File No. S360-25
Form C28-6602-3

OS

IBM Systems Reference Library

IBM System/360 Operating System

FORTRAN IV (H) Programmer's Guide

Program Number 360S-FO-500

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

Fourth Edition

This publication corresponds to OS Release 16. It is a major revision of and makes obsolete Form C28-6602-2 and its associated Technical Newsletter, N28-2307. New material describes the optional Extended Error Message facility which, for certain error occurrences, either provides a standard corrective action or allows the user to correct his error through an installation-written routine, thus enabling execution to continue. Text for certain error messages is added, and there are miscellaneous clarifications and corrections throughout the publication. Changes to the text are indicated by a vertical line to the left of the change; revised illustrations are denoted by the symbol (•) to the left of the caption.

Specifications contained herein are subject to change from time to time. Any such change will be reported in subsequent revisions or Technical Newsletters.

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

Comments regarding this publication may be addressed to IBM Corporation, Programming Systems Publications, 1271 Avenue of the Americas, New York, N.Y. 10020

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 may use the language as described except for the debugging facility.

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

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

   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: 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: Messages and Codes, Form C28-6608

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

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

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

- Descriptions and explanations of compiler invocation from a problem program.
- Examples of job processing.
- Descriptions and explanations for the preparation of subprograms written in assembler language for use with a main program written in FORTRAN.
- Detailed descriptions of the diagnostic messages produced during compilation and load module execution.
- A list of USA carriage control characters.

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.

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.)

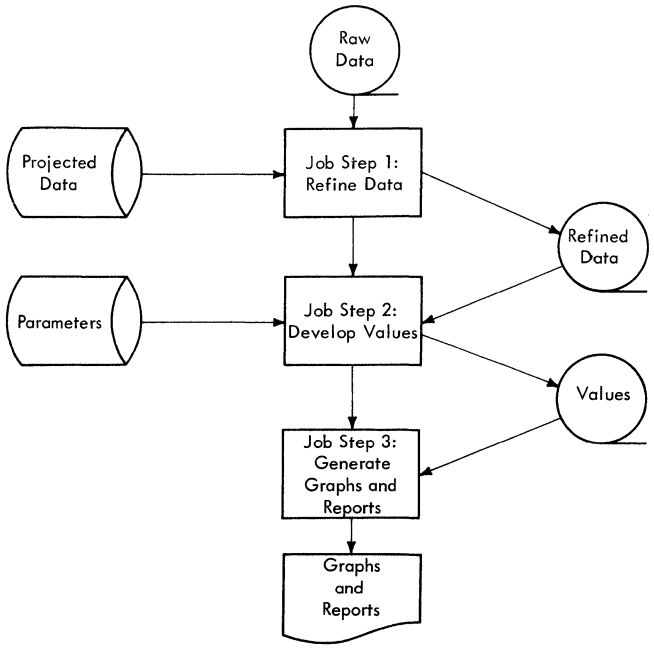


Figure 1. Rocket Firing Job

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:

FORTHC	compile
FORTHCL	compile and link edit
FORTHLG	link edit and execute
FORTHCLG	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. 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 FORTRAN direct access I/O statements; they cannot be transmitted by FORTRAN sequential I/O statements. Records in a direct access data set can be transmitted sequentially by using the associated variable in direct access I/O statements.

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

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

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 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:

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

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

or, 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.

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.

CONTINUING CONTROL STATEMENTS AND COMMENTS

A control statement can be written in card columns 1 through 71. If a control statement exceeds 71 columns, it may be continued onto the next card. If a statement is continued, it 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, columns 3 through 15 must be blank, and the continued portion of the statement must begin in column 16.

Comments are 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.

Note: Excessive continuation cards should be avoided whenever possible to reduce processing time for the control 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 optionally repeated any number of times in succession.

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

    Example: {NEW, OLD, MOD}[1]

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

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

JOB STATEMENT

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

1. Name of the job.
2. Accounting information relative to the job.

Name	Operation	Operand
		Positional Parameters
//jobname	JOB	[([account-number][,accounting-information])[1] [2] [3]]
		[,programmer-name][4] [5] [6]
		Keyword Parameters
		{MSGLEVEL=0 / MSGLEVEL=1}
		[COND=((code,operator)[,(code,operator)]...[7])[8]]
		[PRTY=nn][9]
		[MSGCLASS=x][9]
		[REGION=nnnnnK][9]

[1] If the information specified ("account-number" and/or "accounting-information") contains blanks, parentheses, or equal signs, it must be delimited by apostrophes instead of parentheses.
[2] If only "account-number" is specified, the delimiting parentheses may be omitted.
[3] The maximum number of characters allowed between the delimiting parentheses or apostrophes is 144.
[4] If "programmer-name" contains commas, parentheses, apostrophes, or blanks, it must be enclosed within apostrophes.
[5] When an apostrophe is contained within "programmer-name", the apostrophe must be shown as two consecutive apostrophes.
[6] The maximum number of characters allowed for "programmer-name" is 20.
[7] The maximum number of repetitions allowed is 7.
[8] If only one test is specified, the outer pair of parentheses may be omitted.
[9] This parameter is used with priority schedulers only. The sequential scheduler ignores it.

Figure 3. JOB Statement

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. Direction for where messages should come out.
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.

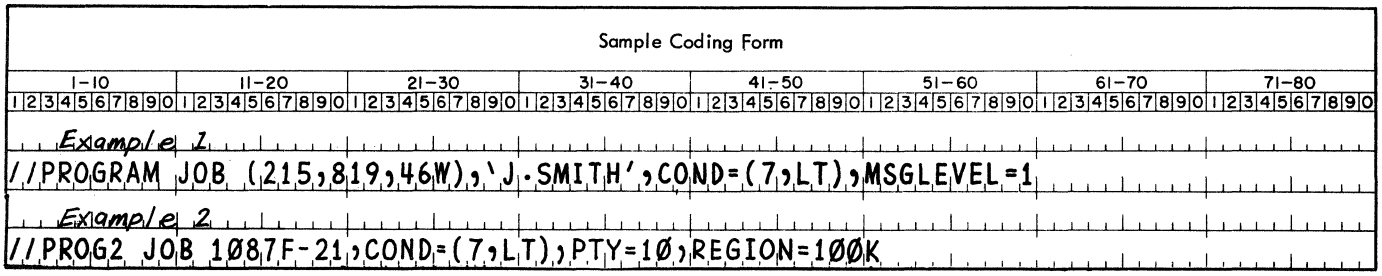


Figure 4. Sample Job Statements

As a system generation option with sequential schedulers, the account number can be established as a required parameter. With priority schedulers, the requirement can be established with a cataloged procedure for the input reader. (Information on the cataloged procedure for the input reader and how to write an accounting routine may be found in the System 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 supplied, its absence must be indicated by a comma preceding the programmer's name. If neither job accounting information nor programmer's name is present, commas need not be used to indicate their absence.

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

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 associated diagnostic messages are written for the programmer.

MSGLEVEL=1
: indicates that all control statements as well as control statement errors, and associated 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 diagnostics.

Conditions for Terminating a Job

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

0 - No errors or warnings detected.

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

8 - Errors detected, execution may fail. (If the LOAD option is specified, compilation continues. However, if the error is found in an executable statement, the statement is replaced by a call to the IBERH routine (IHCIBERH). If the resulting load module is executed, IBERH is called and execution is terminated. If the NOLOAD option is specified, compilation is terminated.

16 - Terminal errors detected, compiler or linkage editor terminated abnormally. (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 relationship between the code placed in the JOB statement and the codes issued by completed job steps. If the relationship is true, the job is terminated. The six operators and their meanings are:

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

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

```
COND=(7,LT)
```

the job is terminated.

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

Assigning Job Priority

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

If the PRTY parameter is omitted, the default job priority is assumed. This parameter is used with the priority scheduler only. The sequential scheduler ignores it.

Requesting a Message Class

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

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

If the MSGCLASS parameter is omitted, the job scheduler messages are routed to the standard output class, A. This parameter is used with priority schedulers only. The sequential scheduler ignores it.

Specifying Main Storage Requirements for a Job

The REGION=nnnnnK parameter is used to specify the amount of main storage to be allocated for the job. The "nnnnn" is to be replaced with the number of 1024-byte areas to be allocated to the job. This number can range from one to five digits.

If the REGION parameter is omitted, the default region size (as established in the input reader procedure) is assumed. This parameter is used with priority schedulers only. The sequential scheduler ignores it.

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

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

NAME FIELD

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

OPERAND FIELD

Positional Parameter

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

Specifying a Cataloged Procedure:

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

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

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

indicates that the cataloged procedure FORTHC is to be executed.

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

[1]If information from this control statement is referred to in a later job step, "stepname" is required.
[2]If this format is selected, it may be repeated in the EXEC statement, once for each step in the cataloged procedure.
[3]If the information specified contains blanks, parentheses, or equal signs, it must be delimited by apostrophes instead of parentheses.
[4]If only one option is specified and it does not contain any blanks, parentheses, or equal signs, the delimiting parentheses may be omitted.
[5]The maximum number of characters allowed between the delimiting apostrophes or parentheses is 40. The PARM parameter cannot occupy more than one card.
[6]If "accounting-information" does not contain commas, blanks, parentheses, or equal signs, the delimiting parentheses may be omitted.
[7]The maximum number of characters allowed between the delimiting apostrophes or parentheses is 144.
[8]The maximum number of repetitions allowed is 7.
[9]If only one test is specified, the outer pair of parentheses may be omitted.
[10]This parameter is used with priority schedulers only. Sequential schedulers ignore it.
[11]If only minutes are given, the parentheses need not be used. If only seconds are given, the parentheses must be used and a comma must precede the seconds.

Figure 5. EXEC Statement

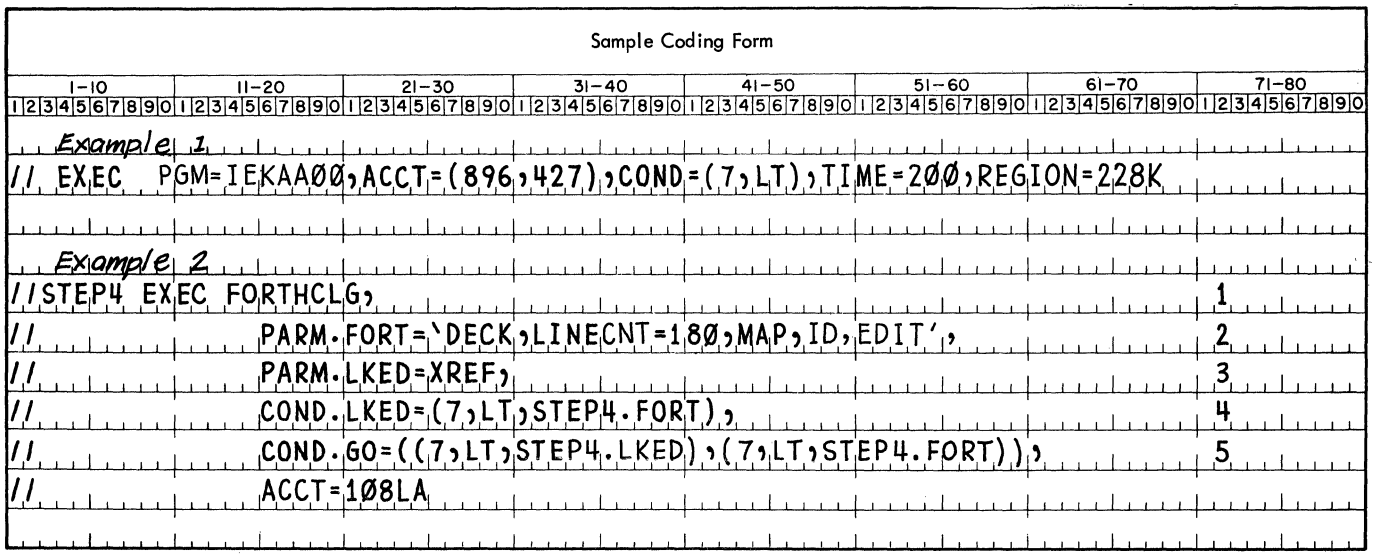


Figure 6. Sample EXEC Statements

Specifying a Program in a Library:

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

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

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

Specifying a Program Described in a Previous Job Step:

PGM=*.stepname.ddname
indicates that the name of the program to be executed is taken from a DD statement of a previous job step. The * indicates the current job; "stepname" is the name of a previous step within the current job; and "ddname" is the name of a DD statement within that previous job step. (The "stepname" cannot refer to a job step in another job.) The program referred to must be a member of a PDS. For example, in the 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=IEKAA00
//SYSPUNCH DD   UNIT=SYSCP
//SYSPRINT DD   SYSOUT=A
//SYSLIN   DD   DSNAME=LINKINP
            .
            .
            .
//LKED     EXEC PGM=IEWL
//SYSLMOD  DD   DSNAME=RESULT(ANS)
            .
            .
            .
```

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

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

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

Keyword Parameters

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

Options for the Compiler and Linkage Editor:

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

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

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

The format for compiler options, and those linkage editor options most applicable to the FORTRAN programmer is shown in Figure 7.

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

Condition for Bypassing a Job Step:

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

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

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

The subparameters for the COND parameter are of the form:

```
(code,operator[,stepname])
```

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

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

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

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.

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.

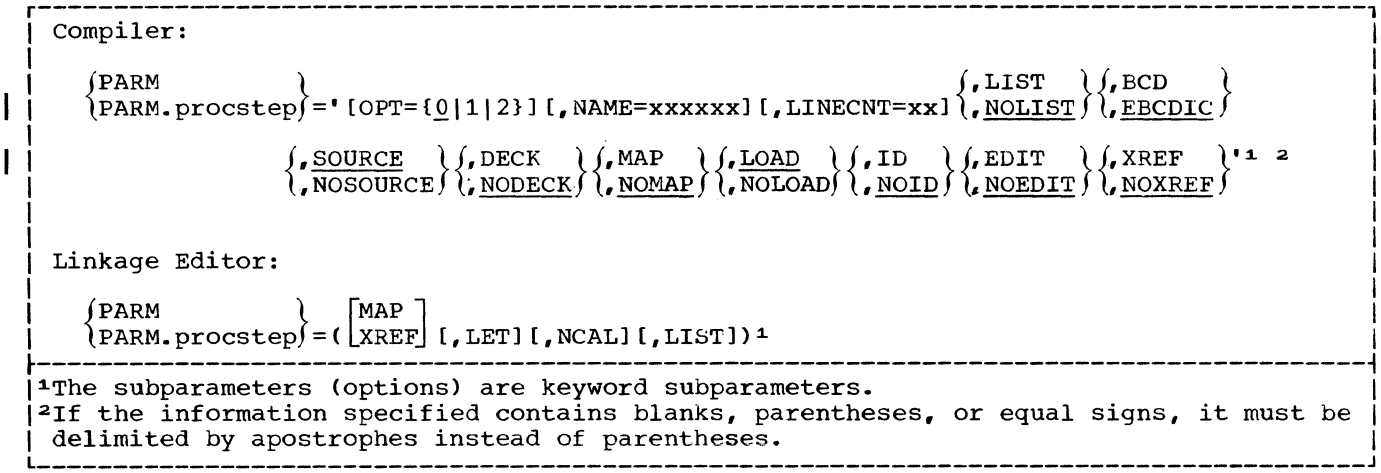


Figure 7. Compiler and Linkage Editor Options

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:

To limit the computing time used by a single job step or cataloged procedure, a maximum time for its completion can be assigned. If the job step is not completed in this time, the entire job is terminated.

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

If the TIME parameter is omitted, the default job step time limit (as established in the cataloged procedure for the input reader) is assumed. This parameter is used with priority schedulers only. Sequential schedulers ignore it.

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

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

Specifying Main Storage Requirements for a Job Step:

The REGION parameter may be specified in the JOB statement, thus overriding REGION parameters specified in any EXEC statements and applying to all steps of the job. However, if it is desired to allot to each step only as much storage as is required, the REGION parameter should be omitted from the JOB statement; each EXEC statement should contain a REGION parameter that specifies the amount of main storage to be allocated to the associated job step. The size is specified in the form "nnnnnK" where the "nnnnn" is to be replaced by the number of 1024-byte areas to be allocated to the job step. This number can range from one to five digits.

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. This parameter is used with priority schedulers only. Sequential schedulers ignore it.

REGION
: is used to specify a region size for the job step or for an entire cataloged procedure. For a cataloged procedure, this parameter overrides all REGION parameters that may have been specified in the procedure.

REGION.procstep
: is used to specify a region size for a single step of a cataloged procedure. This parameter overrides, for the named step, any REGION parameter which is present. One parameter of this form can be written for each step in the procedure. For a discussion of the region size required for FORTRAN

jobs, see the section "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 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 discussion under "Operand Field" 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 library in which the module resides must be specified for the job. The JOBLIB statement concatenates this library with the system library.

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

Blank Name Field

If the name field is blank, the data set defined by the DD statement is concatenated with the data set defined in the preceding DD statement. In effect, these two data sets are combined into one data set. (Private libraries may also be concatenated with the library specified in the JOBLIB DD statement. Therefore, several libraries can be concatenated with the system library.)

Note: Data sets whose records are of different lengths and/or different formats cannot be concatenated.

OPERAND FIELD

For purposes of discussion, parameters for the DD statement are divided into six classes. Parameters are used to:

Specify unit record data sets.

Name	Operation	Operand[1]
//{ddname / procstep.ddname / JOBLIB[3]}[2]	DD	Positional Parameter [* / DUMMY / DATA][4] Keyword Parameters[5] [6] DDNAME=ddname [DSNAME={dsname / dsname(element) / *.ddname / *.stepname.ddname / *.stepname.procstep.ddname / &name / &name(element)}] [UNIT=(subparameter-list)] [DCB=(subparameter-list)] [VOLUME=(subparameter-list)] [SPACE=(subparameter-list) / SPLIT=(subparameter-list) / SUBALLOC=(subparameter-list)] [LABEL=(subparameter-list)] [DISP=(subparameter-list) / SYSOUT=A / SYSOUT=B / SYSOUT=(x[,program-name][,form-number])[7] [8]] [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 the * or DATA the positional parameter is specified, keyword parameters cannot be specified.
[5]If "subparameter-list" consists of only one subparameter and no leading comma (indicating the omission of a positional subparameter) is required, the delimiting parentheses may be omitted.
[6]If "subparameter-list" is omitted, the entire parameter must be omitted.
[7]This form of the parameter is used only with priority schedulers.
[8]If "program-name" and "form-number" are omitted, the delimiting parentheses can be omitted. If only the form number is given, the parentheses must be used and two commas must precede the form number.

Figure 8. Data Definition Statement

- Retrieve a previously created and cataloged data set.

  Retrieve a data set created in a previous job step in the current job and passed to the current job step.

  Retrieve a data set created but not cataloged in a previous job.

  Create data sets that reside on magnetic tape or direct access volumes.

- Optimize I/O operations.

The 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

are described in this section. (See Figure 9 for applicable parameters.)

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

Specifying Data in the Input Stream:

*
: indicates that a data set (e.g., a source module or data) immediately follows this DD statement in the input stream (see Figure 10). If the EXEC statement for the job step invokes a cataloged procedure, a data set may be placed in the input stream for each procedure step. If the EXEC statement specifies execution of a program, only one data set may be placed in the input stream. The DD * statement must be the last DD statement for the procedure step or program. The end of the data set must be indicated by a delimiter statement. The data cannot contain // or /* in the first two characters of the 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 the record.

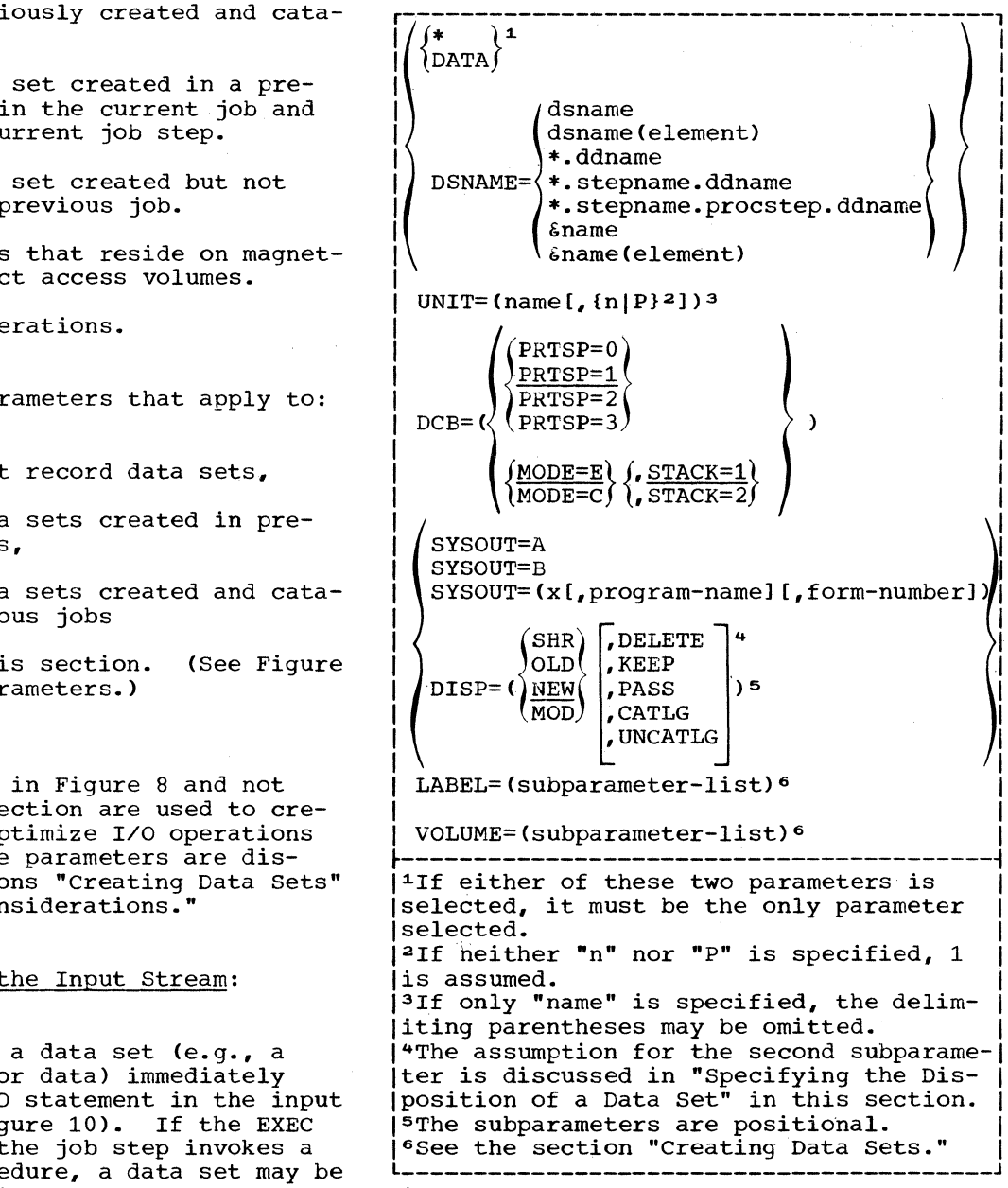


Figure 9. DD Statement

UNIT Parameter:

UNIT=(name[,{n|P}])
: specifies the name and number of I/O devices for a data set (see Figure 10). When the system is generated, the "name" is assigned by the operating system or the installation and represents a device address, a device type, or a device class. (See the System Generation publication.) The programmer can use only the assigned names in his DD statements. For example,

  UNIT=190, UNIT=2311, UNIT=TAPE

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

n|P
: specifies the number of devices allocated to the data set. If a number "n" is specified, the operating system assigns that number of devices to the data set. 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 by 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.

  The carriage control character in a FORTRAN record causes spacing before printing. The PRTSP subparameter causes spacing after printing.

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 or EBCDIC mode; C specifies column binary, and E specifies EBCDIC.

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

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

Sample Coding Form

1-10	11-20	21-30	31-40
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

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

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

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

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

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

Retrieving Previously Created Data Sets

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

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

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

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

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

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

Specifying a Cataloged Data Set by Name:

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

Specifying a Generation Data Group or PDS:

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

  DSNAME=FIRING(-2)

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

  Note: Members of a partitioned data set 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,DISP=(OLD,PASS)
//S1 EXEC PGM=ROCKET
//FT01F001 DD DSNAME=RATES(+1),DISP=OLD
//FT09F001 DD DSNAME=TIME,DISP=(OLD,PASS)
//S2 EXEC PGM=DISTANCE
//FT08F001 DD DSNAME=*.S1.FT09F001,        1
//            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"

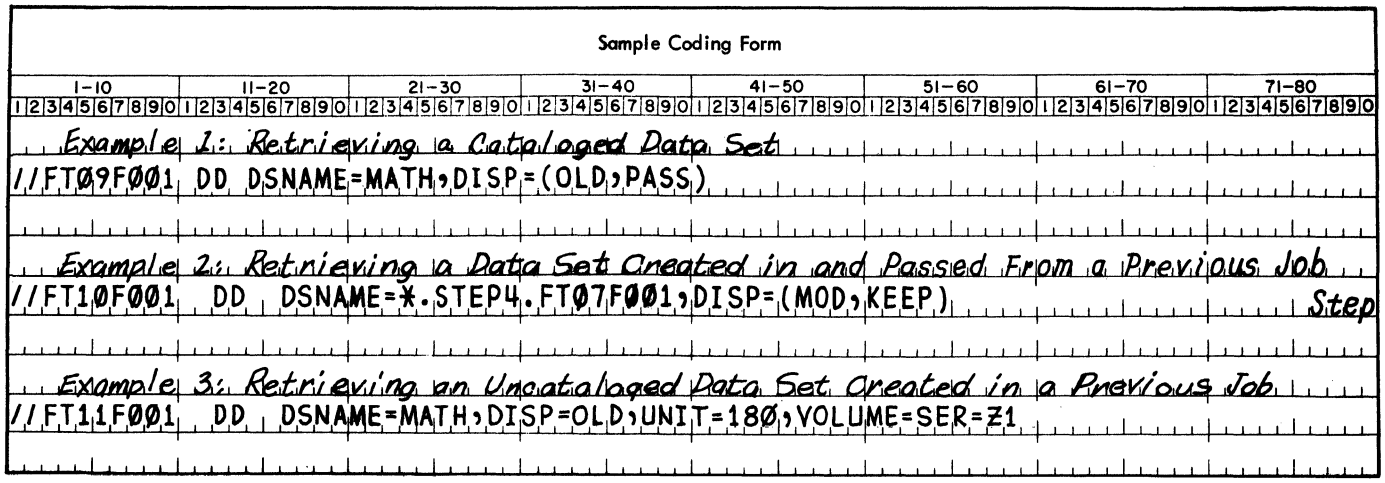


Figure 11. Retrieving Previously Created Data Sets

is the name of a previous job step that invoked the cataloged procedure; "procstep" is the name of a step in the cataloged procedure. The name of the data set is copied from the DSNAME parameter in the DD statement named "ddname".

Assigning Names to Temporary Data Sets:

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

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

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

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

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 of or during the job step.

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

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

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

MOD
: indicates that the data set was created in a previous job or job step, 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 following the last record. If MOD is specified and (1) the volume serial number is not given, and (2) the data set is not cataloged or passed, MOD is ignored and NEW is assumed.

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 parameter is specified in a subsequent job or job step. KEEP is used to retain uncataloged data sets for processing in future jobs. KEEP does not imply PASS.

PASS
indicates that the data set is referred to in a later job step. When a subsequent reference to the data set is made, its PASS status lapses unless another PASS is issued. The final disposition of the data set should be stated in the last job step that uses the data set. When a data set is in PASS status, the volume(s) on which it is mounted is retained. If 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 before this job, it exists after 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.

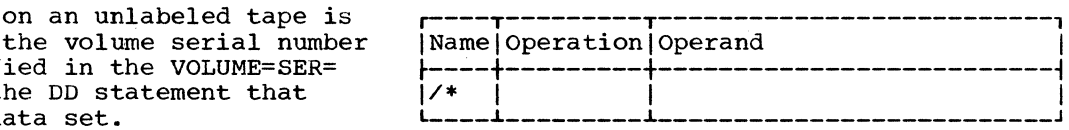


Figure 12. Delimiter Statement

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.

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

USING CATALOGED PROCEDURES

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

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

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

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

COMPILE

The cataloged procedure for compilation is FORTHC. This cataloged procedure consists of the control statements shown in Figure 43 in "Cataloged Procedures." If the EDIT option is specified, a SYSUT1 data set must be defined as a work data set for the compiler; if the compiler XREF option is specified, a SYSUT2 data set must be defined as a work data set.

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

Figure 13. Invoking the Cataloged Procedure FORTHC

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=FORTHC
//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=FORTHC
//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, control returns to the control program at the end of the compilation. If the next card is a FORTRAN statement, control remains with the FORTRAN compiler.

COMPILE AND LINK EDIT

The cataloged procedure to compile FORTRAN source modules and link edit the resulting object modules is FORTHCL. 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 FORTHCL.

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

Figure 16. Invoking the Cataloged Procedure FORTHCL

LINK EDIT AND EXECUTE

The cataloged procedure to link edit FORTRAN object modules and execute the resulting load module is FORTHLG. 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 FORTHLG.

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

Figure 17. Invoking the Cataloged Procedure FORTHLG

A sample deck structure to link edit and execute, as one load module, several object modules entered in the input stream is shown in Figure 18.

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

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

Figure 18. Link Edit and Execute

A sample deck structure is shown in Figure 19 to link edit and execute, as one load module, object modules that are members of the cataloged sequential data set, OBJMODS, which resides on a tape volume. In addition, a data set in the input stream is processed using the SYSIN data set.

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

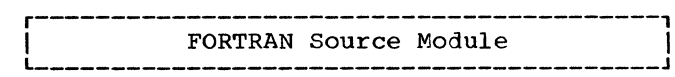
Figure 19. Link Edit and Execute Object Modules in a Cataloged Data Set

COMPILE, LINK EDIT, AND EXECUTE

The fourth cataloged procedure, FORTHCLG, 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 FORTHCLG.

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

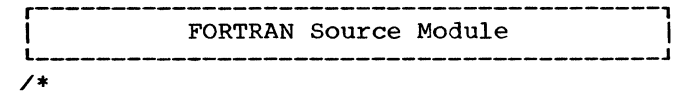



```
/*
```

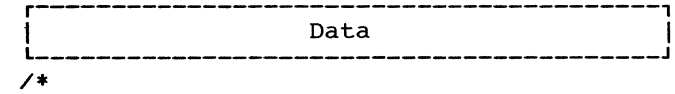
Figure 20. Invoking the Cataloged Procedure FORTHCLG

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 FORTHCLG
//FORT.SYSIN DD *
```




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

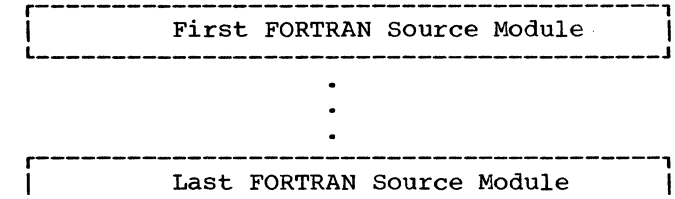



```
/*
```

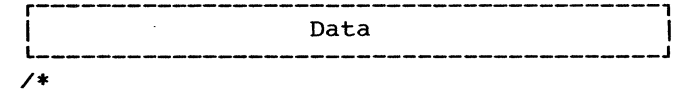
Figure 21. Single Compile, Link Edit, and Execute

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

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




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




```
/*
```

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

Compiler Name

The program name for the compiler is IEKAA00. If the compiler is to be executed without using the supplied cataloged procedures, an EXEC statement of the form

```
// EXEC PGM=IEKAA00
```

must be used. (For more information on procedures and options in coding IEKAA00, refer to Appendix A, "Invoking the FORTRAN Compiler.")

Compiler ddnames

The compiler can use seven data sets. To establish communication between the compiler and the programmer, each data set is assigned a specific ddname. Each data set has a specific function and device requirement. Table 2 lists the ddnames, functions, and device requirements for the data sets.

To compile a FORTRAN source module, two of these data sets are necessary -- SYSIN and SYSPRINT, along with the direct access volume(s) that contains the operating system. However, with these two data sets, only the source listing is generated by the compiler. If a structured source listing is to be generated, a SYSUT1 DD statement must be supplied. If a cross reference listing is to be generated by the compiler, a SYSUT2 DD statement must be supplied. If an object module is to be punched and/or written on a direct access or magnetic tape volume, a SYSLIN and/or SYSPUNCH DD statement must be supplied.

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

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

Table 2. Compiler ddnames

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

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

Compiler Device Classes

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

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

Compiler Data Set Assumptions

Standard assumptions are made for the DCB parameter of the data sets used by the compiler. Table 5 contains the values set for logical record length, record format, and blocksize. Of these, only the value for blocksize may be overridden with a DD statement.

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

Table 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 the card reader
SYSUT1	SYSSQ
SYSUT2	SYSSQ
SYSABEND or SYSUDUMP	A, SYSSQ
SYSPRINT	A, SYSSQ
SYSPUNCH	B, SYSCP[1], SYSSQ, SYSDA
SYSLIN	SYSSQ, SYSDA, SYSCP[1]

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

Table 5. DCB Assumptions for the Compiler Data Sets

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

[1]This value may be increased by overriding the present value in the associated DD statement.

[2]The value is within this range. The actual value is calculated during execution. The size of one of the tables used by the compiler (the address constant table) is compared with the track-size of the device specified by SYSUT2, and the LRECL and BLKSIZE fields are equated to the smaller value. If the BLKSIZE subparameter of the DCB parameter is specified on the SYSUT2 card, it must be an integral multiple of the value chosen by the compiler for LRECL.

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. Type of optimization, if any, desired by the programmer.
2. Name assigned to the program.
3. Number of lines per page for the source listing.
4. Whether an object module listing is printed.
5. Whether the source module is coded in Binary Coded Decimal (BCD) or Extended Binary Coded Decimal Interchange Code (EBCDIC).
6. Whether a list of source statements, with their associated internal statement numbers, is printed.
7. Whether an object module is punched.
8. Whether a table of names and a table of labels, which appear in the source module, are printed.
9. Whether the compiler writes the object module on external storage for input to the link editor.
10. Whether calls and function references are identified with an internal statement number.
11. Whether a structured source listing is printed.
12. Whether a cross reference listing is printed.

There is no specified order for compiler options in the PARM parameter.

