IBM Systems Reference Library
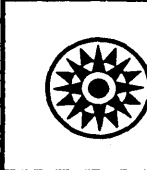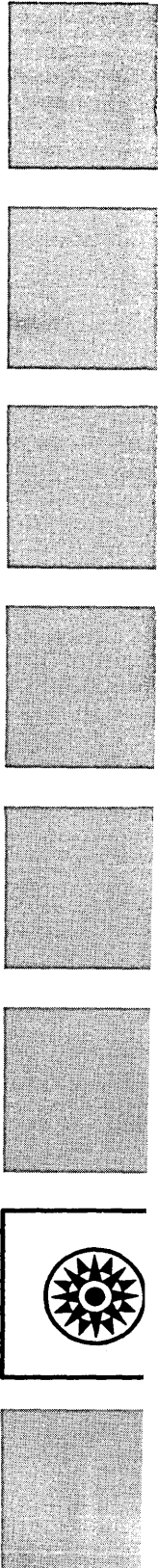
IBM System/360 Operating System

FORTRAN IV (G) Programmer's Guide

Program Number 360S-FO-520

This publication describes how to compile, link edit, and execute a program written in IBM System/360 FORTRAN IV Language.

The purpose of this guide is to enable programmers to compile, link edit, and execute FORTRAN IV programs under control of IBM System/360 Operating System. The FORTRAN IV language is described in the publication IBM System/360 FORTRAN IV Language, Form C28-6515, which is the corequisite to this publication.

The Programmer's Guide is organized to fulfill its purpose at three levels:

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 concerned with making extensive use of the operating system facilities, such as writing their own cataloged procedures, modifying the FORTRAN library, or calculating region size for operating in a multiprogramming environment with a variable number of tasks, should also read the entire Programmer's Guide in conjunction with the following publications, as required:

IBM System/360 Operating System: System Programmer's Guide, Form C28-6550

IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646

IBM System/360 Operating System: Storage Requirements, Form C28-6551

IBM System/360 Operating System: Job Control Language, Form C28-6539

IBM System/360 Operating System: Utilities, Form C28-6586

IBM System/360: FORTRAN IV, Library Subprograms, Form C28-6596

IBM System/360 Operating System: Linkage Editor, Form C28-6538

IBM System/360 Operating System: System Generation, Form C28-6554

IBM System/360 Operating System: Operator's Guide, Form C28-6540

IBM System/360 Operating System: Programmer's Guide to Debugging, Form C28-6670

IBM System/360 Operating System: Messages and Codes, Form C28-6631

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.

• Detailed descriptions of the diagnostic messages produced during compilation and load module execution.

• A list of USA carriage control characters.

• Descriptions of the output from the debug facility.

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 are abbreviated to "Linkage Editor publication."

FIGURES

TABLES

The IBM System/360 Operating System (the 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 to the operating system: the job and the job step, 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 refined 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.)
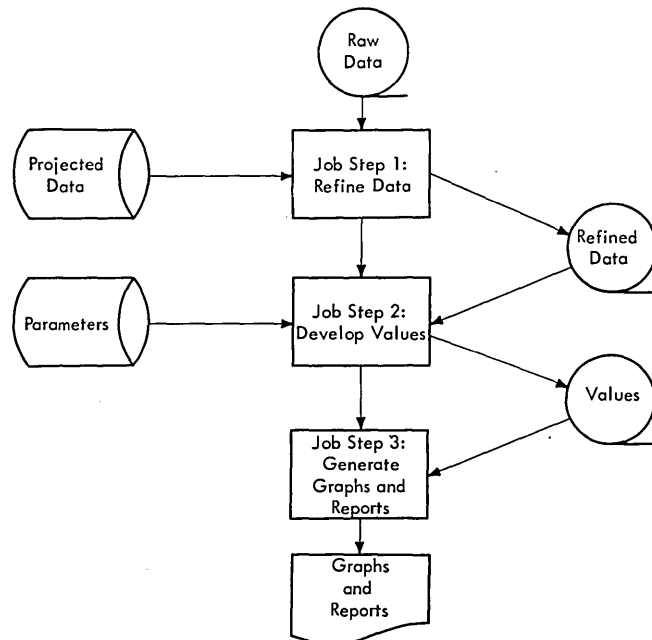
Figure 1. Rocket Firing Job

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 link edited to give a load module. 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 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 link edit or to execute a FORTRAN module. (Note: A JOB statement cannot be cataloged.)

Four FORTRAN cataloged procedures are supplied by IBM. These four cataloged procedures and their uses are:

FORTGC   compile
FORTGCL   compile and link edit
FORTGLG   link edit and execute
FORTGCLG   compile, link edit, and execute

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.

DATA SETS

For FORTRAN processing, a programmer uses DD statements to define the particular data set(s) required for a compile, link 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 is a named collection of data. Several methods are available for internally organizing data sets. Three types of data sets are accessible in FORTRAN processing: sequential data sets, partitioned data sets, and direct access data sets.

A sequential data set is organized in the same way as a data set that resides on a tape volume, but a sequential data set may reside on any type of volume. 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 time method (BSAM) for object time I/O operations.

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. (A member of a PDS is not a data set.) 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 not accessible to a FORTRAN load module.

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.

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 only be transmitted by direct access I/O statements; they cannot be transmitted by 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).

Saying that a data set is sequential, partitioned, or direct access reflects its organization. Saying that a data set is cataloged or that it is a generation data set reflects a method of retrieving the

data set. 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. A generation data set can only be a sequential or direct access data set; a generation data set cannot be a PDS or a member of a PDS. (See the section "Job Control Language" for information on how to specify a generation data set.)

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, den-sity, etc., is stored in the data set labels. The information required in the DD statement used to retrieve a labeled data set is substantially less than that required to retrieve an unlabeled data set.

Data Set Cataloging

To relieve the programmer of the burden of remembering the volume on which a par-ticular 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 program-mer 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.

The FORTRAN programmer uses the job control statements shown in Table 1 to compile, link edit, and execute programs.

Table 1. 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.

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.

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 more powerful 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) and those that provide multiprogramming with a fixed number of tasks (MFT) use sequential schedulers.

Priority schedulers process jobs according to their relative priority and available system resources, and can accept input data from more than one input stream. Systems that provide multiprogramming with a variable number of tasks (MVT) use priority schedulers.

CODING JOB CONTROL STATEMENTS

Job control statements are identified by the initial characters // or /* in card columns 1 and 2, and may contain three fields -- name, operation, and operand (see Figure 2).

NAME FIELD

The name contains between one and eight alphameric or national characters, the first of which must be alphabetic. The name begins in card column 3 and is followed by one or more blanks to separate it from the operation field. The name is used in the following ways:

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 input/output statements in the load module.

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

Figure 2. Job Control Statement Formats

OPERATION FIELD

The operation field contains one of the following operation codes:

JOB
EXEC
DD

If the statement is a delimiter 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 are not order dependent, i.e., they 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 are not order dependent, i.e., they may appear in any order. If a keyword subparameter is omitted, a comma is not required to indicate the omission.

CONTINUING CONTROL STATEMENTS

A control statement can be written in card columns 1 through 72. If a control statement exceeds 71 columns, it may be continued onto the next card. The continuation must be interrupted after the comma that follows the last parameter on the card and a nonblank character must be placed in column 72. The continuation card must contain // in columns 1 and 2, blanks in columns 3 through 15, and the continued portion of the statement must begin in column 16.

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

COMMENTS

Comments must be separated from the last parameter (or the * in a delimiter statement) by one or more blanks and may appear in the remaining columns up to and including column 71.

However, comments may be continued by placing a nonblank character in column 72, // in columns 1 and 2 of the continuation card, and continuing the comment in any column after column 15 (columns 3-15 must be blank). There is no limit to the number of continuation cards that may be used for a single control statement or comment. Also, there is no limitation placed upon the number of comment cards that may be contained in the source program.

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 ^{1}

    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 repeated optionally any number of times in succession.

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

Example: ⎰NEW⎱¹
          ⎱OLD⎰
          ⎰MOD⎱

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. Job name.

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.

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 how to write an accounting routine may be found in IBM System/360 Operating System: Systems Programmer's Guide.) Otherwise, the account number is optional.

Programmer's Name

The "programmer name" is the second positional parameter. If no job accounting information is coded, 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.

Control Statement Messages

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

MSGLEVEL=0
    indicates that only control statement errors and diagnostic messages are written for the programmer.

MSGLEVEL=1
    indicates that all control statements as well as control statement errors and diagnostic messages are written for the programmer.

Note: If an error occurs on a control statement that is continued onto one or more cards, only one of the continuation cards is printed with the diagnostic messages.

Conditions for Terminating a Job

At the completion of a job step, a code is issued indicating the outcome of the job step. The generated code is tested against the conditions stated in control statements. The error codes generated by the FORTRAN compiler are:

0 - No errors or warnings detected.

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

8 - Errors detected, execution may fail. Compilation continues regardless of the errors. If a LOAD option has been specified, a LOAD module will be supplied unless the error code generated is greater than the error level specified by the programmer.

```
r-----------T-----------T--------------------------------------------------------------------------------
|Name       |Operation  |Operand                                                                         |
+-----------+-----------+--------------------------------------------------------------------------------+
|           |           |                              Positional Parameters                             |
|           |           |                                                                                |
|//jobname  |JOB        |[([account-number][,accounting-information])¹ʼ ²ʼ ³]                            |
|           |           |                                                                                |
|           |           |[,programmer-name]⁴ʼ ⁵ʼ ⁶                                                       |
|           |           |                                                                                |
|           |           |                              Keyword Parameters                                |
|           |           |                                                                                |
|           |           | { MSGLEVEL=0 }                                                                 |
|           |           | { MSGLEVEL=1 }                                                                 |
|           |           |                                                                                |
|           |           |[COND=((code,operator)[,(code,operator)]...⁷)⁸]                                 |
|           |           |                                                                                |
|           |           |     [PRTY=nn]⁹                                                                 |
|           |           |                                                                                |
|           |           |     [MSGCLASS=x]⁹                                                              |
|           |           |                                                                                |
|           |           |     [REGION=nnnnnK]⁹                                                           |
+-----------+-----------+--------------------------------------------------------------------------------+
|¹If the information specified ("account-number" and/or "accounting-information") con-|
| tains blanks, parentheses, or equal signs, it must be delimited by apostrophes instead|
| of parentheses.                                                                       |
|²If only "account-number" is specified, the delimiting parentheses may be omitted.     |
|³The maximum number of characters allowed between the delimiting parentheses or|
| apostrophes is 144.                                                                   |
|⁴If "programmer-name" contains commas, parentheses, apostrophes, or blanks, it must be|
| enclosed within apostrophes.                                                          |
|⁵When an apostrophe is contained within "programmer-name", the apostrophe must be shown|
| as two consecutive apostrophes.                                                       |
|⁶The maximum number of characters allowed for "programmer-name" is 20.                 |
|⁷The maximum number of repetitions allowed is 7.                                       |
|⁸If only one test is specified, the outer pair of parentheses may be omitted.          |
|⁹This parameter is used by the priority scheduler only. The sequential scheduler|
| ignores it.                                                                           |
L--------------------------------------------------------------------------------------
```

Figure 3. JOB Statement

Figure 4. Sample JOB Statements

12 - Severe errors detected, execution is impossible.

16 - Terminal errors detected, compiler operation terminated. (If a terminal error is detected during load module execution, a 16 is issued.)

The COND parameter 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 rela-

tionship 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.

Assigning Job Priority (PRTY):
(Used by Priority Schedulers Only)

To assign a priority other than the default job priority (as established in the input reader procedure), PRTY=nn must be coded in the operand field of the JOB statement. The term "nn" is to be replaced with a decimal number from 0 through 14 (the highest priority number is 14).

Whenever possible, avoid using priority 14. This is used by the system to expedite processing of exceptional jobs. 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 by 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 job scheduler to be routed to an output class other than the normal message class, A. Replace the letter "x" with an alphabetic or numeric character. An output writer, which is assigned to process this class, will transfer this data to a specific device.

Specifying Main Storage Requirements for a Job (REGION):
(Used by Priority Schedulers Only)

REGION=nnnnnK can be specified to indicate the amount of main storage to be allocated to the job. Replace the term "nnnnn" with the number of 1024-byte areas

to be allocated to the job; e.g., REGION=100K. This number can range from one to five digits and cannot exceed 16,384.

If the REGION parameter is omitted, the default region size (as established in the input reader procedure) is assumed.

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.

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 the job 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.

Example 1 of Figure 6 shows the EXEC statement used to execute a program. Example 2 in Figure 6 shows an EXEC statement that invokes 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.

```
r----------------T---------T----------------------------------------------------------------------1
|Name            |Operation| Operand                                            |                  |
+----------------+---------+----------------------------------------------------------------------+
|                |         |                          Positional Parameter                        |
|                |         |                                                                       |
|//[stepname]1   |EXEC     | /PROC=cataloged-procedure-name  \                                     |
|                |         | \cataloged-procedure-name        \                                    |
|                |         | <PGM=program-name                 >                                   |
|                |         | /PGM=*.stepname.ddname           /                                    |
|                |         | \PGM=*.stepname.procstep.ddname_/                                     |
|                |         |                                                                       |
|                |         |                          Keyword Parameters                          |
|                |         |                                                                       |
|                |         | [ {PARM               }                            ]                  |
|                |         | [ {PARM.procstep2     }=(option[,option]...)3,4,5  ]                  |
|                |         |                                                                       |
|                |         | [ {ACCT               }                            ]                  |
|                |         | [ {ACCT.procstep2     }=(accounting-information)3,6,7 ]               |
|                |         |                                                                       |
|                |         | [ {COND               }                            ]                  |
|                |         | [ {COND.procstep2     }=((code,operator[,stepname[.procstep]])        |
|                |         |          [,(code,operator[,stepname[.procstep]])]...8)9]              |
|                |         |                                                                       |
|                |         | [ {TIME               }                    10,11 ]                    |
|                |         | [ {TIME.procstep2     } = (minutes,seconds)      ]                    |
|                |         |                                                                       |
|                |         | [ {REGION             }            10 ]                               |
|                |         | [ {REGION.procstep2   } = nnnnnK      ]                               |
+----------------L---------L----------------------------------------------------------------------+
| 1If  information  from  this  control  statement  is  referred  to  in  a  later  job  step,|
|  "stepname" is required.                                                                     |
| 2If this format is selected, it may be repeated in the EXEC statement, once  for  each|
|  step in the cataloged procedure.                                                            |
| 3If the information specified contains blanks, parentheses, or equal signs, it must be|
|  delimited by apostrophes instead of parentheses.                                            |
| 4If  only  one option is specified and it does not contain any blanks, parentheses, or|
|  equal signs, the delimiting parentheses may be omitted.                                     |
| 5The maximum number of  characters  allowed  between  the  delimiting  apostrophes  or|
|  parentheses is 40.  The PARM parameter cannot occupy more than one card.                    |
| 6If  "accounting-information"  does  not contain commas, blanks, parentheses, or equal|
|  signs, the delimiting parentheses may be omitted.                                           |
| 7The maximum number of  characters  allowed  between  the  delimiting  apostrophes  or|
|  parentheses is 144.                                                                         |
| 8The maximum number of repetitions allowed is 7.                                             |
| 9If only one test is specified, the outer pair of parentheses may be omitted.               |
|10This parameter is used by priority schedulers only.  Sequential schedulers ignore it.|
|11If  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.                   |
L----------------------------------------------------------------------------------------------J
```

Figure 5. EXEC Statement

OPERAND FIELD

Positional Parameter

The options in the positional parameter of an EXEC statement 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 pro-

gram 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=FORTGC
            or
    // EXEC FORTGC

| NAME | DEPT. | BLDG. | PHONE NO. | PROJECT NO. | PROJECT I.D. | PROJECT NAME | DATE DUE OUT | SHEET___OF___ |

```
            Example 1
// EXEC PGM=TEYFORT,ACCT=(896,427),COND=(7,LT),TIME=200,REGION=100K

            Example 2
//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=JOBLA
```

• Figure 6.　Sample EXEC Statements

indicates that the cataloged procedure FORTGC is to be executed.

Specifying a Program in a Library:

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

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

　　indicates that the load module IEWL is executed. (A load module in a private library is identified to the operating system 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 statements,

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

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.

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=IEYFORT
//SYSPRINT   DD    SYSOUT=A
//SYSPUNCH   DD    UNIT=SYSCP
//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
```

The statement S2 in the input stream indicates that the name of the program is taken from the DD statement SYSLMOD in the procedure step LKED in the procedure FORT, which was invoked by the EXEC 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 the 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 format for compiler options, and those linkage editor options most applicable to the FORTRAN programmer is shown in Figure 7.

Detail 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.

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 to all codes issued by previous job steps.

