SLITET, return OFF indication, i.e., J=2.

Programmer Response: Probable user error. If CALL SLITE(i) is specified make sure that i is an integer expression with a value of 0-4, inclusive. If CALL SLITET(i,j) is specified, make sure that i is an integer expression with a value of 0-4, inclusive, and j is an integer variable. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Have source and associated listing available.

IHC217I

Explanation: IBC -- An end of data set was sensed during a READ operation; that is, a program attempted to read beyond the data.

If the extended error message facility is specified, the following information is provided:

FIOCS - END OF DATA SET ON UNIT X

Supplemental Data: Unit number.

Standard Corrective Action: Read next file, i.e., increment sequence number by 1.

Programmer Response: Probable user error. Make sure that a FORMAT statement does not define a FORTRAN record longer than the record referred to in the data set. Either keep a counter to avoid exceeding end of record of file, or insert an END= parameter on the READ statement for appropriate transfer of control on end of data set. Check all DD statements. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified on the JOB statement.
- Have source and associated listing available.
- Make sure that LIST has been specified as a parameter on the EXEC statement.
- If a PDS member is being read, make sure that LABEL=(,,,IN) has been specified.

IHC218I I/O ERROR xxx...xxx

Explanation: FIOCS or DIOCS -- One of the following occurred:

- A permanent input/output error has been encountered.
- For sequential I/O, the length of a physical record is inconsistent with the default block size or the blocksize specified on the DD card.
- An attempt has been made to read or write with magnetic tape a record that is fewer than 18 bytes long.

xxx...xxx is the character string formatted by the SYNADAF macro instruction. For an interpretation of this information, see the publication IBM System/360 Operating System: Supervisor and Data Management Macro Instructions, Order No. GC28-6647. After the traceback is completed, control is returned to the call routine statement designated in the ERR parameter of a FORTRAN READ statement if that parameter was specified. (See "Use of ERR Parameter in READ Statement" for additional information.)

Note: If a permanent input/output error has been detected while writing in the object error unit data set, the error message is written in the SYSOUT data set and execution of the job is terminated.

If the extended error message facility is specified, the following information is provided:

FIOCS - I/O ERROR (text provided by data management)

Supplemental Data: Unit number.

Standard Corrective Action: Continue execution and ignore I/O request.

Note: ERR= parameter is honored.

Programmer Response: Probable user error. Make sure that, for sequential I/O, the length of the physical record is consistent with the default or specified blocksize. Check all DD statements. Make sure that no attempt has been made to read or write with magnetic tape from a record that is fewer than 18 bytes in length. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified on the JOB statement.
- Have source and associated listing available.
- Make sure that LIST has been specified as a parameter on the EXEC statement.

IHC219I

Explanation: FIOCS -- Either a data set is referred to in the load module but no DD statement is supplied for it, or a DD statement has an erroneous ddname.

If the extended error message facility is specified, the following information is provided:

FIOCS - MISSING DD CARD FOR (DDname)
or
DIOCS - MISSING DD CARD FOR UNIT X

Supplemental Data: Unit number.

Standard Corrective Action: Continue execution and ignore I/O request.

Programmer Response: Probable user error. Either provide the missing DD statement, or correct any erroneous ddname. Example: If Unit 6 is the installation data set reference number for the printer and an attempt is made to write on Unit 3, then the following DD statement should be included: //GO.FT03F001 DD SYSOUT=A. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified on the JOB statement.
- Have source and associated listing available.
- Make sure that LIST has been specified as a parameter on the EXEC statement.
- Have Stage I SYSGEN output available.

IHC220I

Explanation: FIOCS -- A data set reference number exceeds the limit specified for data set reference numbers when this operating system was generated.

If the extended error message facility is specified, the following information is provided:

FIOCS - UNIT NUMBER OUT OF RANGE. UNIT=X
or
DIOCS - UNIT NUMBER OUT OF RANGE. UNIT=X

Supplemental Data: Unit number.

Standard Corrective Action: Continue execution and ignore I/O request.

Programmer Response: Probable user error. Correct the invalid data set reference number. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified on the JOB statement.
- Have source and associated listing available.
- Have Stage I SYSGEN output available.

IHC221I NAMEL-NAME LARGER THAN EIGHT CHARACTERS. NAME=X

Explanation: NAMEL -- An input variable name exceeds eight characters.

Supplemental Data: Name specified (first eight characters).

Standard Corrective Action: Ignore remainder of namelist request.

Programmer Response: Correct the invalid NAMELIST input variable, or provide any missing delimiters. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IHC222I NAMEL-NAME NOT IN NAMELIST DICTIONARY. NAME=X

Explanation: NAMEL -- An input variable name is not in the NAMELIST dictionary, or an array is specified with an insufficient amount of data.

Supplemental Data: Name specified.

Standard Corrective Action: Ignore remainder of namelist request.

Programmer Response: Probable user error. Make sure that a correct NAMELIST statement is included in the source module for all variable and/or array names read in using NAMELIST. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Have source and associated listing available.

IHC223I NAMEL-END OF RECORD ENCOUNTERED BEFORE EQUAL SIGN. NAME=X

Explanation: NAMEL -- Either an input variable name or a subscript has no delimiter.

Supplemental Data: Name of item.

Standard Corrective Action: Ignore remainder of the namelist request.

Programmer Response: Probable user error. Make sure that all NAMELIST input data is correctly specified and all delimiters are correctly positioned. Check all delimiters. Make sure that sequence numbers are not present in columns 73-80. If the problem recurs, do the following before calling IBM for programming support:

- Have source and associated listing available.

IHC224I NAMEL-SUBSCRIPT FOR NON-DIMENSIONED VARIABLE OR SUBSCRIPT OUT OF RANGE. NAME=X

Explanation: NAMEL -- A subscript is encountered after an undimensioned input name, or the subscript is too big.

Supplemental Data: Name of item.

Standard Corrective Action: Ignore remainder of the namelist request.

Probable Response: Probable user error. Insert any missing DIMENSION statements, or correct the invalid array reference. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Have source and associated listing available.

IHC225I CONVERT-ILLEGAL HEXADECIMAL CHARACTER X

Explanation: FCVTH -- An invalid character is encountered on input under Z format code.

Supplemental Data: Display the record in which the character appeared.

Standard Corrective Action: Zero replaces the encountered character.

Programmer Response: Probable user error. Either correct the invalid character, or correct or delete the Z format code. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Have source and associated listing available.

IHC230I SOURCE ERROR AT ISN xxxx - EXECUTION FAILED [AT SUBROUTINE - name]

Explanation: IBERR -- During load module execution, a source statement error is encountered. The internal statement number for the source statement is xxxx; the routine that contains the statement is specified by "name."

Supplemental Data: None.

Standard Corrective Action: Terminate execution.

Programmer Response: Make sure that all source module code is correct. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Have source and associated listing available.

IHC231I

Explanation: DIOCS -- Direct-access input/output statements are used for a sequential data set, or input/output statements for a sequential data set are used for a direct access data set.

If the extended error message facility is specified, the following information is provided:

IBCOM - DIRECT ACCESS STATEMENT USED WITHOUT DEFINE FILE ON UNIT X
or
DIOCS - DIRECT ACCESS STATEMENT USED FOR SEQUENTIAL DATA SET X
or
FIOCS - SEQUENTIAL I/O STATEMENTS USED FOR DIRECT ACCESS DATA SET X

Supplemental Data: Unit number.

Standard Corrective Action: Ignore I/O request.

Programmer Response: Probable user error. Either include the necessary DEFINE FILE statement for direct access or delete the DEFINE FILE for a sequential data set. Make sure that all DD statements are correct. Verify that all data sets are referenced with valid FORTRAN statements for the data set type. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified on the JOB statement.
- Have source and associated listing available.

IHC232I

Explanation: DIOCS -- Relative position of a record is not a positive integer, or the relative position exceeds the number of records in the data set.

If the extended error message facility is specified, the following information is provided:

DIOCS - RECORD NUMBER X OUT OF RANGE ON UNIT X

Supplemental Data: Unit number and record number.

Standard Corrective Action: Ignore I/O request.

Programmer Response: Probable user error. Make sure that the relative position on the data set has been specified correctly. Check all DD statements. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified on the JOB statement.
- Have source and associated listing available.

IHC233I

Explanation: DIOCS -- The record length specified in the DEFINE FILE statement exceeds the physical limitation of the volume assigned to the data set in the DD statement.

