Form C28-6603-2
Page Revised 11/15/68 by TNL N28-0586

File No. S360-25
Form C28-6603-2

OS

IBM Systems Reference Library

IBM System/360 Operating System
| Basic FORTRAN IV (E) Programmer's Guide

Program Number 360S-FO-092

This publication describes how to compile, link edit, and execute a FORTRAN IV (E) program. The text also describes the output of compilation and execution and how to make optimal use of the compiler and a load module.

Third Edition

This publication is a major revision of, and makes obsolete, Form C28-6603-1 and Technical Newsletters N28-0211, N28-0233, and N28-0235. New material explains how FORTRAN IV handles exponent overflow and underflow in floating-point registers. There are also additions and deletions among input/output messages. Changes to the text are indicated by a vertical line to the left of the change; revised illustrations are denoted by a bullet ( ) to the left of the caption.

PREFACE

The purpose of the Programmer's Guide is to enable programmers to compile, link edit, and execute FORTRAN IV (E) programs under control of IBM System/360 Operating System. The FORTRAN IV (E) language is described in the publication IBM System/360 Basic FORTRAN IV Language, Form C28-6629, a corequisite to this publication.

The Programmer's Guide is organized to fulfill its purpose for three groups of programmers:

1. Programmers who wish to use the cataloged procedures as provided by IBM need read only the "Introduction" and "Job Control Language" sections to understand the job control statements, and the "Job Processing" section to use cataloged procedures for compiling, link editing, and executing FORTRAN programs. The "Programming Considerations" and "System Output" sections are recommended for programmers who want to use the FORTRAN language more effectively.

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

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

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

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

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

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

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

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

   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 C28-6631

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

This publication contains appendixes that:

- Give several examples of executing load modules.

- Describe the preparation of assembler language subprograms for use with a main program written in FORTRAN. To understand this appendix, these publications are prerequisite:

  IBM System/360 Operating System: Assembler Language, Form C28-6514

  IBM System/360 Operating System: Assembler (E) Programmer's Guide, Form C28-6595 or IBM System/360 Operating System: Assembler (F) Programmer's Guide, Form C26-3756

- Describe the diagnostic messages produced during compilation and load module execution.

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 E 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.

Job control language statements define two units of work to the operating system: the job and the job step. The important aspect of jobs and job steps is that they are defined by the programmer. He defines a job to the operating system by using a JOB statement; he defines a job step by using the EXEC statement. Another important statement is the DD (data definition) statement, which gives the operating system information about data used in jobs and job steps. The sequence of control statements and any data placed in this sequence is called the _input stream_. The input stream can be read by either a card reader or a tape device.

JOB AND JOB STEP RELATIONSHIP

When a programmer is given a problem, he analyzes that problem and defines a precise problem-solving procedure; that is, he writes a program or a series of programs. To the operating system, executing a main program (and any subprograms it calls) is a _job step_. A _job_ consists of executing one or more job steps.

At its simplest, 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.

If the problem is complex, 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. Comparing the raw data to projected data and eliminating errors which arise because of intermittent errors in gauges and transmission facilities.
2. Using 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. Using the values to plot the graphs and print the reports.

In this example, each step can be a separate job with one job step in each job. However, designating several 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.) In the rocket firing example, each step may be defined as a job step within one job that encompasses all processing. Figure 1 illustrates the rocket firing job with three job steps.

DATA SETS

In Figure 1, two collections of input data (raw data and projected data) and one collection of output data (refined data) are used in job step 1. In the operating system, a collection of data that can be named by the programmer is called a _data set_. A data set is defined to the operating system by a DD statement.