```
r----------------------------------------------------------------------------------------------------------
|Compiler:
|
| {PARM        }   ' {LIST    }                                                    {,SOURCE    }
| {PARM.procstep} = {NOLIST  }   [,NAME=xxxxxx]        [,LINECNT=xx] {,NOSOURCE }
|
|                    {,DECK   } {,MAP   } {,LOAD    } {,BCD    } {,ID    } '1, 2
|                    {,NODECK } {,NOMAP } {,NOLOAD  } {,EBCDIC } {,NOID  }
|
|Linkage Editor:
|
| {PARM          }  ⌈MAP ⌉
| {PARM. procstep} =(⌊XREF⌋  [,LET]  [,NCAL]  [,LIST])1
+----------------------------------------------------------------------------------------------------------
|1The subparameters (options) are keyword subparameters.
|2If the information specified contains blanks, parentheses, or equal signs, it must
|  not be delimited by parentheses, but by apostrophes.
L----------------------------------------------------------------------------------------------------------
```

• Figure 7.   Compiler and Linkage Editor Options

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 by Priority Schedulers Only)

To limit the computing time used by a single job step or cataloged procedure step, 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 step is not completed in this time, the entire job is terminated. (If the job step execution time is expected to exceed 1439 minutes (24 hours) 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
    assigns 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
    assigns a time limit for a single step of a cataloged procedure. This parameter overrides, for the named step, any TIME parameter which is present. As many parameters of this form as there are steps in the cataloged procedure being executed can be written.

Specifying Main Storage Requirements for a Job Step (REGION):
(Used by Priority Schedulers Only)

The REGION parameter may be specified in the JOB statement, in which case it overrides REGION parameters specified in the EXEC statements and applies to all steps of the job. However, if it is desired to allot to each step only as much storage as it requires, the REGION parameter should be omitted from the JOB statement and EXEC statements should contain a REGION parameter specifying the amount of main storage to be allocated to the associated job step. If the REGION parameter is omitted from both JOB and EXEC statements, the default region size (as established in the cataloged procedure for the input reader) is assumed. The size is specified in the form "nnnnnK" where "nnnnn" is the number of 1024-byte areas to be allocated to the job step; e.g., REGION=100K.

REGION
    specifies a region size for the job step or for the entire cataloged procedure. For a cataloged procedure, this parameter overrides all REGION

parameters that may have been specified in the procedure.

REGION.procstep
specifies a region size for a single step of a cataloged procedure. This parameter overrides the REGION parameter in the named cataloged procedure step, if one is present. As many parameters of this form as there are steps in the cataloged procedure being executed can be written.

For a discussion of the region size required for FORTRAN jobs, see "Cataloged Procedures."

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.

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
is used to concatenate partitioned data sets with the system library; that is, the operating system library and the data sets specified in the JOBLIB DD statement are temporarily combined to form one library. The JOBLIB statement must immediately follow a JOB statement, and the concatenation is in effect only for the duration of the job. In addition, "DISP=(OLD,PASS)" must be specified in the JOBLIB DD statement.

(See the following text concerning the DISP parameter.) Only one JOBLIB statement may be specified for a job.

The "PGM=program name" parameter in the EXEC statement refers to a load module in the system library. However, if this parameter refers to a load module in a private library, a JOBLIB statement identifying the PDS in which the module resides must be specified for the job. The JOBLIB statement concatenates the system library with the private library.

Name	Operation	Operand[1]
//$\left\{\begin{array}{l}\text{ddname}\\\text{procstep.ddname}\\\text{JOBLIB}[3]\end{array}\right\}$[2]	DD	**Positional Parameter** $\left[\begin{array}{l}\text{*}\\\text{DUMMY}\\\text{DATA}\end{array}\right]$[4] **Keyword Parameters**[5] $\left[\begin{array}{l}\text{DDNAME=ddname}\\\\\text{DSNAME= }\left\{\begin{array}{l}\text{dsname}\\\text{dsname(element)}\\\text{*.ddname}\\\text{*.stepname.ddname}\\\text{*.stepname.procstep.ddname}\\\text{\&name}\\\text{\&name(element)}\end{array}\right\}\end{array}\right]$ [UNIT=(subparameter-list)] [DCB=(subparameter-list)] [VOLUME=(subparameter-list)] $\left[\begin{array}{l}\text{SPACE=(subparameter-list)}\\\text{SPLIT=(subparameter-list)}\\\text{SUBALLOC=(subparameter-list)}\end{array}\right]$ [LABEL=(subparameter-list)] $\left[\begin{array}{l}\text{DISP=(subparameter-list)}\\\text{SYSOUT=A}\\\text{SYSOUT=B}\\\text{SYSOUT=(x[,program-name][,form-no.])}\end{array}\right]$[6,7] [SEP=(subparameter-list)]

[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 DATA or * is specified, keyword parameters cannot be specified.

[5]If "subparameter-list" consists of only <u>one</u> subparameter and no leading comma (indicating the omission of a positional subparameter) is required, the delimiting parentheses may be omitted.

[6]This form of the parameter is used only with priority schedulers.

[7]If program-name and form no. are omitted, the delimiting parentheses can be omitted.

Figure 8. Data Definition Statement

The library indicated in the JOBLIB statement is searched before the system library is searched.

Note: A JOBLIB statement does <u>not</u> have to be entered for load modules created in this job, or for permanent members of the system library.

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. Other partitioned data sets (not individual members of a PDS) may also be concatenated with the data set specified in the JOBLIB DD statement. Therefore, the

system library may be concatenated with several partitioned data sets.

Note: In concatenation of data sets, neither of the designated data sets may be in the input stream. Also, data sets whose records are of different length and/or different formats cannot be concatenated.

OPERAND FIELD

For purposes of discussion, parameters for the DD statement are divided into seven functions:

• Specify data in the input stream.

• Specify unit record data sets.

• Retrieve a previously created and cataloged data set.

• Retrieve a data set which was 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:

1. Processing unit record data sets.

2. Retrieving data sets created in previous job steps.

3. Retrieving data sets created and cataloged in previous jobs.

See Figure 9 for applicable parameters.

The method of retrieving uncataloged data sets created in previous jobs is also discussed in this section.


```
 ( {*    } 1                                       )
 ( {DATA }                                         )
 (                                                 )
 (            ( dsname                    )        )
 (            ( dsname(element)           )        )
 ( DSNAME=   <  *.ddname                   >        )
 (            ( *.stepname. ddname         )        )
 (            ( *.stepname.procstep.ddname )        )
 (            ( &name                     )        )
 (            ( &name(element)            )        )

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

         (  (PRTSP=0)                  )
         (  {PRTSP=1{                  )
         (  {PRTSP=2{                  )
 DCB=(  <  (PRTSP=3)              > )
         (  ({MODE=E} {,STACK=1}} )
         (  ({MODE=C} {,STACK=2}} )

 (SYSOUT=A                                         )
 (SYSOUT=B                                         )
 (SYSOUT=(x[,program-name][,form-no.])7, 8         )
 <                                                 >
 (           (OLD)  [,DELETE ]4                    )
 ( DISP=(   {NEW{  [,KEEP    ]     )5              )
 (           {MOD{  [,PASS    ]                    )
 (           (SHR)  [,CATLG   ]                    )
 (                  [,UNCATLG ]                    )

 LABEL=(subparameter-list)6

 VOLUME=(subparameter-list)6
```

[1]If either of these two parameters is selected, it must be the only parameter selected.
[2]If neither "n" nor "P" is specified, 1 is assumed.
[3]If only "name" is specified, the delimiting parentheses may be omitted.
[4]The assumption for the second subparameter is discussed in "Specifying the Disposition of a Data Set" in this section.
[5]The subparameters are positional.
[6]See the section "Creating Data Sets."
[7]This form of the parameter is used only with priority schedulers.
[8]If program-name and form no. are omitted, the delimiting parentheses can be omitted.

Figure 9. DD Statement 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."

```
          1-10        11-20       21-30       31-40       41-50       51-60       61-70       71-80
1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890

   Example 1: Printer
//SYSPRINT DD SYSOUT=A,DCB=PRTSP=2


   Example 2: Card Punch
//SYSPUNCH DD UNIT=SYSCP,DCB=STACK=2


   Example 3: Card Reader
//SYSIN DD *
```

Figure 10. Examples of DD Statements for Unit Record Devices

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 cannot contain // or /* in the first two characters of a record.

DATA

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

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. Parallel, "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 for the data set.

DCB Parameter:

DCB=PRTSP={0|1|2|3}

is used to indicate line spacing for the printer. The digits 0, 1, 2, and 3 indicate no space, single space, double space, and triple space, respectively. This subparameter is not effective if A (for USA carriage control characters) has been specified in the RECFM parameter (refer to the paragraph on Record Format in the section "Creating Data Sets").

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

specify options for the card read punch. The MODE subparameter indicates whether the card is transmitted in column binary mode (C) or EBCDIC mode (E).

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

Routing a Data Set to an Output Stream (SYSOUT):

Through the use of 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. With systems providing multiprogramming with a fixed number of tasks, the processing program that writes the data must be in the lowest priority partition.

SYSOUT=B

can be used with sequential schedulers to indicate the system card punch unit. The priority scheduler will route the output to class B.

SYSOUT=(x[,program-name][,form-no.])

is used with priority schedulers. When priority schedulers are used, 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 character "x" can be alphabetic or numeric, specifying the system output class. 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 describing 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-no." 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. If more than one unit is to be allocated, the UNIT parameter is specified, too.

If a data set is created with standard labels in a previous job but was not cataloged, information pertaining to the data set, such as record format, density, volume sequence number, etc., is stored in the label; 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), 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 fully qualified 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.

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 third most recent member
   of the generation data group FIRING.
   (See the Data Management publication
   for a 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 cannot be read as input to a
   FORTRAN object program or created as
   output from a FORTRAN object program
   even though the member name has been
   specified in the DSNAME parameter of a
   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
   control statements:

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

   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.

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;

Sample Coding Form

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

Example 1: Retrieving a Cataloged Data Set
//FT09F001 DD DSNAME=MATH,DISP=(OLD,PASS)

Example 2: Retrieving a Data Set Created in a Previous Job Step
//FT10F001 DD DSNAME=*.STEP4.FT07F001,DISP=(MOD,KEEP)

Example 3: Retrieving an Uncataloged Data Set Created in a Previous Job
//FT11F001 DD DSNAME=MATH,DISP=OLD,UNIT=180,VOLUME=SER=Z1

Figure 11. Retrieving Previously Created Data Sets

"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." If it is required to refer to this name in a separate job (i.e., because of abnormal termination) the name is "&name.JOBNAME."

DSNAME=&name(element)
   assigns a name to a member of a temporary PDS. The name is assigned in the same manner as the "DSNAME=&name." If it is required to refer to this name in a separate job (i.e., because of abnormal termination) the name is "&name.JOBNAME."

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

$$DISP=\left(\begin{Bmatrix}NEW\\OLD\\MOD\\SHR\end{Bmatrix}\begin{bmatrix},DELETE\\,KEEP\\,PASS\\,CATLG\\,UNCATLG\end{bmatrix}\right)$$

is used for all data sets residing on magnetic tape or direct access volumes.

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

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, but records can be added to the data set. Before the first I/O operation for the data set occurs, the data set is positioned after the last record. If MOD is specified and (1) the volume serial number is omitted, and (2) the data set is not cataloged or passed, then MOD is ignored and NEW assumed.

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.

The second subparameter indicates the disposition of the data set at 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 option 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 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 dismounting 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 prior to this job, it remains in existence at the end of the job.

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. It cannot be placed in a catalog procedure.

```
r-----T---------T---------------------------1
|Name |Operation|Operand                     |
+-----+---------+---------------------------+
|/*   |         |                            |
L-----L---------L---------------------------J
```
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.

A FORTRAN source module may be processed starting with compilation and ending with execution. In this case three steps are required: compile the source module to obtain an object module, link edit the object module to obtain a load module, and execute the load module. Job control statements are required for each of these steps 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 supples 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 also required at the end of this data.)

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

COMPILE

The cataloged procedure for compilation is FORTGC. This cataloged procedure con-

sists of the control statements shown in Figure 43 in "Cataloged Procedures."

Figure 13 shows control statements that can be used to invoke FORTGC. 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
//FORT.SYSIN DD *
```
```
┌─────────────────────────────────────────────┐
│            FORTRAN Source Module              │
└─────────────────────────────────────────────┘
```
```
/*
```

Figure 13. Invoking the Cataloged Procedure FORTGC

Single Compile: A sample deck structure to compile a single source module is shown in Figure 14.

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

Figure 14. Compiling a Single Source Module

Batched Compile: A sample deck structure to batch compile is shown in Figure 15.

```
//JOBBC JOB 00,FORTRANPROG,MSGLEVEL=1
//EXECC EXEC PROC=FORTGC
//FORT.SYSIN DD *
```
```
┌─────────────────────────────────────────────┐
│         First FORTRAN Source Module           │
└─────────────────────────────────────────────┘
                       .
                       .
                       .
┌─────────────────────────────────────────────┐
│          Last FORTRAN Source Module           │
└─────────────────────────────────────────────┘
```
```
/*
```

Figure 15. 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, the control program is called at the end of the compilation. If the next card is a FORTRAN statement, the FORTRAN compiler remains as the controlling program.

COMPILE AND LINK EDIT

The cataloged procedure to compile FORTRAN source modules and link edit the resulting object modules is FORTGCL. This cataloged procedure consists of the control statements shown in Figure 44 in "Cataloged Procedures."

Figure 16 shows control statements that can be used to invoke FORTGCL.

```
//jobname JOB
// EXEC FORTGCL
//FORT.SYSIN DD *
┌─────────────────────────────────────────┐
│           FORTRAN Source Module           │
└─────────────────────────────────────────┘
/*
```

Figure 16. Invoking the Cataloged Procedure FORTGCL

LINK EDIT AND EXECUTE

The cataloged procedure to link edit FORTRAN object modules and execute the resulting load module is FORTGLG. This cataloged procedure consists of the control statements shown in Figure 45 in "Cataloged Procedures."

Figure 17 shows control statements that can be used to invoke FORTGLG.

```
//jobname JOB
// EXEC FORTGLG
//LKED.SYSIN DD *
┌─────────────────────────────────────────┐
│           FORTRAN Object Module           │
└─────────────────────────────────────────┘
/*
```

Figure 17. Invoking the Cataloged Procedure FORTGLG

Figure 18 illustrates a sample deck structure to link edit and execute several object modules in the input stream as one load module.

```
//JOBBLG JOB 00,FORTPROG,MSGLEVEL=1
//EXECLG EXEC PROC=FORTGLG
//LKED.SYSIN DD *
┌─────────────────────────────────────────┐
│         First FORTRAN Object Module        │
└─────────────────────────────────────────┘
                    .
                    .
                    .
┌─────────────────────────────────────────┐
│         Last FORTRAN Object Module         │
└─────────────────────────────────────────┘
/*
//GO.SYSIN DD *
┌─────────────────────────────────────────┐
│                    Data                    │
└─────────────────────────────────────────┘
/*
```

Figure 18. Link Edit and Execute Several Object Modules in the Input Stream

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.

Figure 19 illustrates a sample deck structure that link edits and executes object modules that are members of the cataloged sequential data set, OBJMODS, as a single load module. Reading of a data set in the input stream is accomplished by using data set reference number 5.

```
//JOBBLG JOB 00,FORTPROG,MSGLEVEL=1
//EXECLG EXEC FORTGLG
//LKED.SYSIN DD DSNAME=OBJMODS,DISP=OLD
//GO.SYSIN DD *
┌─────────────────────────────────────────┐
│                    Data                    │
└─────────────────────────────────────────┘
/*
```

Figure 19. Link Edit and Execute Several Object Modules in a Cataloged Data Set

COMPILE, LINK EDIT, AND EXECUTE

The fourth cataloged procedure, FORTGCLG, passes a source module through three procedure steps: compile, link edit, and execute. This cataloged procedure consists of the control statements shown in Figure 46 in "Cataloged Procedures."

Figure 20 shows control statements that can be used to invoke FORTGCLG.

```
//jobname JOB
//    EXEC PROC=FORTGCLG
//FORT.SYSIN DD *
r--------------------------------------------1
|                 FORTRAN Source Module                 |
L--------------------------------------------J
/*
```

Figure 20.   Invoking the Cataloged Proce-
              dure FORTGCLG

Single Compile, Link Edit, and Execute:
Figure 21 shows a sample deck structure to
compile, link edit, and execute a single
source module.

```
//JOBSCLG JOB 00,FORTPROG,MSGLEVEL=1
//EXECC EXEC FORTGCLG
//FORT.SYSIN DD *
r--------------------------------------------1
|                 FORTRAN Source Module                 |
L--------------------------------------------J
/*
//GO.SYSIN DD *
r--------------------------------------------1
|                          Data                          |
L--------------------------------------------J
/*
```

Figure 21.   Single Compile, Link Edit, and
              Execute

Batched Compile, Link Edit, and Execute:
Figure 22 shows a sample deck structure to
batch compile, link edit, and execute. The
source modules are placed in the input
stream along with a data set that is read
using data set reference number 5.

```
//JOBBCLG JOB 00,FORTPROG,MSGLEVEL=1
//EXECCLG EXEC FORTGCLG
//FORT.SYSIN DD *
r--------------------------------------------1
|                First FORTRAN Source Module                |
L--------------------------------------------J
                        .
                        .
                        .
r--------------------------------------------1
|                Last FORTRAN Source Module                |
L--------------------------------------------J
/*
//LKED.SYSIN DD *
r--------------------------------------------1
|                      Object Modules                      |
L--------------------------------------------J
/*
//GO.SYSIN DD *
r--------------------------------------------1
|                          Data                          |
L--------------------------------------------J
/*
```

Figure 22.   Batched Compile, Link Edit, and
              Execute

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
devices classes are also established and
must be used by the programmer.

Compiler Name

   The program name for the compiler is
IEYFORT.   If the compiler is to be executed
without using the supplied cataloged proce-
dures in a job step, an EXEC statement of
the form

//   EXEC PGM=IEYFORT

must be used. (For more detailed informa-
tion on procedures and options in calling
IEYFORT, refer to Appendix A, "Invoking the
FORTRAN Compiler.")

Compiler ddnames

   The compiler can use four data sets. To
establish communication between the compil-
er and the programmer, each data set is
assigned a specific ddname. Each data set
has a specific function and device require-
ment.   Table 2 lists the ddnames, func-
tions, 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 sys-
tem. With these two data sets, only a
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 SYSPUNCH and/or SYSLIN DD state-
ment must be supplied.

Table 2. Compiler ddnames

ddname	FUNCTION	DEVICE REQUIREMENTS
SYSIN	reading the source program	•card reader •magnetic tape •direct-access
SYSPRINT	writing the storage map, listing, and messages	•printer •magnetic tape •direct-access
SYSPUNCH	punching the object module deck	•card punch[1] •magnetic tape •direct-access
SYSLIN	output data set for the object module used as input to the linkage editor	•direct-access •magnetic tape •card punch[1]

[1]These must not be same card punch devices

Table 3. Device Class Names

CLASS NAME	CLASS FUNCTIONS	DEVICE TYPE
SYSSQ	writing, reading, backspacing (sequential)	•magnetic tape •direct-access
SYSDA	writing, reading, backspacing, 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 4. Correspondence Between Compiler ddnames and Device Classes

ddname	Possible Device Classes
SYSIN	SYSSQ, or the input stream device (specified by DD * or DD DATA), or a device specified as a card reader
SYSPRINT	A,SYSSQ
SYSPUNCH	B,SYSCP
SYSLIN	SYSSQ,SYSDA

For the DD statement SYSIN or SYSPRINT, an intermediate storage device (direct-access or magnetic tape) may be specified instead of the card reader or printer.

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 that device; a new job or job step can print the contents of the data set. When the SYSPRINT data set is written on an intermediate storage device, carriage control characters are placed in the records.

Compiler Device Classes

Names for input/output device classes used for compilation are also specified for the operating system when the system is generated. The class names, functions, and types of devices are shown in Table 3.

The data sets used by the compiler must be assigned to the device classes listed in Table 4.