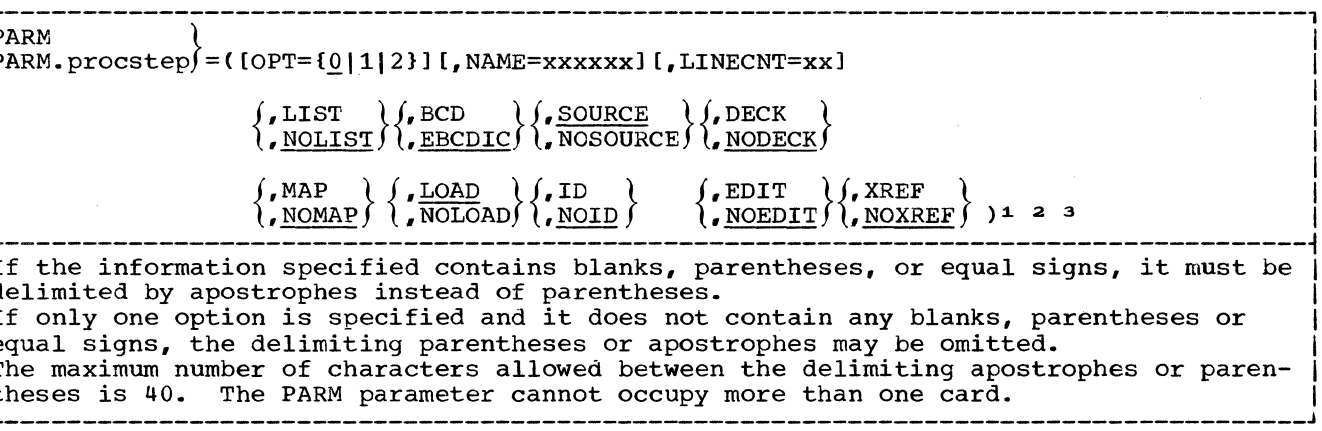


Figure 23. Compiler Options

OPT={0|1|2}

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

Name=xxxxxx

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

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

LINECNT=xx

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

LIST or NOLIST

The LIST option indicates that the object module listing is written in the data set specified by the SYSPRINT DD card. (The statements in the object module listing are in a pseudo 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."

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.

Notes:

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

   &n

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

   $n

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

   (The n represents the statement number.)

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

SOURCE or NOSOURCE

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

DECK or NODECK

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

MAP or NOMAP

The MAP option specifies that a table of names and a table of labels are written in the data set specified by the SYSPRINT DD statement. These tables include those names and labels which are generated by the compiler as well as those which appear in the source module. The NOMAP option specifies that no tables are written. A description of the tables is given in the section "System Output."

LOAD or NOLOAD

The LOAD option indicates that the object module is written in the data set specified by the SYSLIN DD statement. This option must be used if the cataloged procedure to compile and 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 the generated code is to contain an identifier for the compiler-assigned internal statement number associated with each function reference or CALL statement. When ID is specified, the internal statement numbers appear in the diagnostic traceback provided for execution-time errors. (A description of the traceback is given in the section "System Output.") The NOID option specifies the omission of the identifiers in the object program.

EDIT or NOEDIT

The EDIT option specifies that a structured source listing is written in the data set specified by the SYSPRINT DD statement. This listing indicates the loop structure and the logical continuity of the source program. If this option is used, OPT=2 must be specified and a DD statement with the name SYSUT1 must be supplied. The NOEDIT option specifies that no structured source listing is written. A description of the structured source listing is given in the section "System Output."

XREF or NOXREF

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

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

Multiple Compilation Within a Job Step

Several compilations may be performed within one job step. The compiler recognizes the FORTRAN END statement in a source module, compiles the program, and determines if another source module follows the END statement. If there is another source module, another compilation is initiated (see Figure 24).

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

Figure 24. Multiple Compilation Within a Job Step

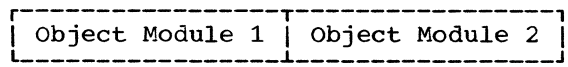
Only one EXEC statement may be used to initiate a job step; therefore, compiler options can be stated only once for all compilations in a job step.

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

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

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

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



LINKAGE EDITOR PROCESSING

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

Linkage Editor Name

The program name for the linkage editor is IEWL. If the linkage editor is executed without using cataloged procedures, an EXEC statement of the form

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

must be used.

Linkage Editor Input and Output

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

Primary input is a sequential data set that contains object modules and linkage editor control statements. (A member of a PDS cannot be the primary input.) Any external references among object modules in the primary input are resolved by the linkage editor as the primary input is processed. Furthermore, the primary input can contain references to the secondary input. These references are linkage editor control statements and/or external references in the FORTRAN modules.

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

An object module used as the main program in the load module being con-

structed. This object module, which can be accompanied by linkage editor control statements, is either a member of a PDS or is a sequential data set. The first record in the primary input data set must be a linkage editor INCLUDE control statement that tells the linkage editor to insert the main program.

- An object module or a load module used to resolve external references made in another module. The object module, which can be accompanied by linkage editor control statements, is a sequential data set or is a member of a PDS. The load module, which is a member of a PDS, cannot be accompanied by linkage editor control statements. An INCLUDE statement that defines the data set must be given.

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

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

The load module created by the linkage editor is always placed in a PDS. Error messages and optional diagnostic messages are written on intermediate storage or a printer. In addition, a work data set is required by the linkage editor to do its processing. Figure 25 shows the I/O flow in linkage editor processing.

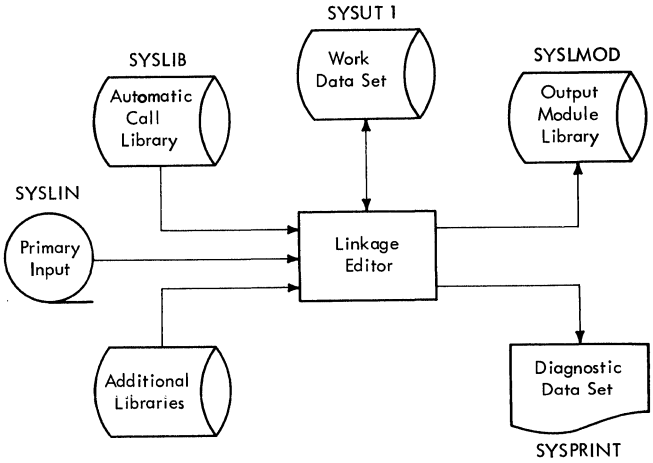


Figure 25. Linkage Editor Input and Output

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 6.

Any 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 retrieves any additional libraries is written in INCLUDE and LIBRARY statements and is not fixed by the linkage editor.

Table 6. Linkage Editor ddnames

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

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 7.

Table 7. Correspondence Between Linkage Editor ddnames and Device Classes

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

Additional Input

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

INCLUDE Statement:

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

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

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

LIBRARY Statement:

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

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

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

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

Figure 26. Linkage Editor Example

Linkage Editor Priority

If modules with the same name appear in the input to linkage editor, the linkage editor inserts only one of the modules. The following priority for modules is established by the linkage editor:

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 an LIBRARY statement and in the automatic call 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 encountered first is inserted in the output load module.

Other Linkage Editor Control Statements

In addition to the LIBRARY and INCLUDE statements, other control statements are available for use with the linkage editor. These statements enable the user to: specify different names for load modules (ALIAS), replace modules within a load module (REPLACE), change program names (CHANGE), and name entry points (ENTRY). In addition, two statements (OVERLAY and INSERT) enable the programmer to overlay load modules. For a detailed description of these control statements, see the Linkage Editor 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:

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

MAP or XREF

The MAP option informs linkage editor to produce a map of the load module; this map indicates the relative location and length of main programs and subprograms. If XREF is specified, a map of the load module is produced and a cross reference list indicating all external references in each main program and subprogram is generated. The map or map and cross reference list are written in the data set specified by the SYSPRINT DD statement. If neither option is specified, no map or cross reference listing is generated. Descriptions of the map and cross reference listing are given in "System Output."

LET

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

NCAL

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

LIST

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

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

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, and yyy is a FORTRAN sequence number.

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

If an installation specifies an upper limit of 99 for its data set reference numbers, the ddnames and data set reference numbers correspond as shown in Table 8. Note that 0 is not a valid data set reference number.

Table 8. 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                                  ddnames

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

Reference Numbers for Data Sets Specified in DEFINE FILE Statements

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

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

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

establishes a data set reference number of 02. All subsequent input/output statements must refer to only one data set with the DD name of FT02F001. (For a more detailed explanation of the DEFINE FILE statement, refer to the FORTRAN IV Language publication.)

If the preceding instructions are used to write a tape, the output tape (unlabeled) has the appearance shown in Figure 27.

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."

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

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,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
//    DISP=(,CATLG),VOLUME=REF=*.FT17F001
//FT17F003 DD DSNAME=N3,LABEL=(3,NL),      X
//    DISP=(,CATLG),VOLUME=REF=*.FT17F002
```

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

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

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

Assume the data sets shown in Figure 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 may be issued. A subsequent READ using the same data set reference number resets the FORTRAN sequence number to 001. When the last

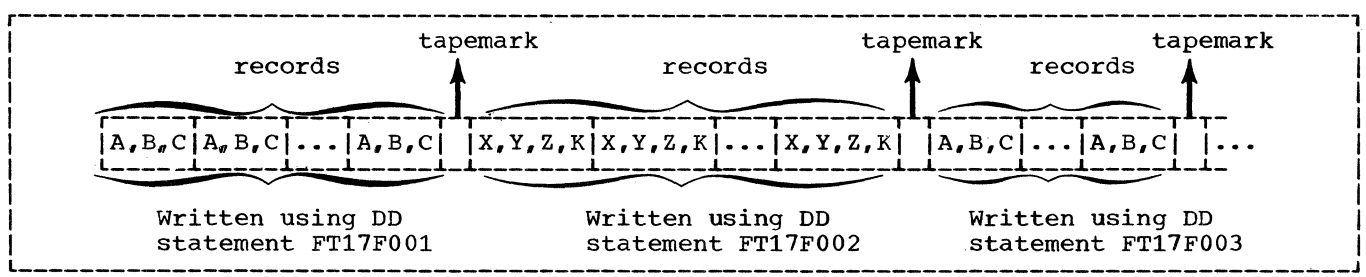


Figure 27. Tape Output for Several Data Sets Using Same Data Set Reference Number

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 REWIND 17
```

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

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."

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. If the error message data set is on tape, the DD statement should contain DCB parameters for BLKSIZE=133 and RECFM=UA. (The publication IBM System/360 Operating System: System Generation, Form C28-6554, explains the method of assigning the data set reference number. See the description of the OBJERR parameter in the section on the FORTLIB macro instruction.)

Execution Device Classes

For load module execution, the programmer can use the same names assigned to device classes used by the compiler (shown in Table 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 "Program Optimization."

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

DSNAME - name of the data set

UNIT - class 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 28.

Sample Coding Form

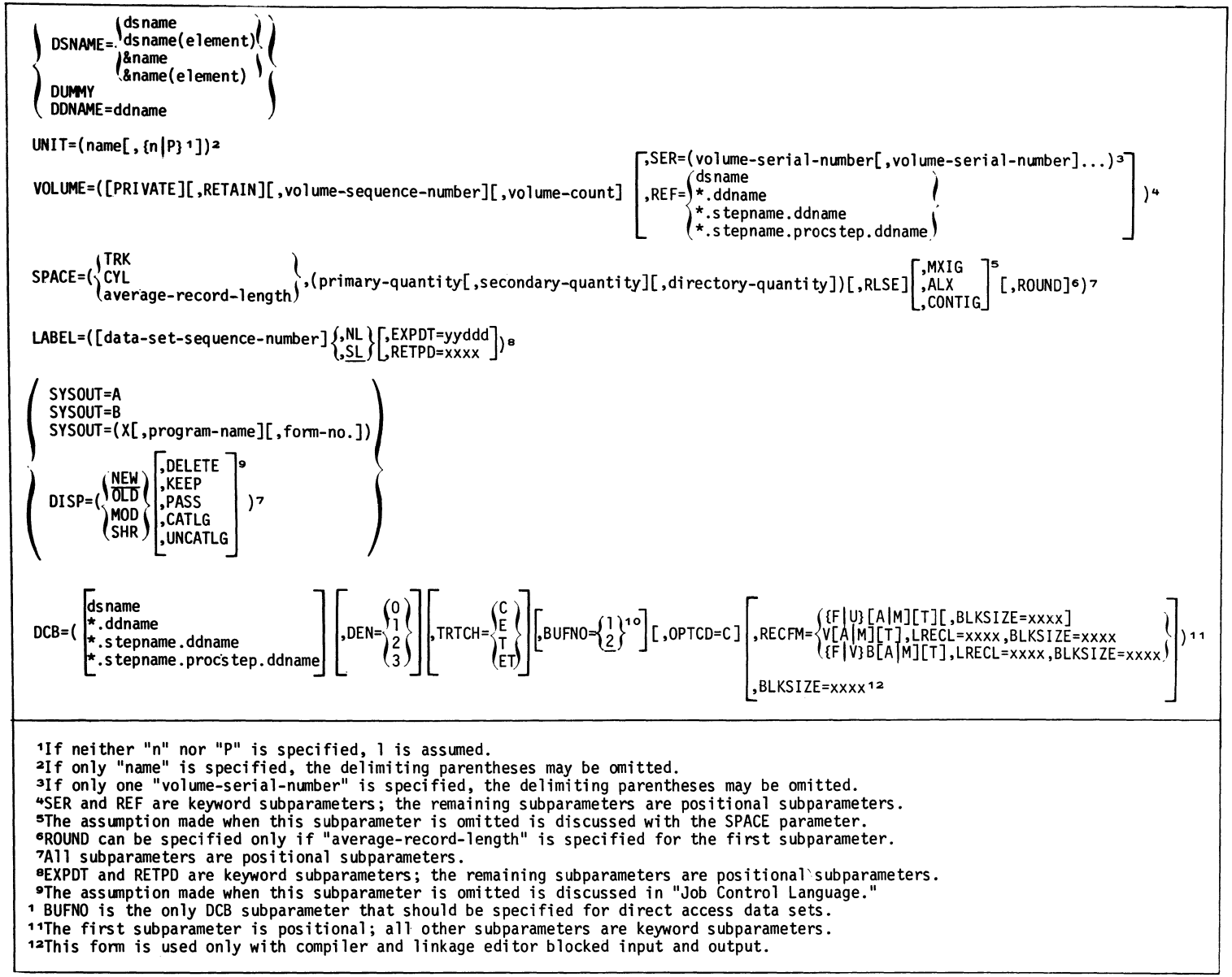
```
1-10      11-20     21-30     31-40     41-50     51-60     61-70     71-80
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
 Example 1: Creating a Cataloged Data Set
//FT31F001 DD DSNAME=MATRIX,DISP=(NEW,CATLG),LABEL=(,SL,EXPDT=67031),        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

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

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

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

Figure 28. Examples of DD Statements



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

Figure 29. DD Parameters for Creating Data Sets

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 29). The DUMMY parameter may not be used with FORTRAN defined direct access data sets 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 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,(50,30),,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.

DDNAME=ddname

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

SPECIFYING I/O DEVICES

The name and number of I/O devices are specified in the UNIT parameter,

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

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

n|P
specifies the number of devices allocated to the data set. If a number, "n" is specified, the operating system assigns that number of devices to the data set. 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 by 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]...)
        ,REF={dsname
              *.ddname
              *.stepname.ddname
              *.stepname.procstep.ddname}])
```

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

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

RETAIN
indicates that this volume is to remain mounted after the job step is completed. Volumes are retained so that data may be transmitted to or from the data set, or so that other data sets may reside on the volume. If the data set requires more than one volume, only the last volume is retained; the other volumes are 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 characters, the serial number is left adjusted and padded with blanks. If SER is not specified, and DISP is not specified as NEW, the data set is assumed to be cataloged and serial numbers are retrieved from the catalog, or inherited from passed data sets in a previous step. A volume serial number is not required for new output data sets.

REF indicates that the data set is to occupy the same volume(s) as the data set identified by "dsname", "*.ddname", "*.stepname.ddname", or *.stepname.procstep.ddname. Table 9 shows the data set references.

Table 9. Data Set References

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

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.

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

SPECIFYING SPACE ON DIRECT-ACCESS VOLUMES

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

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

       ,(primary-quantity

       [,secondary-quantity])

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

The SPACE parameter specifies:

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

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

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

  For example, by specifying:

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

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

  By specifying:

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

  20 cylinders are allocated to the data set. When previously allocated space is exhausted, two additional cylinders are allocated.

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

RLSE
: indicates that all unused external storage assigned to a NEW or MOD data set is released when processing of the data set is completed.

[MXIG / ALX / CONTIG]
: specify the format of the space allocated to the data set, as requested in the "primary quantity".

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

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

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

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

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

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

LABEL INFORMATION

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

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

      [,EXPDT=yyddd]
      [,RETPD=xxxx ])
```

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 identified by data set sequence number 3.) If the data set sequence number is not specified, the operating system assumes 1.

{NL}
{SL}
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.

DISPOSITION OF A DATA SET

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

WRITING A UNIT RECORD DATA SET ON AN INTERMEDIATE DEVICE

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

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

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

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

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

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

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

posed of variable-length records or contains binary information.

DCB PARAMETER

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

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

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

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

┌      ┌{F|U}[A|M][T][,BLKSIZE=xxxx]          ┐
│,RECFM=│ V[A|M][T],LRECL=xxxx,BLKSIZE=xxxx,  │
└      └{F|V}B[A|M][T],LRECL=xxxx,            ┘
         BLKSIZE=xxxx

 ,BLKSIZE=xxxx
```

REFERRING TO PREVIOUSLY SPECIFIED DCB INFORMATION

The first subparameter

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

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

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

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

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

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

DENSITY AND CONVERSION

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

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

DEN={0|1|2|3}
indicates the density used to write a data set (see Table 10).

Table 10. DEN Values

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

If DEN is not specified, the lowest applicable density is assumed.

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

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

C - data conversion feature is used

E - even parity is used

T - translation from BCD to EBCDIC is required

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

NUMBER OF BUFFERS FOR SEQUENTIAL DATA SETS

The number of buffers required for any application is specified by

BUFNO=x

where:

x=1 or 2

CHAINED SCHEDULING

Chained scheduling may be requested by specifying OPTCD=C as a DCB subparameter in the DD statement. Although chained scheduling is not used for direct-access I/O itself, it does produce faster formatting of direct-access data sets. It is this direct-access application of chained scheduling that is its most important FORTRAN function, since use of chained scheduling for sequential data sets does not appreciably reduce FORTRAN execution time. Note, too that when chained scheduling is specified, the system makes use of about 2K additional bytes of main storage to provide the feature.

RECORD FORMAT

```
RECFM=U[A|M][T]
RECFM=V[B][A|M][T]
RECFM=F[B][A|M][T]
```

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 USA carriage control characters. (See Appendix E); the character M indicates the use of machine code control characters.

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

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

RECFM=UT
: Formatted Sequential I/O

RECFM=VT
: Formatted or Unformatted Sequential I/O

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

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

RECORD LENGTH, BUFFER LENGTH, AND BLOCK LENGTH

For unblocked, fixed-length or undefined records, the record length and the buffer length are specified by

BLKSIZE=xxxx (See Table 11.)

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

LRECL=xxxx (1≤xxxx≤3624);

buffer length is specified by

BLKSIZE=xxxx (See Table 11.)

For all blocked records, the record length is specified by

LRECL=xxxx (1≤xxxx≤3624);

block length and buffer length are specified by

BLKSIZE=xxxx (See Table 11.)

Table 11. Load Module DCB Parameter Default Values

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

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

FORTRAN Records and Logical Records

In FORTRAN, records for sequential data sets are defined by specifications in FORMAT statements and by READ/WRITE lists. A record defined by a specification in a FORMAT statement is a FORTRAN record (see the FORTRAN IV Language publication). 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
```

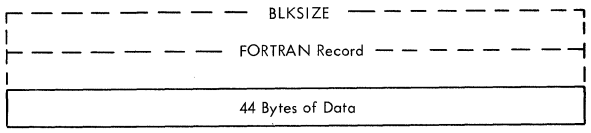


Figure 30. FORTRAN Record (FORMAT Control) Fixed-Length Specification

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
```

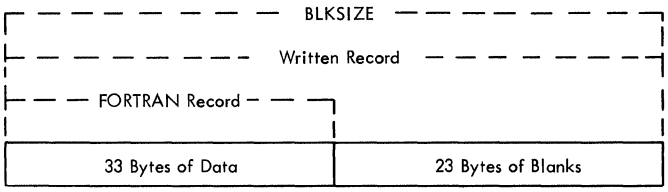


Figure 31. FORTRAN Record (FORMAT Control) Fixed-Length Specification and FORTRAN Record Length Less Than BLKSIZE

Variable-Length Records: For unblocked variable-length records written under FORMAT control, LRECL is specified as four greater than the maximum FORTRAN record length; and BLKSIZE as four greater than LRECL. These extra eight bytes are required for the four-byte block control word (BCW) and the four-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.

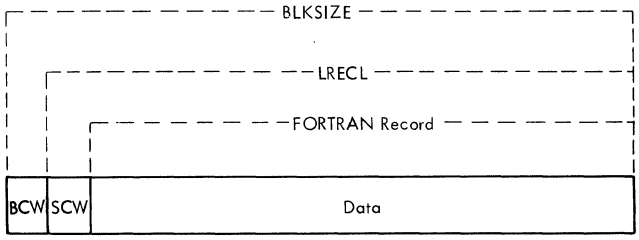


Figure 32. FORTRAN Record (FORMAT Control) Variable-Length Specification

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

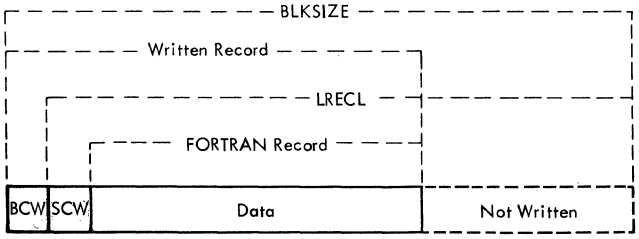


Figure 33. FORTRAN Record (FORMAT Control) With Variable-Length Specification and the FORTRAN Record Length Less Than (LRECL-4)

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).

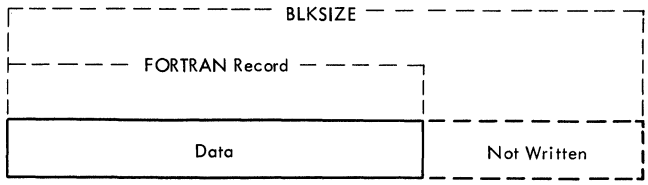


Figure 34. FORTRAN Record (FORMAT Control) With Undefined Specification and the FORTRAN Record Length Less Than BLKSIZE

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

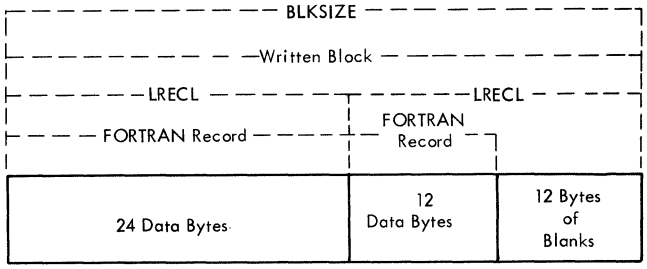


Figure 35. Fixed-Length Blocked Records Written Under FORMAT Control

Variable-Length Records: For blocked variable-length records written under FORMAT control, LRECL is specified as four greater than the maximum FORTRAN record length, and BLKSIZE must be at least four greater than LRECL. The four bytes are

required for the block control word (BCW) that contains the block length. The data length must be equal to or less than (LRECL-4). These four bytes are used for the segment control word (SCW) that contains the record-length indicator.

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. However, if the space remaining in a buffer is greater than LRECL, the buffer is not written, but held for the next WRITE (see Figure 36). If another WRITE is not executed 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
```

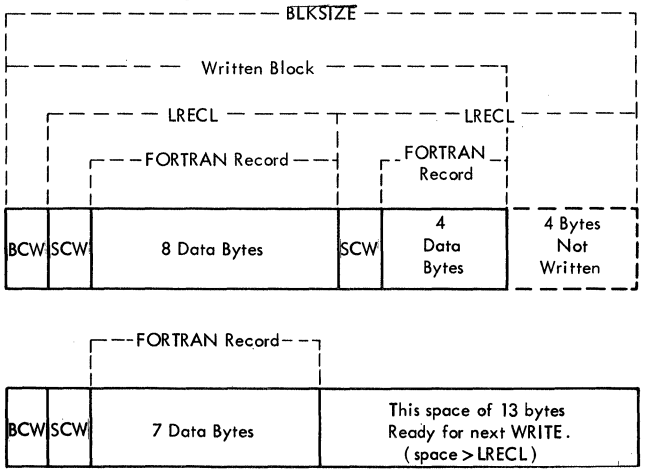


Figure 36. Variable-Length Blocked Records Written Under FORMAT Control

NO FORMAT CONTROL: Only variable-length records can be written without format control; i.e., the RECFM subparameter must be V. (If nothing is specified, V is assumed.)

Records written with no FORMAT control have the following properties:

The length of the logical record is controlled by the type and number of variables in the input/output list of its associated READ or WRITE statement.

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

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 definition (DD) statement. If not specified, 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. Each buffer begins with a four-byte block control word (BCW). The SCWs and BCWs are provided by the system.

The format of a BCW is given in Figure 37.

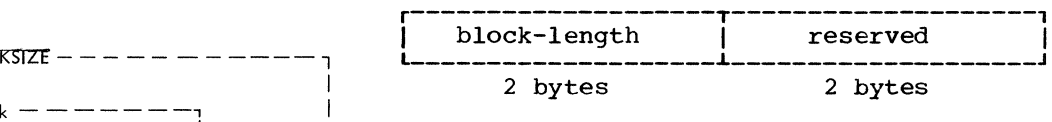


Figure 37. Format of a Block Control Word

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.

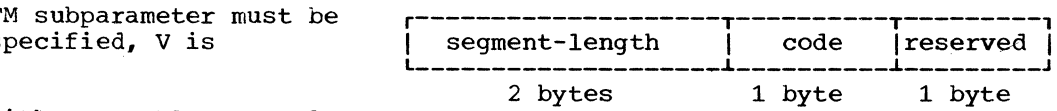


Figure 38. Format of a Segment Control Word

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 multisegment record.
10	This segment is the last of a multisegment record.
11	This segment is neither the first nor last of a multisegment 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 each READ or WRITE statement issued, 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 each READ or WRITE statement issued, 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

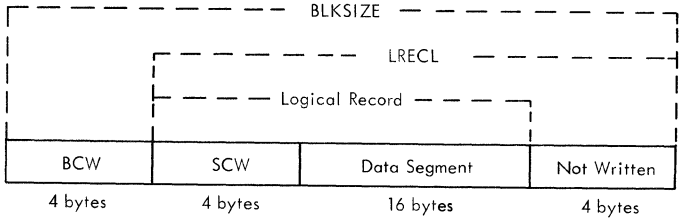
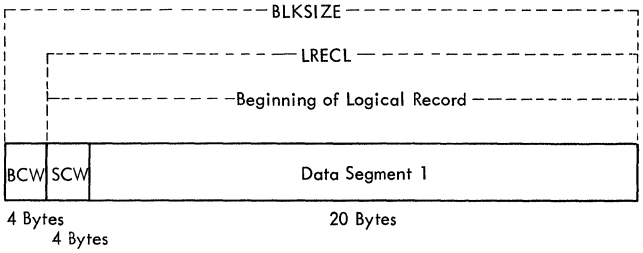


Figure 39. Variable-Length Unblocked Records, No FORMAT Control, One Record Segment

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



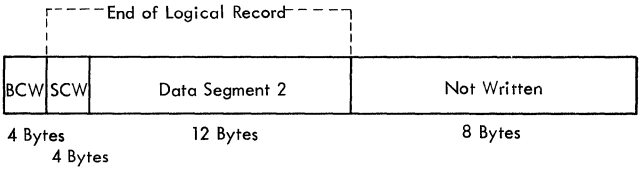


Figure 40. Variable-Length Unblocked Records, No FORMAT Control, Two Record Segments

Blocked Records: For blocked records written without FORMAT control, each block is composed of M record segments, where: M = BLKSIZE-4/LRECL.

If the logical record length is less than or equal to LRECL-4, the logical record comprises one record segment. Hence, for each M READ or WRITE statements issued, one block, i.e., M record segments, are 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 each READ or WRITE statement issued, N record segments (i.e., as many blocks of M segments each as are needed to make up N segments) are transmitted. The unused portion of the last block is held for the next READ or WRITE (see Figure 41).

Example: Assume BLKSIZE=28 and LRECL=12

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

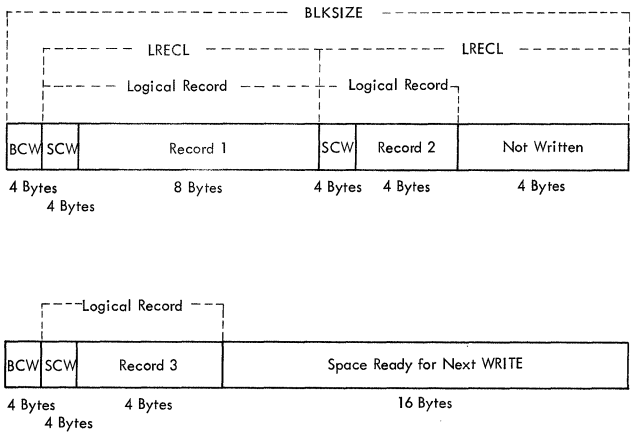


Figure 41. Variable-length, Blocked Records, No FORMAT Control

where: A is a real *8 variable B and E are real *4 variables

BACKSPACE Operations

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

Blocked Records: The programmer is cautioned against backspacing blocked records; the results obtained are unpredictable.

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

Record Length, Buffer Length, and Number of Buffers for Direct Access Data Sets

A direct access data set can contain only fixed-length, unblocked records. Any 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 subparameter that can be supplied for direct access data sets is the number of buffers:

```
BUFNO=x     (x=1 or x=2)
```

Where: x is the number 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).

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

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

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

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

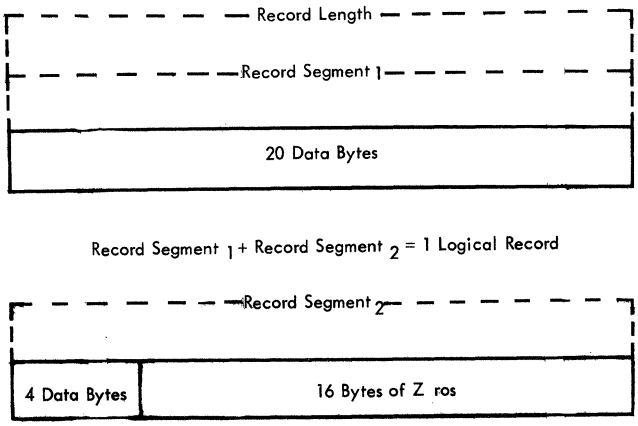


Figure 42. Logical Record (No FORMAT Control) for Direct Access

DCB ASSUMPTIONS FOR LOAD MODULE EXECUTION

The range of values that may be specified for BLKSIZE is established for specific data set reference numbers. If the DCB parameter is not specified, default values are assumed for BLKSIZE and RECFM (see Table 11).

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

Compile

In the three cataloged procedures that have a compile step (see Figures 43, 44, and 46), the EXEC statement named FORT indicates that the operating system is to execute the program IEKAA00 (the FORTRAN IV H compiler).

The REGION parameter is ignored by sequential schedulers. 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 required by the FORTRAN H compiler depends upon the SIZE parameter specified during system generation and on the buffer requirements for the compiler data sets. The SIZE parameter is specified in the FORTRAN macro instruction. The buffer requirements must be determined for each compiler data set, totaled, and rounded to the next highest multiple of 2K (where K = 1024 bytes). (The buffer requirement for a data set is equal to the blocksize multiplied by the number of buffers.) Another 10K for system use must be added to the sum of the SIZE parameter and the buffer requirements.

The region size of 228K in the cataloged procedure assumes the default value for the SIZE parameter and the buffer requirements as given in the procedure.

Note: If different region sizes are to be specified for each step in the job, the REGION parameter should be coded in the EXEC statement associated with each step instead of the JOB statement.

Compiler options are not explicitly specified; default options are assumed -- in particular, SOURCE and LOAD. The source listing and compile-time information and error messages are written in the SYSOUT data set.

The object module is written in the temporary data set &LOADSET. The data set &LOADSET is a sequential data set and is in "pass" status; records can be added to the data set.

The SYSOUT=B parameter on the SYSPUNCH DD statement is interpreted by sequential schedulers as a specification for the system card punch unit. The priority schedulers route the output data set to system output class B. A programmer can get an object module card deck by overriding the default NODECK option with an explicit DECK option.

Several additional DD statements, external to the procedure, may be supplied. If the EDIT option is used, a work data set must be defined with a FORT.SYSUT1 DD statement. If the compiler XREF option is specified, a work data set must be defined with a FORT.SYSUT2 DD statement. Input to the compile step is defined by a FORT.SYSIN DD statement.

Link Edit

In the three cataloged procedures that have a link edit step (see Figures 44, 45, and 46), the EXEC statement named LKED indicates that the operating system is to execute the program IEWL (the linkage editor). The linkage editor requires a region of 54K if used with MVT. The linkage editor step (or the remainder of a procedure) is not executed if a condition code greater than 4 was generated by a compile step in the same procedure.

If the link edit step is executed, a list of linkage editor control statements (in card image format), a map of the load module and a list of linkage editor diagnostic messages are written in the SYSOUT data set. The load module is marked executable even though error conditions are found during linkage editor processing.

If the link edit step is preceded by a compile step (see Figures 44 and 46), the primary input to the linkage editor may consist of concatenated data sets. The first, defined by the SYSLIN DD statement, is the output of the compiler (&LOADSET data set); the second (if present) is the data set defined by a LKED.SYSIN DD statement (external to the procedure). However, if the link edit step is the first step in

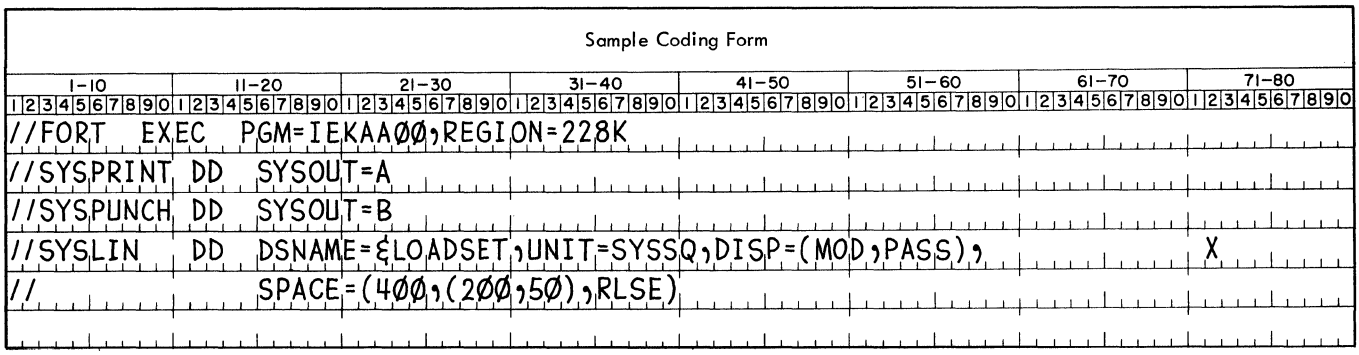


Figure 43. Compile Cataloged Procedure (FORTHC)

a procedure (see Figure 45), the primary input is the data set defined by a LKED.DD statement.

External references made in a FORTRAN object module are resolved by the linkage editor. Some or all of these references can be resolved from the FORTRAN library (SYS1.FORTLIB) which is a system resident PDS.

During processing, the linkage editor requires a work data set which is defined by the SYSUT1 DD statement. This data set is assigned to a direct access device.

The load module produced by the linkage editor is written in the temporary PDS &GOSET with a member name of MAIN. The data set is in "pass" status and is assigned to a direct access device.

Execute

In the two cataloged procedures that have an execute step (see Figures 45 and 46), the EXEC statement named GO indicates that the operating system is to execute the load module (program) produced in a preceding link edit step in the same procedure. However, the execute step is bypassed if a condition code greater than 4 was generated by a compile or link edit step in the same procedure.

Input to the execute step is defined by a GO.SYSIN DD statement (external to the procedure) and is read using data set reference number 5. Execution-time error messages and information for traceback and FORTRAN dumps are written in the SYSOUT data set that is associated with data set reference number 6. (Output from the load module can also be written in the same data set.) The card punch is associated with data set reference number 7.

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.

A list of the execution time routines required for various input/output, interruption, and error procedures is contained in the FORTRAN IV Library Subprograms publication. 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.

Sample Coding Form