If the extended error message facility is specified, the following information is provided:

DIOCS - RECORD LENGTH GREATER THAN 32K-1 SPECIFIED FOR UNIT X

Supplemental Data: Unit number specified.

Standard Corrective Action: Set record length to 32K.

Programmer Response: Probable user error. Make sure that parameters of the DD statement conform to specifications in the DEFINE FILE statement; the record length in both must be equivalent and within the physical limitations of the assigned volume. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified on the JOB statement.
- Have source and associated listing available.

IHC234I

Explanation: DIOCS -- The data set assigned to print execution error messages cannot be a direct access data set.

If the extended error message facility is specified, the following information is provided:

DIOCS - ATTEMPT TO DEFINE THE OBJECT ERROR UNIT AS DIRECT ACCESS DATA SET. UNIT=X

Supplemental Data: Unit number.

Standard Corrective Action: Ignore define file entry.

Programmer Response: Probable user error. Make sure that the object error unit specified is not direct access. See to the publication IBM System/360 Operating System: System Generation, Order No. GC28-6554, for information on assigning the data set reference number. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified on the JOB statement.
- Have source and associated listing available.

IHC235I

Explanation: DIOCS -- A data set reference number assigned to a direct access data set is used for a sequential data set.

If the extended error message facility is specified, the following information is provided:

DIOCS - DEFINE A DATA SET WHICH HAS BEEN USED SEQUENTIALLY AS A DIRECT ACCESS DATA SET. UNIT=X

Supplemental Data: Unit number.

Standard Corrective Action: Ignore define file entry.

Programmer Response: Probable user error. Make sure that use of and/or reference to sequential data sets does not conflict with FORTRAN defined direct access data sets. Verify that device classes assigned by the installation do not conflict with the specification on the UNIT parameter of the DD statement. Make sure that the DEFINE FILE statement defines a direct access data set. Check all DD statements. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified on the JOB statement.
- Have source and associated listing available.

IHC236I

Explanation: DIOCS -- A READ is executed for a direct access data set that has not been created.

If the extended error message facility is specified, the following information is provided:

DIOCS - READ REQUEST FOR AN UNCREATED DATA SET ON UNIT X

Supplemental Data: Unit number.

Standard Corrective Action: Ignore I/O request.

Programmer Response: Probable user error. Make sure that either a data set utility program has been used, or appropriate parameters have been specified on the associated DD statement. Verify that, if a DD statement is used, DSNAME, UNIT, VOLUME, SPACE, LABEL DISP, SYSOUT, and DCB are specified correctly or omitted where appropriate. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified on the JOB statement.
- Have source and associated listing available.

IHC237I

Explanation: DIOCS -- Length of a record did not correspond to length of record specified in DEFINE FILE statement.

If the extended error message facility is specified, the following information is provided:

DIOCS - INCORRECT RECORD LENGTH SPECIFIED FOR UNIT X

Supplemental data: Unit for which error occurred.

Standard Corrective Action: Ignore the I/O request.

Programmer Response: Probable user error. Make sure that parameters on the DD statement conform to specifications in the DEFINE FILE statement. Verify that record length, buffer length, and/or block length as indicated on the DD statement do not conflict with specifications in the DEFINE FILE statement. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified on the JOB statement.
- Have source and associated listing available.

IHC240I STAE--ABEND CODE IS; SYSTEM XXXX, USER YYYY

IO- {NOT RESTORED / RESTORED / NONE} , SCB=xxxxxx,
PSW IS xxxxxxxxxxxxxxxx

Explanation: An abnormal termination occurred. In some instances, pointers to subroutine entry points may have been destroyed causing the traceback map to be incomplete. If an incomplete traceback map is printed, the following additional text appears between the message IHC240I and the traceback map:

TRACEBACK MAY NOT BEGIN WITH ABENDING ROUTINE.

Supplemental Data: XXXX is the completion code if a system code caused termination. The SCB field gives the address of the STAE Control Block, which contains the old PSW and the contents of the general registers at the time of the abnormal termination. For a description of the contents of the STAE Control Block, see the publication IBM System/360 Operating System: Supervisor and Data Management Services, Order No. GC28-6646. The PSW field gives the contents of the last FORTRAN program status word when an abnormal termination occurred.

Input/output operations associated with the error are defined as NOT RESTORED, RESTORED, or NONE as follows:

NOT RESTORED -- Input/output has been halted and cannot be restored.

RESTORED -- Input/output has been halted. FORTRAN will attempt to restart input/output and then close data sets.

NONE -- No active input/output operations were present at abnormal termination time. FORTRAN will close data sets.

Standard Corrective Action: None

Programmer Response: Use the abend code and the contents of the SCB and PSW to determine the nature of the error. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MSGLEVEL=(1,1) was specified on the JOB statement.
- Make sure that the LIST, XREF, and MAP options were specified in the PARM field of the EXEC statement.
- Have source and associated listing available.

IHC241I FIXPI INTEGER BASE=0, INTEGER EXPONENT=X, LE 0

Explanation: LIB -- For an exponentiation operation (I**J) in the subprogram IHCFIXPI(FIXPI#) where I and J represent integer

variables or integer constants, I is equal to zero and J is less than or equal to zero.

Supplemental Data: Exponent specified.

Standard Corrective Action: Result=0.

Programmer Response: Probable user error. Make sure that integer variables and/or integer constants for an exponentiation operation are within the allowable range. If the base and exponent may or will fall outside that range during program execution, then either modify the operand(s), or insert source code to test for the situation and make the compensation appropriate to the program unit. Bypass the exponentiation operation if necessary. Example: Assume I,J,K previously defined integer variables.

```
IF(I.EQ.0.AND.J.LE.0) GO TO 11
K = I**J
11 CONTINUE
```

If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC242I FRXPI REAL*4 BASE=0.0, INTEGER EXPONENT=X, LE 0

Explanation: LIB -- For an exponentiation operation (R**J) in the subprogram IHCFRXPI(FRXPI#), where R represents a real*4 variable or real*4 constant, and J represents an integer variable or integer constant, R is equal to zero and J is less than or equal to zero.

Supplemental Data: Exponent specified.

Standard Corrective Action: Result=0.

Programmer Response: Probable user error. Make sure that both the real variable or constant base and the integer variable or constant exponent for an exponentiation operation are within the allowable range. If the base and exponent may or will fall outside that range during program execution, then either modify the operand(s), or insert source code to test for the situation and make the compensation appropriate to the program unit. Bypass the exponentiation operation if necessary. (See similar example for IHC241I.) If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC243I FDXPI REAL*8 BASE=0.0, INTEGER EXPONENT=X, LE 0

Explanation: LIB -- For an exponentiation operation (D**J) in where D represents a real*8 variable or real*8 constant and J represents an integer variable or integer constant, D is equal to zero and J is less than or equal to zero.

Supplemental Data: Exponent specified.

Standard Corrective Action: Result=0.

Programmer Response: Probable user error. Make sure that both the real variable or constant base and the integer variable or constant exponent for an exponentiation operation are within the allowable range. If the base and exponent may or will fall outside that range during execution, then either modify the operand(s), or insert source code to test for the situation and make the compensation appropriate to the program unit. Bypass the exponentiation operation if

necessary. (See similar example for IHC241I.) If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and the necessary DD statement included.
- Have source and associated listing available.

IHC244I FRXPR REAL*4 BASE=0.0, REAL*4, EXPONENT=X.X, LE 0

Explanation: LIB -- For an exponentiation operation (R**S) in the subprogram IHCFRXPR(FRXPR#), where R and S represent real*4 variables or real*4 constants, R is equal to zero and S is less than or equal to zero.

Supplemental Data: Exponent specified.

Standard Corrective Action: Result=0.

Programmer Response: Probable user error. Make sure that both the real variable or constant base and exponent for an exponentiation operation are within the allowable range. If the base and exponent may or will fall outside that range during program execution, then either modify the operand(s), or insert source code to test for the situation and make compensation appropriate to the program unit. Bypass the exponentiation operation if necessary. (See similar example for IHC241I.) If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC245I FDXPD REAL*8 BASE=0.0, REAL*8 EXPONENT=X.X, LE 0

Explanation: LIB -- For an exponentiation operation (D**P) in the subprogram IHCFDXPD(FDXPD#), where D and P represent real*8 variables or real*8 constants, D is equal to zero and P is less than or equal to zero.

Supplemental Data: Exponent specified.

Standard Corrective Action: Result=0.