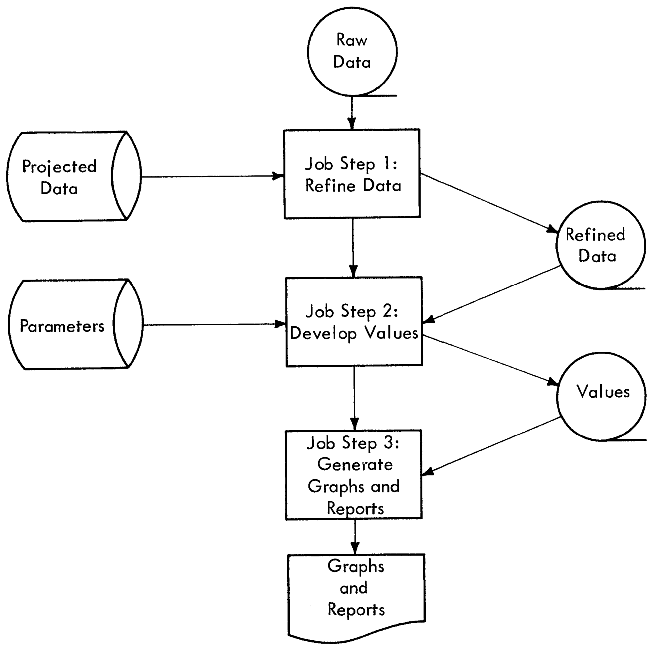


Figure 1. Rocket Firing Job

A data set resides on a volume(s), which is an external storage unit accessible to an input/output device. (For example, a volume may be a reel of tape or a disk pack.)

Several I/O devices grouped together and given a single name when the system is generated constitute a device class. (For example, a device class can consist of all the tape devices in the installation; another can consist of the printer, a direct access device, and a tape device.)

Indexing Data Sets

The name of a data set, information identifying the volume(s) on which the data set resides, device type, and the position of the data set on the volume may be placed in an index to help the control program find the data set. This index, which is part of an index structure called the catalog, resides on direct access volumes with the operating system. Any data set whose name and volume identification are placed in this index is called a cataloged data set.

Furthermore, a hierarchy of indexes may be devised to classify data sets and make names for data sets unique. For example, an installation may divide its cataloged data sets into four groups: SCIENCE, ENGRNG, ACCNTS, and INVNTRY. In turn, each of these groups may be subdivided: the SCIENCE group may be subdivided into MATH, PHYSICS, CHEM, and BIOLOGY; MATH may contain volume identification for the data sets ALGEBRA, CALCULUS, and BOOL. To find the data set BOOL, the programmer specifies all indexes beginning with the largest group -- SCIENCE; then the next largest group, MATH; finally, the data set BOOL. The complete identification needed to find the data set BOOL is SCIENCE.MATH.BOOL.

Data set names are of two classes. An unqualified name is a data set name or an index name not preceded by an index name. A qualified name is a data set name or index name preceded by index names representing index levels; for example, in the preceding text, the qualified name of the data set BOOL is SCIENCE.MATH.BOOL.

Before using a qualified name to name a data set, the programmer must be sure that the index levels specified in a qualified name are placed in the catalog. Index levels are placed in the catalog by a utility program. For more information, see the section "Modifying System Control Data" in the Utilities publication or "Maintaining the Catalog and the Volume Table of Contents" in the Systems Programmer's Guide.

Data Set Labels

Information such as record format, buffer length, density, creation date, expiration date, and an identifier needed to read the data set are stored in the operating system data set labels. If a data set is cataloged and standard labels are specified when the data set is created, the information specified in the DD statement to subsequently retrieve the data set is substantially reduced. In addition to the data set name, the only information needed to retrieve the data set is the current status of the data set (new, old, etc.) and the status the data set is to have when the job step is completed (deleted, kept, passed, etc.).

Generation Data Sets

Data set identification may also be based upon the time of generation. In the operating system, a collection of successive historically related data sets is a generation data group. Each of the data sets is a generation data set. A generation number is attached to the data group name to refer to a particular generation. The most recent generation is 0; the generation previous to 0 is -1; the generation previous to -1 is -2; etc. An index describing a generation data group must exist in the catalog. The index is created by a data set utility program.

For example, a data group named WEATHER might be used for weather reporting and forecasting. The generations for the generation data group WEATHER are:

```
WEATHER(0)
WEATHER(-1)
WEATHER(-2)
     .
     .
     .
```

When a new generation is created, it is called generation (+n), where n is an integer greater than 0. For example, after a job step has created WEATHER (+1), the operating system changes its name to WEATHER(0) at the end of the job. The data set that was WEATHER(0) at the beginning of the job becomes WEATHER(-1).

If more than one generation is created in a job, the first generation created is generation (+1); the next generation created is generation (+2); and so on

FORTRAN PROCESSING

In the operating system environment, a source program is called a source module; a compiled source module is an object module (object program). The object module cannot be executed until it is placed in a format suitable for loading, and all external references to subprograms are resolved. This is done by an IBM supplied program -- the linkage editor.