Compiler Options

Options may be passed to the compiler through the PARM parameter in the EXEC statement (see Figure 23). The following information may be specified:

1. Whether a listing of an object module is printed.

2. 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).
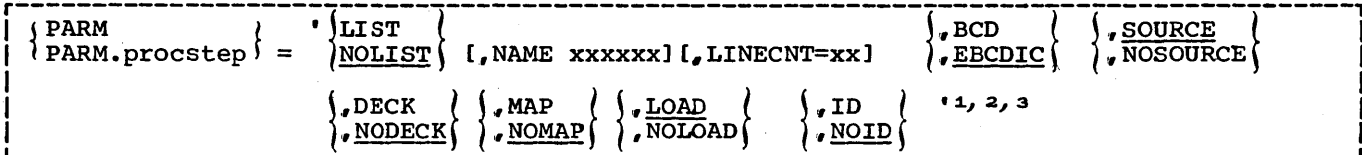
```
r---------------------------------------------------------------------------------------------¬
| (PARM          )   '|LIST  |                                      |,BCD   |  |,SOURCE  |     |
| (PARM.procstep) = |NOLIST| [,NAME xxxxxx][,LINECNT=xx]          |,EBCDIC|  |,NOSOURCE|     |
|                                                                                             |
|                   |,DECK  |  |,MAP  |  |,LOAD  |    |,ID  |  '1, 2, 3                       |
|                   |,NODECK|  |,NOMAP|  |,NOLOAD|    |,NOID|                                 |
+---------------------------------------------------------------------------------------------+
| ¹If the information specified contains blanks, parentheses, or equal signs, it must be      |
|  delimited by apostrophes; otherwise, parentheses may be used.                              |
| ²If only one  option is specified  and it does not  contain any blanks, parentheses or      |
|  equal signs, the delimiting parentheses or apostrophes may be omitted.                     |
| ³The maximum number  of characters allowed between the delimiting apostrophes or paren-     |
|  theses is 40.  The PARM parameter cannot occupy more than one card.                        |
L---------------------------------------------------------------------------------------------J
```

● Figure 23. Compiler Options

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 module is printed.

8. Whether the compiler writes the object module on external storage for input to the linkage editor.

Options specified in the PARM parameter may be in any order.

LIST or NOLIST: The LIST option indicates that the object module listing is written on the data set specified by the SYSPRINT DD card. (The statements in the object module listing are in a pseudo assembly 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 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 (xx) of lines per page that will be written on the data set specified by the SYSPRINT DD statement when a source listing is generated by the com-

piler. The specified number, xx, may be anywhere in the range from 1 to 99. If LINECNT is not specified, the number of lines will be obtained from the system (the default number may be changed by the installation).

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. Intermixing of BCD and EBCDIC in the source module is not allowed.

Note: 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. (n represents the statement number.)

SOURCE or NOSOURCE: The SOURCE option specifies that the source listing is written on the data set specified by the SYSPRINT DD statement. The NOSOURCE option indicates that the source listing is not 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 the object module deck is not 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, which appear in the object module, is written on the data set specified by the SYSPRINT DD statement. The type and location of each name is listed. The NOMAP option specifies that the table of names is not written. A description of the map is given in the section "System Output."

LOAD or NOLOAD: The LOAD option indicates that the object module is written on the data set specified by the SYSLIN DD statement. This option must be used if the cataloged procedure to compile and link edit, or to compile, link 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.

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."

Note: The compiler default options shown in this publication are standard IBM defaults; however, at 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 deck, 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 24).

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.

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

Figure 24. Multiple Compilation Within a Job Step

The first main program in a multiple compilation is given the name specified in the NAME option (only if this program is not preceded by a SUBROUTINE or FUNCTION subprogram); all subsequent main programs are given the name MAIN. However, if the NAME option is not specified, only those main programs that are physically first in a multiple compilation are given the name MAIN. For example, in the multiple compilation,

```
//MULCOM JOB
//     EXEC FORTGC,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. However, had the order of the first two programs been reversed, the name IOR would not have been applied to any of the programs illustrated.

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

```
r----------------------------------r----------------------------------r
| Object Module 1 | Object Module 2 |   ...
L----------------------------------i----------------------------------J
```
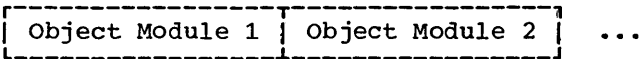
LINKAGE EDITOR PROCESSING

The linkage editor processes FORTRAN object modules, resolves any references to subprograms, and constructs a load module. Communication with the linkage editor is established through a programmer supplied EXEC statement and DD statements that define all required data sets. The user also has the option of supplying linkage editor control statements.

Linkage Editor Name

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

IEWLE150	15,360 bytes
IEWLE180	18,432 bytes
IEWLE440	45,056 bytes

Object modules processed by the linkage editor cannot exceed 512K bytes, because this is the maximum that can be processed by program FETCH.

All facilities described for the linkage editor in this publication are available with all three linkage editors, except that blocking the primary input and primary output is available only with the higher level linkage editor, IEWLE440.

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. 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 FORTRAN external references in the object modules.

Secondary input resolves any 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 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. An object module, which can be accompanied by linkage editor control statements, is a sequential data set or is a member of a PDS. A 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 load 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

Table 5. 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 •magnetic tape •direct-access
SYSLMOD	output data set for the load module	•direct-access
user-specified	additional libraries and object modules	•direct-access •magnetic tape

are written on an intermediate storage device or a printer. In addition, a work data set is required by the linkage editor to do its processing. Figure 25 shows the I/O flow in linkage editor processing.

All data sets specified by SYSLIB or SYSLMOD must be partitioned data sets. (Additional inputs are partitioned data sets or sequential data sets.) The ddname for the DD statement that identifies any additional libraries is written in INCLUDE and LIBRARY statements and is not fixed by the linkage editor.
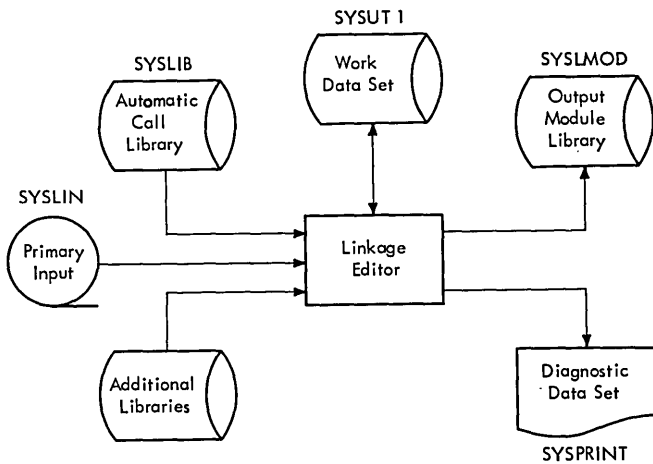
Figure 25. Linkage Editor Input and Output

The device classes used by the compiler (see Table 3) 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 6.

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 5.

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.

Table 6. Correspondence Between Linkage
Editor ddnames and Possible
Device Classes

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

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
(PDS) or a sequential data set. The
"ddname" specifies a DD statement that
defines either a PDS 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 the name
of a member of a PDS and 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 PDS 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 process-
ing is complete.

The LIBRARY statement differs from the
INCLUDE statement in that external referen-
ces specified in the LIBRARY statement are
not resolved until all other processing,
other than those references reserved for
the automatic call library, are 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 26 shows the control statements
for this job. (Note: 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.

Linkage Editor Priority

If modules with the same name appear in
the input to the linkage editor, only one
of the modules is inserted in the output
load module. The following priority for
modules is established by the linkage edi-
tor:

1. Modules appearing in SYSLIN or modules
   identified by INCLUDE statements.

2. Modules identified by the LIBRARY
   statement.

3. Modules appearing in SYSLIB.

For example, if a module named SIN
appears both in a module identified in a
LIBRARY statement and in the automatic call

```
r----------------------------------------------------------------------------------------------------
|//JOBX      JOB
|//STEP1     EXEC      PGM=IEYFORT,PARM='NAME=MAIN,LOAD'
|                               .
|                               .
|                               .
|//SYSLIN    DD        DSNAME=&GOFILE,DISP=(,PASS),UNIT=SYSSQ
|//SYSIN     DD        *
|            Source module for MAIN
|/*
|//STEP2     EXEC      PGM=IEYFORT,PARM='NAME=SUB1,LOAD'
|                               .
|                               .
|                               .
|//SYSLIN    DD        DSNAME=&SUBPROG1,DISP=(,PASS),UNIT=SYSSQ
|//SYSIN     DD        *
|            Source module for SUB1
|/*
|//STEP3     EXEC      PGM=IEYFORT,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)
|/*
L----------------------------------------------------------------------------------------------------
```

Figure 26. Linkage Editor Example

library, only the module identified in the
LIBRARY statement is inserted in the output
load module.

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

Multiple Link Editing Within a Step

Just as the compiler can perform several
compilations within a procedure step or job
step (batched compilation), the linkage
editor can produce several load modules
within a single procedure step or job step.
Another linkage editor control statement,
the NAME statement, is used to delimit the
input for one load module from the input
for another load module.

```
r-----------T--------------------------------------
|Operation  |Operand
|-----------+--------------------------------------
|NAME       |member-name[(R)]
L-----------L--------------------------------------
```

The NAME statement is placed after the
last object module or linkage editor con-
trol statement used as input to a load
module. Any modules or control statements
following a NAME statement are part of the
next load module being
constructed. A NAME statement can be
placed only in the primary input: any NAME
statements in the secondary input are
ignored.

All of the resulting load modules from a
batched linkage editor execution are placed
in the library (PDS) specified in the
SYSLMOD DD statement. The member name for
each of the resulting load modules is
specified as "member name" in the NAME

statement. For example, if the primary input for one of the load modules is followed by a NAME statement containing the member name XALPHA, and the SYSLMOD DD statement for the linkage editor step specifies the PDS MYLIB, the resulting load module is assigned the member name XALPHA and is placed in the PDS MYLIB. The SYSLMOD DD statement should not contain a member name. However, if the SYSLMOD statement contains a member name, that member name must be identical to the member name specified in the first NAME statement appearing in the primary input.

The NAME statement can be used to specify that a load module currently residing in a PDS is to be replaced by the load module constructed from the input immediately preceding the NAME statement. Replacement is specified by coding (R) following the member name in the NAME statement.

When several load modules are created in a single step (multiple link editing), the options specified in the EXEC statement for that step apply to each load module created in that step.

Example: An object module resides on a sequential data set PROGX. A load module is to be constructed from this module, using the FORTRAN library and a private library MYLIB to resolve external references within the module. Another object module resides on a sequential data set PROGY, and a load module is to be constructed from this object module using the same library to resolve external references. Both load modules are to be placed in the library PROGLIB. The first module is to be assigned the member name FUNTST; the second module is assigned the member name SUBTST.

The following text shows the job control statements and the position of INCLUDE, LIBRARY, and NAME linkage editor statements necessary to perform the job.

```
//JOB2 JOB 108,'J.JONES'
//STEP EXEC PGM=IEWL
//SYSLIB DD DSNAME=SYS1.FORTLIB,DISP=OLD
//SYSLMOD DD DSNAME=PROGLIB,DISP=OLD
        .
        .
        .
//DD1 DD DSNAME=PROGX,DISP=OLD
//DD2 DD DSNAME=PROGY,DISP=OLD
//ADDLIB DD DSNAME=MYLIB
//SYSLIN DD *
   INCLUDE DD1
   LIBRARY ADDLIB(X,Z)
   NAME FUNTST
   INCLUDE DD2
   LIBRARY ADDLIB(Y,Z)
   NAME SUBTST
/*
```

The JOB statement JOB2 defines the job, and the EXEC statement STEP instructs the operating system to execute the program IEWL. The DD statement SYSLIB tells the linkage editor that the FORTRAN library is the automatic call library. The SYSLMOD DD statement tells the linkage editor that both modules are written in the PDS PROGLIB.

The first INCLUDE statement and the DD statement DD1 tell the linkage editor that the first load module is to contain the object module that resides on the sequential data set PROGX. The first LIBRARY statement tells linkage editor that the references to X and Z in this module are to be resolved by the library MYLIB. The first NAME statement tells the linkage editor that the resulting module is assigned the member name FUNTST. The control statements are similar for the load module with the member name SUBTST.

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 publication.

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:

$$\left\{ \begin{matrix} PARM \\ PARM.procstep \end{matrix} \right\} = (\left[\begin{matrix} MAP \\ XREF \end{matrix} \right] [,LET] [,NCAL]$$

$$[,LIST])$$

Other options can also be specified for the linkage editor. For a detailed description of all linkgae editor options, see the Linkage Editor publication.

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 and/or cross-reference list are written in the data set specified by the SYSPRINT DD statement. If neither option is specified, no map or cross-reference listing is generated. Descriptions of the map and cross-reference listing are given in the section "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 either in the SYSLIB DD statement or 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 all linkage editor control statements are listed in card-image format in the data set specified by the SYSPRINT DD statement.

LOAD MODULE EXECUTION

The ddnames used in executing load modules must meet the format specified by IBM. When the system is generated, device names are assigned by the operating system and the installation; the programmer chooses devices by specifying either the installation or operating system names.

Program Name

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
    yyy is a FORTRAN sequence number

Data Set Reference Number (xx): When the system is generated, the upper limit for data set reference numbers specified by the installation is 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 7. Note that 0 is not a valid data set reference number.

Table 7. Load Module ddnames

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

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	ddname

```
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 is unlabeled and has the appearance shown in
Figure 27.

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
FORTRAN sequence number of FT02F001. (For
a more detailed explanation of the DEFINE
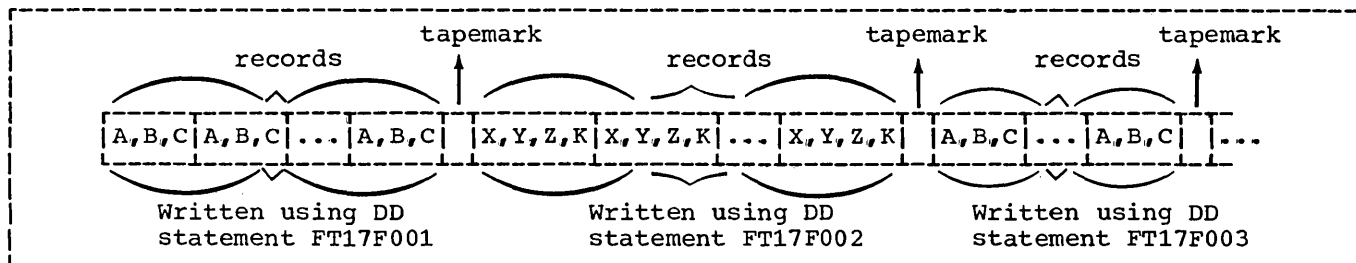FILE statement, refer to the FORTRAN IV
Language publication.)

42

Figure 27. Tape Output for Several Data Sets Using Same Data Set Reference Number

Retrieving Data Sets Written with Varying FORTRAN Sequence Numbers

To retrieve the data sets shown in Figure 27, 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."

$$LABEL=([\text{data-set-sequence-number}] \left\{ \begin{matrix} ,NL \\ ,\underline{SL} \end{matrix} \right\})$$

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.

If one of the data sets shown in Figure 27 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 27 and then read the second data set:

```
//FT17F001 DD UNIT=TAPE,LABEL=(,NL)
//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=*.FT17F002       X
//              DISP=OLD,LABEL=(2,NL)      X
//              DSNAME=*.FT17F002,         X
//              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 27 are cataloged for the purpose of later reading them, and 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
//              VOLUME=REF=*.FT17F001,     X
//              UNIT=TAPE,DISP=(,CATLG)
//FT17F003 DD DSNAME=N3,LABEL=(3,NL),     X
//              VOLUME=REF=*.FT17F002,     X
//              UNIT=TAPE,DISP=(,CATLG)
```

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=(1,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 27 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 is 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 27 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.

```
      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........
      .
      .
      .
```

Concatenation: The data sets shown in Figure 27 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=(1,N)
//        DD DSNAME=N2,DISP=OLD,LABEL=(2,NL)
//        DD DSNAME=N3,DISP=OLD,LABEL=(3,NL)
```

Note: Concatenation of data sets defined by direct-access statements is not allowed.

ERR=Parameter

The ERR= parameter 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.

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 effect of a BACKSPACE statement depends upon the record format and the type of control used to read or write the record (FORMAT control or no FORMAT control). For specific information concerning BACKSPACE, see "Backspace Operations" in the section "Creating Data Sets."

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

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. 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. (The publication IBM System/360 Operating System, System Generation, explains the method of assigning the data set reference number.)

If this data set is not defined and an error condition is encountered during the execution of the job step, the job step is terminated and a condition code of 16 is issued.

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 3). 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 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."

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 discusses data set utility programs.) No consideration is given to optimizing I/O operations; this information is given in the section "Programming Considerations."

To create data sets, the DSNAME, UNIT, VOLUME, SPACE, LABEL, DISP, SYSOUT, and DCB parameters are of special significance (see Figure 28). 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 29.

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.

If the user chooses to exercise this option, caution must be taken in specifying the parameters in the DD statement (Figure 28). With FORTRAN defined direct-access data sets, the DUMMY parameter may not be used because of a conflict in specifications. The remaining parameters of the DD statement must conform to the specifications in the DEFINE FILE statement. The DEN and TRTCH subparameters of the DCB parameter apply only to data sets residing on magnetic tape volumes; consequently, their use with other FORTRAN defined direct-access data sets may also produce a conflict.

The following statements illustrate the possible conflicts that may arise between the DEFINE FILE and DD statements.

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

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

The SPACE parameter must be included for all direct-access data sets, but it must also conform to the DEFINE FILE statement; the record length in both statements must be the same. In the DCB parameter, the subparameter DEN applies only to data sets residing on magnetic tape volumes. If the DUMMY parameter is specified in a DD statement for a direct-access data set, the conflict arises because the disposition of a direct-access data set is always checked and a dummy data set has no disposition.

Note: The name field of the DD statement must contain FTxxF001; where xx is the data set reference number specified in the DEFINE FILE statement.

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 data set cannot be read as input to a FORTRAN object program or created as output from a FORTRAN object program even though the member name has been specified in the DSNAME parameter of a 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.

$$\left\{\begin{array}{l} \text{DSNAME=}\left\{\begin{array}{l} \text{dsname} \\ \text{dsname(element)} \\ \text{\&name} \\ \text{\&name(element)} \end{array}\right\} \\ \text{DUMMY} \\ \text{DDNAME=ddname} \end{array}\right\}$$

UNIT=(name[.{n|P{1}]){2}

VOLUME=([PRIVATE][,RETAIN][,volume-sequence-number][,volume-count] [,SER=(volume-serial-number[,volume-serial-number]...){3} | ,REF={dsname / *.ddname / *.stepname.ddname / *.stepname.procstep.ddname}]){4}

SPACE=({TRK / CYL / average-record-length}, (primary-quantity[,secondary-quantity][,directory-quantity])[,RLSE][{MXIG / ,ALX / CONTIG}]{5}[,ROUND]{6})({7})

LABEL=([data-set-sequence-number]{,NL / ,SL}[,EXPDT=yyddd / ,RETPD=xxxx]){8}

<{ SYSOUT=A / SYSOUT=B / SYSOUT=(X[,program-name][,form-no.]) / DISP=({NEW / OLD / MOD / SHR}{,DELETE / ,KEEP / ,PASS / ,CATLG / ,UNCATLG}{9}){7} }>

DCB=([dsname / *.ddname / *.stepname.ddname / *.stepname.procstep.ddname][,DEN={0 / 1 / 2}][,TRTCH={C / E / T / ET}][,BUFNO={1 / 2}]{10}[,RECFM={{F|U}[A][,BLKSIZE=xxxx] / V[A],LRECL=xxxx.BLKSIZE=xxxx / {F|C{B[A],LRECL=xxxx,BLKSIZE=xxxx} / ,BLKSIZE=xxxx{12}]){11}

[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 keyboard 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 postional 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.

Figure 28. DD Parameters for Creating Data Sets

Sample Coding Form

1-10	11-20	21-30	31-40	41-50	51-60	61-70	71-80
1234567890	1234567890	1234567890	1234567890	1234567890	1234567890	1234567890	1234567890

```
       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 29. Examples of DD Statements

DDNAME=ddname
   indicates a dummy data set that will assume the characteristics specified in a following DD statement of "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 programmer specifies the name and number of input/output devices 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.