```
1-10      11-20     21-30     31-40     41-50     51-60     61-70     71-80
//FORT    EXEC   PGM=IEKAAØØ,REGION=228K
//SYSPRINT DD    SYSOUT=A
//SYSPUNCH DD    SYSOUT=B
//SYSLIN   DD    DSNAME=&LOADSET,UNIT=SYSSQ,DISP=(MOD,PASS),                X
//               SPACE=(4ØØ,(2ØØ,5Ø),RLSE)
//LKED    EXEC   PGM=IEWL,REGION=54K,PARM=(MAP,LET,LIST),COND=(4,LT,FORT)
//SYSLIB   DD    DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSPRINT DD    SYSOUT=A
//SYSLMOD  DD    DSNAME=&GOSET(MAIN),UNIT=SYSDA,DISP=(,PASS),               X
//               SPACE=(3Ø72,(3Ø,1Ø,1),RLSE)
//SYSLIN   DD    DSNAME=&LOADSET,DISP=(OLD,DELETE)
//               DD    DDNAME=SYSIN
//SYSUT1   DD    UNIT=SYSDA,SPACE=(1Ø24,(2ØØ,2Ø)),SEP=SYSLMOD
```

Figure 44. Compile and Link Edit Cataloged Procedure (FORTHCL)

```
//EXAMPLE1 JOB ACCOUNT1,'JOHNSMITH',       X
//             MSGLEVEL=1
//  EXEC FORTHCLG,PARM.FORT=DECK,           X
//             REGION.GO=200K
//FORT.SYSIN DD  *
┌-----------------------------------------┐
|     FORTRAN Source Symbolic Decks       |
└-----------------------------------------┘
/*
//LKED.SYSIN DD  *
┌-----------------------------------------┐
|    Previously Compiled or Assembled     |
|              Object Decks               |
└-----------------------------------------┘
/*
//GO.SYSIN DD  *
┌-----------------------------------------┐
|               Input Data                |
└-----------------------------------------┘
/*
```

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 H define logical unit 05 as SYSIN and 06 as SYSOUT. (See Figures 45 and 46.) If, during system generation, values other than 05 for the ONLNRD parameter and 06 for the OBJERR parameter were specified in the FORTLIB macro instruction, one or both of the following DD cards must be added to the cataloged procedures, either at execution time or permanently.

- For the unit specified as ONLNRD, use the DD card:
  ```
  //GO.FTxxF001 DD DDNAME=SYSIN,
    DCB=(BLKSIZE=80,RECFM=F)
  ```

- For the unit specified as OBJERR, use the DD card:
  ```
  //GO.FTxxF001 DD SYSOUT=A
  ```

where xx is the unit specified. (The System Generation publication describes the FORTLIB macro instruction.)

In addition, the DD card for FT05F001 must be deleted permanently from the procedure. The following section describes the general procedure for adding and deleting cards from cataloged procedures.

OVERRIDING CATALOGED PROCEDURES

Cataloged procedures are composed of EXEC and DD statements. A feature of the operating system is its ability to read control statements and modify a cataloged procedure for the duration of the current job. Overriding is only temporary; that is, the parameters added or modified are in effect only for the duration of the job. The following text discusses the techniques used to modify cataloged procedures.

Sample Coding Form

```
//LKED   EXEC  PGM=IEWL,REGION=54K,PARM=(MAP,LET,LIST)
//SYSLIB   DD  DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSPRINT DD  SYSOUT=A
//SYSLIN   DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&GOSET(MAIN),UNIT=SYSDA,DISP=(,PASS),                X
//            SPACE=(3072,(30,10,1),RLSE)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(1024,(200,20)),SEP=SYSLMOD
//GO     EXEC  PGM=*.LKED.SYSLMOD,COND=(4,LT,LKED)
//FT05F001 DD  DDNAME=SYSIN
//FT06F001 DD  SYSOUT=A
//FT07F001 DD  SYSOUT=B
```

Figure 45. Link Edit and Execute Cataloged Procedure (FORTHLG)

Sample Coding Form

```
//FORT   EXEC  PGM=IEKAA00,REGION=228K
//SYSPRINT DD  SYSOUT=A
//SYSPUNCH DD  SYSOUT=B
//SYSLIN   DD  DSNAME=&LOADSET,UNIT=SYSSQ,DISP=(MOD,PASS),                 X
//            SPACE=(400,(200,50),RLSE)
//LKED   EXEC  PGM=IEWL,REGION=54K,PARM=(MAP,LET,LIST),COND=(4,LT,FORT)
//SYSLIB   DD  DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSPRINT DD  SYSOUT=A
//SYSLMOD  DD  DSNAME=&GOSET(MAIN),UNIT=SYSDA,DISP=(,PASS),                X
//            SPACE=(3072,(30,10,1),RLSE)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(1024,(200,20)),SEP=SYSLMOD
//SYSLIN   DD  DSNAME=&LOADSET(MAIN),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 (FOTHCLG)

Overriding Parameters in the EXEC Statement

Two forms of keyword parameters ("keyword" and "keyword.procstep") are discussed in "Job Control Language." The form "keyword.procstep" is used to add or override parameters in an EXEC statement in a cataloged procedure.

The FORTRAN programmer can, for example, add (or override) compiler or linkage editor options for an execution of a cataloged procedure, or he can state different conditions for bypassing a job step.

Note: When the PARM parameter is overridden, all compiler and/or linkage editor options stated in the EXEC statement in the procedure step are deleted and replaced by those in the overriding PARM parameter.

Example 1: Assume the cataloged procedure FORTHC is used to compile a program, and the programmer wants to specify the name of his program and the MAP option. The following statement can be used to invoke the procedure, and to supply the compiler options.

```
//STEP1 EXEC FORTHC,                              X
//     PARM.FORT='MAP,NAME=MYPROG'
```

The PARM options apply to the procedure step FORT.

Example 2: Assume the cataloged procedure FORTHLG is used to link edit and execute a module. Furthermore, the XREF option overrides MAP, LET, and LIST in the linkage editor step and the COND parameter is changed for the execution of the load module. The following EXEC statement adds and overrides parameters in the procedure.

```
//DO EXEC FORTHLG,PARM.LKED=XREF,                 X
//     COND.GO=(3,LT,DO.LKED)
```

The PARM parameter applies to the linkage editor procedure step LKED, and the COND parameter applies to the execution procedure step GO.

Example 3: Assume a source module is compiled, link edited, and executed using the cataloged procedure FORTHCLG. Furthermore, the compiler option OPT and the linkage editor option XREF are specified, and account number 506 is used for the execution procedure step. The following EXEC statement adds and overrides parameters in the procedure.

```
//STEP1 EXEC FORTHCLG,                            X
//     PARM.FORT='OPT=2',                         X
//     PARM.LKED=XREF,                            X
//     ACCT.GO=506
```

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.

Note: The following additional rules apply to overriding in cataloged procedures:

1. In the DCB parameter, individual subparameters can be overridden.

2. To nullify the use of any particular keyword parameter (except the DCB parameter), the overriding DD statement must specify

   keyword=,

3. A parameter can be overridden by specifying a mutually exclusive parameter in the overriding DD statement. For example, in the FORTHC procedure, the SPACE specification for SYSLIN may be overridden by using either the SPLIT or SUBALLOC parameter.

When the procedures FORTHC, FORTHCL, and FORTHCLG 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 FORTHLG is used, a DD statement must be added to define the SYSLIN data set (see Figures 17 and 18).

When the procedures FORTHCL, FORTHLG, and FORTHCLG are used, an overriding DD statement can be used to write the load module constructed in the linkage editor step in a particular PDS chosen by the programmer, and assign that member of the PDS a particular name.

In execution procedure steps, the programmer can catalog data sets, assign names to data sets, supply DCB information for data sets, add data sets, or specify particular volumes for data sets by using overriding and/or additional DD statements.

Example 1: Assume the data sets identified by ddnames FT04F001 and FT08F001 are named, cataloged, and assigned specific volumes. The following DD statements are used to add this information and indicate the location of the source module.

```
//JOB1 JOB MSGLEVEL=1
//STEP1 EXEC FORTHCLG
//FORT.SYSIN DD *
```

```
┌-----------------------------------------┐
|          FORTRAN Source Module          |
└-----------------------------------------┘
```

```
/*
//GO.FT04F001 DD DSNAME=MATRIX,          X
//   DISP=(NEW,CATLG),UNIT=TAPE,         X
//   VOLUME=SER=987K
//GO.FT08F001 DD DSNAME=INVERT,          X
//   DISP=(NEW,CATLG),UNIT=TAPE,         X
//   VOLUME=SER=1020
//GO.SYSIN DD  *
```

```
┌-----------------------------------------┐
|          Input to Load Module           |
└-----------------------------------------┘
```

```
/*
```

Example 2: Assume the link edit and execute cataloged procedure (FORTHLG) is used. The load module constructed in the linkage editor step is placed in the cataloged partitioned data set MATH and is assigned the member name DERIV.

```
//JOB3 JOB
//STEP1 EXEC FORTHLG
//LKED.SYSLMOD DD DSNAME=MATH(DERIV),    X
//   DISP=(MOD,PASS)
//LKED.SYSIN DD *
```

```
┌-----------------------------------------┐
|          FORTRAN Object Module          |
└-----------------------------------------┘
```

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

```
┌-----------------------------------------┐
|          Input to Load Module           |
└-----------------------------------------┘
```

```
/*
```

Example 3: Assume the compile, link edit, and execute cataloged procedure (FORTHCLG) is used with three data sets in the input stream:

1. A FORTRAN main program MAIN with a series of subprograms, SUB1 through SUBN.

2. A linkage editor control statement that specifies an additional library, MYLIB. MYLIB is used to resolve external references for the symbols ALPHA, BETA, and GAMMA.

3. A data set used by the load module and identified by data set reference number 5 in the source module.

The following example shows the deck structure.

```
//JOBCLG JOB 00,FORTRANPROG,MSGLEVEL=1
//HXECCLGX EXEC FORTHCLG
//FORT.SYSIN DD *
```

```
┌-----------------------------------------┐
|       FORTRAN Source Module MAIN        |
├-----------------------------------------┤
|       FORTRAN Source Module SUB1        |
├-----------------------------------------┤
|                   .                     |
|                   .                     |
|                   .                     |
├-----------------------------------------┤
|       FORTRAN Source Module SUBN        |
└-----------------------------------------┘
```

```
/*
//LKED.ADDLIB DD DSNAME=MYLIB
//LKED.SYSIN DD *
  LIBRARY ADDLIB(ALPHA,BETA,GAMMA)
/*
//GO.SYSIN DD *
```

```
┌-----------------------------------------┐
|       Input to Load Module              |
└-----------------------------------------┘
```

```
/*
```

The DD statement FORT.SYSIN indicates to the compiler that the source modules are in the input stream. The DD statement LKED.ADDLIB defines the additional library MYLIB to the linkage editor. The DD statement LKED.SYSIN defines a data set that is concatenated with the primary input to the linkage editor. The linkage editor control statements and the object modules appear as one data set to the linkage editor. The DD statement GO.SYSIN defines data in the input stream for the load module.

This section discusses 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 12 for storage allocation according to the type and length of the constant or variable.

Table 12. 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 operating system is device independent. In particular, the FORTRAN compiler can operate with any combination of devices (shown in Table 3); however, there are certain requirements.

The FORTRAN IV compiler requires at least a System/360, Model 40 with 256K bytes of storage and the standard instruction set with the floating-point option.

All programs require a device, such as, the 1052 keyboard printer, for direct operator communication.

At least one direct access device must be used for residence of the operating system.

- The printer must have at least a 132 character print line.

PROGRAM OPTIMIZATION

Facilities are available in the FORTRAN compiler that enable a programmer to optimize execution speed and to reduce the size of the object module. However, programs that are compiled using the IBM-supplied cataloged procedures are not optimized; OPT=0 is the default option. A programmer must override this default option with either OPT=1 or OPT=2 to specify the use of the optimization facilities. (See "Cataloged Procedures" for overriding parameters in the EXEC statement.)

When using OPT=1, the entire program is a loop, while individual sections of coding, headed and terminated by labeled statements, are blocks. The object code is improved by:

- Improving local register assignment. (Variables that are defined and used in a block are retained (if possible) in registers during the processing of the block. Time is saved because the number of load and store instructions are reduced.)

- Retaining the most active base addresses and variables in registers across the whole program. (Retention in registers saves time because the number of load instructions are reduced.)

- Improving branching by the use of RX branch instructions. (An RX branch instruction saves a load instruction and reduces the number of required address constants.)

When using OPT=2, the loop structure and data flow of the program are analyzed. The object code is improved over OPT=1 by:

- Assigning registers across a loop to the most active variables, constants, and base addresses within the loop.

  Moving outside the loop many computations which need not be calculated within the loop.

Recognizing and replacing redundant computations.

Replacing (if possible) multiplication of induction variables by addition of those variables.

- Deleting (if possible) references to some variables.
- Using (where possible) the BXLE instruction for loop termination. (The BXLE instruction is the fastest conditional branch; time and space are saved.)

Programming Considerations Using the Optimizer

In general, the specification of OPT=1 or OPT=2 causes compilation time to increase. However, the object code produced is more concise and yields shorter execution times.

The object module logic, when optimized, is identical to the unoptimized logic, except in the following cases:

1. If the list of statement numbers in an Assigned GOTO statement is incomplete, errors, which were not present in the unoptimized code, may arise in the optimized code.

2. The computational reordering done by text optimization may produce a different execution time behavior than unoptimized code. For example, a test of an argument of a FORTRAN library function may be executed after the call to the function. This is caused by the movement of the function call to the back target of the loop when the function argument is not changed within the loop.

```
      DO   11   I=1,10
      DO   12   J=1,10
      IF (B(I).LT.0.)GO TO 11
12    C(J)=SQRT(B(I))
11    CONTINUE
```

   The square root computation will occur before the less-than-zero test, and will result in a message if B(I) is negative. A rearrangement of the program which could avoid this situation can be constructed:

```
      DO   11   I=1,10
      IF (B(I).LT.0.) GO TO 11
      DO   12   J=1,10
12    C(J)=SQRT(B(I))
11    CONTINUE
```

   A similar condition may result with the statements:

```
CALL OVERFL(J)
CALL DVCHK(J)
```

   These may produce different results when optimized, because computations causing overflow, underflow, or divide-check conditions could be moved out of the loop in which the test occurs.

3. If a programmer defines a subprogram with the same name as a FORTRAN-supplied subprogram (e.g., SIN, ATAN, etc.), errors could be introduced during optimization. If the subprogram stores into its arguments, refers to COMMON, performs I/O, or remembers its own variables from one execution to another, the name of the subprogram must be specified in an EXTERNAL statement to allow the program to be optimized without error.

4. In the statements

```
COMMON   X, Y1(10), W, Z
EQUIVALENCE (Y1,Y2)
DIMENSION Y2(12)
```

   there is an implied equivalence of Y2 (11) and W and Y2(12) and Z.

   If the optimization feature is not used, and

   W=Q and A=Y2(I) (where I=11)

   then the value of Q is assigned to A.

   However, if OPT=2 is used, and

   W=Q and A=Y2(I) (where I=11)

   there is no guarantee that the value of Q is assigned to A.

5. With OPT=2, when a subprogram is called at one entry point for initialization of reference-by-name arguments, and at another entry point for subsequent computation, certain argument values may not be transmitted. This applies to either arguments of the second call or any argument values redefined between calls and not explicitly defined in COMMON.

   In the following example the incremented value for I may not be transmitted to the subprogram due to the loop initialization optimization.

```
      CALL INIT (I)   SUBROUTINE  INIT (/J/)
            .                .
            .                .
            .                .
      I = 0           ENTRY   COMP
10    CALL COMP
      I = I + 1
            .
            .
            .
      GO TO 10
```

Definition of a Loop

The term 'loop' is used to refer to DO loops and other configurations of coding that a programmer regards as a loop.

If a programmer writes a loop which is preceded by an IF statement, a conditional GOTO statement, or READ statement with END or ERR parameters, the loop is not identified and efficiency is lost. A CONTINUE statement at the end of the range of a DO also obscures a loop (other than a DO loop) that follows the CONTINUE without intervening initialization. The insertion of a labeled CONTINUE statement or any other suitable re-arrangement allows the loop to be recognized.

The movement of computations from inside a loop to the initialization coding is done on the assumption that every statement in the loop is executed more frequently than the initialization coding. Occasionally, this assumption fails and computations are moved to a position where they are computed more often. One way to prevent such a move is to make a subprogram of the coding (statements and computations) that is executed less frequently within a loop than it would be in the initialization coding.

The recognition of loops may also be obscured when the programmer knows that some paths through the program cannot occur; for example,

```
 10 IF (L) GOTO 200
 20 I=1
 30 ASSIGN 40 TO J
    GOTO 100
 40 I=I +1
 50 IF (I.LE.N) GOTO 30
    .
    .
    .
100 B(I) = FUNCT (I)
110 GOTO J, (40, 220)
200 ASSIGN 220 TO J
210 GOTO 100
220 CONTINUE
```

From the programmer's point of view, the statements 30 to 50 comprise a loop which is initialized by statement 20. The loop causes an internal subprogram consisting of statements 100 and 110 to be executed. From the compiler's point of view, it appears possible to execute statements in the order 10, 200, 210, 100, 110, 40, 50, 30. The compiler does not recognize the loop, because it appears possible to enter it without passing through the initialization coding in statement 20.

A loop can be obscured by the computed GOTO, because the compiler always assumes that one of the possible branches is to the succeeding statement, even though the programmer knows that such a branch is impossible. A loop can also be obscured by a call to the EXIT routine, because the compiler assumes there is a path from such a statement to the next.

Movement of Code Into Initialization of a Loop

Where it is logically possible to do so with OPT=2, the optimizer moves computations from inside the loop to the outside. This movement permits a programmer to do more straightforward coding without penalty in object code efficiency.

If an expression is evaluated inside a loop and all the variables in the expression are unchanged within the loop, the computation is generally moved outside the loop into the coding sequence which initializes the loop. Even if the constant expression is part of a larger expression, this constant expression may still be recognized and moved. However, the movement depends on how the larger expression is written. Table 13 gives examples of expressions and the constant parts which are recognized and moved.

Common Expression Elimination

With OPT=2, if an expression occurs twice in such a way that:

1. any path starting at an entry to the program always passes through the first occurrence of the expression to reach the second occurrence (and any subsequent occurrence), and

Table 13. Constant Expressions

Expression where C1, C2,... are constant in the loop	Constant expression recognized and moved
C1 + C2 * C3/SIN (C4)	C1 + C2 * C3/SIN (C4)
C1 + C2 * C3 + B1	C1 + C2 * C3
C1 + B1 + C2 * C3	C2 * C3
B1 + C1 + C2 * C3	C2 * C3
C1 + B1 + B2 + C2 * C3	C2 * C3
C1 * C2/B1	C1 * C2

2. any evaluation of the second (third, fourth, etc.) expression produces a result identical to the most recent evaluation of the first expression, then the value of the first expression is saved (generally) and used instead of the value of the second (third, fourth, etc.) expression.

In statements such as:

```
A=B + C + D
E=C + D
```

the common expression C + D is not recognized, because the first expression is computed as (B + C) + D.

Induction Variable Optimization

In a loop with OPT=2, an induction variable is a variable that is only incremented by a constant or by a variable whose value is constant in the loop.

When an induction variable is multiplied by a constant in the loop, the optimizer may replace the multiplication with an addition by introducing a new induction variable into the loop. This new induction variable may make it possible to delete all references to the original induction variable. This deletion is likely to occur if the original induction variable is used only as a subscript within the loop, and the value of the subscript is not used on exit from the loop.

Register Allocation

Some variables are assigned to a register on entry to a loop and retained in the register through part or all of the loop to avoid loading and storing the variable in the loop. Within the loop, the variable is modified only in the assigned register, the value of the variable in storage is not changed. If necessary, the latest value of the variable is stored after exit from the loop.

The value in general register 13, which points to the start of a register save area, remains constant during execution of a subprogram. This register is used to refer to address constants and temporary variables, and for branching within the program. The value in general register 12 remains constant and is used to refer to constants, variables, and arrays that are not in COMMON.

General registers 14 and 15 are used for base addresses and index values on a strictly local basis. Floating-point register 0 and general register 0 are used as locally assigned arithmetic accumulators. General register 1 is used in conjunction with general register 0 for fixed-point arithmetic operations, and to point to argument lists in subprogram linkages.

The remaining registers are used for accumulators, index values, base addresses, and high speed storage (a register reference is faster than a main storage reference).

Because general register 13 is not adequate to provide RX branching throughout a large program, general registers 11, 10, and 9 may be pre-empted for RX branching (only if the program exceeds 4K, 8K, and 12K bytes, respectively). (RR branches preceded by loads are required for branching to points beyond the first 16K bytes of the program and possibly to the last part of a program if it exceeds 4K, 8K, or 12K bytes by a small amount.)

COMMON Blocks

Because each COMMON block is independently relocatable, each requires at least one base address to refer to the variables in it. A sequence of coding that refers to a large number of COMMON blocks is slowed down by the need to load base addresses into general registers. Thus, if three

COMMON blocks can be combined into one block whose total size is less than 4096 bytes, one base address can serve to refer to all the variables. (Many register loads can be avoided.)

The order in which data is entered into a COMMON block may also affect the number of base addresses needed. For example, if an array of 5000 bytes is placed in a COMMON block and followed by 200 bytes of variables, two base addresses are needed: the beginning address of the first variable and the beginning address of the last differ by more than 4096 bytes. However, if the variables preceded the array, one base address would suffice.

EQUIVALENCE Statements

Optimization tends to be weakened by the occurrence of variables in EQUIVALENCE statements.

When an array appears in an EQUIVALENCE statement, a reference to one of its elements cannot be eliminated as a common expression, nor can the reference be moved out of a loop. However, the elimination and movement of subscript calculations used for making the reference is not affected.

If a variable is made equivalent only to another variable (not in COMMON) of the same type and length, optimization is not weakened. The net effect is that the compiler accepts the two names as alternate pointers to the same storage location. However, if a variable is made equivalent to another variable in any other way, all references to it are 'immobilized': the references cannot be eliminated, moved, confined to registers, or altered in any way.

Boundary Adjustment of Variables in COMMON Blocks and EQUIVALENCE Groups

Variables in a COMMON block or EQUIVALENCE group may be in any order if the BOUNDRY=ALIGN option is specified in the FORTLIB macro instruction during system generation, because boundary alignment violations are corrected during execution. (The FORTLIB macro instruction is described in the System Generation publication.) If the BOUNDRY=NOALIGN option is specified and boundary violations are encountered during execution of the object program, the job terminates.

If the BOUNDRY=ALIGN option of the FORTLIB macro instruction is specified and a boundary violation occurs in a FORTRAN main program or in a FORTRAN or assembler language subprogram, each instruction that refers to the improperly aligned variable requires that (1) the specification exception resulting from this reference be processed, and (2) the boundary alignment routine be invoked. Therefore, considerable programming efficiency is gained if the programmer ensures that all of the variables have proper boundary alignment. The FORTRAN IV Language publication contains information on boundary alignment.

When boundary alignment is performed, program interrupt message IHC210I is issued. (This message is described completely in the section "Program Interrupt Messages" in Appendix D). For boundary alignment, the letter A appears in the text of the message and the code 6 appears in the old PSW (program status word), which is included in the message. The number of warning messages printed is limited to 10. After 10 boundary alignment adjustments have been made, the message is suppressed, but boundary alignment violations continue to be corrected.

Note: Even if BOUNDRY=ALIGN is specified and a boundary error occurs in an EXECUTE, LM (load multiple), or STM (store multiple) instruction in a subprogram written in assembler language, boundary adjustment does not take place and the job terminates. Therefore, if these instructions refer to improperly aligned data, they should not be used in assembler language subprograms.

Multidimensional Arrays

In general, references to higher dimensional arrays are slower than references to lower dimensional arrays. Thus, a set of one-dimensional arrays is more efficient than a single two-dimensional array in any case where the two-dimensional array can be logically treated as a set of one-dimensional arrays.

Constants occurring in subscript expressions are accounted for at compile time and have no effect at execution time.

Program Structure

If a large number of variables are to be passed among calling and called programs, some of the variables should be placed in the COMMON area. For example, in the main program and subroutine EXAMPL

```
      DIMENSION E(20),I(15)
      READ(10)A,B,C
      CALL EXAMPL(A,B,C,D,E,F,I)
            .
            .
            .
      END

      SUBROUTINE EXAMPL (X,Y,Z,P,Q,R,J)
      DIMENSION Q(20),J(15)
            .
            .
            .
      RETURN
      END
```

time and storage are wasted by allocating storage for variables in both the main program and subprogram and by the subsequent instructions required to transfer variables from one program to another.

The two programs should be written using a COMMON area, as follows:

```
      COMMON A,B,C,D,E(20),F,I(15)
      READ(10)A,B,C
      CALL EXAMPL
            .
            .
            .
      END

      SUBROUTINE EXAMPL
      COMMON X,Y,Z,P,Q(20),R,J(15)
            .
            .
            .
      RETURN
      END
```

Storage is allocated for variables in COMMON only once and fewer instructions are needed to cross reference the variables between programs.

To reduce compilation time for equivalence groups, the entries in the EQUIVALENCE statement should be specified in descending order according to offset. For example, the statement

```
EQUIVALENCE (ARR1(10,10),ARR2(5,5),
            ARR3(1,1),VAR1)
```

compiles faster than the statement

```
EQUIVALENCE (VAR1,ARR3(1,1),ARR2(5,5),
            ARR1(10,10))
```

To reduce compilation time and save internal table space, equivalence groups should be combined, if possible. For example, the statement

```
EQUIVALENCE (ARR1(10,10),ARR2(5,5),VAR1)
```

compiles faster and uses less internal table space than the statement

```
EQUIVALENCE (ARR1(10,10),VAR1),
            (ARR2(5,5),VAR1)
```

Logical IF Statements

A statement such as:

```
IF(A.LT.B.OR.C.GT.F(X).OR..NOT.L)GOTO 10
```

is compiled as though it were written:

```
      IF (A .LT. B) GO TO 10
      IF (C .GT. F(X)) GO TO 10
      IF (.NOT. L) GOTO 10
```

Thus, if A .LT. B is found to be true, the remainder of the logical expression is not evaluated.

Similarly, a statement such as:

```
      IF (D.NE. 7.0 .AND. E.GE.G) I=J
```

is compiled as:

```
      IF (D.EQ. 7.0) GOTO 20
      IF (E.LT.G) GOTO 20
      I=J
20    CONTINUE
```

The order in which a programmer writes logical expressions in an IF statement affects the speed of execution.

If A is more often true than B, then write A .OR. B rather than B .OR. A; and write B .AND. A rather than A .AND. B.

If any of the following occur in a logical expression:

1. a mixture of both .AND. and .OR. operators

2. a .NOT. operator followed by a parenthesized expression

the entire logical expression must be evaluated and efficiency is lost.

Branching

The statement

```
      IF(A.GT.B) GOTO 20
```

gives equivalent or better code than

```
      IF(A-B)10,10,20
10    CONTINUE
```

The Assigned GOTO is the fastest conditional branch.

The computed GOTO should be avoided unless four or more statement labels occur within the parentheses.

The statement

```
      IF(I-2)20,30,40
```

is significantly faster than

```
      GOTO (20,30,40), I
```

Indicators and Sense Lights

At the start of program execution, the divide-check indicator, the overflow indicator, and the sense lights are not initialized. Therefore, if a programmer intends to use the indicators or sense lights, he should initialize them prior to use; otherwise, erroneous results may be obtained.

Name Assignment

For its internal use, the compiler places names used for variables, arrays, and subprograms into a table. This table is divided into six strings and is searched many times during compilation. Names that are one character long are placed in the first string; names two characters long are placed in the second string; and so on. For faster compiling, the names should be distributed equally among the six strings.

Conditional Branching

A test for 0.0 in an IF statement is not recommended. Slight inaccuracies may cause the low-order bit(s) to be set. Therefore, the test for 0.0 may not yield the expected result.

Use of DUMP and PDUMP

Under the operating system, a program may be loaded into different areas of storage for different executions of the same job. The following conventions should be observed when using the DUMP or PDUMP subroutine to insure that the appropriate areas of storage are dumped.

If an array and a variable are to be dumped at the same time, a separate set of arguments should be used for the array and for the variable. The specification of limits for the array should be from the first element in the array to the last element. For example, if an array TABLE is dimensioned as:

```
      DIMENSION TABLE (20)
```

The following statement could be used to dump TABLE and the real variable B in hexadecimal format and terminate execution after the dump is taken:

```
      CALL DUMP (TABLE(1),TABLE(20),0,B,B,0)
```

If an area in COMMON is to be dumped at the same time as an area of storage not in COMMON, the arguments for the area in COMMON should be given separately. For example, if A is a variable in COMMON, the following statement could be used to dump the variables A and B in real format without terminating execution:

```
      CALL PDUMP (A,A,5,B,B,5)
```

If variables not in COMMON are to be dumped, the programs should list each variable separately in the argument list. For example, if R, P, Q are defined implicitly in the program, the statement

```
      CALL PDUMP(R,R,5,P,P,5,Q,Q,5)
```

should be used to dump the three variables. If

```
      CALL PDUMP(R,Q,5)
```

is used, all main storage between R and Q is dumped.

If an array and a variable are passed as arguments to a subroutine, the arguments in the call to DUMP or PDUMP in the subroutine should specify the parameters used in the definition of the subroutine. For example, if the subroutine SUBI is defined as:

```
      SUBROUTINE SUBI(X,Y)
      DIMENSION X(10)
```

and the call to SUBI within the source module is:

```
      DIMENSION A(10)
           .
           .
           .
      CALL SUBI(A,B)
```

Table 14. Additional Built-In Functions

Function	Entry Name	In-Line I	No. of Arguments	Type of Arguments	Type of Function Value
Logical intersection of two arguments	AND	I	2	Real*4 or Integer*4	Real*4
Logical union of two arguments	OR	I	2	Real*4 or Integer*4	Real*4
Logical 1's. Complement of argument.	COMPL	I	1	Real*4 or Integer*4	Real*4

then the following statement in the subroutine should be used to dump the variables in hexadecimal format without terminating execution:

```
CALL PDUMP (X(1),X(10),0,Y,Y,0)
```

If the statement

```
CALL PDUMP (X(1),Y,0)
```

is used, all storage between A(1) and Y is dumped, due to the method of transmitting arguments. (Y does not occupy the same storage location as B.)

Use of ERR Parameter in READ Statement

Use of the optional ERR parameter for a READ statement can indicate the source program statement to which transfer should be made if an error is encountered during data transfer. When transfer has been made to that statement, the first subsequent READ in the source program provides the record that was in error. If this is not the record desired, an additional READ should be be issued.

If the ERR parameter is omitted from the READ statement, an input/output device error terminates program execution.

Support of AND, OR, and COMPL

The functions listed in Table 14 are not part of the standard FORTRAN language, but are currently supported by the compiler. Caution should be exercised in their use since continued support is not assured.

DATA Statements and Literal Constants

These paragraphs describe DATA Statement initialization in cases when data does not exactly match the specifications for variables or arrays involved.

Assume DIMENSION X(10).

```
DATA X/15HABCDEFGHIJKLMNO, 10.0/
```

will be treated exactly the same as

```
DATA X(1)/15HABCDEFGHIJKLMNO/,X(2)/10.0/
```

namely,

```
X(1)=ABCD

X(2)=10.0

X(3)=IJKL

X(4)=MNOb

X(5) through X(10)=bbbb
```

Note the following points:

1. If the array name is used or if the subscript is 1, the entire array is initialized.
2. If the literal in the DATA statement is longer than one array item, and if one of the conditions in point 1 holds, the excess characters of the literal will be placed into succeeding items.
3. A constant following the literal with excess characters may replace some of the excess characters.
4. In DATA X(5)/5HABCDE/, the characters ABCD are placed into X(5). E is not treated as an excess character.

Further examples:

a. Assume DIMENSION Y(5)

```
DATA Y(1),Y(3)/'ABCDEF','XY'/
```

will be initialized as follows:

```
Y(1)=ABCD
Y(2)=EFbb
Y(3)=XYbb
Y(4) through Y(5)=bbbb
```

b. Assume DIMENSION Z(10)

```
Then DATA Z(1),Z(2),Z(6),Z(5)/
  10HABCDEFGHIJ,QRSTU,5H12345,
  5HGGGGG/
```

will be initialized as follows:

```
Z(1)=ABCD
Z(2)=QRST
Z(3)=IJbb
Z(4)=bbbb
Z(5)=GGGG
Z(6)=1234
Z(7)through Z(10)=bbbb
```

The above describes the current implementation for data initialization; however, the implementation is subject to change and it should be used with caution.

Direct Access Programming

Using direct access I/O rather than sequential I/O can decrease load module execution time: the direct access statements in the FORTRAN IV language enable the programmer to retrieve a record from any place on the volume without reading all the records preceding that record in the data set. For efficiency, direct data sets should be pre-formatted. If, however, the NEW subparameter is specified in the DD statement for the data set, a FORTRAN supplied load module will format the data set before the program begins processing.

Note: Direct access I/O statements and sequential I/O statements may not be used to process the same direct data set within the same FORTRAN load module. However, sequential I/O statements may process a direct data set in one load module, while direct access I/O statements process it in another.

Not all applications are suited to direct access I/O, but an application that uses a large table that must be held in external storage can use direct access I/O effectively. An even better example of a direct access application is a data set that is updated frequently. Records in the data set that are updated frequently are called master records. Records in other data sets used to update the master records are called detail records.

Each of the master records should contain a unique identification that 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 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 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 the Supervisor and Data Management Service publication in the section on direct data set processing.

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.