The executable output of the linkage editor is a load module. However, the input to the linkage editor may be either object modules or other load modules. Linkage editor execution can be expanded further: several object modules and/or load modules may be combined to form one load module. The linkage editor inserts the requested library functions and subroutines into the load module. For example, if the compiled object module TEST calls subroutines ALPHA and BETA and the library function SIN, the linkage editor combines the object module TEST and the previously link edited load modules ALPHA, BETA, and SIN into one load module. This process is illustrated in Figure 2.

A program written in FORTRAN may call subprograms written in the assembler language as long as the assembler subprogram uses the linkage conventions shown in Appendix B: "Assembler Language Subprograms." The linkage editor resolves the references between assembler and FORTRAN modules.

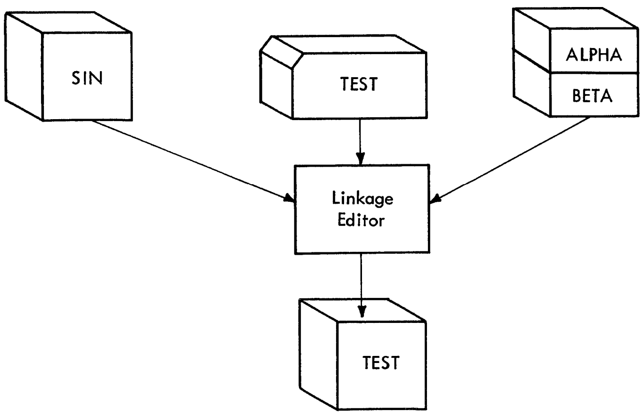


Figure 2. Linkage Editor Execution

Processing a FORTRAN Program

After an object module is processed by the linkage editor, the resulting load module may be executed. Therefore, to compile, link edit, and execute a FORTRAN program, three steps are necessary:

1. Compile the FORTRAN source module and any FORTRAN subprograms not compiled previously to produce one or more object modules.
2. Link edit the resulting object module(s) and any modules needed to resolve external references to form a load module.
3. Execute the load module.

Figure 3 illustrates the problem program processing; FORTRAN subprograms and assembler subprograms (object modules) are used to resolve external references.

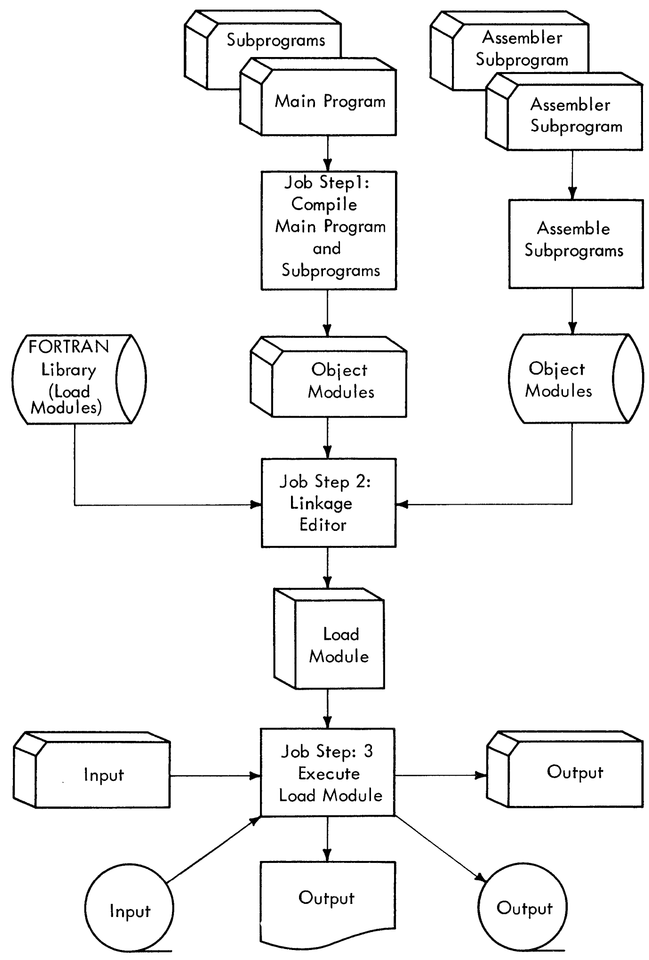


Figure 3. Typical FORTRAN Processing

Efficient Processing

Each compilation, each linkage editor execution, and each load module execution may be defined as separate jobs, but combining the separate jobs into one job is more efficient.