Programmer Response: Probable user error. Make sure that both the real variable or constant base and exponent for an exponentiation operation are within the allowable range. If the base and exponent may or will fall outside that range during program execution, then either modify the operand(s), or insert source code to test for the situation and make compensation appropriate to the program unit. Bypass the exponentiation operation if necessary. (See similar example for IHC241I.) If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC246I FCXPI COMPLEX*8 BASE=0.0+0.0I, INTEGER EXPONENT=X, LE 0

Explanation: LIB -- For an exponentiation operation (Z**J) in the subprogram IHCFCXPI(FCXPI#), where Z represents a complex*8 variable or complex*8 constant and J represents an integer variable or integer constant, Z is equal to zero and J is less than or equal to zero.

Supplemental Data: Exponent specified.

Standard Corrective Action: Result=0.

Programmer Response: Probable user error. Make sure that both the complex variable or constant base and the integer variable or constant exponent for an exponentiation operation are within the allowable range. If the base and exponent may or will fall outside that range during program execution, then either modify the operand(s), or insert source code to test for the situation and make the compensation appropriate to the program unit. Bypass the exponentiation operation if necessary. (See similar example for IHC241I.) If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC247I FCDXI COMPLEX*16 BASE=0.0+0.0I, INTEGER EXPONENT=X, LE 0

Explanation: LIB -- For an exponentiation operation (Z**J) in the subprogram IHCFCDXI(FCDXI#), where Z represents a complex*16 variable or complex*16 constant and J represents an integer variable or integer constant, Z is equal to zero and J is less than or equal to zero.

Supplemental Data: Exponent specified.

Standard Corrective Action: Result=0.

Programmer Response: Probable user error. Make sure that both the complex variable or constant base and the integer variable or constant exponent for an exponentiation operation are within the allowable range. If the base and exponent may or will fall outside that range during program execution, then either modify the operand(s), or insert source code to test for the situation and make the compensation appropriate to the program unit. Bypass the exponentiation operation if necessary. (See similar example for IHC241I.) If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC251I SQRT NEGATIVE ARGUMENT=X

Explanation: LIB -- In the subprogram IHCSSQRT(SQRT), the argument is less than 0.

Supplemental Data: Argument specified.

Standard Corrective Action: Result=$|X|^{1/2}$.

Programmer Response: Probable user error. Make sure that the argument is within the allowable range. Either modify the argument, or insert source code to test for a negative argument and make the necessary compensation. Bypass the function reference if necessary. Example: Assume ARG (REAL*4) is to be the input argument to SORT. Then a simple test might appear:

```
      IF (ARG) 10,20,20
   10 ARG = ABS (ARG)
   20 ANS = SORT (ARG)
```

If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC252I EXP ARG=X.X, GT 174.673

Explanation: LIB -- In the subprogram IHCSEXP(EXP), the argument is greater than 174.673.

Supplemental Data: Argument specified.

Standard Corrective Action: Result=*.

Programmer Response: Probable user error. Make sure that the argument to the exponential function is within the allowable range. If the argument may or will exceed that range during program execution, then provide code to test for the situation and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC spatement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC253I ALOG-ALOG10 ARG=X.X, LE ZERO

Explanation: LIB -- In the subprogram IHCSLOG(ALOG and ALOG10), the argument is less than or equal to zero. Because this subprogram is called by an exponential subprogram, this message also indicates that an attempt has been made to raise a negative base to a real power.

Supplemental Data: Argument specified.

Standard Corrective Action: If X=0, result=-*; if X<0, result=log|X| or $\log_{10}|X|$.

Programmer Response: Probable user error. Make sure that the argument to the logarithmic function is within the allowable range. If the argument may or will be outside that range during program execution, then provide code to test for the situation and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC254I SIN-COS/ARG/=/X.X(HEX=X)/, GE PI*2**18

Explanation: LIB -- In the subprogram IHCSSCN(SIN and COS), the absolute value of an argument is greater than or equal to $2^{18} \bullet \pi$. ($2^{18} \bullet \pi$ =.82354966406249996D+06)

Supplemental Data: Argument specified.

Standard Corrective Action: Result= $\sqrt{2}/2$.

Programmer Response: Probable user error. Make sure that the argument (in radians where 1 radian $\approx$ 57.2957795131°) to the trigonometric sine or cosine function is within the allowable range. If the argument may or will exceed that range during program execution, then provide code to test for the situation and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC255I ATAN2 ARGUMENTS=0.0

Explanation: LIB -- In the subprogram IHCSATN2, when entry name ATAN2 is used, both arguments are equal to zero.

Supplemental Data: Arguments specified.

Standard Corrective Action: Result=0.

Programmer Response: Probable user error. Make sure that both arguments do not become zero during program execution, or are not inadvertently initialized or modified to zero. Provide code to test for the situation and, if necessary, modify the arguments or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC256I SINH-COSH/ARG/=/X.X/, GE 175.366

Explanation: LIB -- In the subprogram IHCSSCNH(SINH or COSH), the argument is greater than or equal to 174.673.

Supplemental Data: Argument specified.

Standard Corrective Action: Result=*.

Programmer Response: Probable user error. Make sure that the argument to the hyperbolic sine or cosine function is within the allowable range. If the argument may or will exceed that range during program execution, then provide code to test for the situation and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC257I ARSIN-ARCOS/ARG/=/X.X/ GT 1

Explanation: LIB -- In the subprogram IHCSASCN (ARCSIN or ARCOS), the absolute value of the argument is greater than 1.

Supplemental Data: Argument specified.

Standard Corrective Action: Result=0.

Programmer Response: Probable user error. Make sure that the argument to the arcsine or arccosing function is between -1 and +1, inclusive. If the argument may or will fall outside that range during program execution, then provide code to test for the situation and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC258I TAN-COTAN/ARG/=/X.X(HEX=X)/, GE PI*2**18

Explanation: LIB -- In the subprogram IHCSTNCT (TAN or COTAN), the absolute value of the argument is greater than or equal to $2^{18} \bullet \pi$. ($2^{18} \bullet \pi$=.82354966406249996D+06)

Supplemental Data: Argument specified.

Standard Corrective Action:
Result=1.

Programmer Response: Probable user error. Make sure that the argument (in radians where 1 radian $\approx$57.2957795131°) to the trigonometric tangent or cotangent function is within the allowable range. If the argument may or will exceed that range during program execution, then provide code to test for the situation and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC259I TAN-COTAN/ARG/=/X.X(HEX=X)/, APPROACHES SINGULARITY

Explanation: LIB -- In the subprogram IHCSTNCT (TAN or COTAN), the argument value is too close to one of the singularities

($\pm\frac{\pi}{2}, \pm\frac{3\pi}{2}, \ldots$ for the tangent;
or $\pm\pi, \pm 2\pi, \ldots$ for the cotangent).

Supplemental Data: Argument specified.

Standard Corrective Action:
Result=*.

Programmer Response: Probable user error. Make sure that the argument (in radians where 1 radian $\approx$ 57.2957795131°) to the trigonometric tangent or cotangent function is within the allowable range. If the argument may or will approach the corresponding singularities for the function during program execution, then provide code to test for the situation and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC261I DSQRT NEGATIVE ARGUMENT=X.X

Explanation: LIB -- In the subprogram IHCLSQRT(DSQRT), the argument is less than 0.

Supplemental Data: Argument specified.

Standard Corrective Action:
Result=$|X|^{1/2}$.

Programmer Response: Probable user error. Make sure that the argument is within the allowable range. Either modify the argument, or insert source code to test for a negative argument and make the necessary compensation. Bypass the function reference if necessary. Example: Assume DARG (REAL*8) is to be the input argument to DSQRT. Then a simple test might appear:

```
      IF (DARG) 10,20,20
   10 DARG = DABS (DARG)
   20 ANS = DSQRT (DARG)
```

If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC262I DEXP ARG=X.X, GT 174.673

Explanation: LIB -- In the subprogram IHCLEXP(DEXP), the argument is greater than 174.673.

Supplemental Data: Argument specified.

Standard Corrective Action: Result=*.

Programmer Response: Probable user error. Make sure that the argument to the exponential function is within the allowable range. If the argument may or will exceed that range during program execution, then provide code to test for the situation and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC263I DLOG-DLOG10 ARG=X.X, LE ZERO

Explanation: LIB -- In the subprogram IHCLLOG(DLOG and DLOG10), the argument is less than or equal to zero. Because the subprogram is called by an exponential subprogram, this message also indicates that an attempt has been made to raise a negative base to a real power.

Supplemental Data: Argument specified.

Standard Corrective Action: If X=0, result=-*; if X<0, result=log|X| or $\log_{10}|X|$.