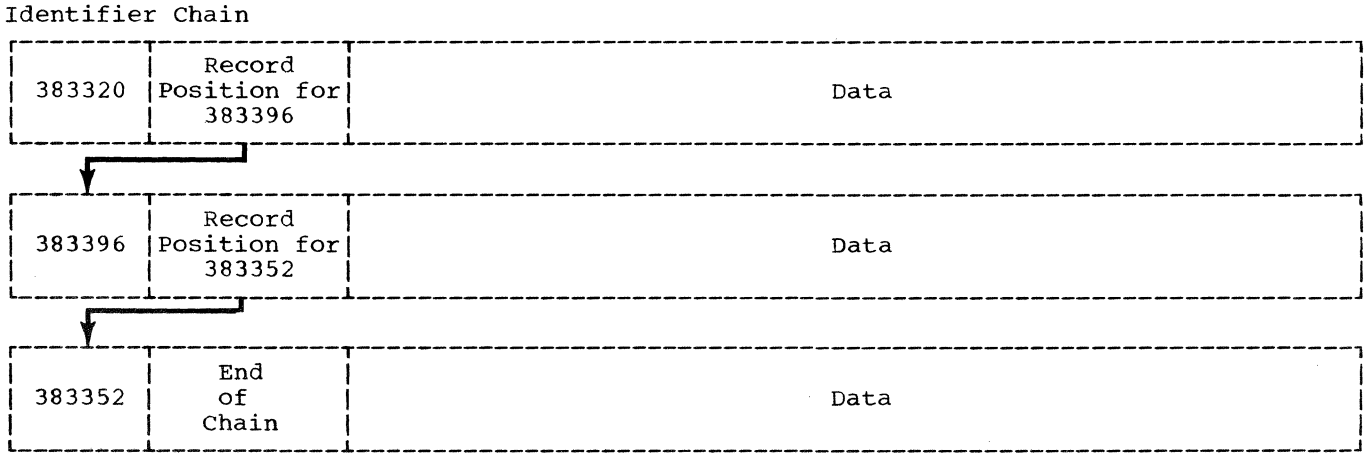


Figure 47. Record Chaining

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. Set for the First Time

The DEFINE FILE statement for a data set 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 to perform the I/O operations on the data set. However, the associated variable in the DEFINE FILE statement is only changed by I/O operations performed in the source module in which the DEFINE FILE statement occurs. Even if the associated variable is passed in COMMON, the associated variable is changed only by I/O operations that occur in the source module in which the DEFINE FILE statement appears.

An associated variable should not be passed as a parameter between a main program and its subprograms because the associated variable is not passed in the same way that other variables are passed. Other variables reflect the result of any operations performed on them in the subprogram. An associated variable is not changed by operations performed on it in the subprogram.

The FIND statement permits record retrieval to occur concurrently with computation or I/O operations performed on different

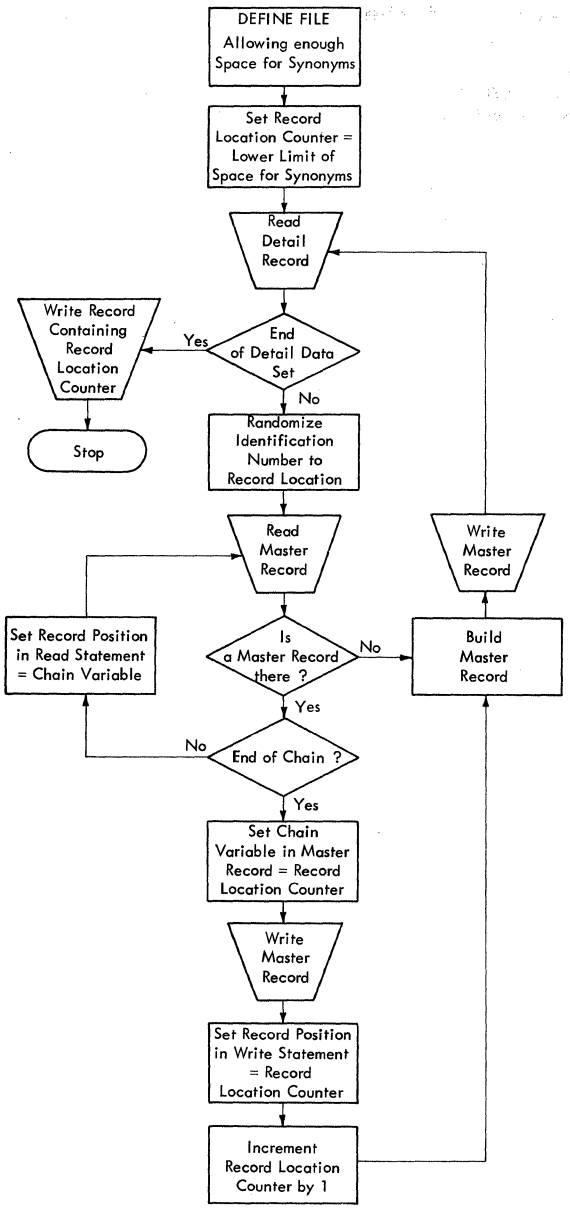


Figure 48. Writing a Direct Access Data Set for the First Time

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 is 25.
- The maximum number of implied DOs per statement is 20.
- The maximum number of arguments in a Statement Function definition is 20.
- The maximum number of characters allowed in a literal constant is 255.
- The maximum number of characters allowed in a PAUSE message is 255.
- The maximum number of nested references to another statement function within a statement function definition is 50; the maximum number of times a statement function may be nested is 50.
- The debug facility is not supported.

Note: In this version of the compiler, Statement Functions are expanded in-line.

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.

When the FORTRAN library is changed for maintenance or to provide additional features, precompiled programs in a user library require special attention to benefit from the changed library modules. This can be accomplished by using the linkage edit facilities to include the current library modules, and storing the resultant load module back into the FORTRAN library. When the facilities of the linkage editor are used to provide an overlay structure or to replace a single control section, care should be taken not to mix FORTRAN library modules that are at diverse operating system levels.

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.

VOLUME MOUNTING AND DEVICE SEPARATION:

```
UNIT=(name{,n}{,P}[,DEFER]
      [,SEP=(ddname[,ddname]...)])
```

can be specified for volume mounting and device separation. The "name" and number of units are discussed in the section "Data Definition Statement".

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 subparameter 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))
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.

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.

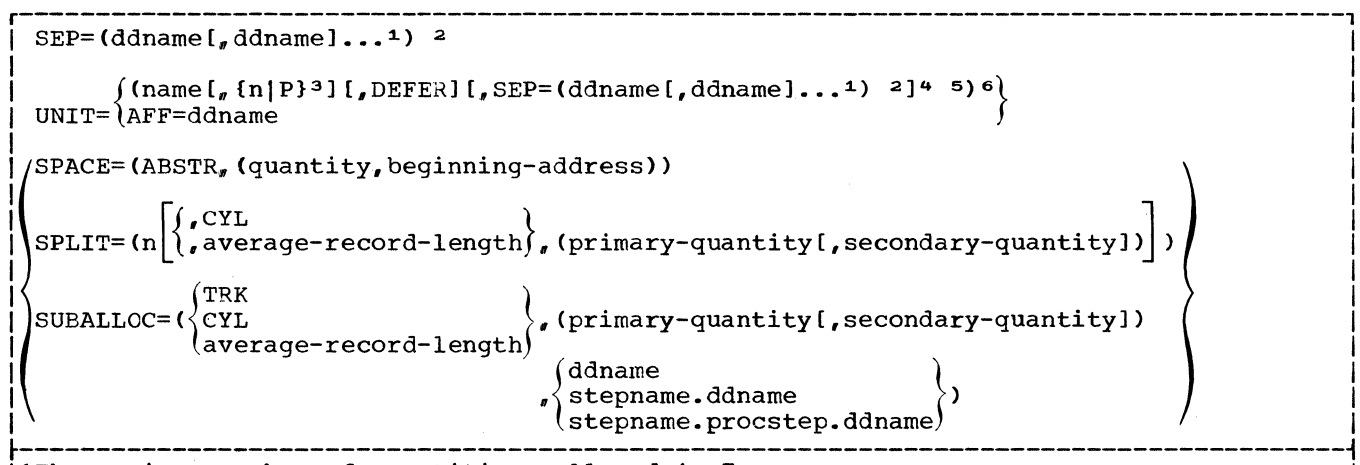


Figure 49. DD Statement Parameters for Optimization

```
SPLIT=(n[{,CYL                    }
        {,average-record-length}
        ,(primary-quantity
        [,secondary-quantity])])
```

is substituted for the SPACE parameter when the use of cylinders is split. If CYL is specified, "n" indicates the number of tracks per cylinder to be used for this data set. If "average record length" is specified, "n" indicates the percentage of tracks per cylinder used for this data set. The remaining subparameters are the same as those specified for SPACE in "Creating Data Sets."

More than one DD statement in a step will use the SPLIT parameter. However, only the first DD statement specifies all the subparameters; the remaining DD statements need only specify "n". For example,

```
//STEP4 EXEC PGM=TESTFI
//FT08F001 DD SPLIT=(45,800,(100,25))
           .
           .
           .
//FT17F001 DD SPLIT=(35)
           .
           .
           .
//FT23F001 DD SPLIT=(20)
```

ACCESSING UNUSED SPACE: Data sets in previous steps may not have used all the space allocated to them in a DD statement. The SUBALLOC parameter may be substituted for the SPACE parameter to permit a new data set to use this unused space.

```
SUBALLOC=({TRK                          }
          {CYL                          }
          {average-record-length        }
          (,primary-quantity
          [,secondary-quantity]),
          {ddname                       }
          {stepname.ddname              }
          {stepname.procstep.ddname     })
```

The data set from which unused space is taken is defined in the DD statement "ddname", which appears in the step "stepname." (The step must be in the current job.) The other subparameters specified in the SUBALLOC parameter are the same as the subparameters described for SPACE in "Creating Data Sets."

SYSTEM OUTPUT

The compiler, linkage editor, and load modules produce aids which may be used to document and debug programs. This section describes the listings, maps, card decks, and error messages produced by these components of the operating system.

COMPILER OUTPUT

The compiler can generate a listing of source statements, a table of source module names, a structured listing of source statements, an object module listing, a table of source module labels, and an object module card deck. 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 program listing is shown in Figure 50. This printout is the source listing of Sample Program 1 shown in Appendix C of the FORTRAN IV Language publication. (This program will be used throughout the remainder of this publication for purposes of illustration.)

Cross Reference Listing

If the compiler XREF option is specified, a cross reference listing of variables and labels is written in the data set specified by the SYSPRINT DD statement. The variable names are listed in alphabetical order, according to length. (Variable names of one character appear first in the listing.) The labels are listed in ascending sequence along with the internal statement number of the statement in which the label is defined.

For both variable names and labels, the listing also contains the internal statement number of each statement in which the variable or label is used. Figure 51 shows a compiler cross reference listing produced for the program in Figure 50.

Structured Source Listing

If the EDIT option is specified, a structured source listing is written in the data set specified by the SYSPRINT DD statement. This listing is independent of the usual source listing and indicates the loop structure and logical continuity of the source program.

Each loop is assigned a unique three-digit number. Entrance to the loop is indicated by (xxx before the internal statement number of the first statement in the loop; exit from the loop is indicated by xxx) on a separate line before the next non-comment line. The xxx is the loop number.

```
                C     PRIME NUMBER PROBLEM
ISN 0002          100 WRITE (6,8)
ISN 0003            8 FORMAT (52H FOLLOWING IS A LIST OF PRIME NUMBERS FROM 1 TO 1000/
                     119X,1H1/19X,1H2/19X,1H3)
ISN 0004          101 I=5
ISN 0005            3 A=I
ISN 0006          102 A=SQRT(A)
ISN 0007          103 J=A
ISN 0008          104 DO 1 K=3,J,2
ISN 0009          105 L=I/K
ISN 0010          106 IF(L*K-I)1,2,4
ISN 0011            1 CONTINUE
ISN 0012          107 WRITE (6,5)I
ISN 0013            5 FORMAT (I20)
ISN 0014            2 I=I+2
ISN 0015          108 IF(1000-I)7,4,3
ISN 0016            4 WRITE (6,9)
ISN 0017            9 FORMAT (14H PROGRAM ERROR)
ISN 0018            7 WRITE (6,6)
ISN 0019            6 FORMAT (31H THIS IS THE END OF THE PROGRAM)
ISN 0020          109 STOP
ISN 0021              END
```

Figure 50. Source Module Listing

```
SYMBOL   INTERNAL STATEMENT NUMBERS
A        0005  0006  0006  0007
I        0004  0005  0009  0010  0012  0014  0014  0015
J        0007  0008
K        0008  0009  0010
L        0009  0010
SQRT     0006

LABEL    DEFINED   REFERENCES
   1     0011      0008  0010
   2     0014      0010
   3     0005      0015
   4     0016      0010  0015
   5     0013      0012
   6     0019      0018
   7     0018      0015
   8     0003      0002
   9     0017      0016
 100     0002
 101     0004
 102     0006
 103     0007
 104     0008
 105     0009
 106     0010
 107     0012
 108     0015
 109     0020
```

Figure 51. Compiler Cross Reference Listing

Indentations are used to show dominance relationships among executable source statements. Statement A dominates statement B if A is the last statement common to all logical paths from which B receives control. Statement A is called a dominator, statement B is called a dominee. By this definition, a statement can have only one dominator, but a dominator may have several dominees. For example, a computed GO TO statement is the last statement through which control passes before reaching three other statements. The GO TO statement is a dominator with three dominees.

A dominee is indented from its dominator unless it is either the only dominee or the last dominee of that dominator. The line of sight between a dominator and its dominee(s) may be obscured by intervening statements. This is a dominance discontinuity and is indicated by C--- on a separate line above the dominee. Listing

Comments and non-executable statements are not involved in dominance relationships; their presence never causes a dominance discontinuity. Comments line up with the last preceding non-comment line; nonexecutable statements line up either with the last preceding executable statement or with the first one following.

Figure 52 shows a structured source listing produced for the program in Figure 50.

Object Module Listing

If the LIST option is specified, the object module listing is written in the data set specified by the SYSPRINT DD statement. The listing is in pseudo-assembly language format; i.e., all instructions are not legal assembly language instructions.

The listing is arranged in a column format as follows:

Column 1: The address (in hexadecimal) of the instruction.

Column 2: The assembly format (in hexadecimal) of the instruction.

```
                                C     PRIME NUMBER PROBLEM
      ISN 0002                100 WRITE (6,8)
      ISN 0003                  8 FORMAT (52H FOLLOWING IS A LIST OF PRIME NUMBERS FROM 1 TO 1000/
                                 119X,1H1/19X,1H2/19X,1H3)
      ISN 0004                101 I=5
(002ISN 0005                    3 A=I
      ISN 0006                102 A=SQRT(A)
      ISN 0007                103 J=A
      ISN 0008                104 DO 1 K=3,J,2
(001ISN 0009                    105 L=I/K
      ISN 0010                106 IF(L*K-I)1,2,4
      ISN 0011                  1    CONTINUE
 001)                                C
      ISN 0012                107    WRITE (6,5)I
      ISN 0013                  5    FORMAT (I20)
      ISN 0014                  2    I=I+2
      ISN 0015                108    IF(1000-I)7,4,3
 002)                                C
      ISN 0016                  4    WRITE (6,9)
      ISN 0017                  9    FORMAT (14H PROGRAM ERROR)
      ISN 0018                  7 WRITE (6,6)
      ISN 0019                  6 FORMAT (31H THIS IS THE END OF THE PROGRAM)
      ISN 0020                109 STOP
      ISN 0021                    END
```

Figure 52. Structured Source Listing

Column 3: Source labels and compiler generated labels (compiler generated labels contain six digits).

Column 4: The actual instruction.

Column 5: Significant items referred to in the corresponding instruction, e.g., entry points, labels, variables, constants, and temporaries (.yxx where y is S, T, or Q and xx is two digits).

Figure 53 shows an object module listing produced for the program in Figure 50.

Storage Map

If the MAP option is specified, a table of names, which appear (or are implied) in the source module, is written in the data set specified by the SYSPRINT DD statement. The table includes:

1. The name of the program.
2. The hexadecimal size of the program in bytes.
3. A list of all variable names, statement function names, subprogram names, and internal function names.
4. An indication of the use of each name, as follows:

   C indicates variables in COMMON
   E indicates variables that appear in an EQUIVALENCE statement
   IF indicates an internal function
   NR indicates variables not referred to
   SF indicates statement functions
   XF indicates subprograms
   XR indicates variables, arrays, or subprograms that are referenced by name

5. An indication of the use of each variable, as follows (alone or in combination):

   A indicates that the variable name was used as an argument; i.e., appeared in a parameter list

   F indicates that the value of the variable was used at some time; i.e., the variable name appeared on the right of an equal sign

   S indicates that a value was stored into the variable; i.e., the variable name appeared on the left of an equal sign

6. The type and length of each variable.
7. The relative address assigned to each variable. (All functions and subroutines have a relative address of 00000.)
8. A map of each COMMON block, followed by a map of any equivalences made for the block. The name of each block is given if a name was assigned, along with the hexadecimal size of the block in bytes, and the name, type, length, and relative address of each variable in the block. For each equivalence, the name of the variable is given along with its displacement (offset) from the beginning of the common block.

Figure 54 shows a storage map produced for the program in Figure 50.

Label Map

If the MAP option is specified, a table of statement numbers, which appear (or are implied) in the source module, is written in the data set specified by the SYSPRINT DD statement. This table includes:

1. The statement number of each source or generated label.
2. The relative address assigned to each.

Figure 55 shows a label map produced for the program in Figure 50.

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.

```
000000  47 F0 F 00C        MAIN  BC    15,12(0,15)
000004  07404040                 DC    XL4'40404040'
000008  40404040                 DC    XL4'40404040'
00000C  90 EC D 00C              STM   14,12,12(13)
000010  98 23 F 020              LM    2,3,32(15)
000014  50 30 D 008              ST    3,8(13)
000018  50 D0 3 004              ST    13,4(0,3)
00001C  07 F2                    BCR   15,2

TEMPORARY FOR FLOAT/FIX
CONSTANTS
000108  4E000000                 DC    XL4'4E000000'
00010C  00000000                 DC    XL4'00000000'
000110  00000002                 DC    XL4'00000002'
000114  00000003                 DC    XL4'00000003'
000118  00000005                 DC    XL4'00000005'
00011C  000003E8                 DC    XL4'000003E8'
ADCONS FOR VARIABLES AND CONSTANTS
ADCONS FOR EXTERNAL REFERENCES
000138  00000000                 DC    XL4'00000000'                    SQRT
00013C  00000000                 DC    XL4'00000000'                    IBCOM=
000148  41 40 0 002      100000  LA     4,    2( 0, 0)                   2
00014C  41 90 0 003              LA     9,    3( 0, 0)                   3
000150  41 A0 0 005              LA     10,   5( 0, 0)                   5
000154  41 80 0 3E8              LA     8,1000( 0, 0)                    3E8
000158  58 F0 D 08C        100   L      15, 140( 0,13)                   IBCOM=
00015C  45 E0 F 004              BAL    14,   4( 0,15)
000160  00000006                DC    XL4'00000006'                    F
000164  00000028                DC    XL4'00000028'                    G+'
000168  45 E0 F 010              BAL    14,  16( 0,15)
00016C  50 A0 D 074        101   ST     10, 116( 0,13)                   I
000170  58 00 D 074          3   L      0, 116( 0,13)                    I
000174  68 00 D 058              LD     0,   88( 0,13)
000178  60 00 D 050              STD    0,   80( 0,13)
00017C  12 00                    LTR    0, 0
00017E  47 40 D 0DE              BC     4, 222( 0,13)
000182  50 00 D 054              ST     0,   84( 0,13)
000186  6A 00 D 050              AD     0,   80( 0,13)
00018A  47 F0 D 0E8              BC     15, 232( 0,13)
00018E  10 00                    LPR    0, 0
000190  50 00 D 054              ST     0,   84( 0,13)
000194  68 00 D 050              SD     0,   80( 0,13)
000198  70 00 D 070              STE    0, 112( 0,13)                    A
00019C  41 10 D 04C        102   LA     1,   76( 0,13)
0001A0  58 F0 D 088              L      15, 136( 0,13)                   SQRT
0001A4  05 EF                    BALR   14,15
0001A6  70 00 D 090              STE    0, 144( 0,13)                    .T00
0001AA  78 00 D 090              LE     0, 144( 0,13)                    .T00
0001AE  70 00 D 070              STE    0, 112( 0,13)                    A
0001B2  28 00              103   SDR    0, 0
0001B4  78 00 D 070              LE     0, 112( 0,13)                    A
0001B8  6E 00 D 058              AW     0,   88( 0,13)
0001BC  60 00 D 050              STD    0,   80( 0,13)
0001C0  58 50 D 054              L      5,   84( 0,13)
0001C4  47 A0 D 11A              BC     10, 282( 0,13)
0001C8  11 55                    LNR    5, 5
0001CA  18 69              104   LR     6, 9
0001CC  58 70 D 074              L      7, 116( 0,13)                    I
0001D0  18 07              105   LR     0, 7
0001D2  8E 00 0 020              SRDA   0,   32( 0, 0)
0001D6  1D 06                    DR     0, 6
0001D8  50 10 D 080              ST     1, 128( 0,13)                    L
0001DC  18 36              106   LR     3, 6
0001DE  5C 20 D 080              M      2, 128( 0,13)                    L
0001E2  1B 37                    SR     3, 7
0001E4  47 80 D 15C              BC     8, 348( 0,13)                    2
0001E8  47 20 D 178              BC     2, 376( 0,13)                    4
0001EC  87 64 D 120          1   BXLE   6, 288( 4,13)                   105
0001F0  58 F0 D 08C        107   L      15, 140( 0,13)                   IBCOM=
0001F4  45 E0 F 004              BAL    14,   4( 0,15)
0001F8  00000006                 DC    XL4'00000006'                    F
0001FC  00000072                 DC    XL4'00000072'                    G -
000200  45 E0 F 008              BAL    14,   8( 0,15)
000204  0450D074                 DC    XL4'0450D074'                    I
000208  45 E0 F 010              BAL    14,  16( 0,15)
00020C  58 00 D 074          2   L      0, 116( 0,13)                    I
000210  1A 04                    AR     0, 4
000212  50 00 D 074              ST     0, 116( 0,13)                    I
000216  18 28              108   LR     2, 8
000218  5B 20 D 074              S      2, 116( 0,13)                    I
00021C  47 40 D 18C              BC     4, 396( 0,13)                    7
000220  47 80 D 178              BC     8, 376( 0,13)                    4
000224  47 20 D 0C0              BC     2, 192( 0,13)                    3
000228  58 F0 D 08C          4   L      15, 140( 0,13)                   IBCOM=
00022C  45 E0 F 004              BAL    14,   4( 0,15)
000230  00000006                 DC    XL4'00000006'                    F
000234  00000076                 DC    XL4'00000076'                    G+8
000238  45 E0 F 010              BAL    14,  16( 0,15)
00023C  58 F0 D 08C          7   L      15, 140( 0,13)                   IBCOM=
000240  45 E0 F 004              BAL    14,   4( 0,15)
000244  00000006                 DC    XL4'00000006'                    F
```

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

```
                    000248  00000088                DC    XL4'00000088'                          G+Q
                    00024C  45 E0 F 010             BAL   14,  16( 0,15)
                    000250  58 F0 D 08C       109   L     15, 140( 0,13)                         IBCOM=
                    000254  45 E0 F 034             BAL   14,  52( 0,15)
                    000258  05                      DC    XL1'00000005'
                    000259  40                      DC    XL1'00000040'
                    00025A  40                      DC    XL1'00000040'
                    00025B  40                      DC    XL1'00000040'
                    00025C  40                      DC    XL1'00000040'
                    00025D  F0                      DC    XL1'000000F0'
ADDRESS OF EPILOGUE
                    00025E  58 F0 D 08C             L     15, 140( 0,13)
                    000262  45 E0 F 034             BAL   14,  52( 0,15)                         IBCOM=
                    000266  0540                    DC    XL2'0540'
                    000268  404040F0                DC    XL4'404040F0'
ADDRESS OF PROLOGUE
                    00026E  58 F0 3 08C             L     15, 140( 0, 3)
                    000272  45 E0 F 040             BAL   14,  64( 0,15)                         IBCOM=
                    000276  18 D3                   LR    13, 3
                    000278  47 F0 D 098             BC    15, 152( 0,13)
ADCON FOR PROLOGUE
                    000020  0000026E                DC    XL4'0000026E'
ADCON FOR SAVE AREA
                    000024  00000080                DC    XL4'00000080'
ADCON FOR EPILOGUE
                    000080  0000025E                DC    XL4'0000025E'
ADCONS FOR PARAMETER LISTS
                    0000FC  80000120                DC    XL4'80000120'                                A
ADCONS FOR TEMPORARIES
                    000140  00000000                DC    XL4'00000000'
                    000144  00000000                DC    XL4'00000000'
ADCONS FOR 3 BLOCK LABELS
```

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

```
                                        /     MAIN /     SIZE OF PROGRAM 00027C HEXADECIMAL BYTES

NAME     TAG   TYPE  ADD.      NAME     TAG   TYPE  ADD.      NAME     TAG   TYPE  ADD.      NAME     TAG   TYPE  ADD.
   A SFA       R*4  000120         I SF        I*4  000124         J SF        I*4  000128         K SF        I*4  00012C
   L S         I*4  000130      SQRT     XF    R*4  000000    IBCOM=  F  XF    I*4  000000
```

Figure 54. Storage Map

```
     LABEL    ADDR          LABEL    ADDR          LABEL    ADDR          LABEL    ADDR

         3  000170              105  0001D0            1  0001EC            107  0001F0
         2  00020C                4  000228            7  00023C
```

Figure 55. Label Map

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

ESD Card: Four types of ESD cards are generated as follows:

ESD, type 0 - contains the name of the program and indicates the beginning of the object module.

ESD, type 1 - contains the entry point name corresponding to an ENTRY statement in a subprogram.

ESD, type 2 - contains the names of subprograms referred to in the source module by CALL statements, EXTERNAL statements, explicit function references, and implicit function references.

ESD, type 5 - contains information about each COMMON block.

The number 0, 1, 2, or 5 is placed in card column 25.

RLD Card: An RLD card is generated for external references indicated in the ESD, type 2 cards. To complete external references, the linkage editor matches the addresses in the RLD card with external symbols in the ESD card. When external references are resolved, the storage at the

address indicated in the RLD card contains the address assigned to the subprogram indicated in the ESD, type 2 card. RLD cards are also generated for a branch list produced for statement numbers.

TXT Card: The TXT card contains the constants and variables used by the programmer in his source module, any constants and variables generated by the compiler, coded information for FORMAT statements, and the machine instructions generated by the compiler from the source module.

END Card: One END card is generated for each compiled source module. This card indicates the end of the object module to the linkage editor, the relative location of the main entry point, and the length (in bytes) of the object module.

OBJECT MODULE DECK STRUCTURE: Figure 56 indicates the FORTRAN object module deck structure.

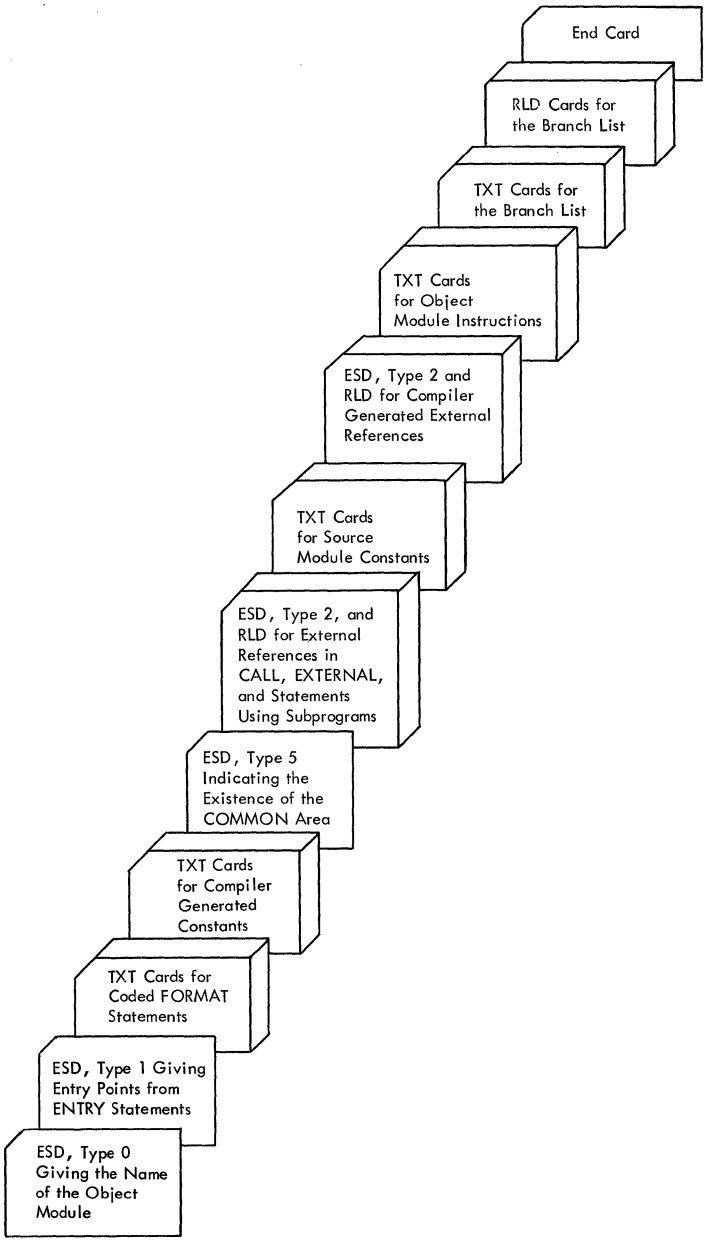


Figure 56. Object Module Deck Structure

Source Module Diagnostics

Two types of diagnostic messages are written by the compiler - informative and error/warning messages.

Source Module Informative Messages: Source module informative messages inform the programmer or operator of the status of the compiler. A message is generated when the compilation has begun, and when the compiler options are processed. For a description of these messages, see Appendix D.

Source Module Error/Warning Messages: All error/warning messages produced are written in a group following the source module listing and object module name table. Figure 57 shows the format of each message as it is written in the data set specified by the SYSPRINT DD statement.

In addition, following the statement in which a serious error is detected, the following appears in the source listing:

ERROR DETECTED - SCAN POINTER = X

where

X

represents the position of the character pointed to by the compiler's scan pointer at the time the error is detected. Any FORTRAN key words and/or meaningless blanks are ignored in determining the position of the character. (If the statement is found to be invalid during the classification process, the value of x always equals one.)

```
                ERROR NO.    ERROR MESSAGE
   ISN a
 LABEL b    IEKxxxI          message
  NAME c
```

a	is the internal statement number of either the statement in error or the statement following the last previous executable statement.
b	is a source label (statement number)
c	is a variable name
xxx	is a three-digit message number
message	is the actual message printed

Figure 57. Format of Diagnostic Messages

There are three types of messages: (1) a terminal error message, (2) serious error messages, and (3) warning messages. The terminal error message returns a condition code of 16; the serious error messages a code of 8; and the warning messages a code of 4. For a description of error/warning 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 cross reference list and a map if the XREF option is specified. The linkage editor also produces diagnostic messages, which are discussed in the Linkage Editor publication.

Module Map

The module map is written in the data set specified in the SYSPRINT DD statement for the linkage editor. To the linkage editor, each program (main or subprogram) and each COMMON (blank or named) block is a control section.

Each control section name is written along with origin and length of the control section. For a program and named COMMON, the name is listed; for blank COMMON, the name $BLANKCOM is listed. The origin and length of a control section is written in hexadecimal numbers. A segment number is also listed for overlay structures (see the Linkage Editor 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 58 shows a load module map produced for the program in Figure 50.

Cross-Reference List

If the linkage editor XREF option is specified, a cross reference list is written with the module map. This cross reference list gives the location from which an external reference is made, the symbol externally referred to in this control section, the control section in which the symbol appears, and the segment number of the control section in which the symbol appears. Unless the linkage editor is building an overlay structure, the cross reference list appears after the module map for all control sections. Figure 59 shows the cross reference list produced for the program in Figure 50.

```
CONTROL SECTION                          ENTRY
  NAME     ORIGIN  LENGTH                 NAME   LOCATION    NAME   LOCATION    NAME   LOCATION    NAME   LOCATION
MAIN          00     27C
IHCSSQRT*    280      AC
                                         SQRT        280
IHCFCOMH*    330     FFD
                                         IBCOM=      330    FDIOCS=     3EC
IHCUOPT *   1330       8
IHCTRCH *   1338     258
IHCFCVTH*   1590     FF3
                                         ADCON=     1590    FCVZO      160C    FCVAO      1782    FCVLO      180A
                                         FCVIO      1818    FCVEO      1FBC    FCVCO      2186
IHCFIOSH*   2588     CF2
                                         FIOCS=     2588
IHCUATBL*   3280     638
```

Figure 58. Load Module Map

```
LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION

     138          SQRT           IHCSSQRT
     13C          IBCOM=         IHCFCOMH
     304          IBCOM=         IHCFCOMH
    11D8          ADCON=         IHCFCVTH
    11D0          FIOCS=         IHCFIOSH
    10E0          IHCUOPT        IHCUOPT
    11DC          FCVEO          IHCFCVTH
    11E0          FCVLO          IHCFCVTH
    11E4          FCVIO          IHCFCVTH
    11E8          FCVCO          IHCFCVTH
    11EC          FCVAO          IHCFCVTH
    11F0          FCVZO          IHCFCVTH
    11C8          IHCTRCH        IHCTRCH
    1490          IBCOM=         IHCFCOMH
    1494          ADCON=         IHCFCVTH
    1498          FIOCS=         IHCFIOSH
    2444          IBCOM=         IHCFCOMH
    2680          IHCUATBL       IHCUATBL
    268C          IBCOM=         IHCFCOMH
ENTRY ADDRESS     00
TOTAL LENGTH    3688
```

Figure 59. Linkage Editor Cross Reference List

LOAD MODULE OUTPUT

At execution time, FORTRAN load module diagnostics are generated in three forms - error code diagnostics, program interrupt messages, and operator messages. An error code indicates an input/output error or a misuse of a FORTRAN library function. A program interrupt message indicates a condition that is beyond the capacity of System/360 to correct. An operator message is generated when a STOP or PAUSE is executed.

Error Code Diagnostics and Traceback without Extended Error Message Facility

If an error is detected during execution of a FORTRAN load module, a message and a diagnostic traceback are written in the error message data set (see "FORTRAN Job Processing"). The message is of the form:

```
    message text
TRACEBACK FOLLOWS-ROUTINE ISN REG.  14,
REG.  15, REG.  0, REG.  1
```

These error messages are described in Appendix D. For the error conditions numbered 211 through 214, 217, 219, 220, and 231 through 237, the message will consist of only IHCxxxI where xxx is a 3-digit error code. The errors detected by the FORTRAN mathematical functions will provide message text describing the error condition. The traceback, which follows the error message, is a list of routines in the direct line of call to the routine in error, in reverse order of use. After the traceback is completed, for error messages IHC217I and IHC218I, control is passed to the call routine statement designated in the END and ERR parameters respectively 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, or IHCFIOSH. The error code in the message indicates the actual origin of the error.