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]...)       │
    │         ⎧ dsname                     ⎫ │  )
    │ ,REF=   ⎨ *.ddname                   ⎬ │
    │         ⎪ *.step.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.

48

RETAIN
  indicates that this volume is to
  remain mounted after the job step is
  completed. (Unless RETAIN is speci-
  fied, the volume is dismounted after
  its last use in the job step.)
  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 dismounted 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 one-to-four 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 charac-
  ters, 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 cata-
  log, 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 8
  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, immediately
behind the data set referred to by this
subparameter. When this subparameter is
used, the UNIT parameter must be omitted.

Table 8. 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 job step "stepname"
REF=*.stepname.procstep.ddname	a data set indicated by DD statement "ddname" in the procedure step "procstep" invoked in the job step "stepname"

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

SPECIFYING SPACE ON DIRECT-ACCESS VOLUMES

$$SPACE=(\begin{cases} TRK \\ CYL \\ average\text{-}record\text{-}length \end{cases}$$

, (primary-quantity

[,secondary-quantity]

[,directory-quantity])

$$[,RLSE]\begin{bmatrix} ,MXIG \\ ,ALX \\ ,CONTIG \end{bmatrix}[,ROUND])$$

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.

$$\begin{cases} TRK \\ CYL \\ average\text{-}record\text{-}length \end{cases}$$
  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
  $\leq 65,535$).

(primary-quantity[,secondary-quantity]
[,directory-quantity])
specifies the amount of space
allocated for the data set. The
"primary quantity" indicates the num-
ber 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 allo-
cated space is exhausted. (Note: The
maximum number of times secondary
allocation will be made is 15.)

The "directory quantity" is used only
when writing a PDS, and it specifies
the number of 256-byte blocks to re-
serve for the directory of the PDS.

For example, by specifying:

SPACE=(120,(400,100))

space is reserved for 400 records, the
average record length is 120 charac-
ters. Each time space is exhausted,
space for 100 additional records is
allocated.

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. In addition, space is
reserved for five 256-byte blocks in
the directory of a PDS.

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

```
┌MXIG  ┐
│ALX   │
└CONTIG┘
```
specify the format of the space allo-
cated 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 non-contiguous 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 non-contiguous areas are allocat-
ed.

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.
$$LABEL=([data-set-sequence-number] \begin{Bmatrix} ,NL \\ ,\underline{SL} \end{Bmatrix}$$
$$\begin{bmatrix} ,EXPDT=yyddd \\ ,RETPD=xxxx \end{bmatrix})$$

data-set-sequence-number
is a four-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 iden-
tified by data set sequence number 3.)
If the data set sequence number is not
specified, the operating system
assumes 1.

$$\begin{Bmatrix} NL \\ \underline{SL} \end{Bmatrix}$$
specifies whether a data set is
labeled or unlabeled. SL indicates
standard labels. NL indicates no
labels (applicable only to data sets
residing on a tape volume).

```
[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.

```
         ┌dsname                        ┐
DCB=(    │*.ddname                      │
         │*.stepname.ddname             │
         └*.stepname.procstep.ddname    ┘

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

┌              ⎧ {F|U}[A][,BLKSIZE=xxxx]              ⎫ ⎤
│ ,RECFM=      ⎨ V[A],LRECL=xxxx,BLKSIZE=xxxx         ⎬ │ )
│              ⎩ {F|V}B[A],LRECL=xxxx,BLKSIZE=xxxx    ⎭ │
│                          [,BUFNO={1|2}]               │
└                                                       ┘
```

DISPOSITION OF A DATA SET

The disposition of a data set is speci-
fied 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

A printed output data set may be written
on an intermediate device and subsequently
written on the printer (ultimate device).

SYSOUT=A
    indicates that the ultimate destina-
    tion for printed output data sets is
    the printer.

Note: If the DEN subparameter is explicit-
ly 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 con-
tains 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.

REFERRING TO PREVIOUSLY SPECIFIED DCB INFORMATION

The first subparameter

```
┌dsname                         ┐
│*.ddname                       │
│*.stepname.ddname              │
└*.stepname.procstep.ddname     ┘
```

is used to retrieve DCB parameter informa-
tion from previously created data sets.
The control program copies the DCB informa-
tion specified for the data set referred to
by this subparameter. The copied informa-
tion 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 proce-
    dure was invoked by the EXEC statement
    "stepname" in the current job.

DENSITY AND CONVERSION

The second subparameter indicates the density and conversion for tape volumes.

DENSITY: Density is only specified for data sets residing on magnetic tape volumes.

DEN={0|1|2}
    indicates the density used to write the data set (refer to Table 9).

•Table 9. DEN Subparameter Values

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

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 either BCD to EBCDIC or EBCDIC to BCD is required

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

RECORD FORMAT

```
┌                    ┐
│RECFM=U[A]          │
│RECFM=V[B][A]       │
│RECFM=F[B][A][M]    │
└                    ┘
```

The characters U, V, F, and B 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

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

RECORD LENGTH, BUFFER LENGTH, BLOCK LENGTH, AND NUMBER OF BUFFERS FOR SEQUENTIAL DATA SETS

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 compiler has a record length of 120 bytes (including the carriage control byte). 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.

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

52

• Table 10. 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	
		Undefined	RECFM=U	BLKSIZE=xxxx	
	Blocked	Fixed-Length	RECFM=FB	LRECL=xxxx	
		Variable-Length	RECFM=VB		
		Undefined	Blocked undefined records are not permitted		

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

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

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

BUFNO=x

where:
    x=1 or x=2

FORTRAN Records and Logical Records for Sequential Data Sets

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 section "Input/Output Statements" in the FORTRAN IV Language publication). A record defined by a READ/WRITE list is a logical record. Within each category,

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

UNBLOCKED RECORDS, FORMAT CONTROL: 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 30).

Example: Assume BLKSIZE=44

10   FORMAT(F10.5,I6,2F12.5,'SUMS')
     WRITE(20,10)AB,NA,AC,AD
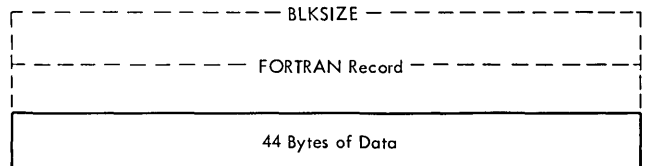
Figure 30. FORTRAN Record (FORMAT Control) Fixed-Length Specification

If the FORTRAN record length is less than BLKSIZE, the record is padded with blanks to fill the remainder of the buffer (see Figure 31). The entire buffer is written.

Example: Assume BLKSIZE=56

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

```
┌── ── ── ── ── ── ──BLKSIZE ── ── ── ── ── ── ──┐
│                                                │
├── ── ── ── ── ── — Written Record ── ── ── ── ──┤
│                                                │
├── ── FORTRAN Record ── ── ──┐                  │
│                             │                  │
│                             │                  │
│                             │                  │
│   33 Bytes of Data          │  23 Bytes of Blanks │
└─────────────────────────────┴──────────────────┘
```
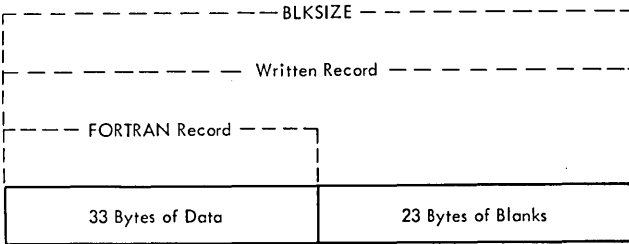
Figure 31. FORTRAN Record (FORMAT Control) With Fixed-Length Specification and FORTRAN Record Length Less Than BLKSIZE

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 control word (BCW) and the 4-byte segment control word (SCW), as shown in Figure 32. The BCW (see Figure 37) contains the length of the block; the SCW (see Figure 38) contains the length of the record segment, i.e., the data length plus four bytes for the SCW.

If the FORTRAN record length is less than (LRECL-4), the unused portion of the buffer is not written (see Figure 33).

```
┌── ── ── ── ── ── ── ──BLKSIZE ── ── ── ── ── ── ──┐
│                                                   │
│  ┌── ── ── ── ── ── ──LRECL ── ── ── ── ── ── ──┤
│  │                                              │
│  │  ┌── ── ── ── ──FORTRAN Record ── ── ── ── ──┤
│  │  │                                           │
│  │  │                                           │
├──┬──┬───────────────────────────────────────────┤
│BCW│SCW│            Data                          │
└──┴──┴───────────────────────────────────────────┘
```
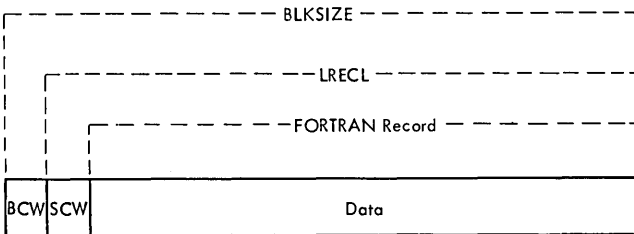
Figure 32. FORTRAN Record (FORMAT Control) Variable-Length Specification

```
┌── ── ── ── ── ──BLKSIZE ── ── ── ── ── ── ── ──┐
│                                                │
│  ┌── ── ── Written Record ── ── ── ──┐         │
│  │                                   │         │
│  │  ┌── ── ── ── ── ──LRECL ── ── ──┤── ── ── ──┤
│  │  │                               │          │
│  │  │  ┌── ── ── FORTRAN Record ── ──┤          │
│  │  │  │                            │          │
├──┬──┬──┴────────────────────────────┤──────────┐
│BCW│SCW│         Data                 │ Not Written│
└──┴──┴─────────────────────────────────┴──────────┘
```
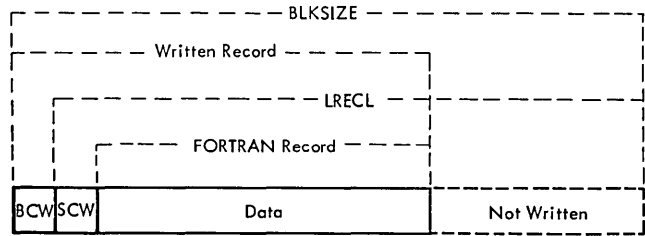
Figure 33. FORTRAN Record (FORMAT Control) With Variable-Length Specification and the FORTRAN Record Length Less Than (LRECL-4)

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 34).

```
┌── ── ── ── ── ── ──BLKSIZE ── ── ── ── ── ──┐
│                                             │
├── ── ── FORTRAN Record ── ── ──┐            │
│                                │            │
│                                │            │
├────────────────────────────────┤────────────┐
│         Data                    │ Not Written │
└─────────────────────────────────┴────────────┘
```
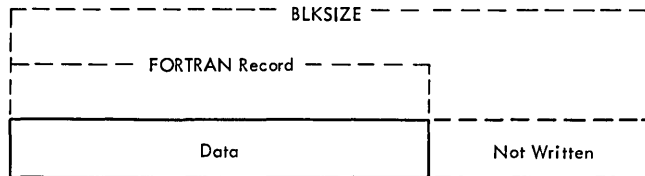
Figure 34. FORTRAN Record (FORMAT Control) With Undefined Specification and the FORTRAN Record Length Less Than BLKSIZE

BLOCKED RECORDS, FORMAT CONTROL: 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 35).

Example: Assume BLKSIZE=48 and LRECL=24

```
10   FORMAT(I8,F16.4)
20   FORMAT(I12)
     .
     .
     .
     WRITE(13,10)N,B
     .
     .
     .
     WRITE(13,20)K
```
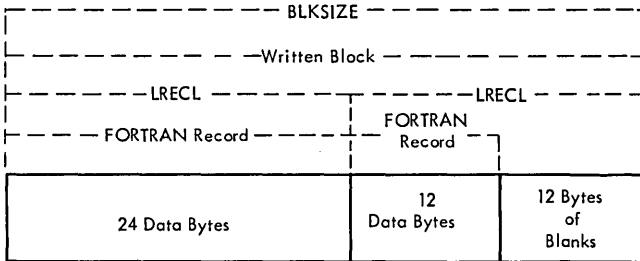

Figure 35.   Fixed-Length   Blocked   Records
             Written Under FORMAT Control

Variable-Length   Records: For   blocked
variable-length records written under FOR-
MAT 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 addi-
tional bytes allocated with BLKSIZE are
required for the block control word (BCW)
that contains the block length.   The four
additional bytes allocated with LRECL are
used for the segment control word (SCW)
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 36); the
new data goes into the next buffer. Howev-
er, if the space remaining in a buffer is
greater than LRECL, the buffer is not
written, but held for the next WRITE (see
Figure 36). If another WRITE is not exe-
cuted before the job step is terminated,
then the filled portion of the buffer is
written.

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
```
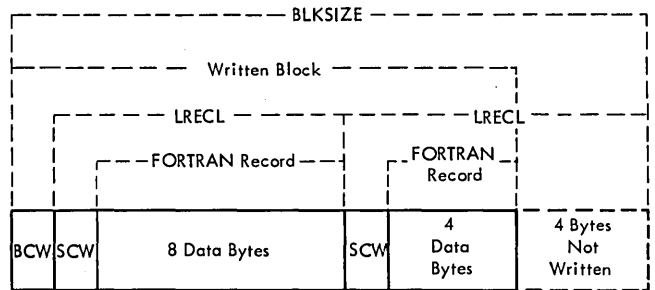

Figure 36.   Variable-Length Blocked Records
             Written Under FORMAT Control

NO FORMAT CONTROL: Only variable-length
records can be written with no FORMAT
control; i.e., the RECFM subparameter must
be V.   (If nothing is specified, V is
assumed.)

Records written without FORMAT control have
the following properties:

* The length of the logical record is
  controlled by the type and number of
  variables in the I/O list of its asso-
  ciated READ or WRITE statement.

* A logical record can be physically
  recorded on an external medium as one
  or more record segments. Not all seg-
  ments of a logical record must fit into
  the same physical record (block).

* Three quantities control the manner in
  which records are placed on an external
  medium: the block size (as specified by
  the BLKSIZE parameter), the segment
  length (as specified by the LRECL
  parameter), and the logical record (as

defined by the length of the I/O list).
BLKSIZE and LRECL are specified as part
of the DCB parameter of the data defi-
nition (DD) statement. If not speci-
fied, FORTRAN provides default values.

Each block begins with a 4-byte block
control word (BCW); each segment begins
with a 4-byte segment control word (SCW).
The SCWs and BCWs are provided by the
system.

The format of a BCW is given in Figure
37.

```
r------------------------T------------------------n
|    block-length        |      reserved          |
L_____i_____J
       2 bytes                  2 bytes
```
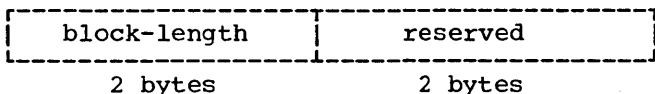
Figure 37. Format of a Block Control Word

where:

block-length
    is a binary count of the total number
    of bytes of information in the block.
    This includes four bytes for the BCW
    plus the sum of the segment lengths
    specified in each SCW in the block.
    (The permissible range is from 8 to
    32,767 bytes.)

reserved
    is two bytes of zeros reserved for
    system use.

The format of an SCW is given in Figure
38.

```
r------------------------T-----------T------------n
|    segment-length      |   code    |  reserved  |
L_____i_____i_____J
       2 bytes              1 byte       1 byte
```
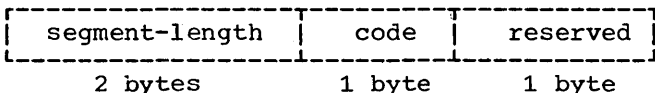
Figure 38. Format of Segment Control Word

where:

segment-length
    is a binary count of the number of
    bytes in the SCW (4 bytes) plus the
    number of bytes in the data portion of
    the segment following the SCW. (The
    permissible range is from 4 to 32,763
    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 multi-segment record.
10	This segment is the last of a multi-segment 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
written without FORMAT control the value of
BLKSIZE is equal to LRECL + 4. (The four
additional bytes are for the BCW.)

If the logical record length is less
than or equal to LRECL-4, the logical
record comprises one record segment.
Hence, for the associated READ or WRITE
statement, one record segment, i.e., one
block, is transmitted (see Figure 39).
Note that the unused portion of the block
is not transmitted.

If the logical record length is greater
than LRECL-4, the logical record comprises
N record segments, where: N=logical record
length/LRECL-4. Hence, for the associated
READ or WRITE statement, N record segments,
i.e., N blocks, are transmitted (see Figure
40).

Example 1: Assume BLKSIZE=28 and LRECL=24

WRITE (18) Q,R

where:
    Q and R are real *8 variables.

```
r----------------- BLKSIZE ----------------¬
|                                           |
|    r------------- LRECL -------------¬    |
|    |                                 |    |
|    |------Logical Record------¬      |    |
|    |                          |           |
+-------+--------+----------------------+-------------+
| BCW   | SCW    |    Data Segment      | Not Written |
+-------+--------+----------------------+-------------+
4 bytes  4 bytes         16 bytes          4 bytes
```

Figure 39.   Variable-length Unblocked Re-
              cords, No FORMAT Control, One
              Record Segment

Example 2: Assume BLKSIZE=28 and LRECL=24

WRITE (18) Q,R,S,V,X

where:
        Q, R, and V are real *8 variables.