Assume that the source module MAIN is to be compiled and executed. MAIN requires the services of two subprograms, SUB1 and SUB2, and neither subprogram is compiled. Since the compiler can perform batched compilations, all the compilations can be combined in one job step. Three job steps are necessary to perform the job:

JOB: Compile, link edit, and execute

JOB STEP 1:	Compile MAIN, SUB1, SUB2
JOB STEP 2:	Link edit the modules
JOB STEP 3:	Execute load module MAIN

Output of Processing

The compiler, linkage editor, and other components of the operating system generate diagnostic output which can be used to debug programs. Among these are listings, module maps, and diagnostic messages.

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.

A _partitioned data set_ (PDS) is composed of named, independent groups of sequential data and resides on a direct access volume. A _directory index_ resides in the PDS and directs the operating system to any group of sequential data. Each group of sequential data is called a _member_. (A member of a PDS is _not_ a data set.) Partitioned data sets are used for storage of any type of sequentially organized data. In particular, they are used for storage of source and load modules (each module is a member).

Only individual members of a PDS are accessible to the compiler. Members of a PDS are accessible to a FORTRAN load module, but only under certain conditions. (See the discussion of the IN, OUT subparameters of the LABEL parameter in the section "Creating Data Sets.")

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

Processing Partitioned Data Sets: To process a member of a partitioned data set, the programmer must use the DD statement to provide information about the data set and the member. The programmer must specify (in the DSNAME parameter) both the name of the data set and of the member, and must indicate (in the LABEL parameter) whether the member is to be created or retrieved. However, if the programmer requests the FORTRAN compiler to process a partitioned data set (for example, to compile a source deck stored as a member of a partitioned data set), no LABEL information need be specified.

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

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

A direct access data set must reside on a direct access volume. Direct access data sets are only processed by FORTRAN load modules; the compiler and linkage editor cannot process direct access data sets.

Catagorizing a data set as sequential, partitioned, or direct access reflects its organization Catagorizing a data set as cataloged or as 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.

Cataloged Procedures

To reduce the possibility of error in the frequent reproduction of job control statements, cataloged procedures can be written. A cataloged procedure is a set of EXEC and DD statements placed in a PDS accessible to the operating system. (The JOB statement cannot be cataloged.) A cataloged procedure consists of a single procedure step or a series of procedure steps defined by EXEC statements. A procedure step in a cataloged procedure is equivalent to a job step in a job. Just as DD statements are included for a job step, DD statements are included in procedure steps.

An EXEC statement in the input stream can invoke a cataloged procedure. There fore, the definition of job step is extended: executing a load module or invoking a cataloged procedure is a job step to the operating system.

To simplify the steps involved in compiling and link editing, four cataloged procedures for FORTRAN (E) are supplied by IBM. These four cataloged procedures and their uses are:

FORTEC	compile
FORTECL	compile and link edit into the FORTRAN library (FORTLIB)
FORTELG	link edit and execute
FORTECLG	compile, link edit, and execute

Any cataloged procedure may be temporarily modified by EXEC and DD statements in the input stream; this temporary modification is called overriding.

JOB CONTROL LANGUAGE

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

JOB MANAGEMENT

Job control statements are processed by a group of operating system routines known collectively as job management routines. These routines interpret control statements, control the flow of jobs, and issue messages to both the operator and the programmer. Job management routines have 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

Like any other computer language, the job control language has a specific structure and must conform to a prescribed format. To make the definition and description of job control statements more understandable, a notation to show the format of the statements has been devised and will be used throughout this publication.

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.
comment	Contains miscellaneous remarks, annotations, etc., written by the programmer; it can appear before or after any control statement.

GENERAL STRUCTURE OF CONTROL STATEMENTS

Except for the comment statement, control statements contain the identifying characters // or /* in card columns 1 and 2. The comment statement is identified by the initial characters //* in card columns 1, 2, and 3. Control statements may contain four fields: name, operation, operand, and comments (see Figure 4).