Note: For an assembler language program or subprogram, the routine name field in the traceback contains the identifier specified in the SAVE macro instruction or equivalent coding. (If the identifier specified is longer than eight characters, only the first eight appear.) If no identifier is specified, the traceback routine name field is either blank or its contents are meaningless in the traceback.

Internal statement number identifiers are generated for function references and calls when the ID option is specified on the EXEC card for the compile step. These identifiers appear in the traceback, except for FORTRAN calls to IBCOM for which no identifiers are generated. If NOID is specified, no identifiers are generated and the internal statement number field will be blank.

Note: For an assembler language program or subprogram, the internal statement number field contains the value of the binary calling sequence identifier specified in the CALL macro instruction or equivalent coding. If no identifier was specified, the field is either blank or its contents are meaningless in the traceback.

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

At the end of the traceback, whether it was completed or not, the entry point of

```
IHC219I
TRACEBACK FOLLOWS   ROUTINE     ISN     REG. 14      REG. 15     REG. 0      REG. 1
                    IBCOM               820068FC     XXXXXXXX    XXXXXXXX    XXXXXXXX
                    MASTR       010     00005378
                    PAYROLL             00003148     XXXXXXXX    XXXXXXXX    XXXXXXXX
ENTRY POINT = 00005000
```

Figure 60. Sample Traceback for Execution-Time Errors

the main FORTRAN program is given in hexadecimal.

Figure 60 shows the traceback information placed in the error message data set for the following example.

Example: A FORTRAN program PAYROLL calls the subroutine MASTR, which contains a READ statement. The IHCFIOSH routine is called to perform the input operation, but an error condition arises because there is no DD statement for the data set.

Explanation: PAYROLL was entered at location 5000 and called MASTR at internal statement number (ISN) 10 in PAYROLL. IBCOM (in this case, the error occurred in the IHCFIOSH routine) would have returned to location 68FC in MASTR; MASTR would have returned to location 5378 in PAYROLL and PAYROLL would have returned to location 3148 in the supervisor. Execution terminates and a condition code of 16 is returned to the operating system.

Program Interrupt Messages

Program interrupt messages containing the old Program Status Word (PSW) are produced when one of the following occurs:

- Protection Exception (4)
- Addressing Exception (5)
- Specification Exception (6)
- Data Exception (7)
- Fixed-Point Divide Exception (9)
- Exponent-Overflow Exception (C)
- Exponent-Underflow Exception (D)
- Floating-Point Divide Exception (F)

The characters in parentheses following the exceptions are PSW codes that appear in the program interrupt message to indicate the type of exception. Appendix D contains a complete description of the message and its format.

The program interrupt messages are written on a data set specified by the programmer. (See "FORTRAN Job Processing.") Operator intervention is not required for any of these interruptions.

ABEND Dump

If a program interrupt occurs that causes abnormal termination of a load module, that part of main storage belonging to his program and significant registers, indicators, etc., are dumped. (For information about interpreting an ABEND dump, see the 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.

This section describes the error diagnostic facilities available during program execution when the extended error message facility has been requested.

If the extended error message facility is specified at system generation time, the user is provided with more information about errors detected in a FORTRAN program at object time. Such errors are either data dependent or program errors which are not syntactical or semantic in nature.

With the extended error message facility, a short message text is printed along with the error number when an error occurs. The data in error is also printed as a supplement to the message text. An option is available to suppress the printing of messages. A summary error count, printed when a job is completed, informs the user how many errors occurred, even though printing of messages may have been suppressed during the run.

The Traceback map is printed after each error occurrence with subroutine flow traced back to the main program before continuing execution. The Traceback map may be optionally suppressed. Unless the extended error message facility is specified, the Traceback map is printed only for terminal errors.

The facility exists for calling a user-written subroutine to correct erroneous data rather than accept a standard correction when using the extended error message facility. Default and dynamic control of the number of times an error may occur before execution is terminated is also offered.

For each error condition, the user has both dynamic and default control over the action that follows.

FUNCTIONAL CHARACTERISTICS

With the extended error message facility, an error monitor with additional features is called when an error is detected by a FORTRAN module. The error monitor uses the Option Table, which contains an entry for each error condition. The Option Table controls the actions that are taken after the detection of the error.

When the error monitor is called, it is passed such information as the error identification number, the text of the appropriate message for printing on the object error unit, and a pointer to the data in error.

Contained in the Option Table is information about how many messages to print, how many errors to allow before terminating the job, indication of whether or not Traceback is desired, and the address of the user-written corrective subroutine, if one exists.

The Option Table consists of a double-word preface, then a double-word entry for each error condition. If the extended error message facility is not specified, the Option Table is reduced to the preface alone. The format of the Option Table is shown in Figure 61, which also lists default values in the preface fields. An Option Table entry is described in Table 15. Default values for each error condition are shown in Table 16.

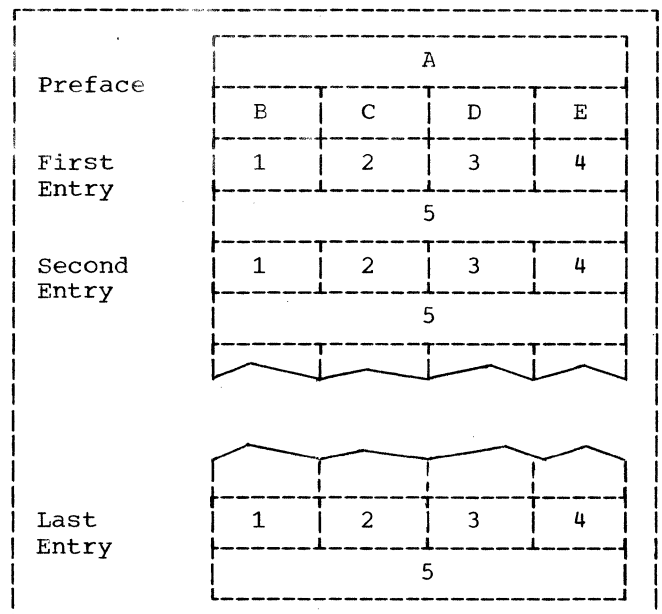
The error monitor tests the Option Table, then acts according to its settings. After printing (or suppressing printing of) the message text and data in error, the action after an error occurrence may be:

1. Terminate the job.
2. Take a standard corrective action, then continue execution.
3. Call a user-written closed subroutine to correct the data in error, then continue execution.

SUBPROGRAMS FOR USING EXTENDED ERROR MESSAGE FACILITY

To aid the programmer in the use of the extended error message facility, several CALL statements may be used in his FORTRAN source program. These statements allow access to the Option Table to alter it dynamically.

All passed parameters, unless otherwise indicated, are 4-byte integers.



Field	Byte Length	Description
A	4	Count of number of entries in the Option Table. Default is 95.
B	1	Boundary alignment switch (bit 1). 0=NOALIGN; 1= ALIGN. Default is ALIGN.
C	1	Error message handling selection indication. FF=NO; OO=YES. Default is YES.
D	1	Align count when error message handling is not selected. Default is 10.
E	1	Not used (reserved).
1	1	Number of errors allowed before termination.
2	1	Number of messages to be printed.
3	1	Number of errors that have occurred.
4	1	Option bits. (See "Option Table Entry Description").
5	4	Address of user exit routine, if specified.

Figure 61. Option Table

Accessing and Altering the Option Table

To access an entire entry from the Option Table without altering it, the programmer issues the following statement:

```
CALL ERRSAV (IERNO,TABENT)
```

where

IERNO
: is an integer equal to the error number to be referenced in the Option Table. Should a negative number be used erroneously, a message will be printed indicating that the number is out of the range of the error table.

TABENT
: is the address of an 8-byte storage area where the accessed table entry is to be stored.

When it is desirable to modify an entry temporarily and save the current entry for later restoration, the CALL ERRSAV statement should be used.

To restore an entry to the Option Table, the statement is:

```
CALL ERRSTR (IERNO,TABENT)
```

where

IERNO
: is an integer equal to the error number for which the entry is to be restored in the Option Table. Should a negative number be used erroneously, a message will be printed indicating that the number is out of range of the error table.

TABENT
: is the address of an 8-byte storage area containing the table entry data.

Note: Certain entries may be protected against alteration because of the way in which the Option Table was originally supplied. If an attempt is made to change an unalterable entry, the request is ignored.

Individual options, numbering up to 5, may be changed in the Option Table by the CALL ERRSET statement. The procedure for altering only one option while leaving the others intact is described in the definition of the parameters. It is accomplished by omitting the final parameter or the last two parameters from the calling sequence, or by supplying zero for a parameter to indicate no change. The CALL ERRSET statement is as follows:

Table 15 OPTION TABLE Entry Description

Field	Length	Default Settings[1]	Description
1	1 byte	10[2]	Contains a number When the count in field 3 matches this number, the job is terminated. The count maximum is 255. A count of zero means unlimited number of occurrences [3] Any count greater than 255 supplied by ERRSET will set this field to zero.
2	1 byte	5[4]	A count of the number of messages to be printed. Message printing is suppressed thereafter. A count of zero means no messages are to be printed.
3	1 byte	0	A count of the number of errors that have occurred where 0 means no errors have occurred.
4	1 byte		8 option bits defined as follows:
	Bit 0	0	Control character indicator: 0 = none, 1 = single space
	1	1	Table entry modifiable: 0=no, 1=yes (See Note 5)
	2	0	Extension of count of field 3
	3	(see Note 6)	Buffer contents to be printed: 0=no, 1=yes
	4	0	Unused
	5	0	Unlimited number of messages allowed: 0=no, 1=yes
	6	1	Traceback required: 0=no, 1=yes
	7	1	Unused
5	4 bytes	1	Address of user's exit routine

[1]The default values shown apply to all error numbers unless excepted by a footnote.
[2]Errors 208, 210, and 215 are set as unlimited, and errors 217 and 230 are set to 1.
[3]When the user sets the count of allowed errors as unlimited, the FORTRAN job may loop endlessly unless the operator intervenes.
[4]Error 210 is set to 10, and errors 217 and 230 are set to 1.
[5]The entry for error 230 is not modifiable.
[6]This entry is set to 0 except for error numbers 212, 215, 218, 221, 222, 223, 224, and 225.

CALL ERRSET (IERNO,INOAL,INOMES,ITRACE, ADDUSE,IRANGE)

where

IERNO
is an integer equal to the error number to be referenced in the Option Table. Should any number that is not within the range of the Option Table be used, a message will be printed.

INOAL
is an integer indicating the number of errors to be allowed before termination of execution. If INOAL is zero or negative, this option is not altered. Any number greater than 255 will mean an unlimited number of occurrences.

INOMES
is an integer indicating the number of messages to be printed A negative value for INOMES is used to suppress all messages; a zero value will leave the entry unaltered. Any number greater than 255 means that the message should always be printed.

ITRACE
is an integer with the value 0, 1, or 2, with these meanings:

0 - Option not to be changed.

1 - A Traceback will not be printed after an error occurrence.

2 - A Traceback will be printed after an error occurrence.

A value for ITRACE which is not 1 or 2 will leave the option unchanged

ADDUSE
is an optional parameter that may contain:

The value 1 as a 4-byte integer, indicating that the Option Table is to be set to show no user exit (i.e., standard corrective action will be used when continuing execution).

The name of a closed subroutine that is to be executed after the occurrence of the error identified by IERNO. This name must appear in an EXTERNAL statement in the source program, and the routine to which control is to be passed must be either in the FORTRAN library or supplied at compile time.

The value 0, indicating that the table entry is not altered.

Table 16. Option Table Default Values

Error Code	Number of Errors Allowed	Number of Messages Allowed	Print Control	Modifiable Entry	Print Buffer Content	Traceback Allowed	Standard Corrective Action	User Exit
207	10	5	NA	Yes	NA	Yes	Yes	No
208	Unlimited	5	NA	Yes	NA	Yes	Yes	No
209	10	5	NA	Yes	NA	Yes	Yes[1]	No[1]
210	Unlimited	10	NA	Yes	NA	Yes	Yes[1]	No
211	10	5	NA	Yes	NA	Yes	Yes	No
212	10	5	No character supplied[2]	Yes	Yes	Yes	Yes	No
213	10	5	NA	Yes	NA	Yes	Yes	No
214	10	5	NA	Yes	NA	Yes	Yes	No
215	Unlimited	5	NA	Yes	Yes	Yes	Yes	No
216	10	5	NA	Yes	NA	Yes	Yes[3]	No
217	1[4]	1	NA	Yes	NA	Yes	Yes	No
218	10[5]	5	NA	Yes	Yes[5]	Yes	Yes	No
219	10[6]	5	NA	Yes	NA	Yes	Yes	No
220	10	5	NA	Yes	NA	Yes	Yes	No
221	10	5	NA	Yes	Yes	Yes	Yes	No
222	10	5	NA	Yes	Yes	Yes	Yes	No
223	10	5	NA	Yes	Yes	Yes	Yes	No
224	10	5	NA	Yes	Yes	Yes	Yes	No
225	10	5	NA	Yes	Yes	Yes	Yes	No
230	1	1	NA	No	NA	Yes	No	No
231	10	5	NA	Yes	NA	Yes	Yes	No
232	10	5	NA	Yes	NA	Yes	Yes	No
233-301	10	5	NA	Yes	NA	Yes	Yes	No

[1]No corrective action is taken except to return to execution. For boundary alignment, the corrective action is part of the support for misalignment. For divide check, the contents of the result register are not altered.

[2]If a print control character is not supplied, the overflow line is not shifted to incorporate the print control character. Thus, if the device is tape, the data is intact, but if the device is a printer, the first character of the overflow line is not printed and is treated as the print control. Unless the user has planned the overflow, the first character would be random and thus the overflow print line control can be any of the possible ones. It is suggested that when the device is a printer, the option be changed to single space supplied.

[3]Corrective action consists of return to execution for SLITE.

[4]If the END parameter is present in a READ statement but the number of occurrences of this error is greater than the number allowed, the END is still honored.

[5]For an I/O error, the buffer may have been partially filled or not filled at all when the error was detected. Thus, the buffer contents could be blank when printed. When an ERR parameter is specified in a READ statement, it is honored even though the error occurrence is greater than the amount allowed.

[6]The count field does not necessarily mean that up to 10 missing DD cards will be detected in a single debugging run, since a single WRITE performed in a loop could cause 10 occurrences of the message for the same missing DD card.

IRANGE
is an optional parameter specifying that the INOAL, INOMES, ITRACE and ADDUSE options are to be applied to the range of error numbers IERNO to IRANGE. If IRANGE is smaller than IERNO, IRANGE is ignored. If IERNO is 212, IRANGE has a special meaning as follows:

1 - Supply single space on overflow line.

If IRANGE is not 1, then the option is set for no print control.

If the print control entry is not to be changed, IRANGE must be omitted from the call to ERRSET.

Obtaining Traceback

Under the extended error message facility, a user may dynamically ask for a Traceback and continued execution. To obtain Traceback, a

```
CALL ERRTRA
```

is issued. The call has no parameters. This statement may not be used when the error message facility has not been specified at system generation time. If the statement is used under such conditions, ERRTRA is assumed to be user-supplied.

USER-SUPPLIED EXIT

Any user-written subprogram may utilize the extended error message facility for error message control. The requirements are:

1. The routine that detects an error will issue a CALL ERRMON statement with appropriate parameters, as described below.

2. An error number is assigned to the error condition and an Option Table entry is made available for the error condition.

The use of the extended error message facility requires certain planning at the installation level. The routine that uses the error monitor for error service should have the status of an installation general purpose function similar to the IBM-supplied mathematical functions. The number of installation error conditions must be known at the time the FORTRAN library is created by system generation so that entries will be provided in the Option Table. The error numbers chosen for user subprograms are restricted in range. IBM-designated error conditions have reserved the error codes from 000 to 301. Error codes for installation-designated error situations must be assigned in the range 302 to 899. The error code is used to find the proper entry in the Option Table.

The CALL ERRMON statement is as follows:

```
CALL ERRMON (IMES,IRETCD,IERNO,DATA1,
  DATA2,...)
```

where

IMES
is the address of an array that contains, in EBCDIC characters, the text of the message to be printed. The first item of the array contains an integer whose value is the length of the message. Thus, the first 4 bytes of the array will not be printed. If the message length is greater than 133 characters, it will be printed on two or more lines of printed output.

IRETCD
is an integer variable that will be set upon return from the error monitor as follows:

0 - The Option Table indicates that standard correction is required.

1 - The Option Table indicates that a user exit to a corrective routine has been executed. The function is to be re-evaluated using arguments supplied in DATA1, DATA2.... For input/output type errors, 1 indicates that the user does not want a standard correction.

IERNO
is the number of the error condition as an integer. This number identifies a unique error situation for which there is an entry in the IERNO field of the Option Table. Should any number that is not within the range of the Option Table be used, a message will be printed.

DATA1
DATA2
are parameters containing the argument or arguments in error upon entry. There must be one parameter for each argument. Upon return, results obtained from user-written corrective action are placed in these variables. Since the variables' content can be altered, they should be used only in the CALL to the error monitor; otherwise, the user of the function may have literals or variables destroyed.

Since DATA1 and DATA2 are the parameters which the error monitor will pass to a user-written routine to correct the detected error, care must be taken to make sure that these parameters agree in type and number in the call to ERRMON and in a user-written corrective routine, if one exists.

Considerations for a User-Supplied Exit Routine

When the error monitor calls a user-written subroutine for corrective action, it uses the equivalent of the following FORTRAN statement:

```
CALL x(IRETCD,IERNO,DATA1,DATA2,...)
```

where

x
is the name of the routine placed in the Option Table. The other parameters are the same as those of ERRMON. If the error occurred during an input/output operation (i.e., error codes 211 to 237), the subroutine x must be one that does not contain any FORTRAN I/O statements -- i.e., READ, WRITE, BACKSPACE, END FILE, REWIND, PDUMP, DEBUG, or ERRTRA.

Although the user-written corrective routine may change the setting of IRETCD, such a change is subject to the following restrictions:

1. If IRETCD is set to 0, then DATA1 and DATA2 must not be altered by the corrective routine, since standard corrective action is requested. If, in fact, DATA1 and DATA2 are altered when IRETCD is set to 0, the operations that follow will have unpredictable results.

2. Only the values 0 and 1 are valid for IRETCD. A user-exit routine must ensure that one of these values is used if it changes the return code setting.

The user-written exit routine can be written in FORTRAN or in assembler language. In either case, it must be able to accept the call to it as shown above. The user-exit routine must be a closed subroutine that returns to the caller.

If a user-written exit routine is written in assembler language, then the end of the parameter list can be checked. The high order byte of the last parameter will be 80. If the routine is written in FORTRAN, the parameter list must match in length the parameter list passed in the CALL to the error monitor.

Meaningful actions the user may take if he wishes to correct an error are described in Tables 17, 18, and 19.

OPTION TABLE CONSIDERATIONS

When a user-written subroutine is to be executed, then the Option Table entry for that error should be altered to contain the address of the user-exit routine.

Although it is not possible to supply user exits when the Option Table is created by system generation, an installation may construct an Option Table to be placed in the FORTRAN library that does contain user-exit addresses. However, this procedure will result in loading all user-exit subroutines mentioned in the Option Table with a V-type adcon.

The user must issue a CALL ERRSET to insert a user-exit address in the Option Table that is created by system generation.

When changing the Option Table, setting error occurrence to be unlimited (i.e., specifying a number greater than 255) may cause the program to loop indefinitely unless the operator intervenes. This use, therefore, should be made with caution.

HOW TO CREATE OR ALTER AN OPTION TABLE

The Option Table supplied during the process of system generation may be altered dynamically for any particular FORTRAN job by the use of the subprograms ERRSET and ERRSTR. To provide a new set of options for the entire installation, the Option Table must be reassembled and link edited into the FORTRAN library after system generation and before the system is used. A procedure for accomplishing this is described below.

An assembler language macro definition can be used to generate an Option Table. The macro definition and use of the macro for each Option Table entry are supplied as input to the assembler procedure ASMFCL to replace the system generated Option Table with the desired one.

An example of the use of such an assembler language macro definition is shown in Figure 62.

Table 17. Corrective Action After I/O Error Occurrence

Error Code	Parameters Passed to User	Standard Corrective Action	User-Supplied Corrective Action
211	A,B,C	Treat format field containing C as end of FORMAT statement	(a) If compiled FORMAT statement, put hexadecimal equivalent of character in C (see Note 1). (b) If variable format, move EBCDIC character into C (See Note 1)
212	A,B,D	Input: Ignore remainder of I/O list. Output: Continue by starting new output record. Supply carriage control character if required by Option Table.	See Note 2
213	A,B,D	Ignore remainder of I/O list	See Note 2
214	A,B,D	Input: Ignore remainder of I/O list. Output: Change record format to V.	If user correction is requested, the remainder of the I/O list is ignored.
215	A,B,E	Substitute zero for the invalid character	The character placed in E will be substituted for the invalid character (see Note 1)
217	A,B,D	Increment FORTRAN sequence number and read next file	See Note 2
218†	A,B,D,F	Ignore remainder of I/O list	See Note 2
219-224	A,B,D	Ignore remainder of I/O list	See Note 2
225	A,B,E	Substitute zero for the invalid character	The character placed in E will be substituted for the invalid character (see Note 1)
231	A,B,D	Ignore remainder of I/O list	See Note 2
232	A,B,D,G	Ignore remainder of I/O list	See Note 2
233	A,B,D	Change record number to list maximum allowed (32,000)	See Note 2
234-236	A,B,D	Ignore remainder of I/O list	See Note 2
237	A,B,D,F	Ignore remainder of I/O list	See Note 2

Meanings:

A - Address of return code field (Integer*4)
B - Address of error number (Integer*4)
C - Address of invalid format character (Logical*1)
D - Address of data set reference number (Integer*4)
E - Address of invalid character (Logical*1)
F - Address of DECB
G - Address of record number requested (Integer *4)

Notes:

1. Alternatively, the user can set the return code to 0, thus requesting a standard corrective action.

2. The user can do anything he wishes except perform another I/O operation - i.e., issue a READ, WRITE, BACKSPACE, END FILE, REWIND, PDUMP, DBUG, or ERRTRA. On return to the Library, the remainder of the I/O request will be ignored.

†If error condition 218 (I/O error detected) occurs while writing error messages on the object error unit, then the job is terminated since there is no place to exhibit the message.

If no DD card has been supplied for the object error unit, error messages cannot be printed and the job is terminated.

Table 18. Corrective Action After Mathematical Subroutines Error Occurrence (Part 1 of 3)

Error Code	FORTRAN Reference	Invalid Argument Range	Options: Standard Corrective Action	Options: User-Supplied Corrective Action (See Note 1)
216	CALL SLITE (I)	I>4	the call is treated as a no operation	I
216	CALL SLITET (I,J)	I>4	J=2	I
241	K=I**J	I=0, J≤0	K=0	I,J
242	Y=X**I	X=0, I≤0	Y=0	X,I
243	DA=D**I	D=0, I≤0	DA=0	D,I
244	XA=X**Y	X=0, Y≤0	XA=0	X,Y
245	DA=D**DB	D=0, DB≤0	DA=0	D,DB
246	CA=C**I	C=0+0i, I≤0	CA=0+0i	C,I
247	CDA=CD*I	C=0+0i, I≤0	CA=0+0i	CD,I
251	Y=SQRT (X)	X<0	$Y=\|X\|^{1/2}$	X
252	Y=EXP (X)	X>174.673	Y=*	X
253	Y=ALOG (X)	X=0 X<0	Y=-* Y=log|X|	X X
	Y=ALOG10 (X)	X=0 x<0	Y=-* $y=\log_{10}\|x\|$	X x
254	Y=COS (X) Y=SIN (X)	$\|X\|\geq 2^{18}*\pi$	$Y=\sqrt{2}/2$	X
255	Y=ATAN2 (X,XA)	X=0, XA=0	Y=0	X,XA
256	Y=SINH (X) Y=COSH (X)	$\|X\|\geq 174.673$	Y=*	X
257	Y=ARSIN (X) Y=ARCOS (X)	|X|>1	Y=0	X
258	Y=TAN (X) Y=COTAN (X)	$\|X\|\geq (2^{18})*\pi$	Y=1	X
259	Y=TAN (X)	X is too close to an odd multiple of $\frac{\pi}{2}$	y=*	X

Variable	Type
I,T	Variables of INTEGER*4
X,XA,I	Variables of REAL*4
D,DA,DB	Variables of REAL*8
C,CA	Variables of COMPLEX*8
Z,X_1,X_2	Complex variables to be given the length of the functioned argument when they appear
CD	Variables of COMPLEX*16

Notes:
1. The user-supplied answer is obtained by recomputation of the function using the value set by the user routine for the parameters listed.
2. The largest number that can be represented in floating point is indicated above by *.

Table 18. Corrective Action After Mathematical Subroutines Error Occurrence (Part 2 of 3)

Error Code	FORTRAN Reference	Invalid Argument Range	Options: Standard Corrective Action	Options: User-Supplied Corrective Action (See Note 1)
	Y=COTAN (X)	X is too close to a multiple of π	Y=*	X
261	DA=DSQRT (D)	D<0	DA=$\|D\|^{1/2}$	D
262	DA=DEXP (D)	D>174.673	DA=*	D
263	DA=DLOG (D)	D=0 D<0	DA=-* DA=log|X|	D
	DA=DLOG10 (D)	D=0 D<0	DA=-* DA=$\log_{10}\|X\|$	D
264	DA=DSIN (D) DA=DCOS (D)	$\|D\| \geq 2^{50} * \pi$	DA=$\sqrt{2}/2$	D
265	DA=DATAN2(D,DB)	D=0,DB=0	DA=0	D,DB
266	DA=DSINH (D) DA=DCOSH (D)	$\|D\| \geq 174.673$	DA=*	D
267	DA=DARSIN (D) DA=DARCOS (D)	$\|D\| > 1$	DA=0	D
268	DA=DTAN (D) DA=DCOTAN (D)	$\|X\| \geq 2^{50} * \pi$	DA=1	D
269	DA=DTAN (D)	D is too close to an odd $\frac{\pi}{2}$ multiple of	DA=*	D
	DA=DCOTAN (D)	D is too close to a multiple of π	DA=*	D
	For errors 271 through 275, $C=X_1+iX_2$			
271	Z=CEXP (C)	$X_1 > 174.673$	Z=*(COS X_2+ SIN X_2)	C
272	Z=CEXP (C)	$\|X_2\| \geq 2^{18} * \pi$	Z=0+0i	C
273	Z=CLOG (C)	C=0+0i	z=-*+0i	C

Variable	Type
I,T	Variables of INTEGER*4
X,XA,I	Variables of REAL*4
D,DA,DB	Variables of REAL*8
C,CA	Variables of COMPLEX*8
Z,X_1,X_2	Complex variables to be given the length of the functioned argument when they appear
CD	Variables of COMPLEX*16

Notes:
1. The user-supplied answer is obtained by recomputation of the function using the value set by the user routine for the parameters listed.
2. The largest number that can be represented in floating point is indicated above by *.

Table 18. Corrective Action After Mathematical Subroutines Error Occurrence (Part 3 of 3)

Error Code	FORTRAN Reference	Invalid Argument Range	Options: Standard Corrective Action	Options: User-Supplied Corrective Action (See Note 1)
274	Z=CSIN (C) Z=CCOS (C)	$\|X_1\| \geq 2^{18} * \pi$	Z=0+0i	C
275	Z=CSIN (C)	$X_2 > 174.673$	$Z = \frac{*}{2}(\text{SIN } X_1 + i\text{COS } X_1)$	C
	Z=CCOS (C)		$Z = \frac{*}{2}(\text{COS } X_1 - i\text{SIN } X_1)$	C
	Z=CSIN (C)	$X_2 < -174.673$	$Z = \frac{*}{2}(\text{SIN } X_1 - i\text{COS } X_1)$	C
	Z=CCOS (C)		$Z = \frac{*}{2}(\text{COS } X_1 + i\text{SIN } X_1)$	C
For errors 281 through 285, CD=X_1+iX_2				
281	Z=CDEXP (CD)	$X_1 > 174.673$	$Z = *(\text{COS } X_2 + i\text{SIN } X_2)$	CD
282	Z=CDEXP (CD)	$\|X_2\| \geq 2^{50} * \pi$	Z=0+0i	CD
283	Z=CDLOG (CD)	CD=0+0i	Z=-*+0i	CD
284	Z=CDSIN (CD) Z=CDCOS (CD)	$\|X_1\| \geq 2^{50} * \pi$	Z=0+0i	CD
285	Z=CDSIN (CD)	$X_2 > 174.673$	$Z = \frac{*}{2}(\text{SIN } X_1 + i\text{COS } X_1)$	CD
	Z=CDCOS (CD)		$Z = \frac{*}{2}(\text{COS } X_1 - i\text{SIN } X_1)$	CD
	Z=CDSIN (CD)	$X_2 < -174.673$	$Z = \frac{*}{2}(\text{SIN } X_1 - i\text{COS } X_1)$	CD
	Z=CDCOS (CD)		$Z = \frac{*}{2}(\text{COS } X_1 + i\text{SIN } X_1)$	CD
290	Y=GAMMA (X)	$X \leq 2^{-252}$ or $X \geq 57.5744$	Y=*	X
291	Y=ALGAMA (X)	$X \leq 0$ or $X \geq 4.2937 * 10^{73}$	Y=*	X
300	DA=DGAMMA (D)	$D \leq 2^{-252}$ or $D \geq 57.5774$	DA=*	D
301	DA=DLGAMA (D)	$D \leq 0$ or $D \geq 4.2937 * 10^{73}$	DA=*	D

Variable	Type
I,T	Variables of INTEGER*4
X,XA,I	Variables of REAL*4
D,DA,DB	Variables of REAL*8
C,CA	Variables of COMPLEX*8
Z,X_1,X_2	Complex variables to be given the length of the functioned argument when they appear
CD	Variables of COMPLEX*16

Notes:
1. The user-supplied answer is obtained by recomputation of the function using the value set by the user routine for the parameters listed.
2. The largest number that can be represented in floating point is indicated above by *.

Table 19. Corrective Action After Program Interrupt Occurrence

Program Interrupt Messages			Options	
Error Code	Parameters Passed to User Exit	Reason for Interrupt (See Note 1)	Standard Corrective Action	User-Supplied Corrective Action
207	D,I	Exponent overflow (Interrupt Code 12)	Result register set to the largest possible floating point number. The sign of the result register is not altered.	User may alter D. See Note 2.
208	D,I	Exponent underflow (Interrupt Code 13)	The result register is set to zero.	User may alter D. See Note 2.
209	None	Divide check, Integer divide (interrupt Code 9), Decimal divide (Interrupt Code 11), Floating point divide (Interrupt Code 15). See Note 3.	There is no standard fixup. Result registers are not touched.	See Note 4.
210	None	Specification interrupt (Interrupt Code 6) occurs for boundary misalignment. Other interrupts occur during boundary alignment adjustment. They will be shown with this error code and the PSW portion of the message will identify the interrupt.	No special corrective action other than correcting boundary misalignments.	See Note 4.

Variable	Type	Description
D	A variable INTEGER*8	This variable contains the contents of the result register after the interrupt.
I	A variable INTEGER*4	This variable contains the "exponent" as an integer value for the number in D. It may be used to determine the amount of the underflow or overflow. The value in I is not the true exponent, but what was left in the exponent field of a floating point number after the interrupt.

Notes:

1. A program interrupt occurs asynchronously. Interrupts are described in IBM System/360 Operating System: Principles of Operation, Form A22-6821.

2. The user exit routine may supply an alternate answer for the setting of the result register. This is accomplished by placing a value for D in the user-exit routine. Although the interrupt may be caused by a long or short floating-point operation, the user-exit routine need not be concerned with this. The user-exit routine should always set a REAL*8 variable and the FORTRAN library will load short or long depending upon the floating-point operation that caused the interrupt.

3. For floating-point divide check, the contents of the result register is shown in the message.

4. The user-exit routine does not have the ability to change result registers after a divide check. The boundary alignment adjustments are informational messages and there is nothing to alter before execution continues.

```
//OPTAB  JOB   1,'SAMPLE MACRO',MSGLEVEL=1   CREATE IHCUOPT
//VER1   EXEC  ASMFC,PARM.ASM=NODECK
//ASM.SYSIN  DD  *
         MACRO
         PREFACE  &ADENT,&ADJST,&SETENT
.*   THIS MACRO GENERATES THE PREFACE TO THE OPTION TABLE AND SETS
.*   GLOBALS FOR SUBSEQUENT CALLS TO THE SETENT MACRO
.*   THE USE OF THIS MACRO GENERATES AN OPTION TABLE AS DEFINED BY IBM
.*   AND ALLOWS CHANGES TO INDIVIDUAL ERROR NUMBERS AS DESIRED, BY USE
.*   OF SETENT
         GBLA  &COUNT,&TOTAL,&SETNR
         LCLA  &A
IHCUOPT  CSECT
&SETNR   SETA  &SETENT
&COUNT   SETA  207           ERROR NUMBER OF FIRST ENTRY IN TABLE
&TOTAL   SETA  &ADENT+301    NUMBER OF LAST ENTRY IN TABLE
&A       SETA  &ADENT+95
         DC    F'&A'         TOTAL NUMBER OF ENTRIES IN TABLE
         DC    B'0&ADJST.000000'
         DC    AL3(0)
         MEND
         MACRO
         SETENT  &E
         GBLA  &COUNT,&TOTAL,&SETNR
         LCLA  &B
&B       SETA  1
&SETNR   SETA  &SETNR-1
.AGAIN   ANOP  START OF LOOP TO GEN ONE ENTRY IN TABLE FOR ERROR NUMBER
         AIF   (&COUNT GT &TOTAL).MEND  HAVE ALL ENTRIES BEEN CREATED
         AIF   (&B LE N'&SYSLIST).TEST
         AIF   (&SETNR EQ 0).DEFAULT
         MEXIT
.TEST    ANOP
.* IF THERE IS NO USER SUPPLIED INFO FOR THIS ERROR NO TAKE DEFAULT
         AIF   (&SYSLIST(&B,1) NE &COUNT).DEFAULT
ERR&COUNT DC   AL1(&SYSLIST(&B,2)) NUMBER OF ERRORS TO ALLOW FR SETENT
         DC    AL1(&SYSLIST(&B,3)) NO OF MSGS TO PRINT FROM SETENT
         DC    X'00'
         DC    X'&SYSLIST(&B,4)'  OPTION BITS SUPPLIED BY SETENT
         DC    F'1'
&COUNT   SETA  &COUNT+1
&B       SETA  &B+1
         AGO   .AGAIN  RETURN TO LOOP
.DEFAULT ANOP IBM DEFAULTS FOR ERRORS NOT INDICATED BY SETENT
.*   IBM SPECIAL CASES FOR MESSAGE COUNT
         AIF   (&COUNT EQ 208).UNLIM
         AIF   (&COUNT EQ 210).UNLIM
         AIF   (&COUNT EQ 215).UNLIM
         AIF   (&COUNT EQ 217).ONE
         AIF   (&COUNT EQ 230).ONE
ERR&COUNT DC   AL1(10)
.BACK1   ANOP
         DC    AL1(5)
.BACK2   ANOP
         DC    X'00'
.*   IBM SPECIAL CASES FOR OPTION BITS
         AIF   (&COUNT EQ 212).SPBITS
         AIF   (&COUNT EQ 215).SPBITS
         AIF   (&COUNT EQ 218).SPBITS
         AIF   (&COUNT EQ 221).SPBITS
         AIF   (&COUNT EQ 222).SPBITS
         AIF   (&COUNT EQ 223).SPBITS
         AIF   (&COUNT EQ 224).SPBITS
         AIF   (&COUNT EQ 225).SPBITS
         DC    X'42'
         AGO   .CONT
.SPBITS  DC    X'52'
.CONT    ANOP
         DC    F'1'
&COUNT   SETA  &COUNT+1
         AGO   .AGAIN RETURN TO LOOP
.UNLIM   ANOP
ERR&COUNT DC   AL1(0)
         AIF   (&COUNT NE 210).BACK1
         DC    AL1(10)
         AGO   .BACK2
.ONE     ANOP
ERR&COUNT DC   AL1(1)
         DC    AL1(1)
         AIF   (&COUNT EQ 217).BACK2
         DC    X'00'
         DC    X'02'
         AGO   .CONT
.MEND    ANOP
         MEND
*        END OF MACRO DEFINITION
*
*        EXAMPLE OF THE USE OF THE MACRO
*
 PREFACE  50,1,2
 SETENT (220,5,2,21),(235,10,5,42),(255,2,0,4)
 SETENT  (300,56,65,3)
         END
/*
 END OF DATA
```

Figure 62. Example of Assembler Language Macro Definition Used to Generate Option Table

The macro parameters are as follows:

PREFACE A,B,C

where

A
is the number of user entries to be created

B
is the boundary alignment desired, with these values

0 = no alignment

1 = alignment

C
is the number of times the SETENT macro is to be issued.