        S and X are real *4 variables

```
r------------------ BLKSIZE ----------------¬
|                                           |
|   r-------------- LRECL --------------¬   |
|   |                                   |   |
|   |---------Beginning of Logical Record---------|
+-------+--------+--------------------------------+
| BCW   | SCW    |        Data Segment 1           |
+-------+--------+--------------------------------+
4 bytes  4 bytes             20 bytes
```

```
            r------End of Logical Record------¬
            |                                 |
+-------+--------+----------------------+-------------+
| BCW   | SCW    |     Data Segment 2   | Not Written |
+-------+--------+----------------------+-------------+
4 bytes  4 bytes        12 bytes           8 bytes
```
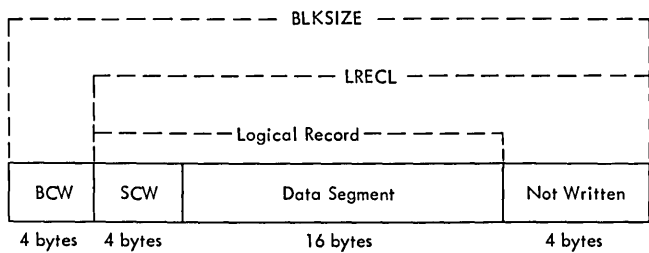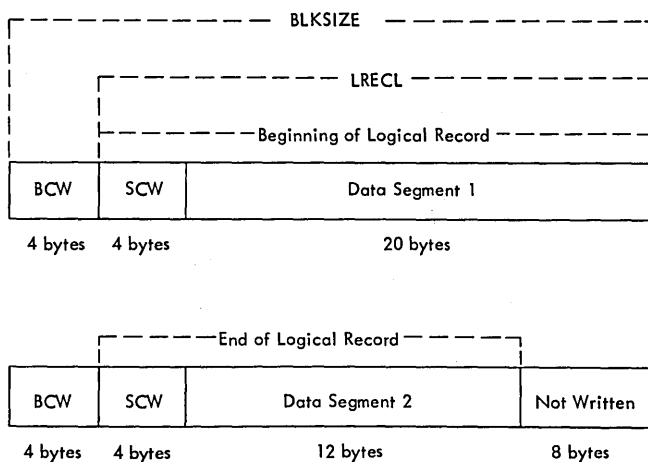
Figure 40.   Variable-length Unblocked Re-
             cords, No FORMAT Control, Two
             Record Segments

Blocked   Records: For   blocked   records
written without FORMAT control, each block,
except the last, is composed of at least M
record   segments,   where: M   =   BLK-
SIZE-4/LRECL.

    If the logical record length is less
than or equal to LRECL-4, the logical
record comprises one   record   segment.
Hence, for the associated M READ or WRITE
statement, one block, i.e., M record seg-
ments, is transmitted.

    If the logical record length is greater
than LRECL-4, the logical record comprises

N record segments, where: N = logical
record   length/LRECL-4.   Hence, for the
associated   READ   or WRITE statement, N
record segments (i.e., as many blocks of M
segments   each as are needed to makeup N
segments) are transmitted. The unused por-
tion of the last block is held for the next
READ or WRITE statement (see Figure 41).

Example: Assume BLKSIZE=28 and LRECL=12

WRITE (18) A
                    .
                    .
                    .
WRITE (18) B
                    .
                    .
                    .
WRITE (18) E

where:
        A is a real *8 variable.

        B and E are real *4 variables.

```
r------------------- BLKSIZE ----------------------¬
|                                                  |
|   r--- LRECL ---¬----- LRECL -----¬              |
|   |             |                 |              |
|   |-- Logical Record --|-- Logical Record--¬     |
+-------+-------+----------+-------+----------+-------------+
| BCW   | SCW   | Record 1 | SCW   | Record 2 | Not Written |
+-------+-------+----------+-------+----------+-------------+
4 bytes  4 bytes   8 bytes  4 bytes   4 bytes     4 bytes
```

```
            r-- Logical Record --¬
            |                     |
+-------+-------+----------+---------------------------+
| BCW   | SCW   | Record 3 |   Space Ready for Next Write |
+-------+-------+----------+---------------------------+
4 bytes  4 bytes  4 bytes             16 bytes
```
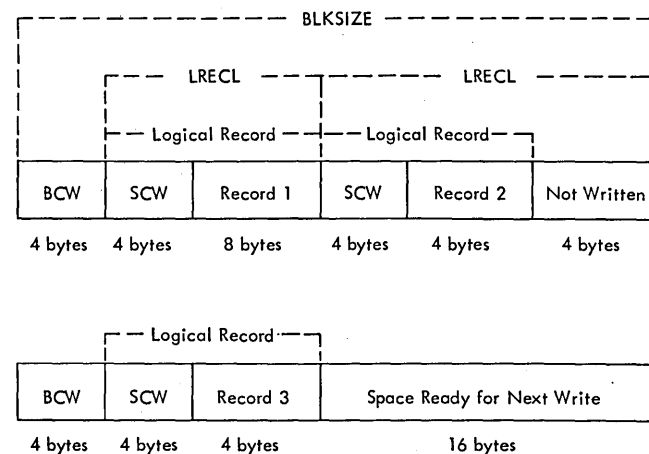
Figure 41.   Variable-length, Blocked Re-
             cords, No FORMAT Control

BACKSPACE Operations

Unblocked Records, FORMAT Control: For all
unblocked records written under FORMAT con-
trol, the volume is positioned so that the
last. record read or written is transmitted
next.

Unblocked Records, No FORMAT Control: For
all unblocked records written without FOR-
MAT control, the volume is positioned so
that the last logical record read or writ-
ten is transmitted next.

Blocked Records: The programmer is warned against backspacing blocked records; the results obtained are unpredictable.
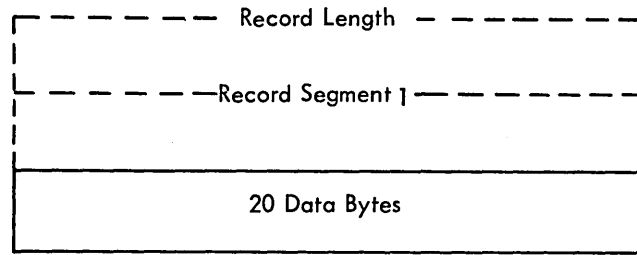
RECORD LENGTH, BUFFER LENGTH, AND NUMBER OF BUFFERS FOR DIRECT ACCESS DATA SETS

A direct access data set can contain only fixed-length, unblocked records. Any attempts to read or write any other record format by specification in the DCB parameter are 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 only DCB parameter that can be supplied for direct access data sets is the number of buffers:

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. For records written with no FORMAT control, the logical record can exceed the record length specified in the DEFINE FILE statement. If it is shorter than the record length, the remaining portion of the record is padded with zeros (see Figure 42).

Record Segment $_1$ + Record Segment $_2$ = 1 Logical Record
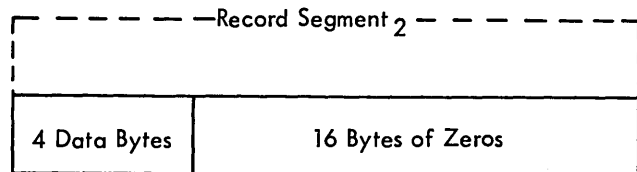
Figure 42. Logical Record (No FORMAT Control) for Direct Access

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 double precision variables
R1 and R2	are real variables
IX	is an integer variable that contains the record position

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

DCB RANGES AND ASSUMPTIONS

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
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 11), 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 (see Table 10.1). Load module DCB parameter default values are shown in Table 12.

Table 11. BLKSIZE Ranges: Device Considerations

| Device Type | BLKSIZE Ranges | |
	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
  132 Spaces
  144 Spaces
	
	1≤x≤121
	1≤x≤133
	1≤x≤145
		
		9≤x≤129
		9≤x≤141
		9≤x≤153

Magnetic Tape	18≤x≤32,000	
Direct Access:	Without Track Overflow[1]	With Track Overflow[1]
2301 2302 2303 2311 2314	1≤x≤20,483 1≤x≤4984 1≤x≤4892 1≤x≤3625 1≤x≤7294	1≤x≤32,763 1≤x≤32,763 1≤x≤32,763 1≤x≤32,763 1≤x≤32,763
[1]If RECFM=V, the minimum block size is 9.		

• Table 12. Load Module DCB Parameter Default Values

Data Set Reference Number	ddname	Sequential Data Sets		Direct Access Data Sets	
		Default BLKSIZE[1]	Default RECFM[2]	Default RECFM	Default LRECL or BLKSIZE
1	FT01Fyyy	800	U	F	
2	FT02Fyyy	800	U	F	The value spec-
3	FT03Fyyy	800	U	FA[3]	ified as the
4	FT04Fyyy	800	U	F	maximum size of
5	FT05Fyyy	80	F	F	a record in the
6	FT06Fyyy	133	UA[3]	F	DEFINE FILE
7	FT07Fyyy	80	F	F	statement.
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.
[2]If the records have no FORMAT control, the default RECFM is V (F if it is direct access).
[3]The first character in the record is for carriage control.

60

This section contains figures illustrating the job control statements used in the FORTRAN IV 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 three cataloged procedures that include the compile step (Figures 44, 45, and 46), the EXEC statement named FORT designates that the operating system is to execute the program IEYFORT FORT (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.

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 23) 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 23.

The control statements contained in the procedure (shown in Figure 43) 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 Link Edit

The cataloged procedure to compile the source module and link edit the resulting FORTRAN object module (FORTGCL) is shown in Figure 44. The control statements for compilation are the same as described above. However, output of the object module is defined by the SYSLIN DD statement.

In each of the cataloged procedures that include a link edit step (Figures 44, 45, and 46), 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.

IBM   FORTRAN Coding Form   X28-7327-5

| PROGRAM | | | | | PUNCHING INSTRUCTIONS | GRAPHIC | | | | | | | | PAGE | OF |
| PROGRAMMER | | | DATE | | | PUNCH | | | | | | | | CARD ELECTRO NUMBER* | |

```
//FORT        EXEC  PGM=IEYFORT,REGION=100K
//SYSPRINT DD  SYSOUT=A
//SYSPUNCH DD  SYSOUT=B
//SYSLIN   DD  DSNAME=&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,       X
//            SPACE=(80,(200,100),RLSE),DCB=BLKSIZE=80
```

● Figure 43. Compile Cataloged Procedure (FORTGC)

Execution of the link edit step produces a list of the linkage editor control statements (in card image format), a map and 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 twenty records and secondary allocation of ten 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.

Link Edit and Execute

This cataloged procedure, FORTGLG, first link edits the FORTRAN object module and then executes the resulting load module. (Procedure is shown in Figure 45.) Since the link 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 45 and 46). 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 link 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 link 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 a 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.

●Figure 44. Compile and Link Edit Cataloged Procedure (FORTGCL)

```
//FORT     EXEC PGM=IEYFORT,REGION=100K
//SYSPRINT DD   SYSOUT=A
//SYSPUNCH DD   SYSOUT=B
//SYSLIN   DD   DSNAME=&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,          X
//              SPACE=(80,(200,100),RLSE),DCB=BLKSIZE=80
//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=(1024,(20,10,1),RLSE),DCB=BLKSIZE=1024
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=SYSDA,SPACE=(1024,(20,10),RLSE),DCB=BLKSIZE=1024
//SYSLIN   DD   DSNAME=&LOADSET,DISP=(OLD,DELETE)
//         DD   DDNAME=SYSIN
```


●Figure 45. Link Edit and Execute Cataloged Procedure (FORTGLG)

```
//LKED EXEC PGM=IEWL,REGION=96K,PARM=(XREF,LET,LIST)
//SYSLIB   DD   DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSLMOD  DD   DSNAME=&GOSET(MAIN),DISP=(NEW,PASS),UNIT=SYSDA,      X
//              SPACE=(1024,(20,10,1),RLSE),DCB=BLKSIZE=1024
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   UNIT=SYSDA,SPACE=(1024,(20,10),RLSE),DCB=BLKSIZE=1024
//SYSLIN   DD   DDNAME=SYSIN
//GO   EXEC PGM=*.LKED.SYSLMOD,COND=(4,LT,LKED)
//FT05F001 DD   DDNAME=SYSIN
//FT06F001 DD   SYSOUT=A
//FT07001  DD   SYSOUT=B
```

A listing of the execution time routines required for various input/output, interruption, and error procedures is contained in IBM System/360 FORTRAN IV Library Subprograms. 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, Link Edit, and Execute

The cataloged procedure (FORTGCLG) to compile, link edit, and execute FORTRAN source modules is shown in Figure 46. 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 link 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.

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 45 and 46). 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 permanently.

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 10). Therefore, if he desires a BLKSIZE of 80 and a REFM 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

The publication IBM System/360 Operating System, System Generation describes the FORTLIB macro instruction.

In addition, the DD card for FT05001 must be deleted permanently from the cataloged procedure.

```
//FORT      EXEC PGM=IEYFORT,REGION=100K
//SYSPRINT  DD SYSOUT=A
//SYSPUNCH  DD SYSOUT=B
//SYSLIN    DD DSNAME=&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,           X
//             SPACE=(80,(200,100),RLSE),DCB=BLKSIZE=80
//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=(1024,(20,10,1),RLSE),DCB=BLKSIZE=1024
//SYSPRINT  DD SYSOUT=A
//SYSUT1    DD UNIT=SYSDA,SPACE=(1024,(20,10),RLSE),DCB=BLKSIZE=1024
//SYSLIN    DD DSNAME=&LOADSET,DISP=(OLD,DELETE)
//          DD DDNAME=SYSIN
//GO EXEC PGM=*.LKED.SYSLMOD,COND=((4,LT,FORT),(4,LT,LKED))
//FT05F001  DD DDNAME=SYSIN
//FT06F001  DD SYSOUT=A
//FT07F001  DD SYSOUT=B
```

•Figure 46. Compile, Link Edit, and Execute Cataloged Procedure (FORTGCLG)

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.

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 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 link 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, link edited, and executed using the cataloged procedure FORTGCLG. Furthermore, the linkage editor option MAP is 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 FORTGCLG,PARM.LKED=MAP,    X
//          ACCT.GO=506
```

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:

1. Present in the step, the parameters in the new DD statement override parameters in the DD statement in the procedure step.

2. 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 appear in the cataloged procedure. Any DD statements that are added to the procedure must follow overriding DD statements.

When the procedures FORTGC, FORTGCL, and FORTGCLG are used, a DD statement must be added to define the SYSIN data set to the compile step in the procedures (see Figures 15 and 21). When the procedure FORTGLG is used, a DD statement must be added to define the SYSLIN data set (see Figure 18).

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.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.FT08F001 DD DCB=(RECFM=F,BLKSIZE=200)
//GO.SYSIN DD *
┌─────────────────────────────────────────────┐
│              Input to Load Module             │
└─────────────────────────────────────────────┘
/*
```

Example 3: Assume the link 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 *
┌───────────────────────────────────────────────┐
│              FORTRAN Object Module              │
└───────────────────────────────────────────────┘
/*
//GO.SYSIN DD *
┌───────────────────────────────────────────────┐
│              Input to Load Module               │
└───────────────────────────────────────────────┘
/*
```

Example 4: Assume the compile, link 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 *
┌───────────────────────────────────────────────┐
│         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.

PROGRAMMING CONSIDERATIONS

This section discusses minimum system requirements for the compiler, program optimization, updating the FORTRAN library, creation of the programmer's private library, and limitations of the compiler.

STORAGE LOCATIONS AND BYTES

Storage locations in System/360 are called bytes, words, and double-words. One word is four bytes long; a double-word is eight bytes long. When data is read into main storage, it is translated into internal format. See Table 13 for storage allocation according to the type and length of the constant or variable.

Table 13. 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 COMPILER

The FORTRAN compiler requires at least a System/360 Model 40 computer with a minimum storage capacity of 128K bytes and a standard instruction set with the floating-point option.

All programs require a device such as an IBM 1052 Keyboard Printer for direct operator communication. Also, at least one direct-access device must be provided for system residence.

The printer must have at least a 120-character print line.

SOURCE PROGRAM CONSIDERATIONS

The FORTRAN compiler automatically provides optimization of certain areas of the source module. Other areas may be optimized by the programmer through his use of the FORTRAN language.

The following paragraphs describe the optimization facilities that are provided by the compiler and those defined by the programmer.

DO Loop Optimization

During the operation of the FORTRAN 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. Standard denotes the requirements of register assignment for the script expression, and non-standard denotes the opposite. 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.

Indicators and Sense Lights

At the start of program execution, the divide-check indicator, the overflow indicator, and the sense lights are <u>not</u> 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.

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 was specified in the FORTLIB macro instruction during system generation, since boundary-alignment violations will be corrected at exection time. (The FORTLIB macro instruction is described in the publication <u>IBM System/360 Operating System, System Generation</u>). If the BOUNDRY=NOALIGN option was specified and boundary violations are encountered during execution of the object program, the job will terminate.

If the BOUNDRY=ALIGN option of the FORTLIB macro instruction was 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 will require that (1) the specification exception resulting from this reference be processed, and (2) the boundary-alignment routine be invoked. Therefore, considerable programming efficiency will be gained if the programmer insures that all of the variables have proper boundary alignment. The publication <u>IBM System/360 FORTRAN IV Language</u> contains information on boundary alignment.

When boundary alignment is performed, program interrupt message IHC210I is issued. (This message is described 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.

<u>Note:</u> Even if BOUNDRY=ALIGN was specified, if a boundary error occurs on an EXECUTE, LOAD MULTIPLE, or STORE MULTIPLE assembler language statement, boundary adjustment

will not take place and the job will terminate. Therfore, if these instructions refer to improperly aligned data, they should not be used in assembler language subprograms.

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 the statement:

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 of SUBI within the source module is:

```
DIMENSION A(10)
        .
        .
        .
    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 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, and input/output device error terminates program execution.