Format	Applicable Control Statements
//Name Operation Operand [Comment]	JOB,EXEC,DD
// Operation Operand [Comment]	EXEC,DD
/* [Comment]	delimiter
//*[Comment]	comment

Figure 4. Job Control Statement Formats

Name Field

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

1. Identify the control statement to the operating system.
2. Enable other control statements in the job to refer to information contained in the named statement.
3. Relate DD statements to I/O statements in the load module.

Operation Field

The operation field contains one of the following operation codes:

JOB
EXEC
DD

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

Operand Field

The operand field contains the parameters that provide information to the operating system. The parameters are separated by commas. 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 and keyword parameters are identified in the definition of control statements.

Positional Parameters: Positional parameters are placed first in the operand field and must appear in a specified order. If a positional parameter is omitted and other positional parameters follow, the omission must be indicated by a comma. If a positional parameter is omitted and only keyword parameters follow, the omission is not indicated by a comma.

Keyword Parameters: A keyword parameter may be placed anywhere in the operand field after the positional parameters. A keyword parameter consists of a keyword, followed by an equal sign, followed by a single value or a list of subparameters. If there is a list of subparameters, the list must be enclosed in parentheses or apostrophes and the subparameters in the list must be separated by commas. Keyword parameters may appear in any order.

Subparameters: Subparameters are either positional or keyword. Positional subparameters appear first in the 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. If a positional subparameter is omitted and only keyword subparameters follow, the omission is not indicated by a comma. Positional and keyword subparameters are noted in the definition of control statements.

Comments Field

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

CONTINUING CONTROL STATEMENTS

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

1. Interrupt the field after a complete parameter or subparameter, including the comma that follows it, at or before column 71. (The coding of a nonblank character in column 72 is optional.)
2. Include comments, if desired, by following the interrupted field with at least one blank.
3. Code the identifying characters // in columns 1 and 2 of the following card or card image.
4. Continue the interrupted operand beginning in any column from 4 through 16.

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

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

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

Note: The comment statement cannot be continued.

NOTATION FOR DEFINING CONTROL STATEMENTS

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

1. The following symbols are used to define control statements but are never used 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 through 10.

2. Upper-case letters and words, numbers, and the symbols 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. Lower-case 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 lower-case 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 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

$$\begin{matrix} \text{A} & \\ \underline{\text{B}} & \text{and A|}\underline{\text{B}}\text{|C} \\ \text{C} & \end{matrix}$$

   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|$\underline{\text{C}}$},D)

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

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

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

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

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

   Example: ALPHA[,BETA]...

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

10. A superscript refers to a footnote.

    Example: $\begin{Bmatrix}\text{NEW}\\ \text{OLD}\\ \text{MOD}\end{Bmatrix}$[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 5) is the first statement in the sequence of control statements that describe a job. The JOB statement contains the following information:

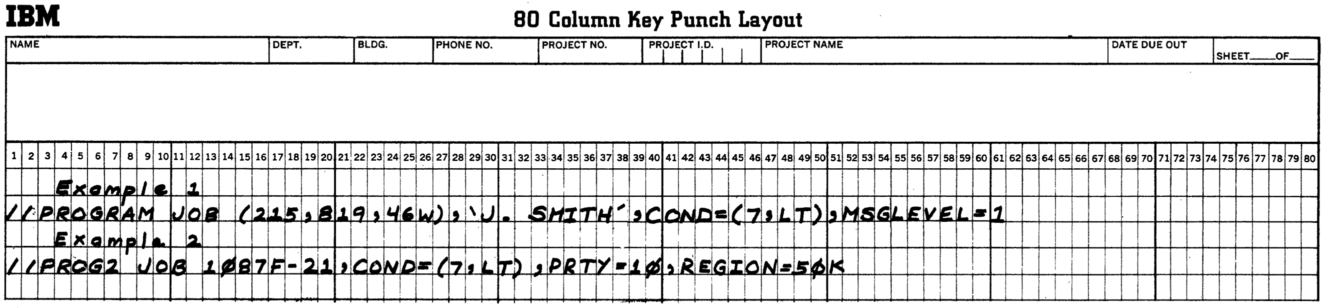
1. The name of the job.
2. Accounting information relative to the job.
3. Programmer's name.
4. Whether the job control statements are printed for the programmer.
5. Conditions for terminating the execution of the job.
6. A job priority assignment.
7. Output class for priority scheduler messages.
8.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

Examples of the JOB statement are shown in Figure 6.