SETENT (A,B,C,D)

where

A
is the error entry to be altered

B
is the count of errors to allow. (A specification of 0 means unlimited error occurrence.)

C
is the count of the number of times the message should be printed before suppression.

D
is two hexadecimal digits that fill the option code field.

The macro instructions are used as follows:

1. Only one PREFACE macro instruction is allowed.

2. As many SETENT macro instructions as are desired may be used. From one to 200 error entries can be specified in the use of a single SETENT macro instruction by using continuation cards.

3. Only error entries that differ from the default options need be mentioned. The default options will be the same as those listed in Table 16.

4. Error codes must be placed in ascending order in the SETENT macro instruction. For IBM-supplied entries, error codes are in the range from 207 to 301. User entries are in the range from 302 to 899.

5. If there is an error in the specification of parameters, the entry will be ignored and a diagnostic message will be printed.

Option Table Default Values

The Option Table controls the extended error message facility.

Table 16 shows the default values for the Option Table. If an option recorded in a table entry does not apply to a particular error condition, it is shown as not applicable (NA).

The field that is defined as the user-exit address also serves as a means of specifying standard corrective action. When the table entry contains an address, the user exit is specified; when it is 1, standard correction is specified. It is not possible to generate or create an Option Table entry with the user-exit address specified. The user exit must be specified by altering the Option Table. To specify that no corrective action -- either standard or user-written -- is to be taken, the table entry must specify that only one error is to be allowed before termination of execution.

ERRORS IN USE OF FACILITY

When the extended error message facility recognizes a situation or request that requires user notification, an informational message is printed.

The error monitor is not recursive; if it has already been called for an error, it cannot be re-entered in the event that the user-written corrective routine causes any of the error conditions that are listed in the Option Table. Boundary misalignment is therefore not allowed in a user-exit routine.

PROGRAMMING EXAMPLE

The programming example in Figure 63 shows how features of the error message facility are used.

A FORTRAN job utilizes a user-supplied library subprogram which makes use of the error message facility to handle a divide by zero situation. A user-written routine is supplied to take corrective action after the detection of an error. Comments in the FORTRAN program describe what is being done.

```
//SAMPLE JOB      1,SAMPLE, MSGLEVEL=1
//STEP1  EXEC     FORTHCLG
//FORT.SYSIN DD *
C       MAIN PROGRAM THAT USES THE SUBROUTINE DIVIDE
        COMMON E
        EXTERNAL FIXDIV
C       SET UP OPTION TABLE WITH ADDRESS OF USER EXIT
        CALL ERRSET(302,30,5,1,FIXDIV)
       E=0
C       GET VALUES TO CALL DIVIDE WITH
        READ(5,10) A,B
        IF(B) 1,2,1
2       E=1.0
1       CALL DIVIDE(A,B,C)
        WRITE(6,10)C
10      FORMAT('1'2E20.8)
        STOP
        END
        SUBROUTINE DIVIDE(A,B,C)
C       ROUTINE TO PERFORM THE CALCULATION C=A/B
C       IF B=0 THEN USE ERROR MESSAGE FACILITY TO SERVICE ERROR
C       PROVIDE MESSAGE TO BE PRINTED
        DIMENSION MES(4)
        DATA MES(1)/12/,MES(2)' DIV'/,MES(3)/'302I/,MES(4)/' B=0'/
        DATA RMAX/Z7FFFFFFF/
C       MESSAGE TO BE PRINTED IS
C       DIV302I B=0
c       ERROR CODE 302 IS AVAILABLE AND ASSIGNED TO THIS ROUTINE
C       STEP1 SAVE A,B IN LOCAL STORAGE
        D=A
        E=B
C       STEP2   TEST FOR ERROR CONDITION
100     IE(E) 1,2,1
C       NORMAL CASE -- COMPUTE FUNCTION
1       C=D/E
        RETURN
C       STEP3  ERROR DETECTED CALL ERROR MONITOR
2       CALL ERRMON(MES,IRETCD,302,D,E)
C       STEP 4   BE READY TO ACCEPT A RETURN FROM THE ERROR MONITOR
        IF(IRETCD) 5,6,5
C        IF IRETCD=) STANDARD RESULT IS DESIRED
C       STANDARD RESULT WILL BE C=LARGEST NUMBER IF D IS NOT ZERO
C       CR C=0 IF E=0 AND D=0
6       IF(D) 7,8,7
8       C=0.0
        GO TO 9
7       C=RMAX
9       RETURN
C       USER FIX UP INDICATED. RECOMPUTE WITH NEW VALUE PLACED IN E
5       GO TO 100
        END
        SUBROUTINE FIXDIV(IRETCD,INO,A,B)
C       THIS IS A USER EXIT TO SERVE THE SUBROUTINE DIVIDE
C       THE PARAMETERS IN THE CALL MATCH THOSE USE IN THE CALL TO
C       ERRMON MADE BY SUBROUTINE DIVIDE
C     STEP1  IS ALTERNATE VALUE FOR B AVAILABLE -- MAIN PROGRAM
C       HAS SUPPLIED A NEW VALUE IN E.  IF E=0 NO NEW VALUE IS AVAILABLE
        COMMON E
        IF(E) 1,2,1
C       NEW VALUE AVAILABLE TAKE USER CORRECTION EXIT
1       B=E
        RETURN
C       NEW VALUE NOT AVAILABLE USE STANDARD FIX UP
2       IRETCD=0
        RETURN
        END
/*
//GO.SYSIN  DD *
              0.1E00
/*
```

Figure 63. Sample Program Using Extended Error Message Facility

FORTRAN can be invoked by a problem program through the use of the CALL, ATTACH, or LINK macro instructions.

The program must supply to the FORTRAN compiler:

- The information usually specified in the PARM parameter of the EXEC statement.
- The ddnames of the data sets to be used during processing by the FORTRAN compiler.

Name	Operation	Operand
[name]	{LINK / ATTACH}	EP=IEKAA00, PARAM=(optionaddr [,ddnameaddr]),VL=1
[name]	CALL	IEKAA00, (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. 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. 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.

The sequence of the 8-byte entries in the ddname list is as follows:

Entry	Alternate Name
1	SYSLIN
2	00000000
3	00000000
4	00000000
5	SYSIN
6	SYSPRINT
7	SYSPUNCH
8	SYSUT1
9	SYSUT2

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

APPENDIX B: EXAMPLES OF JOB PROCESSING

The following examples show several methods to process load modules.

Example 1

Problem Statement: A previously created data set SCIENCE.MATH.MATRICES contains a set of 80 matrices. Each matrix is an array containing real*4 variables. The size of the matrices varies from 2x2 to 25x25; the average size is 10x10. The matrices are inverted by a load module MATINV in the library MATPROGS. Each inverted matrix is written (assume FORMAT control) as a single record on the data set SCIENCE.MATH.INVMATRS.

The I/O flow for the example is shown in Figure 64. The job control statements used to define this job are shown is Figure 65.

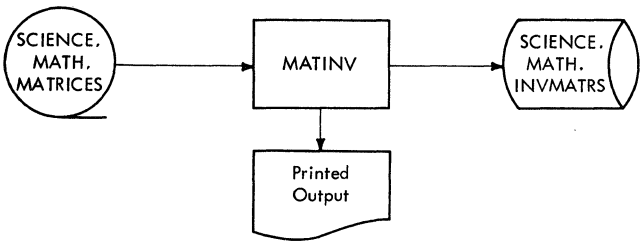


Figure 64. Input/Output Flow for Example

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 written in the SYSOUT data set.

The JOBLIB DD statement indicates that the private library MATPROGS is concatenated with the system library.

The EXEC statement indicates that the load module MATINV is executed.

DD statement FT08F001 identifies the input data set, SCIENCE.MATH.MATRICES. (Data set reference number 8 is used to read the input data set.) Because this data set has been previously created and cataloged, no information other than the data set name and disposition has to be supplied.

DD statement FT10F001 identifies the printed output. (Data set reference number 10 is used for printed output.)

DD statement FT04F001 defines the output data set. (Data set reference number 4 is used to write the data set containing the inverted matrices.) Because the data set is created and cataloged in this job step, a complete data set specification is supplied.

The DSNAME parameter indicates that the data set is named SCIENCE.MATH.INVMATRS. The DISP parameter indicates that the data

Sample Coding Form

```
1-10      11-20     21-30     31-40     41-50     51-60     61-70     71-80
//INVERT JOB 537,JOHNSMITH,MSGLEVEL=1
//JOBLIB DD DSNAME=MATPROGS,DISP=OLD
//INVERT EXEC PGM=MATINV
//FT08F001 DD DSNAME=SCIENCE.MATH.MATRICES,DISP=OLD
//FT10F001 DD SYSOUT=A
//FT04F001 DD DSNAME=SCIENCE.MATH.INVMATRS,                              1
//          DISP=(NEW,CATLG),UNIT=DACLASS,VOLUME=SER=1089W,              2
//          SPACE=(408,(80,9),RLSE,CONTIG,ROUND),SEP=FT08F001,           3
//          DCB=(RECFM=VB,LRECL=2504,BLKSIZE=2508
```

Figure 65. Job Control Statements for Example 1

set is new and is to be cataloged. The SPACE parameter indicates that space is reserved for 80 records, 408 characters long (80 matrices of average size). When space is exhausted, space for 9 more records is allocated. The space is contiguous; any unused space is released, and allocation begins and ends on cylinder boundaries.

The DCB parameter indicates variable-length records, because the size of matrices vary. The record length is specified as 2504, the maximum size of a variable-length record. (The maximum size of a record in this data set is the maximum number of elements (625) in any matrix multiplied by the number of bytes (4) allocated for an element, plus 4 for the segment control word (SCW) that indicates the count of the number of data bytes contained in the record.) The buffer length is specified as 2508 (the 4 bytes are for the block control word (BCW) that contains the length of the block).

The SEP parameter indicates that read and write operations should take place on different channels.

Example 2

Problem Statement: A previously created data set RAWDATA contains raw data from a test firing. A load module PROGRD refines data by comparing the data set RAWDATA against a forecasted result, PROJDATA. The output of PROGRD is a data set &REFDATA, which contains the refined data.

The refined data is used to develop values from which graphs and reports can be generated. The load module ANALYZ contains a series of equations and uses a previously created and cataloged data set PARAMS which contains the parameters for these equations. ANALYZ creates a data set &VALUES, which contains intermediate values.

These values are used as input to the load module REPORT, which prints graphs and reports of the data gathered from the test firing. Figure 1 in the "Introduction" shows the I/O flow for the example. Figure 66 shows the job control statements used to process this job.

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

Explanation: The JOB statement indicates the programmer's name, JOHN SMITH, and that control statements and control statements errors are written in the SYSOUT data set.

The JOBLIB DD statement indicates that the private library FIRING is concatenated with the system library.

The EXEC statement STEP1 defines the first job step in the job and indicates that the load module PROGRD is executed.

The DD statements FT10F001 and FT11F001 identify the data sets containing raw data (RAWDATA) and the forecasted result (PROJDATA), respectively.

DD statement FT12F001 defines a temporary data set, &REFDATA, created for input to the second step. (In the load module, data set reference number 12 is used to write &REFDATA.) The DISP parameter indicates that a data set is new and is passed. The data set is written using the device class TAPECLS. The VOLUME parameter indicates that the volume identified by serial number 2107 is used for this data set. The DCB parameter indicates that the volume is written using high density; the records are fixed-length with FORMAT control and the buffer length is 400.

The EXEC statement STEP2 defines the second job step in the job and indicates that the load module ANALYZ is executed.

DD statement FT17F001 identifies the data set which contains refined data. The DISP parameter indicates that the data set is deleted after execution of this job step. The DD statement FT18F001 identifies the previously created and cataloged data set PARAMS.

DD statement FT20F001 defines the temporary data set &VALUES containing the intermediate values. The DISP parameter indicates that the data set is created in this step, and that it is passed to the next job step. The data set is written on volume 2108 using one of the devices assigned to the class TAPECLS. The DCB parameter indicates high density and fixed-length blocked records (written under FORMAT control). Each record is 204 charac- ters long.

The EXEC statement STEP3 defines the third job step and indicates that the load module REPORT is executed. DD statement FT08F001 identifies the data set containing intermediate values.

DD statement FT06F001 indicates that the data set reference number 06 is used to print the reports and graphs for job step three.

Sample Coding Form

```
1-10      11-20     21-30     31-40     41-50     51-60     61-70     71-80
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
//TESTFIRE JOB ,JOHNSMITH,MSGLEVEL=1
//JOBLIB DD DSNAME=FIRING,DISP=(OLD,PASS)
//STEP1 EXEC PGM=PROGRD
//FT10F001 DD DSNAME=RAWDATA,DISP=OLD
//FT11F001 DD DSNAME=PROJDATA,DISP=OLD
//FT12F001 DD DSNAME=&REFDATA,DISP=(NEW,PASS),UNIT=TAPECLS,                1
//              VOLUME=(,RETAIN,SER=2107),                                 2
//              DCB=(DEN=2,RECFM=F,BLKSIZE=400)
//STEP2 EXEC PGM=ANALYZ
//FT17F001 DD DSNAME=*.STEP1.FT12F001,DISP=OLD
//FT18F001 DD DSNAME=PARAMS,DISP=OLD
//FT20F001 DD DSNAME=&VALUES,DISP=(NEW,PASS),UNIT=TAPECLS,                 1
//              DCB=(DEN=2,RECFM=F,BLKSIZE=204),VOLUME=SER=2108
//STEP3 EXEC PGM=REPORT
//FT08F001 DD DSNAME=*.STEP2.FT20F001,DISP=OLD
//FT06F001 DD UNIT=PRINTER
```

Figure 66. Job Control Statements for Example 2

Example 3

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

Problem Statement: Figure 67 shows a block diagram illustrating the logic for this problem.

A card data set read from the input stream is used to update the star master data set. Each record (detail record) in this data set contains:

1. The star identification field of the star master record that the detail record is used to update.
2. Six variables that are to be used to update the star master.

The following conventions must be observed processing this data set:

1. The star master record that contains the record location counter pointing to space reserved for chained records is assigned to record location 1.
2. A zero in the chain variable indicates that the end of a chain has been reached.
3. The first variable in each star master record is the star identification field; the second variable in each star master is the chain variable.
4. Each record contains six other variables that contain information about that star.

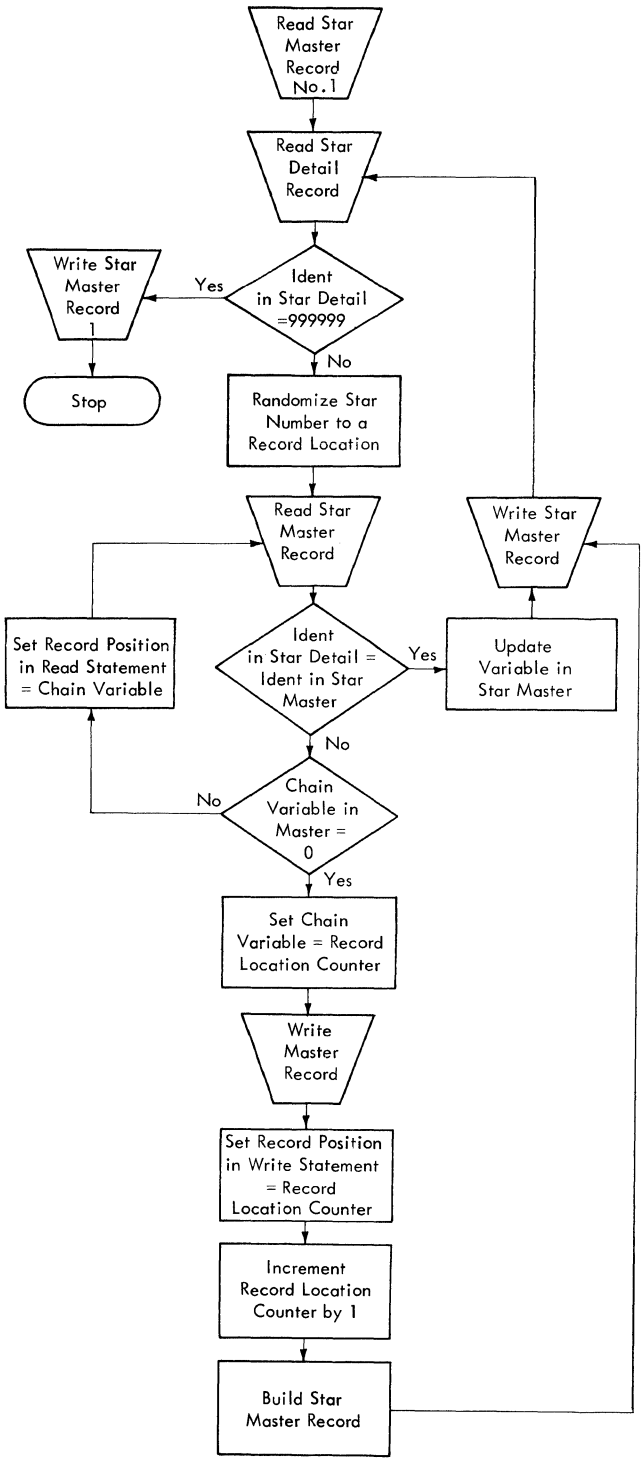


Figure 67. Block Diagram for Example 3

When a star detail record is read, its identification field is randomized, and the appropriate star master record is read. If the correct star master record is found, the record is to be updated. If a star master is not found, then a star master record is to be created for that star.

The last record in the star detail data set contains a star identification number 999999 which indicates that processing the star detail data set is completed.

Explanation: Figure 67 is similar to the diagram shown in Figure 48 except Figure 67 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 67 is adapted to Example 3, while Figure 48 is more general. Figure 69 shows the FORTRAN coding for this program.

The star master record that contains the record counter is read, placing the record location counter in LOCREC. Whenever a detail record is read, the identification variable is checked to determine if the end of the detail data set has been reached. The star detail records contain the variables A, B, C, D, E, and F.

The identification number in the detail record is randomized and the result is placed in the variable NOREC, which is used to read a master record. The master record contains the star identification number (IDSTRM), a chain record location (ICHAIN), and six variables (T, U, V, X, Y, and Z) which are to be updated by the variables in the star detail records. IDSTRM and IDSTRD are compared to see if the correct star master is found. If it is not found, then the variables containing the chain record numbers are followed until the correct star master is found or a new star master is created.

Job Control Statements: The program shown in Figure 69 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 68 shows the job control statements needed to execute the module UPDATE.

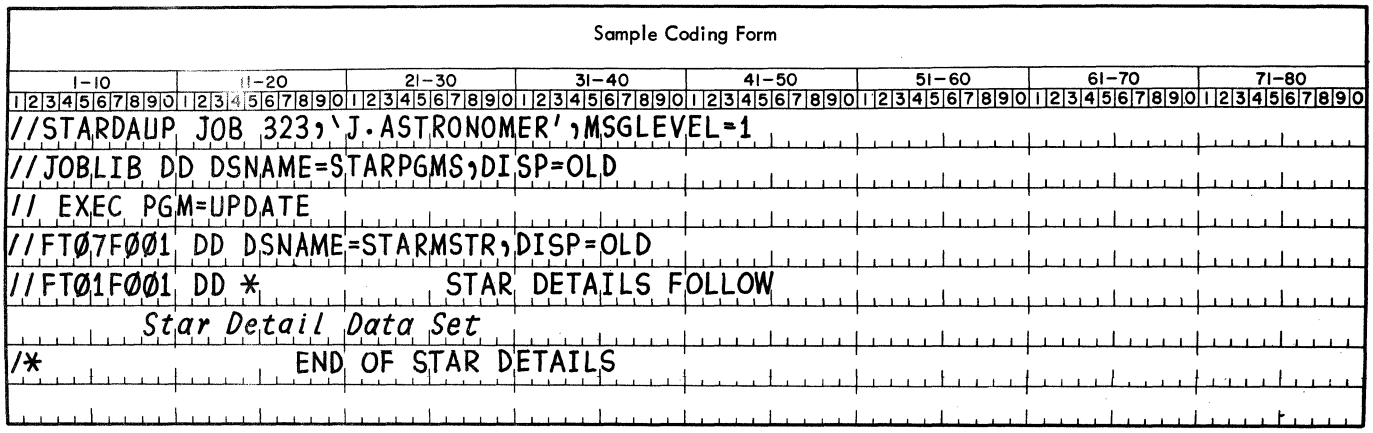
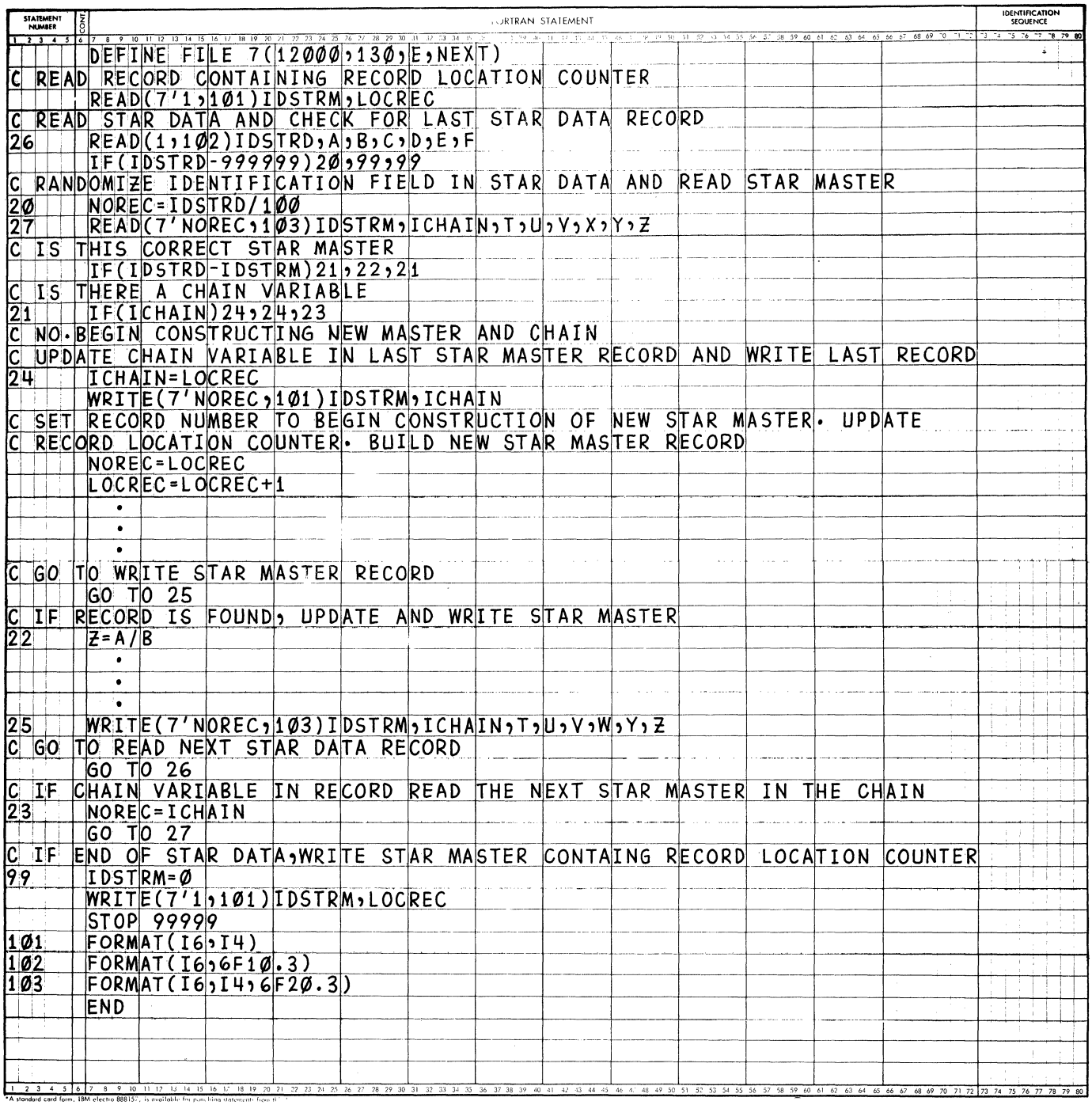


Figure 68. Job Control Statements for Example 3



```
      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,W,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 69. FORTRAN Coding for Example 3

APPENDIX C: ASSEMBLER LANGUAGE SUBPROGRAMS

A FORTRAN programmer can use assembler language subprograms with his FORTRAN main program. This section describes the linkage conventions that must be used by the assembler language subprogram to communicate with the FORTRAN main program. To understand this appendix, the reader must be familiar with the Assembler Language publication 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 on.

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 70 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 20 illustrates the use of the linkage registers.

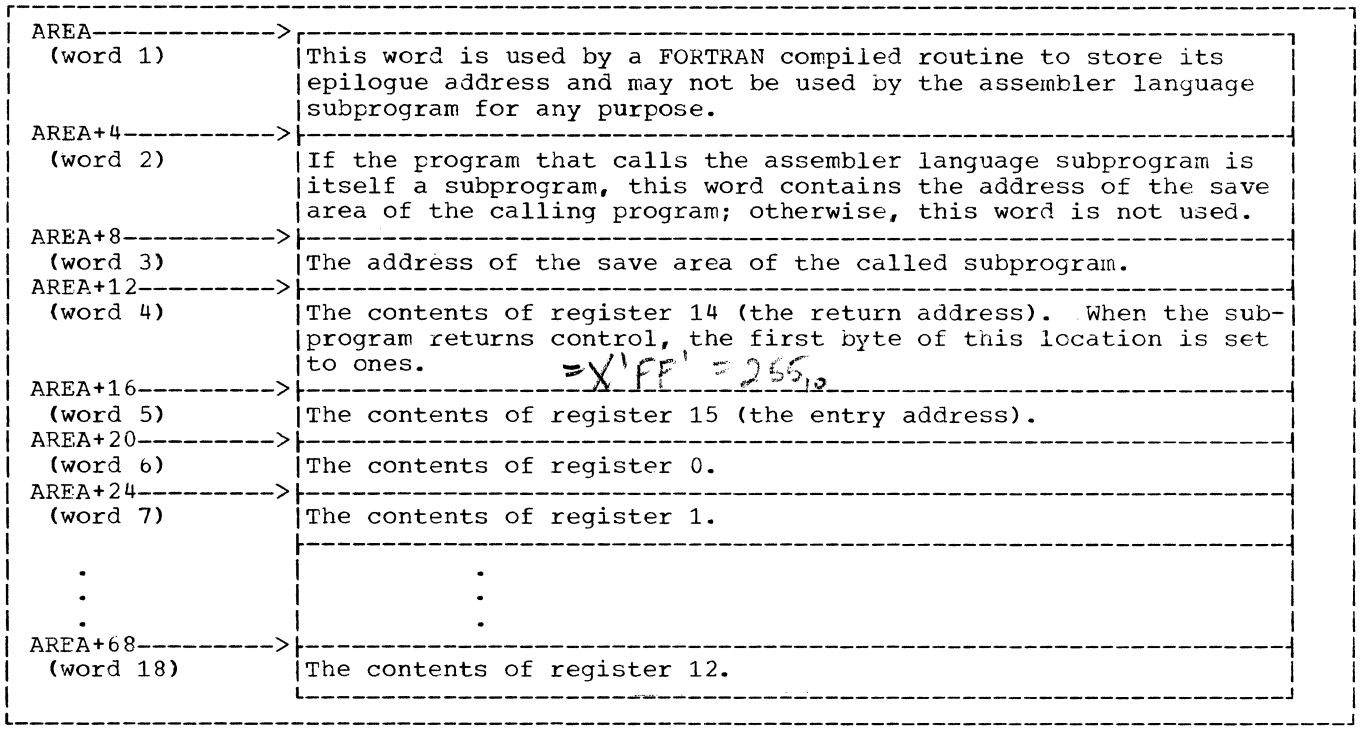


Figure 70. Save Area Layout and Word Contents

Table 20. Linkage Registers

Register Number	Register Name	Function
0	Result Register	Used for function subprograms only. The result is returned in general or floating-point register 0. However, if the result is a complex number, it is returned in floating-point registers 0 (real part) and 2 (imaginary part). (Note: For subroutine subprograms, the result(s) is returned in a variable(s) passed by the programmer.)
1	Argument List Register	Address of the argument list passed to the called subprogram.
2	Result Register	See Function of Register 0.
13	Save Area Register	Address of the area reserved by the calling program in which the contents of certain registers are stored by the called program.
14	Return Register	Address of the location in the calling program to which control is returned after execution of the called program.
15	Entry Point Register	Address of the entry point in the called subprogram. Note: Register 15 is also used as a condition code register, a RETURN code register, and a STOP code register. The particular values that can be contained in the register are 16 - 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 THE ASSEMBLER LANGUAGE SUBPROGRAM

Two types of assembler language subprograms are possible: the first type (lowest level) assembler subprogram does not call another subprogram; the second type (higher level) subprogram does call another subprogram.

Coding a Lowest Level Assembler Language Subprogram

For the lowest level assembler language subprogram, the linkage instructions must include:

1. An assembler instruction that names an entry point for the subprogram.
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 71 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.

Note: If an assembler language main program calls a FORTRAN subprogram, the following instructions must be included in the assembler language program before the FORTRAN subprogram is called:

```
L    15,=V(IBCOM#)
BAL  14,64(15)
```

```
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 71. Linkage Conventions for Lowest Level Subprogram

These instructions cause initialization of return coding and interruption exceptions. If this is not done and the FORTRAN subprogram terminates either with a STOP statement or because of an execution-time error, the data sets opened by FORTRAN are not closed and the result of the termination cannot be predicted. Register 13 must contain the address of the save area that contains the registers to be restored upon termination of the FORTRAN subprogram. If control is to return to the assembler language subprogram, then register 13 contains the address of its save area. If control is to return to the operating system, then register 13 contains the address of its save area.

Figure 72 shows the linkage conventions for an assembler subprogram that calls another assembler subprogram.

```
|Name      |Oper.  |Operand          Comments                                    |
|----------|-------|-------------------------------------------------------------|
|deckname  |START  |0                                                            |
|          |EXTRN  |name2            NAME OF THE SUBPROGRAM CALLED BY THIS SUBPROGRAM
|          |BC     |15,m+1+4(15)                                                 |
|          |DC     |X'm'                                                         |
|          |DC     |CLm'name1                                                    |
|*         |       |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,
|*         |       |                 11, 13, AND 15.                             |
|          |LA     |13,AREA          LOADS THE ADDRESS OF THIS PROGRAM'S SAVE AREA IN 0
|*         |       |                 REGISTER 13.                                |
|          |ST     |13,8(0,Q)        STORES THE ADDRESS OF THIS PROGRAM'S SAVE AREA INTO THE
|*         |       |                 CALLING PROGRAM'S SAVE AREA                 |
|          |ST     |Q,4(0,13)        STORES THE ADDRESS OF THE PREVIOUS SAVE AREA (THE SAVE
|*         |       |                 AREA OF THE CALLING PROGRAM) INTO WORD 2 OF THIS PRO-
|*         |       |                 GRAM'S SAVE AREA                            |
|          |BC     |15,prob1                                                     |
|AREA      |DS     |18F              RESERVES 18 WORDS FOR THE SAVE AREA         |
|*         |       |END OF SAVE ROUTINE                                          |
|          |       |                                                             |
|prob1     |user   |written program statements                                   |
|          |       |                                                             |
|*         |       |CALLING SEQUENCE                                             |
|          |LA     |1,ARGLIST        LOAD ADDRESS OF ARGUMENT LIST               |
|          |L      |15,ADCON                                                     |
|          |BALR   |14,15                                                        |
|          |more   |user written program statements                              |
|*         |       |RETURN ROUTINE                                               |
|          |L      |13,AREA+4        LOADS THE ADDRESS OF THE PREVIOUS SAVE AREA BACK INTO
|*         |       |                 REGISTER 13                                 |
|          |LM     |2,R,28(13)                                                   |
|          |L      |14,12(13)        LOADS THE RETURN ADDRESS INTO REGISTER 14.   |
|          |MVI    |12(13),X'FF'                                                 |
|          |BCR    |15,14            RETURN TO CALLING PROGRAM                   |
|*         |       |END OF RETURN ROUTINE                                        |
|ADCON     |DC     |A(name2)                                                     |
|*         |       |ARGUMENT LIST                                                |
|ARGLIST   |DC     |AL4(arg1)        ADDRESS OF FIRST ARGUMENT                   |
|          | .     |                                                             |
|          | .     |                                                             |
|          | .     |                                                             |
|          |DC     |X'80'            INDICATE LAST ARGUMENT IN ARGUMENT LIST     |
|          |DC     |AL3(argn)        ADDRESS OF LAST ARGUMENT                    |
```

Figure 72. Linkage Conventions for Higher Level 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 73 for that shown in Figure 72.

```
ADCON    DC     A(prob1)
           .
           .
           .
         LA     14,RETURN
         L      15,ADCON
         CNOP   2,4
         BALR   1,15
         DC     AL4(arg1)
         DC     AL4(arg2)
           .
           .
           .
         DC     X'80'
         DC     AL3(argn)
RETURN   BC     0,X'isn'
```

Figure 73. In-Line Argument List

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.

RETRIEVING ARGUMENTS FROM THE ARGUMENT LIST

The argument list contains addresses for the arguments passed to a subprogram. The order of these addresses is the same as the order specified for the arguments in the calling statement in the main program. The address for the argument list is placed in register 1. For example, when the statement

```
CALL MYSUB(A,B,C)
```

is compiled, the following argument list is generated.