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. Direct data sets should be pre-

formatted. If the NEW subparameter is specified in the DD statement for the data set, the FORTRAN 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, via the same unit number, the same direct data set within the same FORTRAN load module. However, sequential I/O statements may process 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 one that uses 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 distinguishes 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 that of 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 six-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
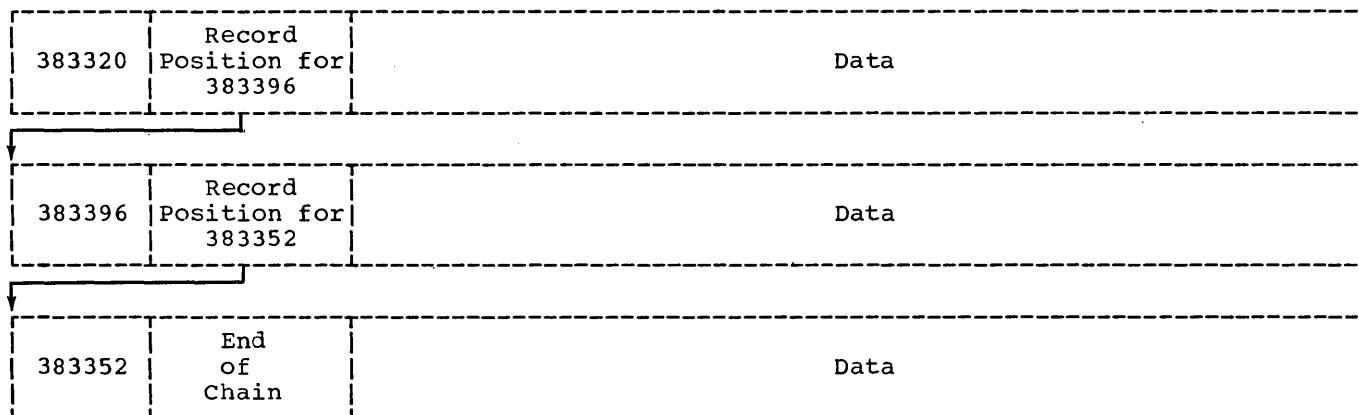
Identifier Chain

```
r----------T-----------T-----------------------------------------------------------------------1
|          |  Record   |                                                                        |
| 383320   |Position for|                          Data                                          |
|          |  383396   |                                                                        |
L_____i_____i_____J
      |  _____|
      | |
      v |
r----------T-----------T-----------------------------------------------------------------------1
|          |  Record   |                                                                        |
| 383396   |Position for|                          Data                                          |
|          |  383352   |                                                                        |
L_____i_____i_____J
      |  _____|
      | |
      v |
r----------T-----------T-----------------------------------------------------------------------1
|          |   End     |                                                                        |
| 383352   |   of      |                          Data                                          |
|          |  Chain    |                                                                        |
L_____i_____i_____J
```

Figure 47.   Record Chaining

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 either 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 47 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.

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. A FORTRAN load module writes skeleton records according to the format described in "WRITE -- Create a Direct Organization Data Set - Format F Records" in "Section 3, Basic Sequential Access Method (BSAM)" in the Control Program Services publication.

Figure 48 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, the DCB subparameter of the DD statement must specify DSORG = DA in order that the label that is created will indicate that this is a direct-access data set (see "Creating a Direct Data Set" in the publication IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646).

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, and 40 bytes for each record.

The DEFINE FILE statement for a data set must be in a source module in the root

segment (i.e., it cannot be overlaid), but does not have to be in the same source module in which I/O operations occur. 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 input/output operations, the subprogram cannot get to the updated value to make use of it in its operations unless the associated variable is in COMMON.

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
```

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.
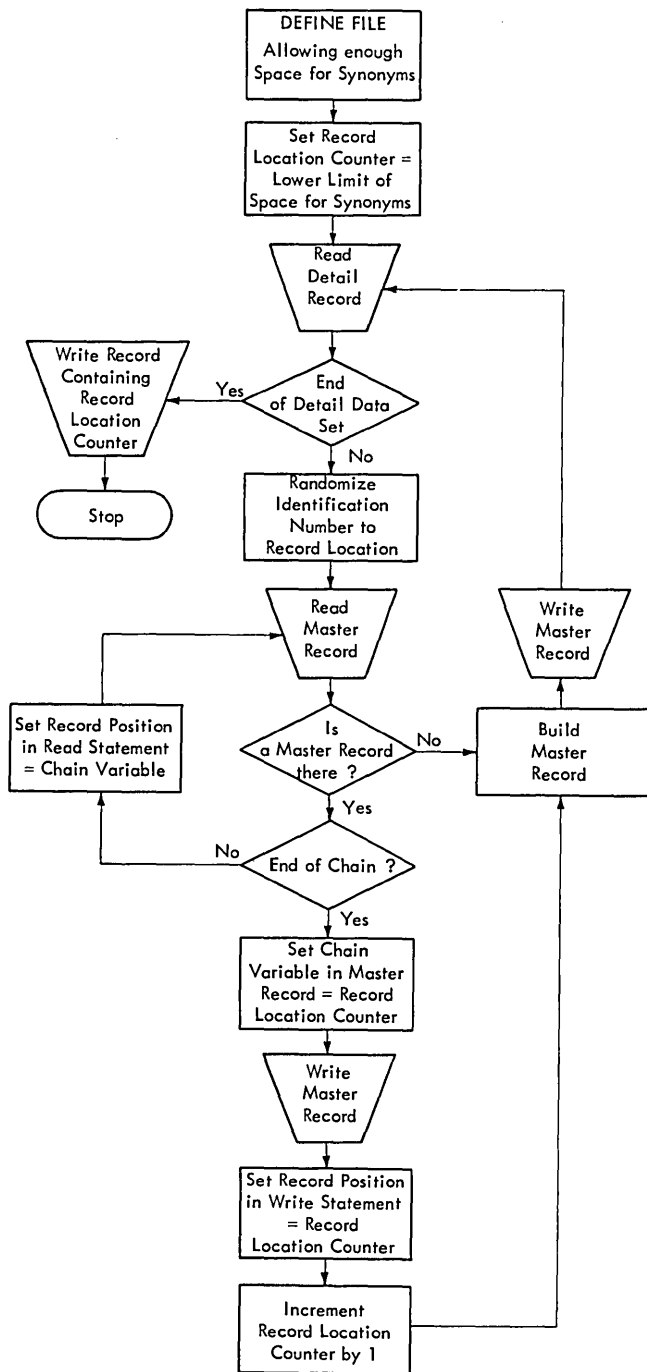
DEFINE FILE
Allowing enough
Space for Synonyms

Set Record
Location Counter =
Lower Limit of
Space for Synonyms

Read
Detail
Record

End
of Detail Data
Set

Yes → Write Record
Containing
Record
Location
Counter

Stop

No

Randomize
Identification
Number to
Record Location

Read
Master
Record

Write
Master
Record

Set Record Position
in Read Statement
= Chain Variable

Is
a Master Record
there ?   No → Build
Master
Record

Yes

No ← End of Chain ?

Yes

Set Chain
Variable in Master
Record = Record
Location Counter

Write
Master
Record

Set Record Position
in Write Statement
= Record
Location Counter

Increment
Record Location
Counter by 1

Figure 48.   Writing a Direct Access Data
Set for the First Time

- The maximum number of source cards for one compilation is dependent upon the amount of storage available to the compiler. A 400 statement program will

require approximately 90K bytes in PCP or MFT systems and 100K bytes in MVT systems.

- The maximum number of contiguous comment and/or continuation cards after the first statement is 49. There is no restriction on the number of comments at the beginning of the deck.

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. 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.

DD STATEMENT CONSIDERATIONS

Several DD statement parameters and subparameters are provided for I/O optimization (see Figure 49). 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.

```
┌────────────────────────────────────────────────────────────────────────────────┐
│SEP=(ddname[,ddname]...¹) ²                                                       │
│                                                                                  │
│       ⎰(name[, n|P ³][,DEFER][,SEP=(ddname[,ddname]...¹) ²]⁴ ⁵)⁶⎱               │
│UNIT=⎱AFF=ddname                                                   ⎰              │
│                                                                                  │
│⎛SPACE=(ABSTR,(quantity,beginning-address[,directory-quantity]) )          ⎞     │
│⎜                                                                          ⎟     │
│⎜          ⎡⎰,CYL                  ⎱                                     ⎤ ⎟     │
│⎜SPLIT=(n  ⎢⎱,average-record-length⎰,(primary-quantity[,secondary-quantity])⎥ )⎟ │
│⎜          ⎣                                                             ⎦ ⎟     │
│⎜          ⎰TRK                  ⎱                                          ⎟     │
│⎜SUBALLOC=(⎱CYL                  ⎰,(primary-quantity[,secondary-quantity]   ⎟     │
│⎜          ⎱average-record-length⎰                                          ⎟     │
│⎜                                          ⎰ddname                ⎱         ⎟     │
│⎜              [,directory-quantity]),⎰stepname.ddname         ⎰) )         ⎟     │
│⎝                                          ⎱stepname.procstep.ddname⎰       ⎠     │
├────────────────────────────────────────────────────────────────────────────────┤
│¹The maximum number of repetitions allowed is 7.                                  │
│²If only one "ddname" is specified, the delimiting parentheses may be omitted.    │
│³If neither "n" nor "P" is specified, 1 is assumed.                               │
│⁴This subparameter is applicable only for direct-access devices.                  │
│⁵This subparameter is the only keyword subparameter shown in this figure. All the │
│ remaining subparameters shown in the UNIT, SPACE, SPLIT, and SUBALLOC parameters are│
│ positional subparameters.                                                        │
│⁶If only "name" is specified, the delimiting parentheses may be omitted.          │
└────────────────────────────────────────────────────────────────────────────────┘
```

Figure 49. DD Statement Parameters for Optimization

VOLUME MOUNTING AND DEVICE SEPARATION:

UNIT=(name $\begin{bmatrix} ,n \\ ,P \end{bmatrix}$ [,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."

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 honors the request for device separation if possible, but ignores the SEP sub-parameter if an insufficient number of access arms are 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.

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, the "directory quantity" specifies the number of 256-byte blocks that are allocated to the directory of a new PDS.

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.

$$SPLIT=(n \left[\left\{ \begin{matrix} ,CYL \\ ,average\text{-}record\text{-}length \end{matrix} \right\} \right.$$

$$,(primary\text{-}quantity$$

$$\left. [,secondary\text{-}quantity]) \right])$$

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.

$$SUBALLOC=(\left\{ \begin{matrix} TRK \\ CYL \\ average\text{-}record\text{-}length \end{matrix} \right\}$$

$$(primary\text{-}quantity,$$

$$[,secondary\text{-}quantity]$$

$$[,directory\text{-}quantity]),$$

$$\left\{ \begin{matrix} ddname \\ stepname.ddname \\ stepname.procstep.ddname \end{matrix} \right\})$$

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."

The compiler, linkage editor, and load module 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

A listing of the source statements, a table of the source module names, an object module listing, and an object module card deck will be generated by the compiler, depending on the options specified by the user. Source module diagnostic messages are also produced during compilation.

Source Listing

If the SOURCE option is specified, the source listing is written in the data set specified by the SYSPRINT DD statement. An example of a source module listing is shown in Figure 51. This printout is the source listing of the sample program illustrated in Figure 50.

(This program will be used throughout the remainder of the publication for purposes of example illustration.)

Figure 50. Sample FORTRAN IV Program

Storage Map

If the MAP option is specified, 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 names are arranged across the page; six to a line. However, storage maps of variables not used in the source module are not produced.

Figure 52 is an example of a storage map produced for the sample program in Figure 50.

Object Module Listing

If the LIST option is specified, an object module listing is produced. An example of an object module listing is given in Figure 53.

```
                C       PRIME NUMBER PROBLEM
 C001          100 WRITE (6,8)
 C002            8 FORMAT (52H FOLLOWING IS A LIST OF PRIME NUMBERS FROM 1 TO 1000/
                   119X,1H1/19X,1H2/19X,1H3)
 0003          101 I=5
 CC04            3 A=I
 C005          102 A=SQRT(A)
 0006          103 J=A
 C007          104 DO 1 K=3,J,2
 0008          105 L=I/K
 CCC9          106 IF(L*K-I)1,2,4
 CC10            1 CONTINUE
 0011          107 WRITE (6,5)I
 0012            5 FORMAT (I20)
 0013            2 I=I+2
 CC14          108 IF(1000-I)7,4,3
 0015            4 WRITE (6,9)
 0016            9 FORMAT (14H PKOGRAM ERROR)
 0017            7 WRITE (6,6)
 0018            6 FORMAT (31H THIS IS THE END OF THE PROGRAM)
 CC19          109 STOP
 C020              END
```

Figure 51. Source Module Listing

		SCALAR MAP							
SYMBCL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATICN	SYMBOL	LOCATION	SYMBOL	LOCATION
I	BC	A	C0	J	C4	K	C8	L	CC
		SUBPROGRAMS CALLED							
SYMBCL	LCCATION	SYMBUL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION
IBCOM=	D0	SQRT	D4						
		FORMAT STATEMENT MAP							
SYMBCL	LOCATICN	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION	SYMBOL	LOCATION
8	DC	5	126	9	12A	6	13C		

Figure 52. Storage Map

LOCATION	STA NUM	LABEL	OP	OPERAND	BCD OPERAND
C00000			BC	15,12(0,15)	
C0C004			DC	06D4C1C9	
CCC008			DC	D5404C40	
00000C			STM	14,12,12(13)	
000010			LM	2,3,40(15)	
C00014			LR	4,13	
CC0016			L	13,36(0,15)	
0G001A			ST	13,8(0,4)	
CCC01E			STM	3,4,0(13)	
000022			BCR	15,2	
000024			DC	00000000	A4
CC0028			DC	00000000	A20
00002C			DC	00000000	A36
000198		A36	L	13,4(0,13)	
00019C			L	14,12(0,13)	
0001A0			LM	2,12,28(13)	
0001A4			MVI	12(13),255	
0001A8			BCR	15,14	
0001AA		A20	L	15,160(0,13)	IBCOM=
0001AE			LR	12,13	
CC01B0			LR	13,4	
C001B2			BAL	14,64(0,15)	
0001B6			LR	13,12	
CC01B8	1	100	L	15,160(0,13)	IBCOM=
0001BC			BAL	14,4(0,15)	
0CC1C0			DC	00000006	
0001C4			DC	000000DC	
0001C8			L	15,160(0,13)	IBCOM=
C001CC			BAL	14,16(0,15)	
0001D0	3	101	L	0,344(0,13)	
0001C4			ST	0,140(0,13)	I
C001D8	4	3	L	0,140(0,13)	I
0001CC			LPR	1,0	
0001DE			ST	1,324(0,13)	
CC01E2			LD	0,320(0,13)	
C0C1E6			AD	0,304(0,13)	
C001EA			LTR	0,0	
C001EC			BALR	14,0	
C001EE			BC	11,6(0,14)	
0001F2			LCDR	0,0	
0001F4			STE	0,144(0,13)	A
CC01F8	5	102	LA	1,168(0,13)	
0001FC			L	15,164(0,13)	SQRT
0C0200			BALR	14,15	
CC0202			STE	0,144(0,13)	A
000206	6	103	SDR	0,0	
CC0208			LE	0,144(0,13)	A
00020C			AW	0,336(0,13)	
CC0210			STD	0,328(0,13)	
000214			L	0,332(0,13)	
C00218			LTDR	0,0	
C0021A			BALR	14,0	
0C021C			BC	11,6(0,14)	
000220			LCR	0,0	
000222			ST	0,148(0,13)	J
0C0226	7	104	L	0,348(0,13)	
00022A		L44	ST	0,152(0,13)	K
C0022E	8	105	L	0,140(0,13)	I
CCC232			SRDA	0,32(0)	
C00236			D	0,152(0,13)	K
00023A			ST	1,156(0,13)	L
C0023E	9	106	L	1,156(0,13)	L
000242			M	0,152(0,13)	K
CC0246			S	1,140(0,13)	I
0C024A			LTR	1,1	
00024C			L	14,104(0,13)	2
0CC250			BCR	8,14	
000252			L	14,108(0,13)	4
000256			BCR	2,14	
C00258	10	1	L	0,152(0,13)	K
00025C			L	1,116(0,13)	L44
0C0260			LA	2,2(0,0)	
000264			L	3,148(0,13)	J
000268			BXLE	0,2,0(1)	
C0026C	11	107	L	15,160(0,13)	IBCOM=

Figure 53. Object Module Listing (Part 1 of 2)

```
CC0270                    BAL      14,4(0,15)
000274                    DC       C0000006
0C0278                    DC       00000126
0C027C                    L        15,160(0,13)        IBCOM=
0C0280                    BAL      14,8(0,15)
000284                    DC       0450D08C
C00288                    BAL      14,16(0,15)
00028C        13    2     L        0,140(0,13)         I
000290                    A        0,352(0,13)
0C0294                    ST       0,140(0,13)         I
000298        14    108   L        0,356(0,13)
0CC29C                    S        0,140(0,13)         I
C002A0                    LTR      0,0
0002A2                    L        14,112(0,13)        7
0002A6                    BCR      4,14
0002A8                    L        14,96(0,13)         3
0002AC                    BCR      2,14
0002AE        15    4     L        15,160(0,13)        IBCOM=
00C2B2                    BCR      0,0
0002B4                    BAL      14,4(0,15)

0C02B8                    DC       0000C006
0002BC                    DC       000C012A
CC02C0                    BAL      14,16(0,15)
CC02C4        17    7     L        15,160(0,13)        IBCOM=
0002C8                    BAL      14,4(0,15)
0C02CC                    DC       00000006
000200                    DC       0000013C
C002C4                    BAL      14,16(0,15)
0002D8        19    109   L        15,160(0,13)        IBCOM=
C002CC                    BAL      14,52(0,15)
CC02E0                    DC       05404040
0002E4                    DC       40FC
                          END




    TUTAL MEMORY REQUIREMENTS 0002E6 BYTES
```

Figure 53. Object Module Listing **(Part 2 of 2)**

Object Module Card Deck

If the DECK option 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.

OBJECT MODULE CARDS: Every card in the
object module deck contains a 12-2-9 punch
in column 1. The identifier consists of
the characters ESD, RLD, TXT or END in
columns 2 through 4. The first four char-
acters of the name of the program are
placed in columns 73 through 76 with the
sequence number of the card in columns
77-80.

ESD CARD: Four types of ESD cards are
generated, as follows:

ESD, type 0 - contains the name of the pro-
              gram and indicates the begin-
              ning of the object module.
              The name is the module name
              followed by a #.

ESD, type 1 - contains the entry point
              (where control is given to
              begin execution of the
              module). The entry point is
              the module name on a SUBROU-
              TINE or FUNCTION statement,
              or the name specified in the
              NAME option, or the name
              MAIN.

ESD, type 2 - contains the names of subpro-
grams referred to in the
source module by CALL state-
ments, EXTERNAL statements,
explicit function references,
and implicit function ref-
erences.

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 ref-
erences, 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 con-
stants and variables used by the programmer
in his source module, any constants and
variables used by the programmer in his
source module, any constants and variables
generated by the compiler, coded informa-
tion 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: Figure 54
indicates the FORTRAN object module deck
structure.

Source Module Diagnostics

Two types of diagnostic messages are
written by the compiler -- error/warning
messages and status.
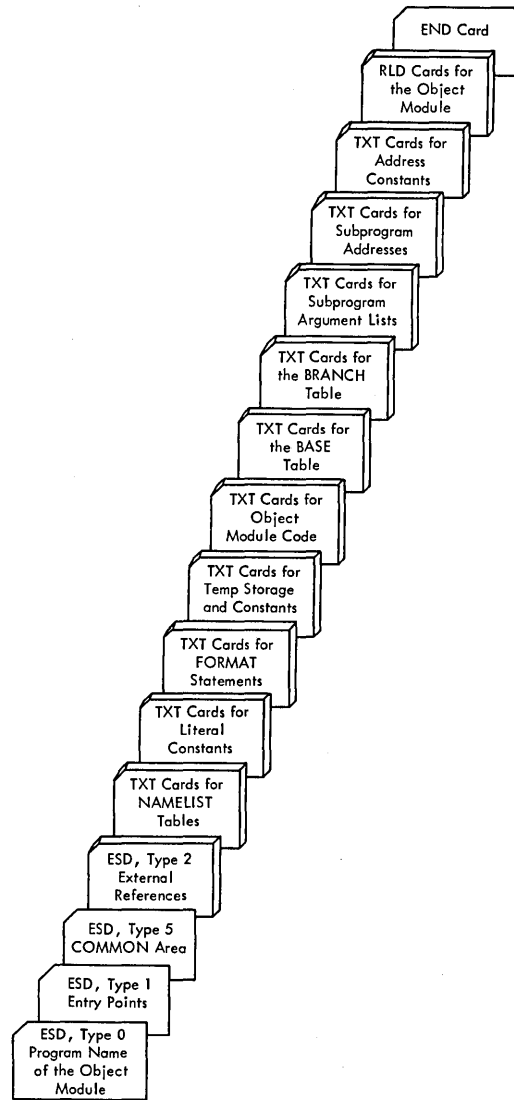
Figure 54. Object Module Deck Structure

Source Module Error/Warning Messages: The
error/warning messages produced by the com-
piler are noted on the source listing
immediately after the statement in which
they occur. A maximum of four messages
appears on each line. Figure 55 illus-
trates 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 warn-
ing messages. The serious error messages
have a condition code of eight and the
warning messages a code of four or zero.

For a description of error/warning mes-
sages, see Appendix D.

80