Programmer Response: Probable user error. Make sure that the argument to the logarithmic function is within the allowable range. If the argument may or will be outside that range during program execution, then provide code to test for the situation and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC264I DSIN-DCOS/ARG/=/X.X(HEX=X)/, GE PI*2**50

Explanation: LIB -- In the subprogram IHCLSCN(DSIN and DCOS), the absolute value of the argument is greater than or equal to $2^{50} \cdot \pi$. ($2^{50} \cdot \pi$ =.35371188737802239D+16)

Supplemental Data: Argument specified.

Standard Corrective Action: Result = $\sqrt{2}/2$.

Programmer Response: Probable user error. Make sure that the argument (in radians where 1 radian=57.2957795131°) to the trigonometric sine or cosine function is within the allowable range. If the argument may or will exceed that range during program execution, then provide code to test for the situation and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC265I DATAN2 ARGUMENTS=0.0

Explanation: LIB -- In the subprogram IHCLATN2, when entry name DATAN2 is used, both arguments are equal to zero.

Supplemental Data: Arguments specified.

Standard Corrective Action: Result=0.

Programmer Response: Probable user error. Make sure that both arguments do not become zero during program execution, or are not inadvertently initialized or modified to zero. Provide code to test for the situation and, if necessary, modify the arguments or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC266I DSINH-DCOSH/ARG/=/X.X/, GE 175.366

Explanation: LIB -- In the subprogram IHCLSCNH (DSINH or DCOSH), the absolute value of the argument is greater than or equal to 175.366.

Supplemental Data: Argument specified.

Standard Corrective Action: Result=*.

Programmer Response: Probable user error. Make sure that the argument to the hyperbolic sine or cosine function is within the allowable range. If the argument may or will exceed that range during program execution, then provide code to test for the situation and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC267I DARSIN-DARCOS/ARG/=/X.X/, GT 1

Explanation: LIB -- In the subprogram IHCLASCN (DARSIN or DARCOS), the absolute value of the argument is greater than 1.

Supplemental Data: Argument specified.

Standard Corrective Action: Result=0.

Programmer Response: Probable user error. Make sure that the argument to the arcsine or arccosine function is between -1 and +1, inclusive. If the argument may or will fall outside that range during program execution, then provide code to test for the situation and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC268I DTAN-DCOTAN/ARG/=/X.X(HEX=X)/ GE PI*(2**50)

Explanation: LIB -- In the subprogram IHCLTNCT (DTAN or DCOTAN), the absolute value of the argument is greater than or equal to $2^{50} \cdot \pi$. ($2^{50} \cdot \pi$ =.35371188737802239D+16)

Supplemental Data: Argument specified.

Standard Corrective Action: Result=1.

Programmer Response: Probable user error. Make sure that the argument (in radians where 1 radian ≈ 57.2957795131°) to the trigonometric tangent or cotangent function is within the allowable range. If the argument may or will exceed that range during program execution, then provide code to test for the situation and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC269I DTAN-DCOTAN/ARG/=/X.X(HEX=X)/, APPROACHES SINGULARITY

Explanation: LIB -- In the subprogram IHCLTNCT (DTAN or DCOTAN), the argument value is too close to one of the singularities

($\pm\frac{\pi}{2}, \pm\frac{3\pi}{2}, \ldots$ for the tangent; or $\pm\pi, \pm 2\pi, \ldots$ for the cotangent).

Supplemental Data: Argument specified.

Standard Corrective Action: Result=*.

Programmer Response: Probable user error. Make sure that the argument (in radians where 1 radian ≈ 57.2957795131°) to the trigonometric tangent or cotangent function is within the allowable range. If the argument may or will approach the corresponding singularities for the function during program execution, then provide code to test for the situation and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC271I CEXP REAL ARG=X.X(HEX=X), GT 174.673

Explanation: LIB -- In the subprogram IHCCSEXP (CEXP), the value of the real part of the argument is greater than 174.673.

Supplemental Data: Argument specified.

Standard Corrective Action: Result=*(COS X + iSIN X) where X is the imaginary portion of the argument.

Programmer Response: Probable user error. Make sure that the argument is within the allowable range. If the argument may or will exceed that range during program execution, then provide code to test for the situation, and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC272I CEXP IMAG ARG=X(HEX=X), ABS VALUE GE PI*2**18

Explanation: LIB -- In the subprogram IHCCSEXP (CEXP), the absolute value of the imaginary part of the argument is greater than or equal to $2^{18} \cdot \pi$. ($2^{18} \cdot \pi$=.82354966406249996D+06)

Supplemental Data: Argument specified.

Standard Corrective Action: Result=0+0i.

Programmer Response: Probable user error. Make sure that the argument to the exponential function is within the allowable range. If the argument may or will exceed that range during program execution, then provide code to test for the situation, and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC273I CLOG ARGUMENT=0.0+0.0I

Explanation: LIB -- In the subprogram IHCCSLOG (CLOG), the real and imaginary parts of the argument are equal to zero.

Supplemental Data: Argument specified.

Standard Corrective Action: Result=-*+0i.

Programmer Response: Probable user error. Make sure that both the real and imaginary parts of the argument do not become zero during program execution, or are not inadvertently initialized or modified to zero. Provide code to test for the situation and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC274I CSIN-CCOS/REAL ARG/=/X.X (HEX=X)/, GE PI*2**18

Explanation: LIB -- In the subprogram IHCCSSCN (CSIN or CCOS), the absolute value of the real part of the argument is greater than or equal to $2^{18} \cdot \pi$. ($2^{18} \cdot \pi$=.82354966406249996D+06)

Supplemental Data: Argument specified.

Standard Corrective Action: Result=0+0i.

Programmer Response: Probable user error. Make sure that the real part of the argument (in radians where 1 radian $\approx$ 57.2957795131°) to the trigonometric sine or cosine function is within the allowable range. If the real part of the argument may or will exceed the range during program execution, then provide code to test for the situation and, if necessary, modify the real part of the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC275I CSIN-CCOS/IMAG ARG/=/X.X (HEX=X)/ GT 174.673

Explanation: LIB -- In the subprogram IHCCSSCN (CSIN or CCOS), the absolute value of the imaginary part of the argument is greater than 174.673.

Supplemental Data: Argument specified.

Standard Corrective Data: If imaginary part >0, (X is real portion of argument):

- For sine, result=*/2(SIN X + iCOS X).
- For cosine, result=*/2(COS X - iSIN X).

If imaginary part <0, (X is real portion of argument):

- For sine, result=*/2(SIN X - iCOS X).
- For cosine, result=*/2(COS X + iSIN X).

Programmer Response: Probable user error. Make sure that the imaginary part of the argument (in radians where 1 radian ≈ 57.2957797131°) to the trigonometric sine or cosine function is within the allowable range. If the imaginary part of the argument may or will exceed that range during program execution, then provide code to test for the situation and, if necessary, modify the imaginary part of the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC281I CDEXP-REAL ARG=X.X(HEX=X) GT 174.673

Explanation: LIB -- In the subprogram IHCCLEXP (CDEXP), the value of the real part of the argument is greater than 174.673.

Supplemental Data: Argument specified.

Standard Corrective Action: Result=*(COS X + iSIN X) where X is the imaginary portion of the argument.

Programmer Response: Probable user error. Make sure that the real part of the argument to the exponential function is within the allowable range. If the real part of the argument may or will exceed that range during program execution, then provide code to test for the situation and, if necessary, modify the real part of the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC282I CDEXP IMAG ARG=X.X(HEX=X) ABS VALUE GE PI*2**50

Explanation: LIB -- In the subprogram IHCCLEXP (CDEXP), the absolute value of the imaginary part of the argument is greater than or equal to $2^{50} \cdot \pi$. ($2^{50} \cdot \pi$=.3537118873780239D+16)

Supplemental Data: Argument specified.

Standard Corrective Action: Result=*+0i.

Programmer Response: Probable user error. Make sure that the imaginary part of the argument to the exponential function is within the allowable range. If the imaginary part of the argument may or will exceed that range during program execution, then provide code to test for the situation and, if necessary, modify the imaginary part of the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.

- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.

- Have source and associated listing available.

IHC283I CDLOG ARGUMENT=0.0+0.0I

Explanation: LIB -- In the subprogram IHCCLLOG (CDLOG), the real and imaginary parts of the argument are equal to zero.

Supplemental Data: Argument specified.

Standard Corrective Action: Result=*+0i.