00000000	address for A
00000000	address for B
10000000	address for C

For purposes of discussion, A is a real*8 variable, B is a subprogram name, and C is an array.

The address of a variable in the calling program is placed in the argument list. The following instructions in an assembler language subprogram can be used to move the real*8 variable A to location VAR in the subprogram.

```
L      Q,0(1)
MVC    VAR(8),0(Q)
```

where
Q is any general register except 0.

For a subprogram reference, an address of a storage location is placed in the argument list. The address at this storage location is the entry point to the subprogram. The following instructions can be used to enter subprogram B from the subprogram to which B is passed as an argument.

```
L      Q,4(1)
L      15,0(Q)
BALR   14,15
```

where
Q is any general register except 0.

For an array, the address of the first variable in the array is placed in the argument list. An array [for example, a three-dimensional array C(3,2,2)] appears in this format in main storage.

```
C(1,1,1)  C(2,1,1)  C(3,1,1)  C(1,2,1)--┐
┌-------------------------------------┘
└-C(2,2,1)  C(3,2,1)  C(1,1,2)  C(2,1,2)--┐
┌-------------------------------------┘
└-C(3,1,2)  C(1,2,2)  C(2,2,2)  C(3,2,2)
```

Table 21 shows the general subscript format for arrays of 1, 2, and 3 dimensions.

Table 21. 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 integer variables in a three-dimensional array, the displacement (DISP) is calculated to be 28. The following instructions can be used to move the variable,

```
L    Q,8(1)
L    R,DISP
L    S,0(Q,R)
ST   S,ARVAR
```

where

Q and R are any general register except 0.
S is any general register.

APPENDIX D: SYSTEM DIAGNOSTICS

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 - informative and error/warning.

Compiler Informative Messages

Two informative messages are generated by the compiler to inform the programmer or operator of the status of the compilation. The messages are shown with any compiler action taken.

LEVEL- dmthyr OS/360 FORTRAN IV
DATE- yy.ddd/HH.mm.ss

Explanation: This message is generated at the beginning of every compilation. The level date of the compiler is given by "dmthyr", where d is the day of the month (mth) in the year (yr). The number of the day (ddd) in the year (yy) that the compilation takes place is given by "yy.ddd"; the time of day in hours (HH), minutes (mm), and seconds (ss) (based on a 24-hour clock) is given by "HH.mm.ss". The time is also punched into the END card of the object deck.

COMPILER OPTIONS - option{option}...

Explanation: This message occurs for every compilation. Options explicitly specified in the PARM parameter and any default options appear in the message.

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, and any compiler action or user action that is required.

COMPILATION DELETED. n

where n can be 1, 2, 3, 4, 5, 6, or 7

Explanation: The message is generated by the FORTRAN System Director. The compilation is deleted because of the reason indicated by the value of n. (Condition code - 16)

n=1 Phase 10 Program too large to compile

n=2 A program interrupt occurred during execution of the compiler. A program interrupt message and the contents of all registers are written preceding the message.

Figure 74 shows the format of the compile-time program interrupt message when the extended error message facility has not been specified at system generation time. In the old PSW, c is a hexadecimal number that indicates the cause of the interruption; c may be one of the following values:

c	Cause
1	Operation
2	Privileged operation
3	Execute
4	Protection
5	Addressing
6	Specification
7	Data
8	Fixed-point overflow
9	Fixed-point divide
A	Decimal overflow
B	Decimal divide
C	Exponent overflow
D	Exponent underflow
E	Significance
F	Floating-point divide

Following PHASE SWITCH, m is a hexadecimal number that indicates which phase of the compiler was executing when the interrupt occurred; m may be one of the following values:

m	Phase
1	Phase 10
2	Phase 10 (STALL routine)
4	Phase 15 (PHAZ15 routine)
8	Phase 15 (CORAL routine)
10	Phase 20
20	Phase 25
40	Phase 30

n=3 Phase 15 Program too large to compile

n=4 Phase 20 Program too large to compile

n=5 Phase in control requested System Director to terminate compilation immediately. (Any error messages

```
IHC210I  PROGRAM INTERRUPT - OLD PSW IS xxxxxxxcxxxxxxxx - PHASE SWITCH m
```

Figure 74. Compile-Time Program Interrupt Message

generated by the calling phase will also be written.)

n=6 Error detected by IHCFCOMH (IBCOM) I/O error detected during compilation - an IHCxxxI message may also be generated

n=7 End of file, no END statement in source module

IEK001I THE NUMBER OF ENTRIES IN THE ERROR TABLE HAS EXCEEDED THE MAXIMUM.

(Condition code - 8)

IEK002I THE DO LOOPS ARE INCORRECTLY NESTED.

(Condition code - 8)

IEK003I THE EXPRESSION HAS AN INVALID LOGICAL OPERATOR.

(Condition code - 8)

IEK005I THE STATEMENT HAS AN INVALID USE OF PARENTHESES.

(Condition code - 8)

IEK006I THE STATEMENT HAS AN INVALID LABEL.

(Condition code - 8)

IEK007I THE EXPRESSION HAS AN INVALID DOUBLE DELIMITER.

(Condition code - 8)

IEK008I THE EXPRESSION HAS A CONSTANT WHICH IS GREATER THAN THE ALLOWABLE MAGNITUDE.

(Condition code - 8)

IEK009I THE EXPRESSION HAS A NON-NUMERIC CHARACTER IN A NUMERIC CONSTANT.

(Condition code - 8)

IEK010I THE EXPRESSION HAS A CONSTANT WITH AN INVALID EXPONENT.

(Condition code - 8)

IEK011I THE ARITHMETIC OR LOGICAL EXPRESSION USES AN EXTERNAL FUNCTION NAME AS A VARIABLE NAME.

(Condition code - 8)

IEK012I THE EXPRESSION HAS A COMPLEX CONSTANT WHICH IS NOT COMPOSED OF REAL CONSTANTS.

(Condition code - 8)

IEK013I AN INVALID CHARACTER IS USED AS A DELIMITER.

(Condition code - 8)

IEK014I THE STATEMENT HAS AN INVALID NON-INTEGER CONSTANT.

(Condition code - 8)

IEK016I THE GO TO STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

IEK017I THE ASSIGNED OR COMPUTED GO TO HAS AN INVALID ELEMENT IN ITS STATEMENT NUMBER LIST.

(Condition code - 8)

IEK019I THE ASSIGNED GO TO HAS THE OPENING PARENTHESIS MISPLACED OR MISSING.

(Condition code - 8)

IEK020I THE ASSIGNED GO TO HAS AN INVALID DELIMITER FOLLOWING THE ASSIGNED VARIABLE.

(Condition code - 8)

IEK021I THE COMPUTED GO TO HAS AN INVALID COMPUTED VARIABLE.

(Condition code - 8)

IEK022I THE VARIABLE IN THE ASSIGNED GO TO STATEMENT IS NOT INTEGRAL.

(Condition code - 8)

IEK023I THE DEFINE FILE STATEMENT HAS AN INVALID DATA SET REFERENCE NUMBER.

(Condition code - 8)

IEK024I THE DEFINE FILE STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

IEK025I THE DEFINE FILE STATEMENT HAS AN INVALID INTEGER CONSTANT AS THE RECORD NUMBER OR SIZE.

(Condition code - 8)

IEK026I THE DEFINE FILE STATEMENT HAS AN INVALID FORMAT CONTROL CHARACTER.

(Condition code - 8)

IEK027I THE ASSIGN STATEMENT HAS AN INVALID INTEGER VARIABLE.

(Condition code - 8)

IEK028I THE ASSIGN STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

IEK030I THE DO STATEMENT HAS AN INVALID END OF RANGE STATEMENT NUMBER.

(Condition code - 8)

IEK031I THE DO STATEMENT OR IMPLIED DO HAS AN INVALID INITIAL VALUE.

(Condition code - 8)

IEK034I THE ASSIGNMENT STATEMENT BEGINS WITH A NON-VARIABLE.

(Condition code - 8)

IEK035I THE NUMBER OF CONTINUATION CARDS EXCEEDS THE COMPILER LIMIT.

(Condition code - 8)

IEK036I THE STATEMENT CONTAINS INVALID SYNTAX. THE STATEMENT CANNOT BE CLASSIFIED.

(Condition code - 8)

IEK039I THE DEFINE FILE STATEMENT HAS AN INVALID ASSOCIATED VARIABLE.

(Condition code - 8)

IEK040I IT IS ILLEGAL TO HAVE A & STATEMENT NUMBER PARAMETER OUTSIDE A CALL STATEMENT.

(Condition code - 8)

IEK044I ONLY THE CALL, FORMAT, OR DATA STATEMENTS MAY HAVE LITERAL FIELDS.

(Condition code - 8)

IEK045I THE EXPRESSION HAS A LITERAL WHICH IS MISSING A DELIMITER.

(Condition code - 8)

IEK047I THE LITERAL HAS MORE THAN 255 CHARACTERS IN IT.

(Condition code - 8)

IEK050I THE ARITHMETIC IF HAS THE SYNTAX OF THE BRANCH LABELS INCORRECT.

(Condition code - 8)

IEK052I THE EXPRESSION HAS AN INCORRECT PAIRING OF PARENTHESES OR QUOTES.

(Condition code - 8)

IEK053I THE STATEMENT HAS A MISPLACED EQUAL SIGN.

(Condition code - 8)

IEK056I THE FUNCTION STATEMENT MUST HAVE AT LEAST ONE ARGUMENT.

(Condition code - 8)

IEK057I THE STATEMENT HAS A NON-VARIABLE SPECIFIED AS A SUBPROGRAM NAME.

(Condition code - 8)

IEK058I THE SUBPROGRAM STATEMENT HAS AN INVALID ARGUMENT.

(Condition code - 8)

IEK059I THE FUNCTION STATEMENT HAS AN INVALID LENGTH SPECIFICATION.

(Condition code - 8)

IEK062I THE EQUIVALENCE STATEMENT HAS AN ARRAY WITH AN INVALID NUMBER OF SUBSCRIPTS.

(Condition code - 8)

IEK064I THE NAMELIST STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

IEK065I THE NAMELIST STATEMENT HAS A NAMELIST NAME NOT BEGINNING WITH AN ALPHABETIC CHARACTER.

(Condition code - 8)

IEK066I THE NAMELIST STATEMENT HAS A NON-UNIQUE NAMELIST NAME.

(Condition code - 8)

IEK067I THE NAMELIST STATEMENT HAS AN INVALID LIST ITEM.

(Condition code - 8)

IEK069I THE COMMON STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

IEK070I THE EQUIVALENCE STATEMENT HAS A MISSING OR MISPLACED DELIMITER.

(Condition code - 8)

IEK071I THE EQUIVALENCE STATEMENT DOES NOT SPECIFY AT LEAST TWO VARIABLES TO BE EQUIVALENCED.

(Condition code - 8)

IEK072I THE EQUIVALENCE STATEMENT HAS AN INVALID VARIABLE NAME.

(Condition code - 8)

IEK073I THE EQUIVALENCE STATEMENT HAS A SUBSCRIPT WHICH IS NOT AN INTEGER CONSTANT.

(Condition code - 8)

IEK074I THE STATEMENT HAS A VARIABLE WITH MORE THAN SEVEN SUBSCRIPTS.

(Condition code - 8)

IEK075I THE COMMON STATEMENT HAS A VARIABLE THAT HAS BEEN REFERENCED IN A PREVIOUS COMMON STATEMENT.

(Condition code - 8)

IEK076I THE IMPLICIT STATEMENT IS NOT THE FIRST STATEMENT IN A MAIN PROGRAM OR THE SECOND STATEMENT IN A SUBPROGRAM.

(Condition Code - 8)

IEK077I THE IMPLICIT STATEMENT HAS A MISPLACED DELIMITER IN THE TYPE SPECIFICATION FIELD.

(Condition code - 8)

IEK078I THE IMPLICIT STATEMENT HAS AN INVALID TYPE.

(Condition code - 8)

IEK079I THE IMPLICIT STATEMENT HAS A MISSING LETTER SPECIFICATION.

(Condition code - 8)

IEK080I THE IMPLICIT STATEMENT HAS AN INVALID LETTER SPECIFICATION.

(Condition code - 8)

IEK081I THE IMPLICIT STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

IEK082I THE IMPLICIT STATEMENT DOES NOT END WITH A RIGHT PARENTHESIS.

(Condition code - 8)

IEK083I THE IMPLICIT STATEMENT HAS A MISPLACED DELIMITER IN ITS PARAMETER FIELD.

(Condition code - 8)

IEK084I THE IMPLICIT STATEMENT CONTAINS A LITERAL FIELD.

(Condition code - 8)

IEK086I THE COMMON STATEMENT SPECIFIES A NON-VARIABLE TO BE ENTERED.

(Condition code - 8)

IEK087I THE COMMON STATEMENT SPECIFIES A NON-VARIABLE COMMON BLOCK NAME.

(Condition code - 8)

IEK090I THE EXTERNAL STATEMENT HAS A NON-VARIABLE DECLARED AS EXTERNAL.

(Condition code - 8)

IEK091I THE EXTERNAL STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

IEK092I THE TYPE STATEMENT MULTIPLY DEFINES THE VARIABLE.

Condition code - 8)

IEK093I THE TYPE STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

IEK094I THE TYPE STATEMENT HAS A NON-VARIABLE TO BE TYPED.

(Condition code - 8)

IEK095I THE TYPE STATEMENT HAS THE WRONG LENGTH FOR THE GIVEN TYPE.

(Condition code - 8)

IEK096I THE TYPE STATEMENT HAS A MISSING DELIMITER.

(Condition code - 8)

IEK101I THE DO STATEMENT OR IMPLIED DO HAS AN INVALID DELIMITER.

(Condition code - 8)

IEK102I THE BACKSPACE/REWIND/END FILE STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

IEK104I THE BACKSPACE/REWIND/END FILE STATEMENT HAS A DATA SET REFERENCE NUMBER THAT IS EITHER A NON-INTEGER OR AN ARRAY NAME.

(Condition code - 8)

IEK109I THE PAUSE STATEMENT HAS A MISPLACED DELIMITER.

(Condition code - 8)

IEK110I THE PAUSE STATEMENT SPECIFIES A VALUE WHICH IS NEITHER A LITERAL NOR AN INTEGER CONSTANT.

(Condition code - 8)

IEK111I THE PAUSE STATEMENT HAS MORE THAN 255 CHARACTERS IN ITS LITERAL FIELD.

(Condition code - 8)

IEK112I THE DICTIONARY HAS OVERFLOWED.

(Condition code - 16)

IEK115I THE VARIABLE RETURN STATEMENT HAS NEITHER AN INTEGER CONSTANT NOR VARIABLE FOLLOWING THE KEYWORD.

(Condition code - 8)

IEK116I THE DO STATEMENT OR IMPLIED DO HAS AN INVALID PARAMETER.

(Condition code - 8)

IEK117I THE BLOCK DATA STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

IEK120I THE BLOCK DATA STATEMENT WAS NOT THE FIRST STATEMENT OF THE SUBPROGRAM.

(Condition code - 8)

IEK121I THE DATA STATEMENT HAS A VARIABLE WHICH HAS A NON-ALPHABETIC FIRST CHARACTER.

(Condition code - 8)

IEK122I THE DATA STATEMENT CONTAINS A SUBSCRIPTED VARIABLE WHICH HAS NOT BEEN DEFINED AS AN ARRAY.

(Condition code - 8)

IEK123I THE DATA STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 8)

IEK124I THE DATA STATEMENT HAS A VARIABLE WITH AN INVALID INTEGER SUBSCRIPT.

(Condition code - 8)

IEK125I THE DATA STATEMENT HAS A VARIABLE WITH A SUBSCRIPT THAT CONTAINS AN INVALID DELIMITER.

(Condition code - 8)

IEK129I THE STATEMENT CONTAINS AN INVALID DATA CONSTANT.

(Condition code - 8)

IEK132I THE DATA STATEMENT HAS AN INVALID DELIMITER IN ITS INITIALIZATION VALUES.

(Condition code - 8)

IEK133I THE DO STATEMENT CANNOT FOLLOW A LOGICAL IF STATEMENT.

(Condition code - 8)

IEK134I THE DO STATEMENT HAS AN INVALID INTEGER DO-VARIABLE.

(Condition code - 8)

IEK135I THE DO STATEMENT OR IMPLIED DO HAS AN INVALID TEST VALUE.

(Condition code - 8)

IEK136I THE NUMBER OF NESTED DO'S EXCEEDS THE COMPILER LIMIT.

(Condition code - 8)

IEK137I THE DO STATEMENT OR IMPLIED DO HAS AN INVALID INCREMENT VALUE.

(Condition code - 8)

IEK138I THE DO STATEMENT HAS A PREVIOUSLY DEFINED STATEMENT NUMBER SPECIFIED TO END THE DO RANGE.

(Condition code - 8)

IEK139I A LOGICAL IF IS FOLLOWED BY ANOTHER LOGICAL IF OR A SPECIFICATION STATEMENT.

(Condition code - 8)

IEK140I THE IF STATEMENT BEGINS WITH AN INVALID CHARACTER.

(Condition code - 8)

IEK141I THE FORMAT STATEMENT DOES NOT END WITH A RIGHT PARENTHESIS.

(Condition code - 8)

IEK143I THE STATEMENT FUNCTION HAS AN ARGUMENT WHICH IS NOT A VARIABLE.

(Condition code - 8)

IEK144I THE STATEMENT FUNCTION HAS MORE THAN 20 ARGUMENTS.

(Condition code - 8)

IEK145I THE STATEMENT FUNCTION HAS AN INVALID DELIMITER.

(Condition code - 8)

IEK146I THE STATEMENT FUNCTION HAS A MISPLACED EQUAL SIGN.

(Condition code - 8)

IEK147I A STATEMENT FUNCTION DEFINITION MUST PRECEDE THE FIRST EXECUTABLE STATEMENT.

(Condition code - 8)

IEK148I THE DIMENSIONED ITEM HAS A NON-INTEGER SUBSCRIPT.

(Condition code - 8)

IEK149I A VARIABLE TO BE DIMENSIONED USING ADJUSTABLE DIMENSIONS MUST HAVE BEEN PASSED AS AN ARGUMENT AND MUST NOT APPEAR IN COMMON.

(Condition code - 8)

IEK150I THE DIMENSIONED ITEM HAS AN INVALID DELIMITER.

(Condition code - 8)

IEK151I THE STATEMENT SPECIFIES A NON-VARIABLE TO BE DIMENSIONED.

(Condition code - 8)

IEK152I THE SUBPROGRAM STATEMENT HAS AN INVALID DELIMITER IN THE ARGUMENT LIST.

(Condition code - 8)

IEK153I THE STATEMENT HAS AN INVALID NAME SPECIFIED AS A FUNCTION REFERENCE.

(Condition code - 8)

IEK156I THE I/O STATEMENT HAS AN INVALID NAME PRECEDING THE EQUAL SIGN.

(Condition code - 8)

IEK157I THE I/O STATEMENT HAS A NON-VARIABLE SPECIFIED AS A LIST ITEM.

(Condition code - 8)

IEK158I THE I/O STATEMENT HAS AN IMPROPFR PAIRING OF PARENTHESES IN AN IMPLIED DO, OR A NON-INTEGRAL INDEX.

(Condition code - 8)

IEK159I THE FORMAT STATEMENT DOES NOT HAVE A STATEMENT NUMBER.

(Condition code - 4)

IEK160I THE I/O STATEMENT HAS AN INVALID DELIMITER IN THE PARAMETERS.

(Condition code - 8)

IEK161I THE I/O STATEMENT HAS A DUPLICATE PARAMETER.

(Condition code - 8)

IEK163I THE I/O STATEMENT HAS AN ARRAY WHICH IS NOT DIMENSIONED.

(Condition code - 8)

IEK165I THE I/O STATEMENT HAS A PARAMETER WHICH IS NOT AN ARRAY AND NOT A NAMELIST NAME.

(Condition code - 8)

IEK166I THE I/O STATEMENT HAS A NON-INTEGER CONSTANT OR VARIABLE REPRESENTING THE DATA SET REFERENCE NUMBER.

(Condition code - 8)

IEK167I THE STATEMENT HAS AN INVALID USE OF A STATEMENT FUNCTION NAME.

(Condition code - 8)

IEK168I THE STATEMENT SPECIFIES AS A SUBPROGRAM NAME A VARIABLE WHICH HAS BEEN PREVIOUSLY USED AS A NON-SUBPROGRAM NAME.

(Condition code - 8)

IEK169I THE DIRECT ACCESS I/O STATEMENT MAY NOT SPECIFY A NAMELIST NAME.

(Condition code - 8)

IEK170I THE DIRECT ACCESS I/O STATEMENT HAS A NON-INTEGER SPECIFYING THE RECORD'S RELATIVE POSITION.

(Condition code - 8)

IEK171I THE NAME SPECIFIED FOR AN ENTRY POINT HAS ALREADY BEEN USED AS EITHER A VARIABLE SUBROUTINE OR FUNCTION NAME.

(Condition code - 8)

IEK176I THE I/O STATEMENT CONTAINS INVALID SYNTAX IN ITS IMPLIED DO.

(Condition code - 8)

IEK192I THE STATEMENT HAS A LABEL WHICH IS SPECIFIED AS BOTH THE LABEL OF A FORMAT STATEMENT AND THE OBJECT OF A BRANCH.

(Condition code - 8)

IEK193I THE STATEMENT NUMBER HAS BEEN PREVIOUSLY DEFINED.

(Condition code - 8)

IEK194I THE TYPE STATEMENT HAS A MISSING DELIMITER IN THE INITIALIZATION VALUES.

(Condition code - 8)

IEK197I THE STOP STATEMENT HAS A NON-INTEGER CONSTANT AFTER THE KEYWORD.

(Condition code - 8)

IEK199I THE SUBROUTINE OR FUNCTION STATEMENT WAS NOT THE FIRST STATEMENT.

(Condition code - 8)

IEK200I QUOTE LITERALS MAY APPEAR ONLY IN CALL, DATA, FUNCTION AND FORMAT STATEMENTS.

(Condition code - 8)

IEK202I THE STATEMENT HAS A VARIABLE WHICH HAS BEEN PREVIOUSLY DIMENSIONED. THE INITIAL DIMENSION FACTORS ARE USED.

(Condition code - 4)

IEK204I THE STOP STATEMENT HAS AN INVALID DELIMITER.

(Condition code - 4)

IEK205I THE ASSIGNED OR COMPUTED GO TO HAS AN INVALID ELEMENT FOLLOWING THE CLOSING PARENTHESIS.

(Condition code - 4)

IEK206I THE STATEMENT HAS A NON-SUBSCRIPTED ARRAY ITEM.

(Condition code - 4)

IEK207I THE CONTINUE STATEMENT DOES NOT END AFTER THE KEY WORD CONTINUE.

(Condition code - 4)

IEK208I THE CONTINUE STATEMENT DOES NOT HAVE A STATEMENT NUMBER.

(Condition code - 4)

IEK209I THE STATEMENT HAS AN OCTAL CONSTANT SPECIFIED AS AN INITIAL VALUE. THE VALUE IS REPLACED BY ZERO.

(Condition code - 4)

IEK211I THE STATEMENT HAS A COMPLEX CONSTANT WHOSE REAL CONSTANTS DIFFER IN LENGTH.

(Condition code - 4)

IEK212I THE BLOCK DATA SUBPROGRAM CONTAINS EXECUTABLE STATEMENT(S). THE EXECUTABLE STATEMENT(S) IS IGNORED.

(Condition code - 4)

IEK222I THE EXPRESSION HAS A LITERAL WITH A MISSING DELIMITER.

(Condition code - 4)

IEK224I THE STATEMENT AFTER AN ARITHMETIC IF, GO TO, OR RETURN HAS NO LABEL.

(Condition code - 4)

IEK225I A LABEL APPEARS ON A NON-EXECUTABLE STATEMENT. THE LABEL IS IGNORED.

(Condition code - 4)

IEK226I THE STATEMENT HAS A VARIABLE WITH MORE THAN SIX CHARACTERS. THE RIGHTMOST CHARACTERS ARE TRUNCATED.

(Condition code - 4)

IEK229I ALL THE ARGUMENTS OF AN ARITHMETIC STATEMENT FUNCTION ARE NOT USED IN THE DEFINITION.

(Condition code - 4)

IEK302I THE EQUIVALENCE STATEMENT HAS EXTENDED COMMON BACKWARDS.

(Condition code - 8)

IEK303I THE EQUIVALENCE STATEMENT CONTAINS AN ARRAY WHICH IS NOT DIMENSIONED.

(Condition code - 8)

IEK304I THE EQUIVALENCE STATEMENT HAS LINKED BLOCKS OF COMMON TOGETHER.

(Condition code - 8)

IEK305I THE COMMON STATEMENT CONTAINS AN ARRAY WHICH IS NOT DIMENSIONED.

(Condition code - 8)

IEK306I THE EQUIVALENCE STATEMENT HAS AN INCONSISTENCY.

(Condition code - 8)

IEK307I THE DATA STATEMENT CONTAINS A VARIABLE THAT IS NOT REFERENCED.

(Condition code - 4)

IEK308I THE EQUIVALENCE STATEMENT HAS EQUIVALENCED TWO VARIABLES IN THE SAME COMMON BLOCK.

(Condition code - 8)

IEK310I THE EQUIVALENCE STATEMENT HAS A VARIABLE WITH A VARIABLE DIMENSION.

(Condition code - 8)

IEK312I THE EQUIVALENCE STATEMENT CONTAINS AN EXTERNAL REFERENCE.

(Condition code - 8)

IEK314I THE EQUIVALENCE STATEMENT MAY CAUSE WORD BOUNDARY ERRORS.

(Condition code - 4)

IEK315I THE EQUIVALENCE STATEMENT WILL CAUSE WORD BOUNDARY ERRORS.

(Condition code - 4)

IEK317I THE BLOCK DATA PROGRAM DOES NOT CONTAIN A COMMON STATEMENT.

(Condition code - 8)

IEK318I THE DATA STATEMENT IS USED TO ENTER DATA INTO COMMON OUTSIDE A BLOCK DATA SUBPROGRAM.

(Condition code - 8)

IEK319I DATA IS ENTERED INTO A LOCAL VARIABLE IN A BLOCK DATA PROGRAM.

(Condition code - 8)

IEK320I DATA MAY NOT BE ENTERED INTO A VARIABLE WHICH HAS BEEN PASSED AS AN ARGUMENT.

(Condition code - 8)

IEK322I THE COMMON STATEMENT WILL CAUSE WORD BOUNDARY ERRORS.

(Condition code - 4)

IEK323I THE COMMON STATEMENT MAY CAUSE A WORD BOUNDARY ERROR.

(Condition code - 4)

IEK332I THE STATEMENT NUMBER IS UNDEFINED.

(Condition code - 8)

IEK334I THE COMMON STATEMENT HAS A VARIABLE WITH A VARIABLE DIMENSION.

(Condition code - 8)

IEK350I THE DATA STATEMENT HAS A MISSING PARENTHESIS.

(Condition code - 8)

IEK352I THE DATA TYPE STATEMENT HAS TOO MANY INITIALIZATION VALUES.

(Condition code - 4)

IEK353I THE DIMENSION STATEMENT HAS A VARIABLE WHICH HAS A SUBSCRIPT OF REAL MODE.

(Condition code - 8)

IEK355I ADCON TABLE EXCEEDED.

(Condition code - 8)

IEK356I A PARAMETER CANNOT ALSO BE IN COMMON.

(Condition code - 8)

IEK500I AN ARGUMENT TO A FORTRAN SUPPLIED FUNCTION IS OF THE WRONG TYPE. THE FUNCTION IS TREATED AS EXTERNAL.

(Condition code - 4)

IEK501I THE EXPRESSION HAS A COMPLEX EXPONENT.

(Condition code - 8)

IEK502I THE EXPRESSION HAS A BASE WHICH IS COMPLEX BUT THE EXPONENT IS NON-INTEGER.

(Condition code - 8)

IEK503I A NON-SUBSCRIPTED ARRAY ITEM APPEARS IMPROPERLY WITHIN A FUNCTION REFERENCE OR A CALL.

(Condition code - 8)

IEK504I THE BASE AND/OR EXPONENT IS A LOGICAL VARIABLE.

(Condition code - 8)

IEK505I THE INPUT/OUTPUT STATEMENT REFERS TO THE STATEMENT NUMBER OF A NON-FORMAT STATEMENT.

(Condition code - 8)

IEK506I THERE IS A MISSING OPERAND PRECEEDING A RIGHT PARENTHESIS.

(Condition code - 8)

IEK507I A NON-SUBSCRIPTED ARRAY ITEM IS USED AS AN ARGUMENT TO AN IN-LINE FUNCTION.

(Condition code - 8)

IEK508I THE NUMBER OF ARGUMENTS TO AN IN-LINE FUNCTION IS INCORRECT.

(Condition code - 8)

IEK509I THE PROGRAM DOES NOT END WITH A STOP, RETURN, OR GO TO.

(Condition code - 4)

IEK510I THE EXPRESSION HAS A LOGICAL OPERATOR WITH A NON-LOGICAL OPERAND.

(Condition code - 8)

IEK511I THE EXPRESSION HAS A LOGICAL OPERATOR WITH A NON-LOGICAL OPERAND.

(Condition code - 8)

IEK512I THE LOGICAL IF DOES NOT CONTAIN A LOGICAL EXPRESSION.

(Condition code - 8)

IEK515I THE EXPRESSION HAS A RELATIONAL OPERATOR WITH A COMPLEX OPERAND.

(Condition code - 8)

IEK516I THE ARITHMETIC IF CONTAINS A COMPLEX EXPRESSION.

(Condition code - 8)

IEK520I THERE IS A COMMA IN AN INVALID POSITION.

(Condition code - 8)

IEK521I THE EXPRESSION HAS AT LEAST ONE EXTRA RIGHT PARENTHESIS.

(Condition code - 8)

IEK522I THE EXPRESSION HAS AT LEAST ONE TOO FEW RIGHT PARENTHESES.

(Condition code - 8)

IEK523I THE EQUAL SIGN IS IMPROPERLY USED.

(Condition code - 8)

IEK524I THE EXPRESSION HAS AN OPERATOR MISSING AFTER A RIGHT PARENTHESIS.

(Condition code - 8)

IEK525I THE EXPRESSION USES A LOGICAL OR RELATIONAL OPERATOR INCORRECTLY.

(Condition code - 8)

IEK529I A FUNCTION NAME APPEARING AS AN ARGUMENT HAS NOT BEEN DECLARED EXTERNAL.

(Condition code - 8)

IEK530I THE EXPRESSION HAS A VARIABLE WITH AN IMPROPER NUMBER OF SUBSCRIPTS.

(Condition code - 8)

IEK531I THE EXPRESSION HAS A STATEMENT FUNCTION REFERENCE WITH AN IMPROPER NUMBER OF ARGUMENTS.

(Condition code - 8)

IEK541I AN ARGUMENT TO A LIBRARY FUNCTION HAS AN INVALID TYPE.

(Condition code - 8)

IEK542I A LOGICAL EXPRESSION APPEARS IN INVALID CONTEXT.

(Condition code - 8)

IEK550I PUSHDOWN, ADCON, OR ASF ARGUMENT

(Condition code - 16)

IEK552I SOURCE PROGRAM IS TOO LARGE.

(Condition code - 16)

IEK570I TABLE EXCEEDED. OPTIMIZATION DOWNGRADED.

Explanation: The program is too large to permit optimization. This is a warning message and appears in the source listing at the point where the table (RMAJOR) overflows. The compiler performs OPT=1 register allocation only; no other optimization is performed.

(Condition code - 0)

User Response: Either the program should be segmented or the size of the table RMAJOR should be increased. Documentation of how to increase RMAJOR is available through a local IBM branch office.

IEK580I COMPILER ERROR.

Explanation: One of the following four conditions occurred: an invalid adjective code was detected; an illegal element length was detected; no equivalence group was found; an unusual primary adjective code was detected.

(Condition code - 16)

IEK600I INTERNAL COMPILER ERROR. LOGICALLY IMPOSSIBLE BRANCH TAKEN IN A COMPILER SUBROUTINE.

(Condition code - 16)

IEK610I THE STATEMENT NUMBER OR GENERATED LABEL IS UNREACHABLE.

(Condition code - 4)

IEK620I THE STATEMENT NUMBER OR GENERATED LABEL IS A MEMBER OF AN UNREACHABLE LOOP.

(Condition code - 4)

IEK630I INTERNAL TOPOLOGICAL ANALYSIS TABLE EXCEEDED.

(Condition code - 16)

IEK640I COVERAGE BY BASE REGISTER 12 IN OBJECT MODULE EXCEEDED.

(Condition code - 16)

IEK650I INTERNAL ADCON TABLE EXCEEDED.

(Condition code - 16)

IEK660I INTERNAL COMPILER ERROR. TEMPORARY FETCHED BUT NEVER STORED.

(Condition code - 16)

IEK670I LOGICALLY IMPOSSIBLE BRANCH TAKEN IN A COMPILER SUBROUTINE.

(Condition code - 16)

IEK671I LOGICALLY IMPOSSIBLE BRANCH TAKEN IN A COMPILER SUBROUTINE.

(Condition code - 16)

IEK710I THE FORMAT STATEMENT SPECIFIES A FIELD WIDTH OF ZERO.

(Condition code - 8)

IEK720I THE FORMAT STATEMENT CONTAINS AN INVALID CHARACTER.

(Condition code - 8)

IEK730I THE FORMAT STATEMENT HAS UNBALANCED PARENTHESES.

(Condition code - 8)

IEK740I THE FORMAT STATEMENT HAS NO BEGINNING LEFT PARENTHESIS.

(Condition code - 8)

IEK750I THE FORMAT STATEMENT SPECIFIES A COUNT OF ZERO FOR A LITERAL FIELD.

(Condition code - 8)

IEK760I THE FORMAT STATEMENT CONTAINS A MEANINGLESS NUMBER.

(Condition code - 8)

IEK770I THE FORMAT STATEMENT HAS A MISSING DELIMITER.

(Condition code - 8)

IEK780I THE FORMAT STATEMENT CONTAINS A NUMERIC SPECIFICATION GREATER THAN 255.

(Condition code - 8)

IEK800I SOURCE PROGRAM IS TOO LARGE.

(Condition code - 16)

IEK1000I INTERNAL COMPILER ERROR

Explanation: An erroneous error number has been placed in the error table.

(Condition code - 4)

LOAD MODULE EXECUTION DIAGNOSTIC MESSAGES

The load module produces three types of diagnostic messages:

Program interrupt messages.
Execution error messages.
Operator message.

Program Interrupt Messages

Program interrupt messages containing the old Program Status Word (PSW) are written when an exception occurs. The format of the program interrupt message when the extended error message facility has not been specified at system generation time is given in Figure 75.

Note: Codes 4, 5, 6, and 7 are associated with the execution-time adjustment of boundary alignment errors and appear only when the system is generated to provide boundary alignment adjustment; i.e., when BOUNDRY=ALIGN is specified in the FORTLIB macro instruction during system generation.