Name	Operation	Operand
		Positional Parameters
//jobname	JOB	[([account-number][,accounting-information])[1] [2] [3]]
		[,programmer-name][4] [5] [6]
		Keyword Parameters
		$\begin{Bmatrix}\text{MSGLFVEL=0}\\ \text{MSGLFVEL=1}\end{Bmatrix}$
		[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, the information must be delimited by apostrophes instead of parentheses.
[2]If only "account-number" is specified, the delimiting parentheses may be omitted.
[3]The maximum number of characters allowed between the delimiting parentheses or apostrophes is 142.
[4]If "programmer-name" contains 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 tests allowed is 8.
[8]If only one test is specified, the outer pair of parentheses may be omitted.
[9]This parameter is used by the priority scheduler only. The sequential scheduler ignores it.

Figure 5. JOB Statement


IBM
80 Column Key Punch Layout
NAME
DEPT.
BLDG.
PHONE NO.
PROJECT NO.
PROJECT I.D.
PROJECT NAME
DATE DUE OUT
SHEET____OF____

```
   Example 1
//PROGRAM JOB (215,819,46W),'J. SMITH',COND=(7,LT),MSGLEVEL=1
   Example 2
//PROG2 JOB 1Ø87F-21,COND=(7,LT),PRTY=1Ø,REGION=5ØK
```

Figure 6. Sample JOB Statements

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 precise format of the job accounting information is specified by the installation.

As a system generation option with sequential schedulers, the account number can be established as a required parameter. With priority schedulers, the requirement can be established with a cataloged procedure for the input reader. (Information about how to write an accounting routine may be found in IBM System/360 Operating System: Systems Programmer's Guide.) Otherwise, the account number is optional.

Programmer's Name

The "programmer name" is the second positional parameter. If no job accounting information is coded, its absence must be indicated by a comma preceding the programmer's name. If neither job accounting information nor programmer's name is coded, 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 diagnostic messages are written for the programmer.

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

Note: If an error occurs in 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 the job step. Instructions, written by the programmer, in a FORTRAN program cannot generate the code. The 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

8 - Serious errors detected

12 - For the compiler, the SYSLIN DD state-

ment is omitted, or the NOLOAD option is specified. For the linkage editor, severe errors are detected.

16 - For the compiler, the SYSIN or SYSPRINT DD statement is omitted, a permanent I/O error is encountered, the source module is nonexistent, or the compiler, linkage editor, or a load module terminated abnormally. If any error message (except a program interrupt message) is issued 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 of the conditions are satisfied, the job is terminated.

Assigning Job Priority (PRTY)

(Used by Priority Schedulers Only)

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

Whenever possible, avoid using priority 14. This is used by the system to expedite processing of exceptional jobs It is also intended for other special uses by future features of systems with priority schedulers

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

Requesting a Message Class (MSGCLASS)

(Used by Priority Schedulers Only)

With a quantity and diversity of data in the output stream, an installation may want to separate different types of output data into different classes. Each class is directed to an output writer associated with a specific output unit. The MSGCLASS=x parameter allows the messages issued by the job scheduler to be routed to an output class other than the normal message class, A. Replace the letter "x" with an alphabetic or numeric character. An output writer, which is assigned to process this class, will transfer this data to a specific device.

Specifying Main Storage Requirements for a Job (REGION)

(Used by Priority Schedulers Only)

REGION=nnnnnK can be specified to indicate the amount of main storage to be allocated to the job. Replace the term "nnnnn" with the number of 1024-byte areas to be allocated to the job; e.g., REGION=50K. This number can range from one to five digits and cannot exceed 16,384.