Programmer Response: Probable user error. Make sure that both the real and imaginary parts of the argument do not become zero during program execution, or are not inadvertently initialized or modified to zero. Provide code to test for the situation and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.

- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.

- Have source and associated listing available.

IHC284I CDSIN-CDCOS/REAL ARG/=/X.X (HEX=X)/, GE PI*2**50

Explanation: LIB -- In the subprogram IHCCLSCN (CDSIN or CDCOS), the absolute value of the real part of the argument is greater than or equal to $2^{50} \cdot \pi$. ($2^{50} \cdot \pi$=.35371188737802239D+16)

Supplemental Data: Argument specified.

Standard Corrective Action: Result=0+0i.

Programmer Response: Probable user error. Make sure that the real part of the argument (in radians where 1 radian $\approx$ 57.2957795131°) to the trigonometric sine or cosine function is within the allowable range. If the part of the argument may or will exceed the range during program execution, then provide code to test for the situation and, if necessary, modify the real part of the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.

- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.

- Have source and associated listing available.

IHC285I CDSIN-CDCOS/IMAG ARG/=/X.X (HEX=X)/, GT 174.673

Explanation: LIB -- In the subprogram IHCCLSCN (CDSIN or CDCOS), the absolute value of the imaginary part of the argument is greater than 174.673.

Supplemental Data: Argument specified.

Standard Corrective Action: If imaginary part >0, (X is real portion of argument):

- For sine, result=*/2(SIN X + iCOS X).

- For cosine, result=*/2(COS X - iSIN S).

If imaginary part <0, (X is real portion of argument):

- For sine, result=*/2(SIN X - iCOS X).

- For cosine, result=*/2(COS X + iSIN X).

Programmer Response: Probable user error. Make sure that the imaginary part of the argument (in radians where 1 radian $\approx$ 57.2957795131°) to the trigonometric sine or cosine

function is within the allowable range. If the imaginary part of the argument may or will exceed that range during program execution, then provide code to test for the situation and, if necessary, modify the imaginary part of the argument or bypass the source referencing the function subprogram. If the program recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the 'XREF' option is specified in the parm field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC290I GAMMA ARG=X.X(HEX=X), LE 2**-252 OR GE 57.5744

Explanation: LIB -- In the subprogram IHCSGAMA (GAMMA), the value of the argument is outside the valid range. (Valid range: $2^{-252}<x<57.5744$)

Supplemental Data: Argument specified.

Standard Corrective Action: Result=*.

Programmer Response: Probable user error. Make sure that the argument to the gamma function is within the allowable range. If the argument may or will be outside that range during program execution, then provide code to test for the situation and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC291I ALGAMA ARG=X.X(HEX=X), LE 0. OR GE 4.2937*10**73

Explanation: LIB -- In the subprogram IHCSGAMA (ALGAMA), the value of the argument is outside the valid range. (Valid range: $0<x<4.2937x10^{73}$)

Supplemental Data: Argument specified.

Standard Corrective Action: Result=*.

Programmer Response: Probable user error. Make sure that the argument to the algama function is within the allowable range. If the argument may or will be outside that range during program execution, then provide code to test for the situation and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the necessary DD statement included.
- Have source and associated listing available.

IHC300I DGAMMA ARG=X.X(HEX=X), LE 2**-252 OR GE 57.5744

Explanation: LIB -- In the subprogram IHCLGAMA (DGAMMA), the value of the argument is outside the valid range. (Valid range: $2^{-252}<x<57.5744$)

Supplemental Data: Argument specified.

Standard Corrective Action: Result=*.

Programmer Response: Probable user error. Make sure that the argument to the dgamma function is within the allowable range. If the argument may or will be outside that range during program execution, then provide code to test for the situation and, if

necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

IHC301I DLGAMA ARG=X.X(HEX=X), LE 0. OR GE 4.2937*10**73

Explanation: LIB -- In the subprogram IHCLGAMA (DLGAMA), the value of the argument is outside the valid range. (Valid range: $0<x<4.2937x10^{73}$)

Supplemental Data: Argument specified.

Standard Corrective Action: Result=*.

Programmer Response: Probable user error. Make sure that the argument to the dlgama function is within the allowable range. If the argument may or will be outside that range during program execution, then provide code to test for the situation and, if necessary, modify the argument or bypass the source referencing the function subprogram. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Have source and associated listing available.

Even though message printing may be suppressed when the extended error message facility is available, a summary of errors is printed when the job is completed. Its format is shown in Figure 108 . The format of a traceback map is shown in Figure 109.

The headings in the traceback map may be described as follows:

ROUTINE The name of the routine entered, which was called by the next routine in the list.

ISN When the compiler's ID option supplies an Internal Statement Number (ISN), the ISN entry is a symbolic reference to the point from which the routine was called.

REG. 14 This is the absolute location reference to the point from which ROUTINE was called. By using the ENTRY POINT location, a relative location can be computed.

REG. 15 This is the address of the entry point in ROUTINE.

REG. 0 This is the result register used by function subprograms.

REG. 1 This is the address of the argument list passed to ROUTINE.

If the user specifies that an installation-supplied routine is to be used for corrective action, this line is added to the message:

```
USER FIXUP TAKEN, EXECUTION CONTINUING
```

For a standard corrective action, the message addition reads:

```
STANDARD FIXUP TAKEN, EXECUTION
CONTINUING
```

```
SUMMARY OF ERRORS FOR THIS JOB  ERROR NUMBER  NUMBER OR ERRORS
                                219           1
                                217           1
                                211           57
```

Figure 108. Summary of Error and Traceback

```
TRACEBACK FOLLOWS-  ROUTINE  ISN  REG.  14  REG.  15  REG.   0  REG.   1
                    IBCOM        000083B4  000089B8  00000005  000081A6
                    MAIN         00004918  50008020  00000030  0003FF04
ENTRY POINT- 50008020
```

Figure 109. Example of Traceback Map

If the extended error message facility detects an error condition, an informational message is printed and the job may be terminated. The following text contains a description of such messages.

IHC900I EXECUTION TERMINATING DUE TO ERROR COUNT FOR ERROR NUMBER X

Explanation: This error has occurred frequently enough to reach the count specified as the number at which execution should be terminated.

System Action: The job is terminated.

Programmer Response: Probable user error. Make sure that occurrences of the error number indicated are eliminated. For alternative action, see the Extended Error Handling Facility section. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.
- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.
- Make sure that MSGLEVEL=(1,1) was specified on the JOB statement.
- Have source and associated listing available.

IHC901I EXECUTION TERMINATING DUE TO SECONDARY ENTRY TO ERROR MONITOR FOR ERROR X WHILE PROCESSING ERROR X

Explanation: In a user's corrective action routine, an error has occurred that has called the error monitor before it has returned from processing a diagnosed error.

System Action: The job is terminated.

Note: If Traceback follows this message, it may be unreliable.

Programmer Response: Probable user error. Make sure that the error monitor is not called prior to processing the diagnosed error. Example: A statement such as R=A**B cannot be used in the exit routine for error 252, because FRXPR# uses EXP, which detects error 252. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.

- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.

- Have source and associated listing available.

IHC902I ERROR NUMBER X OUT OF RANGE OF ERROR TABLE

Explanation: A request has been made to reference a non-existent Option Table entry.

System Action: The request is ignored and execution continues. IRETCD is set to zero.

Programmer Response: Probable user error. Make sure that the value assigned to an error condition is within the range of entries in the option table. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.

- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement in included.

- Have source and associated listing available.

- Have information from system generation time on the extended error handling facility available.

IHC903I ATTEMPT TO CHANGE UNMODIFIABLE TABLE ENTRY, NUMBER=X

Explanation: The Option Table specifies that no changes may be made in this entry, but a change request has been made by use of CALL ERRSET or CALL ERRSTR.

System Action: The request is ignored and execution continues.

Programmer Response: Probable user error. Make sure that no attempt has been made to alter dynamically an unmodifiable entry in the Option Table. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.

- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.

- Have source and associated listing available.

- Have information from system generation time on the extended error handling facility available.

IHC904I ATTEMPT TO DO I/O DURING FIXUP ROUTINE FOR AN I/O TYPE ERROR

Explanation: When attempting to correct an input/output error, the user may not issue a READ, WRITE, BACKSPACE, ENDFILE, REWIND, PDUMP, DEBUG, or ERRTRA.

System Action: The job is terminated.

Programmer Response: Probable user error. Make sure that, if an I/O error is detected, the user exit routine does not attempt to execute any FORTRAN I/O statement. If the problem recurs, do the following before calling IBM for programming support:

- Make sure that MAP and LIST have been specified as parameters on the EXEC statement.