```
┌─────────────────────────────────────────────┐
│XX = A+B+-C/(X**3-A**-75)                      │
│         $                 $                   │
│                                               │
│n) y message, n) y message                     │
│                                               │
│Where:  n is an integer noting the posi-       │
│        tional 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 partic-      │
│        ular error in the statement (this      │
│        symbol is always noted on the          │
│        line following the source state-       │
│        ment and underneath the error).        │
│                                               │
│        message is the actual message          │
│        printed                                │
└─────────────────────────────────────────────┘
```

Figure 55. Format of Diagnostic Messages

Status Messages: During operation of the compiler, messages may occur which indicate termination of compilation. These messages are noted as a result of internal compiler errors which render continuation of compilation impossible. These messages are terminal error messages and have a condition code of 16. For a description of these messages see Appendix D.

LINKAGE EDITOR OUTPUT

The linkage editor produces a map of a load module if the MAP option is specified, or a map and a cross-reference list if the XREF option is specified. The linkage editor also produces diagnostic messages, which are discussed in the Linkage Editor 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) area 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 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 56 shows a load module map for the Sample Program shown in Figure 50.

Cross-Reference List

If the option XREF 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 referenced 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.

CONTROL SECTION			ENTRY							
NAME	ORIGIN	LENGTH	NAME	LOCATION	NAME	LOCATION	NAME	LOCATION	NAME	LOCATION
MAIN#	00	2E6								
			MAIN	00						
IHCFCCMH*	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	1072	FCVCO	1F6C		
IHCFIOSH*	2340	C30								
			FIOCS=	2340						
IHCUATBL*	2F70	108								

Figure 56. Module Map

Figure 57 shows the cross-reference list for the Sample Program shown in Figure 50.

LCCATION	REFERS TO SYMBOL	IN 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=	IHCFCCMH
21FC	IBCOM=	IHCFCOMH
2464	IHCUATBL	IHCUATBL
2470	IBCOM=	IHCFCOMH
ENTRY ADDRESS	00	
TOTAL LENGTH	3078	

Figure 57. Linkage Editor Cross-Reference List

LOAD MODULE OUTPUT

The programmer defines the output data sets for load module execution in READ, WRITE, and FORMAT statements. 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.

Error Code Diagnostic Messages and Traceback

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 (see "FORTRAN Job Processing"). The message is of the form:

IHCxxxI   [message text]

TRACEBACK FOLLOWS ROUTINE ISN REG. 14

where:
        xxx is a 3-digit error code

These error codes are described in Appendix D. 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, IHCNAMEL IHCDIOSE, 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 will contain the identifier specified in the SAVE macro instruction or equivalent coding. (If the identifier specified is longer than eight characters, only the first eight will appear.) If no identifier is specified, the traceback routine name field will be blank or have content meaningless for the traceback.

Internal statement numbers are generated for function references and calls when the ID option is specified on the EXEC card at compile time. These numbers appear in the traceback, except for FORTRAN calls to IBCOM for which internal statement numbers are not generated.

For an assembler language program or subprogram, the internal statement number field will contain the value of the binary calling sequence identifier specified in the CALL macro instruction or equivalent coding. If no identifier was specified, the field will be blank or its contents will be meaningless in the traceback.

If the traceback cannot be completed, the message TRACEBACK TERMINATED is issued and the job step is terminated. This message will appear only if (1) 13 names of subprograms appear in the traceback or (2) a calling loop has been detected, e.g., subprogram A calling B calling A.

```
r----------------------------------------------------------------------------------
|    IHC219I
|    TRACEBACK FOLLOWS               ROUTINE        ISN           REG. 14
|                                    IBCOM                        820068FC
|                                    MASTR                        00005378
|                                    PAYROLL                      00003148
|    ENTRY POINT = 5000
L----------------------------------------------------------------------------------
```

Figure 58.   Sample Traceback for Execution-Time Errors

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 58 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. 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.

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 Control Program Messages, Completion Codes, and Storage Dumps 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.

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  ⎫   │EP=IEYFORT                    │
│       │⎩ATTACH⎭   │ PARAM=(optionaddr            │
│       │           │  [,ddnameaddr]),VL=1         │
│       │           │                              │
│[name] │CALL       │IEYFORT                       │
│       │           │ PARAM=(optionaddr            │
│       │           │  [,ddnameaddr]),VL           │
└───────┴───────────┴──────────────────────────────┘
```

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 (one that is not also a full-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 half-word boundary (one that is not also a full-word 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 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

VL=1
    specifies that the sign bit of the last full-word of the address parameter list is to be set to 1.

The   following   examples   show   several methods to process load modules.

Example 1:

Problem   Statement: A   previously created   and   cataloged   data   set SCIENCE.MATH.MATRICES   contains a set of 80 matrices.   Each matrix is an array containing real 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   PDS   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.

The I/O flow for the example is shown in Figure 59.   The job control statements used to define this job are shown is Figure   60.
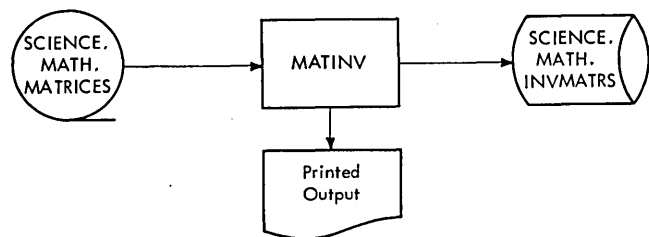
Figure 59.   Input/Output Flow for Example 1

Explanation: The JOB statement identifies the programmer as JOHN SMITH   and   supplies the   account   number   537.   Both   control statements and control statement error messages are printed on 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.   (In the load module, 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.   (In the load module, data set reference number 10 is used for printed output.) The data set is   written   on   the class   of   devices   specified in the SYSOUT parameter.

DD statement FT04F001 defines the output data set.   (In the load   module,   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 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   2508,   the   maximum   size   of   a variable-length record   for   this   application.   (The   maximum   size   of a record in this data set   is   the   maximum   number   of elements   (25) in a matrix multipled by the number of bytes   (4) allocated for   an   element,   plus   4   for the variable that indicates the size of the matrix,   plus   4   for the   segment   control   word   (SCW) that contains a count of the number of   data   bytes in   the   record.)   The   buffer   length   is specified as 2512 (the 4   extra   bytes   are for   the   block   control word that contains the length of the block.)

The   SEP   parameter   indicates   that   the data set SCIENCE.MATH.INVMATRS should use a different   channel   from that used for data set SCIENCE.MATH.MATRICES.

Sample Coding Form

1-10	11-20	21-30	31-40	41-50	51-60	61-70	71-80
1234567890	1234567890	1234567890	1234567890	1234567890	1234567890	1234567890	1234567890

```
//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=908,BLKSIZE=2728)
```

Figure 60. Job Control Statements for Example 1

Sample Coding Form

1-10	11-20	21-30	31-40	41-50	51-60	61-70	71-80
1234567890	1234567890	1234567890	1234567890	1234567890	1234567890	1234567890	1234567890

```
//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
//FT10F001 DD UNIT=PRINTER
```

Figure 61. Job Control Statements for Example 2

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 61 shows the job control statements used to process this job.

The load modules REFDAT, ANALYZ, and REPORT are contained in the private library FIRING.

Explanation: The JOB statement indicates the programmer's name, JOHN SMITH, and that control statements and control statement error are printed on the console typewriter.

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 blocked; the record length is 400; and the buffer length is 2000.

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 DSNAME parameter indicates that the data set name is copied from DD statement

FT12F001 in job step STEP1. 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. 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. The DSNAME parameter indicates that the data set name is copied from the DD statement FT20F001 in job step STEP2.

DD statement FT10F001 indicates that the data set reference number 10 is used to print the reports and graphs for job step three.

Example 3:

A data set has been created that contains master records for an index of stars. Each star is identified by a unique six-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.

The following conventions must be observed 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.

Problem Statement: Figure 62 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.

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 62 is similar to the diagram shown in Figure 48, except Figure 62 includes blocks that describe updating variables in master records already present in the data set. (Figure 48 includes blocks describing certain operations that must be performed when a direct access data set is first written.) Also, Figure 62 is adapted to Example 3, while Figure 48 is more general. Figure 64 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
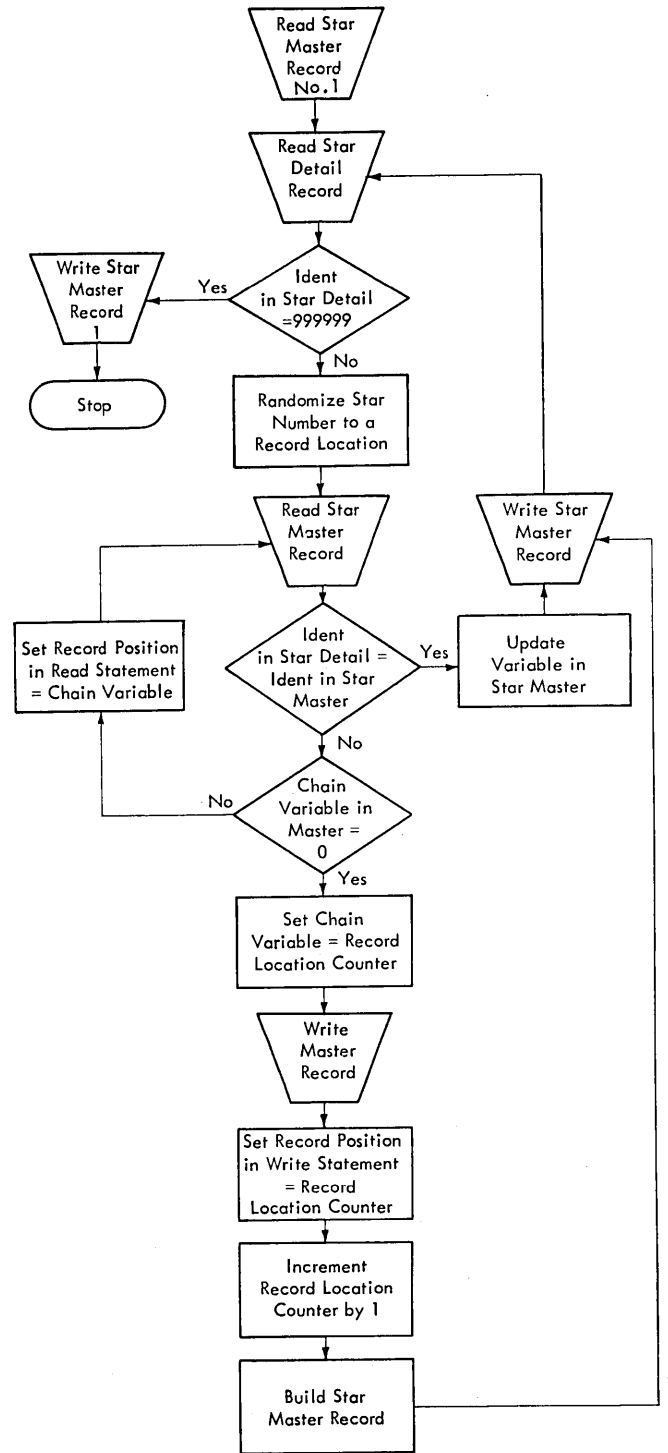
Figure 62. Block Diagram for Example 3

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 64 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 63 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

```
//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 63. 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,101)IDSTRM,ICHAIN
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 64. FORTRAN Coding for Example 3

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 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 full-word 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 one.

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 65 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. A branch is made to the address in register 15 and the return address is saved in general register 14. Table 14 illustrates the use of the linkage registers.

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.

```
 r---------------------------------------------------------------------------------------------¬
| AREA------------->  r-----------------------------------------------------------------------¬ |
|    (word 1)         | This word is used by a FORTRAN-compiled routine to store              | |
|                     | its epilogue address and may not be used by the assem-               | |
|                     | bler language subprogram for any purpose.                            | |
| AREA+4----------->  +----------------------------------------------------------------------+ |
|    (word 2)         | If the  program that  calls  the  assembler  language                | |
|                     | subprogram  is itself a subprogram, this word.contains               | |
|                     | the address of the save area of the  calling program;               | |
|                     | otherwise, this word is not used.                                    | |
| AREA+8----------->  +----------------------------------------------------------------------+ |
|    (word 3)         | The address of the save area of the called subprogram.               | |
| AREA+12---------->  +----------------------------------------------------------------------+ |
|    (word 4)         | The contents of register 14(the return address).  When              | |
|                     | the subprogram returns control, the first byte of this              | |
|                     | location is set to ones.                                             | |
| AREA+16---------->  +----------------------------------------------------------------------+ |
|    (word 5)         | The contents of register 15(the entry address).                     | |
| AREA+20---------->  +----------------------------------------------------------------------+ |
|    (word 6)         | The contents of register 0.                                          | |
| AREA+24---------->  +----------------------------------------------------------------------+ |
|    (word 7)         | The contents of register 1.                                          | |
|                     +----------------------------------------------------------------------+ |
|                     |                             .                                        | |
|                     |                             .                                        | |
|                     |                             .                                        | |
| AREA+68---------->  +----------------------------------------------------------------------+ |
|    (word 18)        | The contents of register 12.                                         | |
|                     L----------------------------------------------------------------------- |
 L---------------------------------------------------------------------------------------------
```

•Figure 65. Save Area

Table 14. 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 reg- isters 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 particu- lar values that can be contained in the register are: 16 - terminal error detected during execution of a subprogram (an IHCxxxI message is generated) 4*i - a RETURN i statement is executed n - a STOP n statement is executed 0 - a RETURN or a STOP statement is executed

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.

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 66 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 Assembly 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.

Figure 67 shows the linkage conventions for an assembler subprogram that calls another assembler subprogram.

In-Line Argument List

The assembler programmer may establish an in-line argument list instead of out-of-line list. In this case, he may substitute the calling sequence and argument list shown in Figure 68 for that shown in Figure 67.

Name	Oper.	Operand	Comments
deckname	START	0	
	BC	15,m+1+4(15)	BRANCH AROUND CONSTANTS IN CALLING SEQUENCE
	DC	X'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 66. Linkage Conventions for Lowest Level Subprogram

Name	Oper.	Operand	Comments
deckname	START	0	
	EXTRN	$name_2$	NAME OF THE SUBPROGRAM CALLED BY THIS SUBPROGRAM
	BC	15,m+1+4(15)	
	DC	X'm'	
	DC	CLm'$name_1$	
*		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 15.
*	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,$prob_1$	
AREA	DS	18F	RESERVES 18 WORDS FOR THE SAVE AREA
*		END OF SAVE ROUTINE	
$prob_1$	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($name_2$)	
*		ARGUMENT LIST	
ARGLIST	DC	AL4(arg_1)	ADDRESS OF FIRST ARGUMENT
		.	
		.	
		.	
	DC	X'80'	INDICATE LAST ARGUMENT IN ARGUMENT LIST
	DC	AL3(arg_n)	ADDRESS OF LAST ARGUMENT

Figure 67. Higher Level Assembler Subprogram

Sharing Data in COMMON

Both named and blank COMMON in a FORTRAN IV program can be referred to by an assembly 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 assembly language subprogram, only one blank COMMON area is generated for the output load module. Data in this blank COMMON is accessible to both programs.

```
ADCON       DC     A(prob_1)
             .
             .
             .
            LA     14,RETURN
            L      15,ADCON
            CNOP   2,4
            BALR   1,15
            DC     AL4(arg_1)
            DC     AL4(arg_2)
             .
             .
             .
            DC     X'80'
            DC     AL3(arg_n)
RETURN      BC     0,X'isn'
```

Figure 68. In-Line Argument List

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. (If statement numbers are passed as arguments, they are not placed in the argument list. See "Return i in an Assembler Language Subprogram.") 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 15 shows the general subscript format for arrays of 1, 2, and 3 dimensions.

Table 15. 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 the 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 variables in a three-dimensional array, the displacement (DISPLC) is calculated to be 28. The following instructions can be used to move the variable,

```
LA    Q,8(13)
LA    R,DISPLC
L     S,0(Q,R)
ST    S,ARVAR
```

where:

Q, R, and S are general registers;
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 69 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
```

Name	Oper.	Operand	
ADDARR	START	0	
B	EQU	8	
	BC	15,12(15)	
	DC	X'7'	
	DC	CL7'ADDARR'	
ADDARR	STM	14,12,12(13)	
	BALR	B,0	
	USING	*,B	
	L	2,8(1)	MOVE THIRD ARGUMENT TO THE LOCATION CALLED INDEX IN
	MVC	INDEX(4),0(2)	THE ASSEMBLER LANGUAGE SUBPROGRAM.
	L	3,0(1)	MOVE FIRST ARGUMENT TO THE LOCATION CALLED VAR IN THE
	MVC	VAR(4),0(3)	ASSEMBLER LANGUAGE SUBPROGRAM
	L	4,4(1)	LOAD THE ADDRESS OF THE ARRAY TO GENERAL REGISTER 4.
	L	4,4(4)	
	user.	written statements	
		.	
		.	
		.	
	LM	14,12,28(13)	
	MVI	12(13),X'FF'	
	BCR	15,14	
	DS	0F	
INDEX	DS	1F	
VAR	DS	1F	

Figure 69. Assembler Subprogram Example

This appendix contains a detailed description of the diagnostic messages produced during compilation and load module execution.