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

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

EXEC STATEMENT

The EXEC statement (Figure 7) indicates the beginning of a job step and describes that job step. The statement contains the following information:

1 The name of the job step or procedure step.

2 Name of the cataloged procedure or load module to be executed.

3. Compiler and/or linkage editor options passed to the job step.

4. Accounting information relative to this job step.

5 Conditions for bypassing the execution of this job step.

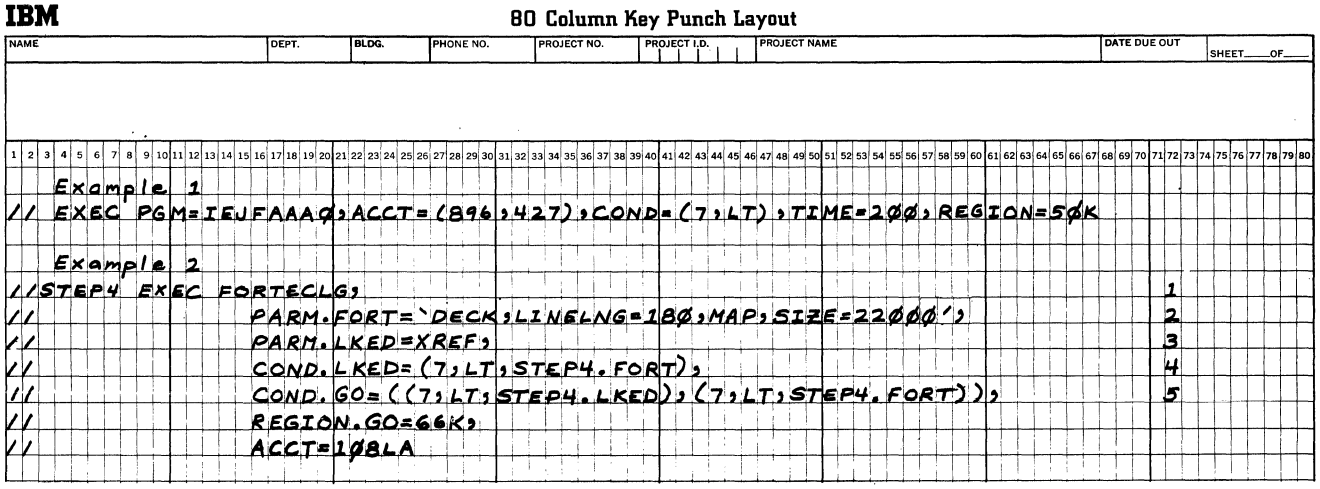
6. A time limit for the job step or an entire cataloged procedure.

7. Specification of main storage requirements for a job step or an entire cataloged procedure.

```
┌──────────────┬─────────┬──────────────────────────────────────────────────────────────┐
│Name          │Operation│Operand                                                       │
├──────────────┼─────────┼──────────────────────────────────────────────────────────────┤
│              │         │       Positional Parameter                                   │
│              │         │                                                              │
│//[stepname]¹ │EXEC     │ ⎧PROC=cataloged-procedure-name   ⎫                           │
│              │         │ ⎪cataloged-procedure-name        ⎪                           │
│              │         │ ⎨PGM=program-name                ⎬                           │
│              │         │ ⎪PGM=*.stepname.ddname           ⎪                           │
│              │         │ ⎩PGM=*.stepname.procstep.ddname  ⎭                           │
│              │         │                                                              │
│              │         │       Keyword Parameters                                     │
│              │         │                                                              │
│              │         │ ⎡⎧PARM          ⎫                                 ⎤          │
│              │         │ ⎣⎩PARM.procstep²⎭=(option[,option]...)³ ⁴ ⁵       ⎦          │
│              │         │                                                              │
│              │         │ ⎡⎧ACCT          ⎫                                 ⎤          │
│              │         │ ⎣⎩ACCT.procstep²⎭=(accounting-information)³ ⁶ ⁷   ⎦          │
│              │         │                                                              │
│              │         │ ⎡⎧COND          ⎫                                            │
│              │         │ ⎣⎩COND.procstep²⎭=((code,operator[,stepname[.procstep]])  ⎤  │
│              │         │          [,(code,operator[,stepname[.procstep]])]...⁸)⁹   ⎦  │
│              │         │                                                              │
│              │         │ ⎡                                         10,11⎤             │
│              │         │ ⎢⎧TIME          ⎫                              ⎥             │
│              │         │ ⎣⎩TIME.procstep²⎭   =(minutes,seconds)         ⎦             │
│              │         │                                                              │
│              │         │ ⎡                           10⎤                              │
│              │         │ ⎢⎧REGION          ⎫           ⎥                              │
│              │         │ ⎣⎩REGION.procstep²⎭=nnnnnK    ⎦                              │
└──────────────┴─────────┴──────────────────────────────────────────────────────────────┘
```