- Make sure that the XREF option is specified in the PARM field of the EXEC statement and that the necessary DD statement is included.

- Have source and associated listing available.

- Have information from system generation time on the extended error handling facility available.

Operator Messages

Operator messages for STOP and PAUSE are generated during load module execution.

The message for a PAUSE can be one of the forms:

```
                    ⎧PAUSE n          ⎫
yy IHC001A          ⎨PAUSE 'message'  ⎬
                    ⎩PAUSE 0          ⎭
```

where: yy — is the identification number

n — is the unsigned 1-5 digit integer constant specified in a PAUSE source statement

'message' — is the literal constant specified in a PAUSE source statement

0 — is printed out when a

PAUSE

statement is executed

Explanation: The programmer should give instructions that indicate the action to be taken by the operator when the PAUSE is encountered.

User Response: To resume execution, the operator presses the REQUEST key. When the PROCEED light comes on, the operator types

REPLY yy,'Z'

where yy is the identification number and Z is any letter or number. To resume program execution, the operator must press the alternate coding key and a numeric 5.

The message for a STOP statement is of the form:

IHC002I STOP n

where: n — is the unsigned 1-5 digit integer constant specified in a STOP source statement. This value is placed in register 15 when the STOP statement is executed.

A STOP or STOP 0 message is not displayed on the console.

User Response: None

APPENDIX E: EXTENDED AMERICAN NATIONAL STANDARD CARRIAGE CONTROL CHARACTERS

Code	Interpretation
* blank	Space one line before printing
* 0	Space two lines before printing
-	Space three lines before printing
* +	Suppress space before printing
* 1	Skip to first line of a new page
2	Skip to channel 2
3	Skip to channel 3
4	Skip to channel 4
5	Skip to channel 5
6	Skip to channel 6
7	Skip to channel 7
8	Skip to channel 8
9	Skip to channel 9
A	Skip to channel 10
B	Skip to channel 11
C	Skip to channel 12
V	Select punch pocket 1
W	Select punch pocket 2

* These carriage control characters are identical to the FORTRAN carriage control characters specified in the FORTRAN IV Language publication.

The UNIT parameter of the DD statement can identify an input or output unit by its actual address, its type number, or its group name. Type numbers, automatically established at system generation, correspond to units entered into system configurations. Type numbers and corresponding units are listed here for the reader's convenience.

Tape Units

Unit Type	Unit
2400	2400 series 9-Track Magnetic Tape Drive that can be allocated to a data set written or to be written in 800 bpi density
2400-1	2400 series Magnetic Tape Drive with 7-Track Compatibility and without Data Conversion
2400-2	2400 series Magnetic Tape Drive with 7-Track Compatibility and Data Conversion
2400-3	2400 series 9-Track Magnetic Tape Drive that can be allocated to a data set written or to be written in 1600 bpi density
2400-4	2400 series 9-Track Magnetic Tape Drive having an 800 and 1600 bpi (density) capability

Direct Access Units

Unit Type	Unit
2301	2301 Drum Storage Unit
2302	2302 Disk Storage Drive
2303	2303 Drum Storage Unit
2305	2305 Drum Storage Unit
2311	Any 2311 Disk Storage Drive
2314	2314 Storage Facility
2321	Any bin mounted on a 2321 data cell drive
3330	3330 Disk Storage Facility

Unit Record Equipment

1052	1052 Printer-Keyboard
1403	1403 Printer or 1404 Printer (continuous form only)
1442	1442 Card Read Punch
1443	any 1443 Printer
2501	2501 Card Reader
2520	2520 Card Read Punch
2540	2540 Card Read Punch (read feed)
2540-2	2540 Card Read Punch (punch feed)
2671	2671 Paper Tape Reader
3211	3211 Printer

Graphic Units

1053	1053 Model 4 Printer
2250-1	2250 Display Unit, Model 1
2250-3	2250 Display Unit, Model 3
2260-1	2260 Model 1 Display Station (Local Attachment)
2260-2	2260 Model 2 Display Station (Local Attachment)
2280	2280 Film Recorder
2282	2282 Film Recorder-Scanner

APPENDIX G: FORTRAN IV (G) DEBUG FACILITY

The FORTRAN IV (G) Debug Facility statements (DEBUG, AT, DISPLAY, TRACE ON and TRACE OFF) are described in the FORTRAN IV Language publication. This section describes the output produced when these statements are used in a FORTRAN source module.

DEBUG STATEMENT

The options UNIT, TRACE, SUBTRACE, INIT, and SUBCHK may be specified in the DEBUG statement. The UNIT option indicates the unit on which the DEBUG output is to be written; if this option is omitted, DEBUG output is written on SYSOUT.

TRACE

TRACE output is written only when TRACE is on as a result of the TRACE ON statement. For each labeled statement that is executed, the line

-DEBUG-TRACE statement-label

is written.

SUBTRACE

SUBTRACE is used to trace program flow from one routine to another. For each subprogram called, the line

-DEBUG-SUBTRACE subprogram-name

is written on entry to the subprogram, and the line

-DEBUG-SUBTRACE *RETURN*

is written on exit from the subprogram.

INIT

The output produced as a result of the INIT option is written regardless of any TRACE ON or TRACE OFF statements in the source module. When the value of an unsubscripted variable listed in the INIT option changes, the line

-DEBUG-variable-name = value

is written, with the value given in the proper format for the variable type. When the value of an element of an array listed in the INIT option changes, the line

-DEBUG-array-name(element-number) = value

is written, with the format of the value determined by the type of the array element. The single element number subscript is used regardless of the number of dimensions on the array.

SUBCHK

SUBCHK output is not affected by TRACE ON and TRACE OFF statements in the source module. When a reference to an array listed in the SUBCHK option includes subscripts such that the reference is outside the array, the line

-DEBUG-SUBCHK array-name(element-number)

is printed. The statement including the out-of-bounds reference is operated nonetheless.

DISPLAY STATEMENT

DISPLAY statement output is identical to NAMELIST WRITE output. The first line written is the name of the NAMELIST created by the compiler for the DISPLAY statement, preceded by the ampersand character:

&DBGnn#

where:

nn is the 2-digit decimal value assigned to the DISPLAY statement; this value begins at 01 for the first DISPLAY statement in the source module and increases by one for each subsequent DISPLAY statement.

The NAMELIST name is followed by the DISPLAY list, in NAMELIST format. The output is terminated with the line

&END

SPECIAL CONSIDERATIONS

Any DEBUG output which is produced during an input/output operation is saved in storage until the input or output operation is complete, when it is written out. Saving this information may require a request for additional storage space from the system. If the request cannot be satisfied, some of the DEBUG output may be lost. If this situation occurs, the message

-DEBUG-SOME DEBUG OUTPUT MISSING

is written after the output which was saved.

If a subscript appearing in an input/output list includes a function reference, and the FUNCTION contains a DISPLAY statement, the DISPLAY cannot be performed. In this case the message

-DEBUG-DISPLAY DURING I/O SKIPPED

is written in the DEBUG output.

APPENDIX H: FORTRAN IV (H) OPTIMIZATION FACILITIES

This appendix contains information relating to the use of the FORTRAN IV (H) compiler optimization facilities.

PROGRAM OPTIMIZATION

Facilities are available in the FORTRAN IV (H) compiler that enable a programmer to optimize execution speed and to reduce the size of the object module. However, programs that are compiled using the IBM-supplied cataloged procedures are not optimized; OPT=0 is the default option. A programmer must override this default option with either OPT=1 or OPT=2 to specify the use of the optimization facilities. (See "Cataloged Procedures" for overriding par- ameters in the EXEC statement.)

When using OPT=1, the entire program is a loop, while individual sections of coding, headed and terminated by labeled statements, are blocks. The object code is improved by:

- Improving local register assignment. (Variables that are defined and used in a block are retained (if possible) in registers during the processing of the block. Time is saved because the number of load and store instructions are reduced.)
- Retaining the most active base addresses and variables in registers across the whole program. (Retention in registers saves time because the number of load instructions are reduced.)
- Improving branching by the use of RX branch instructions. (An RX branch instruction saves a load instruction and reduces the number of required address constants.)

When using OPT=2, the loop structure and data flow of the program are analyzed. The object code is improved over OPT=1 by:

- Assigning registers across a loop to the most active variables, constants, and base addresses within the loop.
- Moving outside the loop many computations which need not be calculated within the loop.
- Recognizing and replacing redundant computations.
- Replacing (if possible) multiplication of induction variables by addition of those variables.
- Deleting (if possible) references to some variables.
- Using (where possible) the BXLE instruction for loop termination. (The BXLE instruction is the fastest conditional branch; time and space are saved.)