COMPILER DIAGNOSTIC MESSAGES

Two types of compiler diagnostic messages are generated -- error/warning and status.

Compiler 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 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)

IEY002I   LABEL

> Explanation: The statement in question is unlabeled and follows a transfer of control; the statement therefore cannot be executed. (Condition Code 0)

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 4)

IEY004I   COMMA

> Explanation: The comma required in the statement has been omitted. (Condition Code 0)

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)

IEY006I   DUPLICATE LABEL

> Explanation: The label appearing in the label field of a statement has previously been defined for another statement. (Condition Code 8)

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.
>
> 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. (Condition Code 8)

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.
>
> 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. (Condition Code 8)

IEY009I ORDER

> Explanation: The statements contained in the source module are used in an improper 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. (Condition Code 8)

IEY010I SIZE

> Explanation: A number used in the source module does not conform 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 Specification statement is not acceptable; an integer constant is too large. (Condition Code 8)

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)

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)

IEY013I SYNTAX

> Explanation: The statement or part of a statement to which this message refers does not conform 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. (Condition Code 8)

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)

IEY015I NO END CARD

> Explanation: The source module does not contain an END statement. (Condition Code 0)

IEY016I ILLEGAL STA.

> Explanation: The context in which the statement in question has been used is illegal.
>
> Examples: The statement "S" in a Logical IF statement is a Specification statement, a DO statement, etc.; an ENTRY statement appears in the source module and the source module is not a subprogram. (Condition Code 8)

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)

IEY018I NUMBER ARG

> Explanation: The reference to a library subprogram specifies an incorrect number of arguments. (Condition Code 4)

IEY019I FUNCTION ENTRIES UNDEFINED

> Explanation: If the program being compiled is a FUNCTION subprogram, and there is no scalar with the same name as the FUNCTION nor is there a definition for each ENTRY, the message appears on the SYS-PRINT data set. A list of the names in error is printed following the message. (Condition Code 0)

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)

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)

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. However, if there are no undefined labels, the word NONE appears on the same line as the message. (Condition Code 8)

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)

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)

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)

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)

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)

Compiler Status Messages

The following paragraphs describe the messages that are produced during the operation of the 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)

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.

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)

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)

IEY033I   EXCESSIVE COMMENTS

Explanation: If more than 49 contiguous comment and/or continuation cards are read, this message is printed, and the remainder of the cards are skipped until a card that is not a comment or continuation card is encountered. (There is no restriction on the number of comments at the beginning of the deck.) No condition code is generated for this message.

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, Form C28-6647.

IEY035I   UNABLE TO OPEN ddname

Explanation: This message is produced when the required ddname data definition card is missing or the ddname is misspelled.

LOAD MODULE EXECUTION DIAGNOSTIC MESSAGES

The load module produces three types of diagnostic messages:

• Program interrupt messages.

• Execution error messages.

• Operator messages.

Program Interrupt Messages

Program interrupt messages containing the old Program Status Word (PSW) are written when an exception occurs. The format is:

IHC2101   PROGRAM INTERRUPT $\left(\begin{smallmatrix} A \\ P \end{smallmatrix}\right)$ OLD PSW IS xxxxxxx $\left\{\begin{smallmatrix} 4 \\ 5 \\ 6 \\ 7 \\ 9 \\ C \\ D \\ F \end{smallmatrix}\right\}$ xxxxxxxx
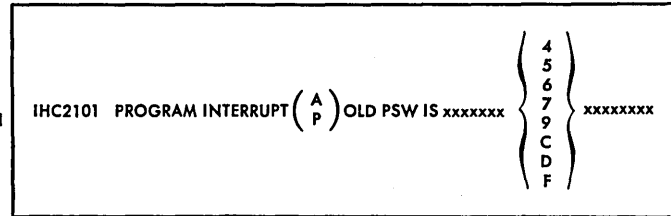
Note: Codes 4, 5, 6, and 7 are associated with the execution-time adjustment of boundary alignment errors and appear only when the system is generated to provide boundary alignment adjustment; i.e., when BOUNDRY=ALIGN is specified in the FORTLIB macro instruction (see the publication IBM System/360 Operating System, System Generation).

If the letter A appears in the message, 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, 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, assigned 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.

Addressing Exception: The addressing exception, assigned 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.

Specification Exception: The specification exception, assigned code 6, is recognized when a data address does not specify an integral boundary for that unit of informa-

tion. A specification error would occur during execution of the following instructions:

```
DOUBLE-PRECISION D, E
COMMON A, B, C
EQUIVALENCE (B, D)
D = 3.0D02
```

Note: If an instruction contains a boundary violation, a specification interrupt will occur and the message will be issued with code 6. The boundary-adjustment routine will then be invoked, if the BOUNDRY=ALIGN option was specified in the FORTLIB macro instruction during system generation. If an instruction which has been processed for boundary misalignment also contains a protection, addressing, or data error, the interrupt message will be reissued with the appropriate code (4, 5, or 7). The job will then terminate because both a specification error and a protection, addressing, or data error have been detected. The completion code in the dump will specify that the job terminated because of the specification error.

Data Exception: The data exception, assigned code 7, is recognized when the sign or 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.

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. For example, a divide exception would occur during execution of the following statement:

K=I/J

where:

J=0 and
I=7

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 7.2×10^{75}). For example, an exponent-overflow would 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 and 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

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 (TE) 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 5.4×10^{-79}). For example an exponent-underflow exception would 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 and 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

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 (TE) may be computed as follows:

TE=(Bits 1 to 7) - 128 - 64

Before program execution continues, the FORTRAN 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 which 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 would occur during execution of the following statement:

C=A/B

where:

        B=0.0 and
        A=1.0

Execution Error Messages

Execution error messages have the form:

IHCxxxI [message text]

TRACEBACK FOLLOWS ...

where:
        xxx is the error code

Traceback, which is a diagnostic tool containing a list of the routines in the direct line of call to the routine in error, is discussed in the section "System Output."

In the following text, the error codes are given with an explanation describing the type of error. Preceding the explanation, an abbreviated name is given indicating the origin of the error. For any load module execution error, unless otherwise stated below, a condition code of 16 is generated and the job step is terminated.

The abbreviated name for the origin of the error is:

IBC -- IHCFCOMH routine (performs interruption, conversion, and error procedures).

FIOCS -- IHCFIOSH routine (performs I/O operations for FORTRAN load module execution).

NAMEL -- IHCNAMEL routine (performs the processing of NAMELIST specifications).

IBERR -- IHCIBERH routine (performs the processing of errors detected during execution of the load module).

DIOCS -- IHCDIOSE routine (performs direct-access I/O operations for FORTRAN load module execution).

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).

IHC211I

Explanation: IBC -- An invalid character has been detected in a FORMAT statement.

IHC212I

Explanation: IBC -- An attempt has been made to read or write a record, under FORMAT control, that exceeds the buffer length.

IHC213I

Explanation: IBC -- The input list in an I/O statement without a FORMAT specification is larger than the logical record.

104

IHC214I

> Explanation: FIOCS -- For records in sequential data sets written with no FORMAT control, for which the RECFM subparameter must be V (variable), either U (undefined) or F (fixed) was specified.

IHC215I

> Explanation: FCVTH -- An invalid character exists for the decimal input corresponding to an I, E, F, or D format code.

IHC216I

> Explanation: LIB -- An illegal sense light number was detected in the argument list in a call to the SLITE or SLITET subroutine.

IHC217I

> Explanation: IBC -- An end of data set was sensed during a READ operation; that is, a program attempted to read beyond the data.

| IHC218I   I/O ERROR xxx...xxx

> Explanation: IBC -- A permanent input/output error has been encountered, or an attempt has been made to read or write with magnetic tape a record that is less than 18 bytes long. xxx...xxx is the character string formatted by the SYNADAF macro instruction. For an interpretion of this information, see the publication IBM System/360 Operating System: Supervisor and Data Management Macro Instructions, Form C28-6647. 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.

IHC219I

> Explanation: FIOCS -- 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.

IHC220I

> Explanation: FIOCS -- A data set reference number exceeds the limit specified for data set reference numbers when this operating system was generated.

IHC221I

> Explanation: NAMEL -- An input variable name exceeds eight characters.

IHC222I

> Explanation: NAMEL -- An input variable name is not in the NAMELIST dictionary, or an array is specified with an insufficient amount of data.

IHC223I

> Explanation: NAMEL -- An input variable name or a subscript has no delimiter.

IHC224I

> Explanation: NAMEL -- A subscript is encountered after an undimensioned input name.

IHC225I

> Explanation: FCVTH -- An illegal character is encountered on input for the Z format code.

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".

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.

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.

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.

IHC234I

> Explanation: DIOCS -- The data set assigned to print execution error messages cannot be a direct access data set.

IHC235I

> Explanation: DIOCS -- A data set reference number assigned to a direct access data set has been used for a sequential data set.

IHC236I

> Explanation: IBC -- A READ is executed for a data set that has not been created.

IHC237I

> Explanation: DIOCS -- Length of record read did not correspond to length of record specified in DEFINE FILE statement.

IHC241I

> 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.

IHC242I

> Explanation: LIB -- For an exponentiation operation (R**J) in the subprogram IHCFRXPI (FRXPI#),

where R represents a real*4 variable or integer constant, R is equal to zero and J is less than or equal to zero.

IHC243I

> Explanation: LIB -- For an exponentiation operation (D**J) in the subprogram IHCFDXPI (FDXPI#), 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.

IHC244I

> 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.

IHC245I

> 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.

IHC246I

> Explanation: LIB -- For an exponentiation operation (Z**J) in the subprogram IHCFCXPI(FCXPI#), where Z represents a complex*8 variable or integer constant, Z is equal to zero and J is less than or equal to zero.

IHC247I

> 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.

106

IHC251I

Explanation: LIB -- In the sub-
program IHCSSQRT(SQRT), the argu-
ment is less than 0.

IHC252I

Explanation: LIB -- In the sub-
program IHCSEXP(EXP), the argument
is greater than 174.673.

IHC253I

Explanation: LIB -- In the sub-
program IHCSLOG(ALOG and ALOG10),
the argument is less than or equal
to zero. Because this subprogram
is called by an exponential sub-
program, this message also indi-
cates that an attempt has been
made to raise a negative base to a
real power.

IHC254I

Explanation: LIB -- In the sub-
program IHCSSCN(SIN and COS), the
absolute value of an argument is
greater than or equal to $2^{18} \cdot \pi$
($2^{18} \cdot \pi = .82354966406249996D+06$).

IHC255I

Explanation: LIB -- In the sub-
program IHCSATN2, when entry name
ATAN2 is used, both arguments are
equal to zero.

IHC256I

Explanation: LIB -- In the sub-
program IHCSSCNH(SINH or COSH),
the argument is greater than or
equal to 174.673.

IHC257I

Explanation: LIB -- In the sub-
program IHCSASCN (ARCSIN or
ARCOS), the absolute value of the
argument is greater than 1.

IHC258I

Explanation: LIB -- In the sub-
program IHCSTNCT (TAN or COTAN),
the absolute value of the argument
is greater than or equal to $2^{18} \cdot \pi$
($2^{18} \cdot \pi = .82354966406249996D+06$).

IHC259I

Explanation: LIB -- In the subpro-
gram IHCSTNCT (TAN or COTAN), the
argument value is too close to one
of the singularities

($\pm\frac{\pi}{2}, \pm\frac{3\pi}{2}, \cdots$ for the tangent
or $\pm\pi, \pm2\pi, \cdots$ for the cotangent).

IHC261I

Explanation: LIB -- In the sub-
program IHCLSQRT(DSQRT), the argu-
ment is less than 0.

IHC262I

Explanation: LIB -- In the sub-
program IHCLEXP(DEXP), the argu-
ment is greater than 174.673.

IHC263I

Explanation: LIB -- In the sub-
program IHCLLOG(DLOG and DLOG10),
the argument is less than or equal
to zero. Because the subprogram
is called by an exponential sub-
program, this message also indi-
cates that an attempt has been
made to raise a negative base to a
real power.

IHC264I

Explanation: LIB -- In the sub-
program IHCLSCN(DSIN and DCOS),
the absolute value of the argument
is greater than or equal to $2^{50} \cdot \pi$
($2^{50} \cdot \pi = .35371188737802239D+16$).

IHC265I

Explanation: LIB -- In subprogram
IHCLATN2, when entry name DATAN2
is used, both arguments are equal
to zero.

IHC266I

Explanation: LIB -- In the sub-
program IHCLSCNH (DSINH or DCOSH),
the absolute value of the argument
is greater than or equal to
174.673.

IHC267I

Explanation: LIB -- In the sub-
program IHCLASCN (DARSIN or

DARCOS), the absolute value of the argument is greater than 1.

IHC268I

Explanation: LIB -- In the subprogram IHCLTNCNT (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).

IHC269I

Explanation: LIB -- In the subprogram IHCLTNCT (DTAN or DCOTAN), the argument value is too close to one of the singularities

$(\pm \frac{\pi}{2}, \pm \frac{3\pi}{2}, \dots$ for the tangent;
$\pm \pi, \pm 2\pi, \dots$ for the cotangent).

IHC271I

Explanation: LIB -- In the subprogram IHCCSEXP (CEXP), the value of the real part of the argument is greater than 174.673.

IHC272I

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).

IHC273I

Explanation: LIB -- In the subprogram IHCCSLOG (CLOG), the real and imaginary parts of the argument are equal to zero.

IHC274I

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).

IHC275I

Explanation: LIB -- In the subprogram IHCCSSCN (CSIN or CCOS), the absolute value of the imag-

inary part of the argument is greater than 174.673.

IHC281I

Explanation: LIB -- In the subprogram IHCCLEXP (CDEXP), the value of the real part of the argument is greater than 174.673.

IHC282I

Explanation: LIB -- In the subprogram IHCCLEXP (CDEXP), the absolute value of the imaginary part of the argument is greater than or equal to $2^{50} \cdot \pi$
($2^{50} \cdot \pi$ =.35371188737802239D+16).

IHC283I

Explanation: LIB -- In the subprogram IHCCLLOG (CDLOG), the real and imaginary parts of the argument are equal to zero.

IHC284I

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).

IHC285I

Explanation: LIB -- In the subprogram IHCCLSCN (CDSIN or CDCOS), the absolute value of the imaginary part of the argument is greater than 174.673.

IHC290I

Explanation: LIB -- In the subprogram IHCSGAMA (GAMMA), the value of the argument is outside the valid range ($2^{-252} < x < 57.5744$).

IHC291I

Explanation: LIB -- In the subprogram IHCSGAMA (ALGAMA), the value of the argument is outside the valid range ($0 < x < 4.2937 \times 10^{73}$).

IHC300I

Explanation: LIB -- In the subprogram IHCLGAMA (DGAMMA), the

value of the argument is outside the valid range ($2^{-252} < x < 57.5744$).

IHC301I

Explanation: LIB -- In the subprogram IHCLGAMA (DLGAMA), the value of the argument is outside the valid range ($0 < x < 4.2937 \times 10^{73}$).

Operator Messages

Operator messages for STOP and PAUSE are generated during load module execution.

The message for a PAUSE can be one of the following formats:

$$yy\ \text{IHC001A} \left\{ \begin{array}{l} \text{PAUSE } \underline{n} \\ \text{PAUSE '}\underline{message}\text{'} \\ \text{PAUSE 0} \end{array} \right\}$$

where:

yy   is the identification number

n   is the unsigned 1- to 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 ẏy,'z'

where:

yy   is the identification number

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 can be one of the following formats:

$$\text{IHC002I} \left\{ \begin{array}{l} \text{STOP } \underline{n} \\ \text{STOP 0} \end{array} \right\}$$

where:

n   is the unsigned 1- to 5-digit integer constant specified in a STOP source statement

0   is printed when a STOP statement is executed

User Response: None

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 FORTRAN 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

TRACE statement-label

is written.

SUBTRACE

SUBTRACE is used to trace program flow from one routine to another. For each subprogram called, the line

SUBTRACE subprogram-name

is written on entry to the subprogram, and the line

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

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

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

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

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

DISPLAY DURING I/O SKIPPED

is written in the DEBUG output.

PDUMP subroutine 69
PGM parameter 18-19
Positional parameters and subparameters 13
Primary input 36
Primary quanity subparameter 49,75
Printer spacing 25,51,92
Priority schedulers 12
PRIVATE subparameter 48
Private volume 48
PROC parameter 18-19
Procedure,cataloged 9-10
Procstep 19,20
Procstep.ddname 22
Procstep.SYSIN 30
Programmer's name parameter 15
PRTSP subparameter 25
PRTY parameter 16

Randomizing techniques 70
Real constants and variables 68
RECFM subparameter 51
Record chaining 70
REF subparameter 48
REGION parameter 17,21
Register use 93
REPLACE linkage editor control
  statement 40
RETAIN subparameter 49
Retention period for data sets 49
RETPD subparameter 51
Retrieving data sets 26,43
REWIND statement 44,57
RLD cards 80
RLSE subparameter 50
ROUND subparameter 50

Save area 92
Secondary input 38
Secondary quantity subparameter 49,75
SEP subparameter 74
Sequential data set 10
SER subparameter 49
Serial number, volume 49
Skeleton records 71
SL subparameter 43,50
SOURCE compiler option 34
Source listing 34,77
Space on direct-access volumes 49,74
SPACE parameter 49,74
Specifying execution of a program
    described in a cataloged procedure 19
    described in a previous job step 19
    in a library 19
SPLIT parameter 74
STACK subparameter 25

Stacker selection 25
Standard labels 11,26,43,50
Step
    job 9
    procedure 9
Stepname 17
STOP statement 82,109
Storage map 77
SUBALLOC subparameter 75
Subparameters 14
Subprograms, assembler language 92
SYSCP device class 33
SYSDA device class 33
SYSIN ddname 30,32,33
SYSLIB ddname 37,38
SYSLIN ddname 32,37,38
SYSLMOD ddname 37,38
SYSOUT parameter 33,51
SYSPRINT ddname 33,37,38
SYSPUNCH ddname 32,33
SYSSQ device class 33
SYSUT1 ddname 37,38
SYS1.FORTLIB 10,36,40

Tape density 52
Temporary names for data sets 28,46
Terminating a job 15
TRK subparameter 49,75
TRTCH subparameter 52
TXT card 80

Unblocked records
    direct-access data set 58
    sequential data set
        fixed-length 52-58
        undefined 52-58
        variable-length 52-58
UNCATLG subparameter 29
Undefined logical record 54
UNIT parameter 25-26,48,73-74
Unit record data sets 25-26,51

Variable-length logical record 54,55,57
Variables 69
Volume count subparameter 49
VOLUME parameter 48
Volume sequence number subparameter 48
Volume serial number 48

Warning messages
    (see error/warning messages)
Word 68

XREF linkage editor option 41,81

C28-6639-1

IBM