The letter A in the message indicates that boundary adjustment has taken place. The letter P in the message indicates that the interruption was precise. This will always be the case for non-specification interrupt messages in FORTRAN except when using machines with special hardware on which imprecise interruptions may occur. The eighth character in the PSW (i.e., 4, 5, 6, 7, 9, C, D, or F) represents the code number (in hexadecimal) associated with the type of interruption. The following text describes these interruptions.

Protection Exception: The protection exception (code 4), is recognized when the key of an operand in storage does not match the protection key in the PSW. A message is issued only if a specification exception (code 6) has already been recognized in the same instruction. Otherwise, the job terminates abnormally.

Addressing Exception: The addressing exception (code 5) is recognized when the address of the data is outside of the available storage for the particular installation. A message is issued only if a specification exception (code 6) has already been recognized in the same instruction. Otherwise, the job terminates abnormally.

Specification Exception: The specification exception (code 6) is recognized when a data address does not specify an integral boundary for that unit of information. A specification error would occur during execution of the following instructions.

```
REAL*8 D, E
COMMON A, B, C
EQUIVALENCE (B, D)
D = 3.0D02
```

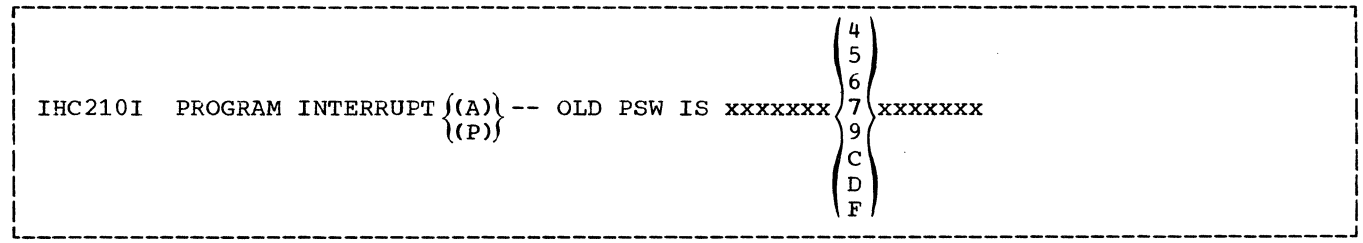


Figure 75. Program Interrupt Message Format Without Extended Error Message Facility

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

Data Exception: The data exception (code 7), is recognized when the sign and digit codes for a CONVERT TO BINARY instruction are incorrect. A message is issued only if a specification exception (code 6) has already been recognized in the same instruction. Otherwise, the job terminates abnormally.

Fixed-Point-Divide Exception: The fixed-point-divide exception, assigned code number 9, is recognized when division of a fixed-point number by zero is attempted. A fixed-point divide exception will occur during execution of the following statements:

```
J=0
I=7
K=I/J
```

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 will occur during execution of the statement:

```
A = 1.0E+75 + 7.2E+75
```

When the interrupt occurs, the result register contains a floating-point number whose fraction is normalized and whose sign is correct. However, the number is not usable for further computation since its characteristic field no longer reflects the true exponent. The content of the result register as it existed when the interrupt occurred is printed following the program interrupt message with the format.

```
REGISTER CONTAINED hhhhhhhhhhhhhhhh
```

where: hhhhhhhhhhhhhhhh is the floating-point number in hexadecimal notation.

If the improved floating-point engineering change is not in effect, the register content cannot be used to calculate the true value.

If the improved floating-point engineering change is in effect, exponent overflow causes "exponent wraparound" - i.e., the characteristic field represents an exponent that is 128 smaller than the correct one. Treating bits 1 to 7 (the exponent characteristic field) of the floating-point number as a binary integer, the true exponent may be computed as follows:

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

Before program execution continues, the FORTRAN library sets the result register to the largest possible floating-point number that can be represented in short precision ($16^{63}*(1-16^{-6})$) or in long precision ($16^{63}*(1-16^{-14})$), but the sign of the result is not changed. The condition code is not altered.

Exponent-Underflow Exception: The exponent-underflow exception, assigned code number D, is recognized when the result of a floating-point addition, subtraction, multiplication, or division is less than 16^{-65} (approximately 5.4×10^{-79}). For example, an exponent-underflow exception will occur during execution of the statement:

```
A = 1.0E-50 * 1.0E-50
```

Although exponent underflows are maskable, FORTRAN jobs are executed without the mask so that the library will handle such interrupts.

When the interrupt occurs, the result register contains a floating-point number whose fraction is normalized and whose sign is correct. However, the number is not usable for further computation since its characteristic field no longer reflects the true exponent. The content of the result register as it existed when the interrupt occurred is printed following the program interrupt message with the format:

```
REGISTER CONTAINED hhhhhhhhhhhhhhhh
```

where: hhhhhhhhhhhhhhhh is the floating-point number in hexadecimal notation.

If the improved floating-point engineering change is not in effect, the exponent underflow always leaves a zero in the result register.

If the improved floating-point engineering change is in effect, exponent underflow causes "exponent wraparound" - i.e., the

characteristic field represents an exponent that is 128 larger than the correct one. Treating bits 1 to 7 (the exponent characteristic field) of the floating-point number as a binary integer, the true exponent may be computed as follows:

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

Before program execution continues, the library sets the result register to a true zero of correct precision. If the interrupt resulted from a floating-point addition or subtraction operation, the condition code is set to zero to reflect the setting of the result register.

Note: The System/360 operating System FORTRAN programmer who wishes to take advantage of the "exponent wraparound" feature and handle the interrupt in his own program must call an assembly language subroutine to issue a SPIE macro instruction that will override the FORTRAN interruption routine.

Floating-Point-Divide Exception: The floating-point-divide exception, assigned code number F, is recognized when division of a floating-point number by zero is attempted. A floating-point divide exception will occur during execution of the following statements:

```
B=0.0
A=1.0
C=A/B
```

Execution Error Messages

Execution error messages have the form:

```
IHCxxxI [message text]
   TRACEBACK FOLLOWS-ROUTINE ISN REG.  14,
     REG.  15, REG.  0, REG.  1
```

The facility for error detection and diagnostic messages is controlled by a system generation option. When the parameter OPTERR=EXCLUDE is specified in the FORTLIB macro-instruction at system generation time (or assumed when no parameter is specified), the features described in "Extended Error Message Facility" are not available.

The following text indicates the messages printed on the object error unit when OPTERR=EXCLUDE. Variable information in the message is shown as X. 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, 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, and error procedures).

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

NAMEL - IHCNAMEL routine (performs namelist processing).

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

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

LIB - SYS1.FORTLIB. In the explanation of the messages, the module name is given followed by the entry point name(s) enclosed in parentheses.

FCVTH - IHCFCVTH routine (performs conversions).

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.

IHC214I

Explanation: FIOCS -- For records in sequential data sets read or written without FORMAT control, for which the RECFM subparameter must be V (variable), either U (undefined) or F (fixed) was specified.

IHC215I CONVERT-ILLEGAL DECIMAL CHARACTER X

Explanation: FCVTH -- An invalid character exists for the decimal

input corresponding to an I, E, F, or D format code.

IHC216I SLITE-SLITET X IS AN ILLEGAL VALUE

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

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. The xxx ... xxx is the character string formatted by the SYNADAF macro instruction. For an interpretation of this information, see the publication 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. (See "Use of ERR Parameter in READ Statement" for additional information.)

IHC219I

Explanation: FIOCS -- Either a data set is referred to in the load module but no DD statement is supplied for it, or a DD statement has an erroneous ddname.

IHC220I

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

IHC221I NAMEL-NAME LARGER THAN EIGHT CHARACTERS. NAME=X

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

IHC222I NAMEL-NAME NOT IN NAMELIST DICTIONARY. NAME=X

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

IHC223I NAMEL-END OF RECORD ENCOUNTERED BEFORE EQUAL SIGN. NAME=X

Explanation: NAMEL -- Either an input variable name or a subscript has no delimiter.

IHC224I NAMEL-SUBSCRIPT FOR NON-DIMENSIONED VARIABLE OR SUBSCRIPT OUT OF RANGE. NAME=X

Explanation: NAMEL -- A subscript is encountered after an undimensioned input name, or the subscript is too big.

IHC225I CONVERT-ILLEGAL HEXADECIMAL CHARACTER X

Explanation: FCVTH -- An illegal character encountered on input under 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: DIOCS -- A READ is executed for a direct access data set that has not been created.

IHC237I

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

IHC241I FIXPI INTEGER BASE=0, INTEGER EXPONENT=X, LE 0

Explanation: LIB -- For an exponentiation operation (I**J) in the subprogram IHCFIXPI(FIXPI#) where I and J represent integer variables or integer constants, I is equal to zero and J is less than or equal to zero.

IHC242I FRXPI REAL*4 BASE=0.0, INTEGER EXPONENT=X, LE 0

Explanation: LIB -- For an exponentiation operation (R**J) in the subprogram IHCFRXPI(FRXPI#), where R represents a real*4 variable or real*4 constant, and J represents an integer variable or integer constant, R is equal to zero and J is less than or equal to zero.

IHC243I FDXPI REAL*8 BASE=0.0, INTEGER EXPONENT=X, LE 0

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 FRXPR REAL*4 BASE=0.0, REAL*4, EXPONENT=X.X, LE 0

Explanation: LIB -- For an exponentiation operation (R**S) in the subprogram IHCFRXPR(FRXPR#), where R and S represent real*4 variables or real*4 constants, R is equal to zero and S is less than or equal to zero.

IHC245I FDXPD REAL*8 BASE=0.0, REAL*8 EXPONENT=X.X, LE 0

Explanation: LIB -- For an exponentiation operation (D**P) in the subprogram IHCFDXPD(FDXPD#), where D and P represent real*8 variables or real*8 constants, D is equal to zero and P is less than or equal to zero.

IHC246I FCXPI COMPLEX*8 BASE=0.0+0.0I, INTEGER EXPONENT=X, LE 0

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

IHC247I FCDXI COMPLEX*16 BASE=0.0+0.0I, INTEGER EXPONENT=X, LE 0

Explanation: LIB -- For an exponentiation operation (Z**J) in the subprogram IHCFCDXI(FCDXI#), where Z represents a complex*16 variable or complex*16 constant and J represents an integer variable or integer constant, Z is equal to zero and J is less than or equal to zero.

IHC251I SQRT NEGATIVE ARGUMENT=X

Explanation: LIB -- In the subprogram IHCSSQRT(SQRT), the argument is less than 0.

IHC252I EXP ARG=X.X, GT 174.673

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

IHC253I ALOG-ALOG10 ARG=X.X, LE ZERO

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

IHC254I SIN-COS/ARG/=/X.X(HEX=X)/, GE PI*2**18

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

IHC255I ATAN2 ARGUMENTS=0.0

Explanation: LIB -- In the subprogram IHCSATN2, when entry name

ATAN2 is used, both arguments are equal to zero.

IHC256I SINH-COSH/ARG/=/X.X/, GE 174.673

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

IHC257I ARSIN-ARCOS/ARG/=/X.X/ GT 1

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

IHC258I TAN-COTAN/ARG/=/X.X(HEX=X)/, GE PI*2**18

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

IHC259I TAN-COTAN/ARG/=/X.X(HEX=X)/, APPROACHES SINGULARITY

Explanation: LIB -- In the subprogram IHCSTNCT (TAN or COTAN), the argument value is too close to one of the singularities. ($\pm\frac{\pi}{2}, \pm\frac{3\pi}{2}, \ldots$ for the tangent or $\pm\pi, \pm 2\pi, \ldots$ for the cotangent)

IHC261I DSQRT NEGATIVE ARGUMENT=X.X

Explanation: LIB -- In the subprogram IHCLSQRT(DSQRT), the argument is less than 0.

IHC262I DEXP ARG=X.X, GT 174.673

Explanation: LIB -- In the subprogram IHCLEXP(DEXP), the argument is greater than 174.673.

IHC263I DLOG-DLOG10 ARG=X.X, LE ZERO

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

IHC264I DSIN-DCOS/ARG/=/X.X(HEX=X)/, GE PI*2**50

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

IHC265I DATAN2 ARGUMENTS=0.0

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

IHC266I DSINH-DCOSH/ARG/=/X.X/, GE 174.673

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

IHC267I DARSIN-DARCOS/ARG/=/X.X/, GT 1

Explanation: LIB -- In the subprogram IHCLASCN (DARSIN or DARCOS), the absolute value of the argument is greater than 1.

IHC268I DTAN-DCOTAN/ARG/=/X.X(HEX=X)/ GE PI*(2**50)

Explanation: LIB -- In the subprogram IHCLTNCT (DTAN or DCOTAN), the absolute value of the argument is greater than or equal to $2^{50}\ \pi$. ($2^{50} \bullet \pi$ =.35371188737802239D+16)

IHC269I DTAN-DCOTAN/ARG/=/X.X(HEX=X)/, APPROACHES SINGULARITY

Explanation: LIB -- In the subprogram IHCLTNCT (DTAN or DCOTAN), the argument value is too close to one of the singularities ($\pm\frac{\pi}{2}, \pm\frac{3\pi}{2}, \ldots$ for the tangent or $\pm\pi, \pm 2\pi, \ldots$ for the cotangent).

IHC271I CEXP REAL ARG=X.X(HEX=X), GT 174.673

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

IHC272I CEXP IMAG ARG=X(HEX=X), ABS VALUE GE PI*2**18

Explanation: LIB -- In the subprogram IHCCSEXP (CEXP), the absolute value of the imaginary part of the argument is greater than or equal to $2^{18} \bullet \pi$. ($2^{18} \bullet \pi$=.82354966406249996D+06)

IHC273I CLOG ARGUMENT=0.0+0.0I

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

IHC274I CSIN-CCOS/REAL ARG/=/X.X (HEX=X)/, GE PI*2**18

Explanation: LIB -- In the subprogram IHCCSSCN (CSIN or CCOS), the absolute value of the real part of the argument is greater than or equal to $2^{18}\ \pi$. ($2^{18}\bullet\pi$=.82354966406249996D+06)

IHC275I CSIN-CCOS/IMAG ARG/=/X.X (HEX=X)/ GT 174.673

Explanation: LIB -- In the subprogram IHCCSSCN (CSIN or CCOS), the absolute value of the imaginary part of the argument is greater than 174.673.

IHC281I CDEXP-REAL ARG=X.X(HEX=X) GT 174.673

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

IHC282I CDEXP IMAG ARG=X.X(HEX=X) ABS VALUE GE PI*2**50

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

IHC283I CDLOG ARGUMENT=0.0+0.0I

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

IHC284I CDSIN-CDCOS/REAL ARG/=/X.X (HEX=X) /, GE PI*2**50

Explanation: LIB -- In the subprogram IHCCLSCN (CDSIN or CDCOS), the absolute value of the real part of the argument is greater than or equal to $2^{50}\bullet\pi$. ($2^{50}\bullet\pi$=.35371188737802239D+16)

IHC285I CDSIN-CDCOS/IMAG ARG/=/X.X (HEX=X) /, GT 174.673

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

IHC290I GAMMA ARG=X.X(HEX=X), LE 2**-252 OR GE 57.5744

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

IHC291I ALGAMA ARG=X.X(HEX=X), LE 0. OR GE 4.2937*10**73

Explanation: LIB -- In the subprogram IHCSGAMA (ALGAMA), the value of the argument is outside the valid range. (Valid range: $0<x<4.2937x10^{73}$)

IHC300I DGAMMA ARG=X.X(HEX=X), LE 2**-252 OR GE 57.5744

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

IHC301I DLGAMA ARG=X.X(HEX=X), LE 0. OR GE 4.2937*10**73

Explanation: LIB -- In the subprogram IHCLGAMA (DLGAMA), the value of the argument is outside the valid range. (Valid range: $0<x<4.2937x10^{73}$)

Extended Error Messages for Execution Errors

When the extended error message facility is specified by use of the OPTERR=INCLUDE option in the FORTLIB macro-instruction at system generation time, the message text that follows is printed on the object error unit. Explanations of individual error situations are the same as those described previously for "Execution Error Messages." Variable information in the message is shown as X, and in the corrective action descriptions, * denotes the largest possible number that can be represented in floating point.

Supplemental data is identified. The user should refer to Tables 17, 18, and 19 for a listing of the data available to the user-exit routine. Provided here is a summary of data available in the message which is also passed to the user-exit routine, as well as any additional information concerning the data in error that appears in the message. The standard action to correct the error is also described.

ERROR	DESCRIPTION
207	IHC207I IBCOM - PROGRAM INTERRUPT-OVERFLOW OLD PSW IS X REGISTER CONTAINED X

Supplemental Data: The following point number before alteration.

Standard Corrective Action: Continue execution at point of interrupt with result register set to the largest possible floating-point number that can be represented in short precision ($16^{63}*(1-16^{-6})$) or in long precision ($16^{63}*(1-16^{-14})$).

208 IHC208I IBCOM - PROGRAM INTERRUPT-UNDERFLOW OLD PSW IS X REGISTER CONTAINED X

Supplemental Data: The floating point number before alteration.

Standard Corrective Action: Continue execution at point of interrupt with result register set to a true zero of correct precision.

209 IHC209I IBCOM - PROGRAM INTERRUPT-DIVIDE CHECK OLD PSW IS X

Supplemental Data: None.

Standard Corrective Action: Leave register unmodified.

210 IHC210I IBCOM - PROGRAM INTERRUPT - ALIGNMENT OLD PSW IS X

Supplemental Data: None.

Standard Corrective Action: None.

IHC210I IBCOM - PROGRAM INTERRUPT ($\begin{matrix}P\\A\end{matrix}$) OLD PSW IS (contents of old PSW)

Supplemental Data: None.

Standard Corrective Action: Continue execution at point of interrupt.

211 IHC211I IBCOM - ILLEGAL COMPILED FORMAT CHARACTER SPECIFIED

IHC211I IBCOM - ILLEGAL VARIABLE FORMAT CHARACTER SPECIFIED X

Supplemental Data: Character in error.

Standard Corrective Action: Format field treated as an end of format.

212 IHC212I IBCOM - FORMATTED I/O, END OF RECORD ON UNIT X

Supplemental Data: Unit number.

Standard Corrective Action: For a read, ignore remainder of I/O list; for a write, start new record with no control character.

213 IHC213I IBCOM - UNFORMATTED READ, END OF RECORD ON UNIT X

Supplemental Data: Unit number.

Standard Corrective Action: Ignore remainder of I/O list.

214 IHC214I FIOCS - UNFORMATTED I/O, RECORD FORMAT SPECIFIED AS F OR U ON UNIT X

Supplemental Data: Unit number.

Standard Corrective Action: For read, ignore I/O request; for write, change record form to V.

215 IHC215I CONVERT - ILLEGAL DECIMAL CHARACTER X

Supplemental Data: Display the record in which character appeared.

Standard Corrective Action: O replaces the character encountered.

Note: If the standard or corrective user action results in a null format, no output will result. If the FORMAT statement is terminated in such a way that no conversion type is called for, an alphanumeric literal may be repeated for each list item.

216 IHC216I SLITE - SLITET X IS AN ILLEGAL VALUE

Supplemental Data: The sense light value supplied.

Standard Corrective Action: For SLITE, no action; for SLITET, return OFF indication, i.e., J=2.

217 IHC217I FIOCS - END OF DATA SET ON UNIT X

Supplemental Data: Unit number.

Standard Corrective Action: Read next file, i.e., increment sequence number by 1.

Note: END=parameter is honored.

218 IHC218I - FIOCS - I/O ERROR (text provided by data management)

Supplemental Data: Unit number.

Standard Corrective Action: Continue execution and ignore I/O request.

Note: ERR=parameter is honored.

219 IHC219I FIOCS - MISSING DD CARD FOR (DDname)

IHC219I DIOCS - MISSING DD CARD FOR UNIT X

Supplemental Data: Unit number.

Standard Corrective Action: Continue execution and ignore I/O request.

220 IHC220I FIOCS - UNIT NUMBER OUT OF RANGE. UNIT=X

IHC220I DIOCS - UNIT NUMBER OUT OF RANGE. UNIT=X

Supplemental Data: Unit number.

Standard Corrective Action: Continue execution and ignore I/O request.

221 IHC221I NAMEL - NAME LARGER THAN EIGHT CHARACTERS. NAME=X

Supplemental Data: Name specified (first eight characters).

Standard Corrective Action: Ignore remainder of namelist request.

222 IHC222I NAMEL - NAME NOT IN NAMELIST DICTIONARY. NAME=X

Supplemental Data: Name specified.

Standard Corrective Action: Ignore remainder of namelist request.

223 IHC223I NAMEL - END OF RECORD ENCOUNTERED BEFORE EQUAL SIGN. NAME=X

Supplemental Data: Name of item.

Standard Corrective Action: Ignore remainder of the namelist request.

224 IHC224I NAMEL - SUBSCRIPT FOR NON-DIMENSIONED VARIABLE OR SUBSCRIPT OUT OF RANGE. NAME=X

Supplemental Data: Name of item.

Standard Corrective Action: Ignore remainder of the namelist request.

225 IHC225I CONVERT - ILLEGAL HEXADECIMAL CHARACTER X

Supplemental Data: Display the record in which the character appeared.

Standard Corrective Action: 0 replaces the encountered character.

230 IHC230 - SOURCE ERROR AT ISN X - EXECUTION FAILED AT SUBROUTINE X

Supplemental Data: None.

Standard Corrective Action: Terminate execution.

231 IHC231I IBCOM - DIRECT ACCESS STATEMENT USED WITHOUT DEFINE FILE ON UNIT X

IHC231I DIOCS - DIRECT ACCESS STATEMENT USED FOR SEQUENTIAL DATA SET X

IHC231I FIOCS - SEQUENTIAL I/O STATEMENTS USED FOR DIRECT ACCESS DATA SET X

Supplemental Data: Unit number X

Standard Corrective Action: Ignore I/O request.

232 IHC232I DIOCS - RECORD NUMBER X OUT OF RANGE ON UNIT X

Supplemental Data: Unit number and record number.

Standard Corrective Action: Ignore I/O request.

233 IHC233I DIOCS - RECORD LENGTH GREATER THAN 32K-1 SPECIFIED FOR UNIT X

Supplemental Data: Unit number specified.

Standard Corrective Action: Set record length to 32,000.

234 IHC234I DIOCS - ATTEMPT TO DEFINE THE OBJECT ERROR UNIT AS DIRECT ACCESS DATA SET. UNIT=X

Supplemental Data: Unit number.

Standard Corrective Action: Ignore define file entry.

235 IHC235I DIOCS - DEFINE A DATA SET WHICH HAS BEEN USED SEQUENTIALLY AS A DIRECT ACCESS DATA SET. UNIT=X

Supplemental Data: Unit number.

Standard Corrective Action: Ignore define file entry.

236 IHC236I DIOCS - READ REQUEST FOR AN UNCREATED DATA SET ON UNIT X

Supplemental Data: Unit number.

Standard Corrective Action: Ignore I/O request.

237 IHC237I DIOCS - INCORRECT RECORD LENGTH SPECIFIED FOR UNIT X

Supplemental Data: Unit for which error occurred.

Standard Correct Action: Ignore the I/O request.

241 IHC241I FIXPI INTEGER BASE=0, INTEGER EXPONENT=X, LE 0

Supplemental Data: Exponent specified.

Standard Corrective Action: Result=0.

242 IHC242I FRXPI REAL*4 BASE=0.0, INTEGER EXPONENT=X, LE 0

Supplemental Data: Exponent supplied.

Standard Corrective Action: Result=0.

243 IHC243I FDXPI REAL*8 BASE=0.0, INTEGER EXPONENT=X, LE 0

Supplemental Data: Exponent specified.

Standard Corrective Action: Result=0.

244 IHC244I FRXPR REAL*4 BASE=0.0, REAL *4 EXPONENT=X.X, LE 0

Supplemental Data: Exponent specified.

Standard Corrective Action: Result=0.

245 IHC245I FDXPD REAL*8 BASE=0.0, REAL*8 EXPONENT=X.X, LE 0

Supplemental Data: Exponent specified.

Standard Corrective Action: Result=0.

246 IHC246I FCXPI COMPLEX*8 BASE=0.0+ 0.0I, INTEGER EXPONENT=X, LE 0

Supplemental Data: Exponent specified.

Standard Corrective Action: Result=0.

247 IHC247I FCDXI COMPLEX*16 BASE=0.0+ 0.0I, INTEGER EXPONENT=X, LE 0

Supplemental Data: Exponent specified.

Standard Corrective Action: Result=0.

251 IHC251I SQRT NEGATIVE ARGUMENT=X

Supplemental Data: Argument specified.

Standard Corrective Action: Result=$|X|^{1/2}$.

252 IHC252I EXP ARG=X.X, GT 174.673

Supplemental Data: Argument specified.

Standard Corrective Action: Result=*.

253 IHC253I ALOG-ALOG10 ARG=X.X,LE ZERO

Supplemental Data: Argument specified.

Standard Corrective Action: if X=0, result=-*; if X<0, result=log |X| or $\log_{10}$ |X|.

254 IHC254I SIN-COS/ARG/=/X.X (HEX=X)/ , GE PI*2**18

Supplemental Data: Argument specified.

Standard Corrective Action: Result=$\sqrt{2}/2$

255 IHC255I ATAN2 ARGUMENTS=0.0

Supplemental Data: Arguments specified.

Standard Corrective Action: Result=0.

256 IHC256I SINH-COSH/ARG/=/X.X/, GE 174.673

Supplemental Data: Argument specified.

Standard Corrective Action: Result=*.

257 IHC257I ARSIN-ARCOS/ARG/=/X.X/ GT 1

Supplemental Data: Argument specified.

Standard Corrective Action: Result=0.

258 IHC258I TAN-COTAN/ARG/=/X.X/ (HEX=X)/, GE PI*2**18

Supplemental Data: Argument specified.

Standard Corrective Action: Result=1.

259 IHC259I TAN-COTAN/ARG/=/X.X/ (HEX=X)/, APPROACHES SINGULARITY

Supplemental Data: Argument specified

Standard Corrective Action: Result=*.

261 IHC261I DSQRT NEGATIVE ARGUMENT=X.X

Supplemental Data: Argument specified

Standard Corrective Action: Result=$|X|^{1/2}$

262 IHC262I DEXP ARG=X.X, GT 174.673

Supplemental Data: Argument specified.

Standard Corrective Action: Result=*.

263 IHC263I DLOG-DLOG10 ARG=X.X, LE ZERO

Supplemental Data: Argument specified.

Standard Corrective Action: if X=0, result=-*; if X<0, result=log |X| or $\log_{10}|X|$.

264 IHC264I DSIN-DCOS/ARG/=/X.X (HEX=X)/, GE PI*2**50

Supplemental Data: Argument specified.

Standard Corrective Action: Result=0. $\sqrt{2}/2$

265 IHC265I DATAN2 ARGUMENTS=0.0

Supplemental Data: Arguments specified.

Standard Corrective Action: Result=0.

266 IHC266I DSINH-DCOSH/ARG/=/X.X/, GE 174.673

Supplemental Data: Argument specified.

Standard Corrective Action: Result=*.

267 IHC267I DARSIN-DARCOS/ARG/=/X.X/, GT 1

Supplemental Data: Argument specified.

Standard Corrective Action: Result=0.

268 IHC268I DTAN-DCOTAN/ARG/=/X.X (HEX=X)/ GE PI*(2**50)

Supplemental Data: Argument specified.

Standard Corrective Action: Result=1.

269 IHC269I DTAN-DCOTAN/ARG/=/X.X (HEX=X)/, APPROACHES SINGULARITY

Supplemental Data: Argument specified.

Standard Corrective Action: Result=*.

271 IHC271I CEXP REAL ARG=X.X (HEX=X), GT 174.673

Supplemental Data: Argument specified.

Standard Corrective Action: Result=* (COS X + iSIN X) where X is the imaginary portion of the argument.

272 IHC272I CEXP IMAG ARG=X (HEX=X), ABS VALUE GE PI*2**18

Supplemental Data: Argument specified.

Standard Corrective Action: Result=0+0i.

273 IHC273I CLOG ARGUMENT=0.0+0.0I

Supplemental Data: Argument specified.

Standard Corrective Action:
Result=-*+0i.

274 IHC274I CSIN-CCOS/REAL ARG/=/X.X (HEX=X)/, GE PI*2**18

Supplemental Data: Argument specified.

Standard Corrective Action:
Result=0+0i.

275 IHC275I CSIN-CCOS/IMAG ARG/=/X.X (HEX=X)/, GT 174.673

Supplemental Data: Argument specified.
Standard Corrective Action: If imaginary part >0, (X is real portion of argument):

For sine, result= $\frac{*}{2}$ (SIN X + iCOS X).

For cosine, result= $\frac{*}{2}$ (COS X - iSIN X).

If imaginary part <0, (X is real portion of argument):

For sine, result= $\frac{*}{2}$ (SIN X - iCOS X).

For cosine, result= $\frac{*}{2}$ (COS X + iSIN X).

281 IHC28I CDEXP REAL ARG=X.X(HEX=X) GT 174.673

Supplemental Data: Argument specified.
Standard Corrective Action:
Result=* (COS X+iSIN X) where X is the imaginary portion of the argument.

282 IHC282I CDEXP IMAG ARG=X.X(HEX=X) ABS VALUE GE PI*2**50

Supplemental Data: Argument specified.
Standard Corrective Action:
Result=0+0i.

283 IHC283I CDLOG ARGUMENT=0.D0+0.D0I

Supplemental Data: Argument specified.
Standard Corrective Action:
Result=-*+0i.

284 IHC284I CDSIN-CDCOS/REAL ARG/=/X.X(HEX=X)/, GE PI*2**50

Supplemental Data: Argument specified.
Standard Corrective Action:
Result=0+0i

285 IHC285I CDSIN-CDCOS/IMAG ARG/=/X.X(HEX=X)/, GT 174.673

Supplemental Data: Argument specified.
Standard Corrective Action: If imaginary part >0, (X is real portion of argument):

For sine, result= $\frac{*}{2}$ (SIN X + iCOS X).

For cosine, result= $\frac{*}{2}$ (COS X - iSIN X).

If imaginary part <0, (X is real portion of argument):

For sine, result= $\frac{*}{2}$ (SIN X - iCOS X)

For cosine, result= $\frac{*}{2}$ (COS X + iSIN X)

290 IHC290I GAMMA ARG=X.X(HEX=X), LE2**-252 OR GE 57.5744

Supplemental Data: Argument specified.
Standard Corrective Action:
Result=*.

291 IHC291I ALGAMA ARG=X.X(HEX=X), LE 0 OR GE 4.2937*10**73

Supplemental Data: Argument specified
Standard Corrective Action:
Result=*.

300 IHC300I DGAMMA ARG=X.X(HEX=X), LE 2**-252 OR GE 57.5744

Supplemental Data: Argument specified.
Standard Corrective Action:
Result=*.

301 IHC301I DLGAMA ARG=X.X(HEX=X), LE 0. OR GE 4.2937*10**73

Supplemental Data: Argument specified.
Standard Corrective Action:
Result=*.

*is used to denote the largest number representable in floating point.

```
SUMMARY OF ERRORS FOR THIS JOB  ERROR NUMBER  NUMBER OR ERRORS
                                219           1
                                217           1
                                211           57
```

Figure 76. Summary of Error and Traceback

```
TRACEBACK FOLLOWS-  ROUTINE  ISN  REG.  14  REG.  15  REG.   0  REG.   1
                    IBCOM        000083B4    000089B8    00000005    000081A6
                    MAIN         00004918    50008020    00000030    0003FF04
ENTRY POINT- 50008020
```

Figure 77. Example of Traceback Map

Even though message printing may be suppressed when the extended error message facility is available, a summary of errors is printed when the job is completed. Its format is shown in Figure 76 . The format of a Traceback Map is shown in Figure 77.

The headings in the Traceback Map may be described as follows:

ROUTINE — The name of the routine entered, which was called by the next routine in the list.

ISN — When the compiler's ID option supplies an Internal Statement Number (ISN), the ISN entry is a symbolic reference to the point from which the routine was called.

REG. 14 — This is the absolute location reference to the point from which ROUTINE was called. By using the ENTRY POINT location, a relative location can be computed.

REG. 15 — This is the address of the entry point in ROUTINE.

REG. 0 — This is the result register used by function subprograms.

REG. 1 — This is the address of the argument list passed to ROUTINE.

If the user specifies that an installation-supplied routine is to be used for corrective action, this line is added to the message:

USER FIXUP TAKEN, EXECUTION CONTINUING

For a standard corrective action, the message addition reads:

STANDARD FIXUP TAKEN, EXECUTION CONTINUING

If the extended error message facility detects an error condition, an informational message is printed and the job may be terminated. The following text contains a description of such messages.

IHC900I EXECUTION TERMINATING DUE TO ERROR COUNT FOR ERROR NUMBER X

Explanation: This error has occurred frequently enough to reach the count specified as the number at which execution should be terminated.

System Action: The job is terminated.

IHC901I EXECUTION TERMINATING DUE TO SECONDARY ENTRY TO ERROR MONITOR FOR ERROR X WHILE PROCESSING ERROR X

Explanation: In a user's corrective action routine, an error has occurred that has called the error monitor before it has returned from processing a diagnosed error.

System Action: The job is terminated.

Note: If Traceback follows this message, it may be unreliable.

IHC902I ERROR NUMBER X OUT OF RANGE OF ERROR TABLE

Explanation: A request has been made to reference a non-existent Option Table entry.

System Action: The request is ignored and execution continues.

IHC903I ATTEMPT TO CHANGE UNMODIFIABLE TABLE ENTRY, NUMBER=X

Explanation: The Option Table specifies that no changes may be made in this entry, but a change request has been made by use of CALL ERRSET or CALL ERRSTR.

System Action: The request is ignored and execution continues.

IHC904I ATTEMPT TO DO I/O DURING FIXUP ROUTINE FOR AN I/O TYPE ERROR

Explanation: When attempting to correct an input/output error, the user may not issue a READ, WRITE, BACKSPACE, ENDFILE, REWIND, PDUMP, DEBUG, or ERRTRA.

System Action: The job is terminated.

Operator Messages

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

The message for a PAUSE can be one of the forms:

```
yy IHC001A{PAUSE n          }
          {PAUSE 'message'  }
          {PAUSE 0          }
```

where: yy is the identification number

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

'message' is the literal constant specified in a PAUSE source statement

0 is printed out when a

PAUSE

statement is executed

Explanation: The programmer should give instructions that indicate the action to be taken by the operator when the PAUSE is encountered.

User Response: To resume execution, the operator presses the REQUEST key. When the PROCEED light comes on, the operator types

REPLY yy,'Z'

where yy is the identification number and Z is any letter or number. To resume program execution, the operator must press the alternate coding key and a numeric 5.

The message for a STOP statement can be one of the forms:

```
IHC002I{STOP n}
       {STOP 0}
```

where: n is the unsigned 1-5 digit integer constant specified in a STOP source statement. This value is placed in register 15 when the STOP statement is executed.

0 is printed when a STOP statement is executed.

User Response: None

APPENDIX E: EXTENDED USA CARRIAGE CONTROL CHARACTERS

1130

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.

C28-6602-3

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]