Programming Considerations Using the Optimizer

In general, the specification of OPT=1 or OPT=2 causes compilation time to increase. However, the object code produced is more concise and yields shorter execution times.

The object module logic, when optimized, is identical to the unoptimized logic, except in the following cases:

1. If the list of statement numbers in an Assigned GOTO statement is incomplete, errors, which were not present in the unoptimized code, may arise in the optimized code.
2. With OPT=2, the computational reordering done may produce a different execution time behavior than unoptimized code. For example, a test of an argument of a FORTRAN library function may be executed after the call to the function. This is caused by the movement of the function call to the back target of the loop when the function argument is not changed within the loop.

```
      DO   11   I=1,10
      DO   12   J=1,10
      IF (B(I).LT.0.)GO TO 11
   12 C(J)=SQRT(B(I))
   11 CONTINUE
```

The square root computation will occur before the less-than-zero test, and will result in a message if B(I) is negative. A rearrangement of the program which could avoid this situation can be constructed:

```
      DO   11   I=1,10
      IF (B(I).LT.0.) GO TO 11
      DO   12   J=1,10
12    C(J)=SQRT(B(I))
11    CONTINUE
```

A similar condition may result with the statements:

```
CALL OVERFL(J)
CALL DVCHK(J)
```

These may produce different results when optimized, because computations causing overflow, underflow, or divide-check conditions could be moved out of the loop in which the test occurs.

3. If a programmer defines a subprogram with the same name as a FORTRAN-supplied subprogram (e.g., SIN, ATAN, etc.), errors could be introduced during optimization. If the subprogram stores into its arguments, refers to COMMON, performs I/O, or remembers its own variables from one execution to another, the name of the subprogram must be specified in an EXTERNAL statement to allow the program to be optimized without error.

4. In the statements

```
COMMON   X, Y1(10), W, Z
EQUIVALENCE (Y1,Y2)
DIMENSION Y2(12)
```

there is an implied equivalence of Y2 (11) and W and Y2(12) and Z.

If the optimization feature is not used, and

W=Q and A=Y2(I) (where I=11)

then the value of Q is assigned to A.

However, if OPT=2 is used, and

W=Q and A=Y2(I) (where I=11)

there is no guarantee that the value of Q is assigned to A.

5. When a subprogram is called at one entry point for initialization of reference-by-name arguments, and at another entry point for subsequent computation, certain argument values may not be transmitted. This applies to either arguments of the second call or any argument values redefined between calls and not explicitly defined in COMMON.

   In the following example the incremented value for I may not be transmitted to the subprogram due to the loop initialization optimization.

```
      CALL INIT (I)   SUBROUTINE  INIT (/J/)
            .                .
            .                .
            .                .
      I = 0           ENTRY   COMP
   10  CALL COMP
      I = I + 1
            .
            .
            .
      GO TO 10
```

6. With OPT=2, variables in named COMMON arrays may not be stored on exit from a FORTRAN main program if these variables have not been used in an I/O statement in that main program, or if there is no subroutine call following the definition of these variables.

7. With OPT=2, implied DO variables should not be stored if an END= transfer was made out of a READ statement.

Definition of a Loop

The term 'loop' is used to refer to DO loops and other configurations of coding that a programmer regards as a loop.

If a programmer writes a loop which is preceded by an IF statement, a conditional GOTO statement, or READ statement with END or ERR parameters, the loop is not identified and efficiency is lost. A CONTINUE statement at the end of the range of a DO also obscures a loop (other than a DO loop) that follows the CONTINUE without intervening initialization. The insertion of a labeled CONTINUE statement or any other suitable rearrangement allows the loop to be recognized.

The movement of computations from inside a loop to the initialization coding is done on the assumption that every statement in the loop is executed more frequently than the initialization coding. Occasionally, this assumption fails and computations are moved to a position where they are computed more often. One way to prevent such a move is to make a subprogram of the coding (statements and computations) that is executed less frequently within a loop than it would be in the initialization coding.

The recognition of loops may also be obscured when the programmer knows that some paths through the program cannot occur; for example,

```
 10 IF (L) GOTO 200
 20 I=1
 30 ASSIGN 40 TO J
    GOTO 100
 40 I=I +1
 50 IF (I.LE.N) GOTO 30
    .
    .
    .
100 B(I) = FUNCT (I)
110 GOTO J, (40, 220)
200 ASSIGN 220 TO J
210 GOTO 100
220 CONTINUE
```

From the programmer's point of view, the statements 30 to 50 comprise a loop which is initialized by statement 20. The loop causes an internal subprogram consisting of statements 100 and 110 to be executed. From the compiler's point of view, it appears possible to execute statements in the order 10, 200, 210, 100, 110, 40, 50, 30. The compiler does not recognize the loop, because it appears possible to enter it without passing through the initialization coding in statement 20.

A loop can be obscured by the computed GOTO, because the compiler always assumes that one of the possible branches is to the succeeding statement, even though the programmer knows that such a branch is impos- sible. A loop can also be obscured by a call to the EXIT routine, because the compiler assumes there is a path from such a statement to the next.

Movement of Code Into Initialization of a Loop

Where it is logically possible to do so with OPT=2, the optimizer moves computations from inside the loop to the outside. This movement permits a programmer to do more straightforward coding without penalty in object code efficiency.

If an expression is evaluated inside a loop and all the variables in the expression are unchanged within the loop, the computation is generally moved outside the loop into the coding sequence which initializes the loop. Even if the constant expression is part of a larger expression, this constant expression may still be recognized and moved. However, the movement depends on how the larger expression is written. Table 26 gives examples of expressions and the constant parts which are recognized and moved.

Common Expression Elimination

With OPT=2, if an expression occurs twice in such a way that:

1. any path starting at an entry to the program always passes through the first occurrence of the expression to reach the second occurrence (and any subsequent occurrence), and

2. any evaluation of the second (third, fourth, etc.) expression produces a result identical to the most recent evaluation of the first expression, then the value of the first expression is saved (generally) and used instead of the value of the second (third, fourth, etc.) expression.

In statements such as:

```
A=B + C + D
E=C + D
```

the common expression C + D is not recognized, because the first expression is computed as (B + C) + D.

Induction Variable Optimization

In a loop with OPT=2, an induction variable is a variable that is only incremented by a constant or by a variable whose value is constant in the loop.

When an induction variable is multiplied by a constant in the loop, the optimizer

Table 26. Constant Expressions

Expression where C1, C2,... are constant in the loop	Constant expression recognized and moved
C1 + C2 * C3/SIN (C4)	C1 + C2 * C3/SIN (C4)
C1 + C2 * C3 + B1	C1 + C2 * C3
C1 + B1 + C2 * C3	C2 * C3
B1 + C1 + C2 * C3	C2 * C3
C1 + B1 + B2 + C2 * C3	C2 * C3
C1 * C2/B1	C1 * C2

may replace the multiplication with an addition by introducing a new induction variable into the loop. This new induction variable may make it possible to delete all references to the original induction variable. This deletion is likely to occur if the original induction variable is used only as a subscript within the loop, and the value of the subscript is not used on exit from the loop.

Register Allocation

Some variables are assigned to a register on entry to a loop and retained in the register through part or all of the loop to avoid loading and storing the variable in the loop. Within the loop, the variable is modified only in the assigned register, the value of the variable in storage is not changed. If necessary, the latest value of the variable is stored after exit from the loop.

The value in general register 13, which points to the start of a register save area, remains constant during execution of a subprogram. This register is used to refer to data, and possibly to branch within the program. The value in general register 12 remains constant and is used to branch within the program, and possibly is used to refer to data.

General registers 14 and 15 are used for base addresses and index values on a strictly local basis. Floating-point register 0 and general register 0 are used as locally assigned arithmetic accumulators. General register 1 is used in conjunction with general register 0 for fixed-point arithmetic operations, and to point to argument lists in subprogram linkages.

The remaining registers are used for accumulators, index values, base addresses, and high speed storage (a register reference is faster than a main storage reference).

Because general registers 12 and 13 are not adequate to provide RX branching throughout a large program, general registers 11, 10, and 9 may be pre-empted for RX branching (only if the program exceeds 8K, 12K, and 16K bytes, respectively). (RR branches preceded by loads are required for branching to points beyond the first 16K bytes of the program and possibly to the last part of a program if it exceeds 8K, 12K, or 16K bytes by a small amount.)

COMMON Blocks

Because each COMMON block is independently relocatable, each requires at least one base address to refer to the variables in it. A sequence of coding that refers to a large number of COMMON blocks is slowed down by the need to load base addresses into general registers. Thus, if three COMMON blocks can be combined into one block whose total size is less than 4096 bytes, one base address can serve to refer to all the variables. (Many register loads can be avoided.)

The order in which data is entered into a COMMON block may also affect the number of base addresses needed. For example, if an array of 5000 bytes is placed in a COMMON block and followed by 200 bytes of variables, two base addresses are needed: the beginning address of the first variable and the beginning address of the last differ by more than 4096 bytes. However, if the variables preceded the array, one base address would suffice.

EQUIVALENCE Statements

Optimization tends to be weakened by the occurrence of variables in EQUIVALENCE statements.

When an array appears in an EQUIVALENCE statement, a reference to one of its elements cannot be eliminated as a common expression, nor can the reference be moved out of a loop. However, the elimination and movement of subscript calculations used for making the reference is not affected.

If a variable is made equivalent only to another variable (not in COMMON) of the same type and length, optimization is not weakened. The net effect is that the compiler accepts the two names as alternate pointers to the same storage location. However, if a variable is made equivalent to another variable in any other way, all references to it are 'immobilized': the references cannot be eliminated, moved, confined to registers, or altered in any way.

Multidimensional Arrays

In general, references to higher dimensional arrays are slower than references to lower dimensional arrays. Thus, a set of one-dimensional arrays is more efficient than a single

two-dimensional array in any case where the two-dimensional array can be logically treated as a set of one-dimensional arrays.

Constants occurring in subscript expressions are accounted for at compile time and have no effect at execution time.

Program Structure

If a large number of variables are to be passed among calling and called programs, some of the variables should be placed in the COMMON area. For example, in the main program and subroutine EXAMPL

```
      DIMENSION E(20),I(15)
      READ(10)A,B,C
      CALL EXAMPL(A,B,C,D,E,F,I)
            .
            .
            .
      END

      SUBROUTINE EXAMPL (X,Y,Z,P,Q,R,J)
      DIMENSION Q(20),J(15)
            .
            .
            .
      RETURN
      END
```

time and storage are wasted by allocating storage for variables in both the main program and subprogram and by the subsequent instructions required to transfer variables from one program to another.

The two programs should be written using a COMMON area, as follows:

```
      COMMON A,B,C,D,E(20),F,I(15)
      READ(10)A,B,C
      CALL EXAMPL
            .
            .
            .
      END

      SUBROUTINE EXAMPL
      COMMON X,Y,Z,P,Q(20),R,J(15)
            .
            .
            .
      RETURN
      END
```

Storage is allocated for variables in COMMON only once and fewer instructions are needed to cross reference the variables between programs.

To reduce compilation time for equivalence groups, the entries in the EQUIVALENCE statement should be specified in descending order according to offset. For example, the statement

```
EQUIVALENCE (ARR1(10,10),ARR2(5,5),
            ARR3(1,1),VAR1)
```

compiles faster than the statement

```
EQUIVALENCE (VAR1,ARR3(1,1),ARR2(5,5),
            ARR1(10,10))
```

To reduce compilation time and save internal table space, equivalence groups should be combined, if possible. For example, the statement

```
EQUIVALENCE (ARR1(10,10),ARR2(5,5),VAR1)
```

compiles faster and uses less internal table space than the statement

```
EQUIVALENCE (ARR1(10,10),VAR1),
            (ARR2(5,5),VAR1)
```

Logical IF Statements

A statement such as:

```
IF(A.LT.B.OR.C.GT.F(X).OR..NOT.L)GOTO 10
```

is compiled as though it were written:

```
      IF (A .LT. B) GO TO 10
      IF (C .GT. F(X)) GO TO 10
      IF (.NOT. L) GOTO 10
```

Thus, if A .LT. B is found to be true, the remainder of the logical expression is not evaluated.

Similarly, a statement such as:

```
      IF (D.NE. 7.0 .AND. E.GE.G) I=J
```

is compiled as:

```
      IF (D.EQ. 7.0) GOTO 20
      IF (E.LT.G) GOTO 20
      I=J
20    CONTINUE
```

The order in which a programmer writes logical expressions in an IF statement affects the speed of execution.

If A is more often true than B, then write A .OR. B rather than B .OR. A; and write B .AND. A rather than A .AND. B.

If any of the following occur in a logical expression:

1. a mixture of both .AND. and .OR. operators

2. a .NOT. operator followed by a parenthesized expression

the entire logical expression must be evaluated and efficiency is lost.

Branching

The statement

```
IF(A.GT.B) GOTO 20
```

gives equivalent or better code than

```
   IF(A-B)10,10,20
10 CONTINUE
```

The Assigned GO TO is the fastest conditional branch.

The computed GO TO should be avoided unless four or more statement labels occur within the parentheses.

The statement

```
IF(I-2)20,30,40
```

is significantly faster than

```
GOTO (20,30,40), I
```

Name Assignment

For its internal use, the compiler places names used for variables, arrays, and subprograms into a table. This table is divided into six strings and is searched many times during compilation. Names that are one character long are placed in the first string; names two characters long are placed in the second string; and so on. For faster compiling, the names should be distributed equally among the six strings.

APPENDIX I: CONSIDERATIONS FOR MODELS 91 AND 195

This appendix discusses FORTRAN programming factors that are of special concern to users of IBM System/360 Models 91 and 195.

PROGRAM INTERRUPTION EXIT ROUTINE

The library subroutine that handles interruptions has been modified to recognize precise, imprecise, and multiple imprecise interruptions. Multiple imprecise interruptions may require that the subroutine set more than one of the indicators (for divide check, exponent overflow, and exponent underflow).

Modifications are made to the message that is issued for the following program exceptions:

- Fixed-point divide
- Decimal divide
- Floating-point exponent overflow
- Floating-point exponent underflow
- Floating-point divide

The format of the message issued is

```
IHC210I PROGRAM INTERRUPT (x) OLD PSW
  IS y
```

where x represents one of the letters P (for precise), I (for imprecise), or M (for multiple imprecise of different classes), and y is the hexadecimal representation of the old PSW.

BOUNDARY ADJUSTMENT ROUTINES (MODEL 91 ONLY)

Specification of a system generation option, BOUNDRY=ALIGN, will provide boundary adjustment routines for correction of instructions that cause specification exceptions. However, the nature of these fix-up routines requires the identification of the instruction that causes the exception. Since specification exceptions on the Model 91 generate imprecise interruptions, boundary adjustments cannot be made. Thus, when the FORTRAN library is specified for the Model 91, boundary alignment must not be requested. Because BOUNDRY=ALIGN is the default option, BOUNDRY=NOALIGN must be specified. If BOUNDRY=NOALIGN is not specified during system generation for the Model 91, an error message will occur.

Note: If boundary alignment were allowed, the related task eventually could be terminated for the following reason: Boundary alignment is made with respect to the instruction addressed by the program old PSW, but since the related interruption may have been imprecise, the old PSW may not contain the address of the incorrect instruction.

FLOATING-POINT OPERATIONS

For the Models 91 and 195, floating-point operations are somewhat different from what they are on other models. Discussed below are two of these differences.

Exponent Overflow

A floating-point exponent overflow exception results in the maximum floating-point number (see the publication IBM System/360 Principles of Operation) being placed in the result register. The correct sign of the result is appended to the result in the register. For operations using long precision, all 56 of the fraction bits in the register are set to one. For operations using short precision, the low-order 32 bits in the register remain unchanged. In addition and subtraction, the condition code reflects the sign of the result. This exception produces an imprecise interruption.

Exponent Underflow

The result of an exponent underflow exception is that a true zero is placed in the result register. For long-precision operations, all 56 of the fraction bits in the register are set to zero. For short-precision operations, the low-order 32 bits in the register remain unchanged. In addition and subtraction, the condition

code is set to zero. This exception produces an imprecise interruption if the mask bit (bit 38) in the PSW is set to one.

Note: Whenever an interruption occurs on other models of System/360, system routines provide the setting of the result register when requested. To maintain compatibility, these operations are performed in the hardware of the Models 91 and 195 since the imprecise interruption prohibits the programming technique.

(Where more than one page reference is given, the major reference is first.)

IBM OS FORTRAN IV (G&H) P.G. Printed in U.S.A. GC28-6817-4

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)