[1]"stepname" is required when information from this control statement is referred to in a later job step.
[2]If this alternative 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 delimiting parentheses is 100. If the option list is enclosed in apostrophes, however, the PARM parameter must be coded on one card.
[6]If "accounting-information" does not contain commas, blanks, parentheses, or equal signs, the delimiting parentheses may be omitted.
[7]The maximum number of characters allowed between the delimiting apostrophes or parentheses is 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 by MVT 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 7. EXEC Statement


IBM 80 Column Key Punch Layout

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

```
   Example 1
// EXEC PGM=IEJFAAAØ,ACCT=(896,427),COND=(7,LT),TIME=2ØØ,REGION=5ØK

   Example 2
//STEP4 EXEC FORTECLG,                                                 1
//               PARM.FORT='DECK,LINELNG=18Ø,MAP,SIZE=22ØØØ',          2
//               PARM.LKED=XREF,                                       3
//               COND.LKED=(7,LT,STEP4.FORT),                          4
//               COND.GO=((7,LT,STEP4.LKED),(7,LT,STEP4.FORT)),        5
//               REGION.GO=66K,
//               ACCT=1Ø8LA
```

Figure 8. Sample EXEC Statements

Example 1 of Figure 8 shows the EXEC statement used to execute a program. Example 2 in Figure 8 shows an EXEC statement that invokes a cataloged procedure.

NAME FIELD

The "stepname" is the name of the job step. It is required when information from this job step is referred to in a later job step.

OPERAND FIELD

Positional Parameter

The options in the positional parameter of an EXEC statement specify either the name of the cataloged procedure or program to be executed. Each program (load module) to be executed must be a member of a library (PDS). The library can be the system library (SYS1.LINKLIB), a private library, or a temporary library created to store a 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 unqualified name of the cataloged procedure. For example,

    PROC=FORTEC

    indicates that the cataloged procedure FORTEC is to be executed.

Specifying a Program in a Library:

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

    PGM=IEWL

    indicates that the load module IEWL is executed. A load module in a private PDS (private library) is executed by joining the private library with the system library through the use of a JOBLIB DD statement. See the discussion concerning JOBLIB.

Specifying a Program Described in a Previous Job Step:

PGM=*.stepname.ddname
    indicates that a program is executed, but the program is taken from a data set specified in 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 step. (The "stepname" cannot refer to a job step in another job.) For example, in the statements,

```
//LXIX 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 a program is executed, but the program is taken from the data set specified in a DD statement of a previously executed 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 cataloged procedure; "ddname" is the name of a DD statement within that procedure step. (The "stepname" cannot refer to a job step in another job.) For example, consider a cataloged procedure FORT,

```
//COMPIL   EXEC PGM=IEJFAAA0
//SYSUT1   DD   UNIT=TAPE
//SYSLIN   DD   DSNAME=LINKINP
           •
           •
           •
//LKED     EXEC PGM=IEWL
//SYSLMOD  DD   DSNAME=RESULT(ANS)
           •
           •
           •
```

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

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

The statement S2 in the input stream indicates that the name of the program is taken from the DD statement SYSLMOD in the procedure step LKED in the procedure invoked by the EXEC statement S1 Consequently, the load module ANS in the PDS RESULT is executed.

Keyword Parameters

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

If the parameter refers to a program or to an entire cataloged procedure (with the PARM parameter only the first procedure step is affected), the keyword is written followed by an equal sign and the list of subparameters. (In example 2, Figure 8, the parameter ACCT applies to the entire procedure.) When parameters are overridden in a cataloged procedure step, the keyword is written, a period is placed after the keyword, and the stepname follows immediately. (In example 2, Figure 8, the cataloged procedure FORTECLG is invoked. Two sets of PARM options apply to two different procedure steps; one applies to the procedure step FORT and the other to the procedure step LKED.) More information about overriding cataloged procedures is given in the section "Cataloged Procedures."

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 their execution is indicated by the PGM parameter in the EXEC statement. If the execution of a cataloged procedure is indicated, the options specified in the first procedure step are overridden by the options in the new PARM parameter; any options specified in other procedure steps are deleted.

PARM.procstep
: passes options to a compiler or linkage editor step within the named cataloged procedure step. Any PARM parameter in the procedure step is deleted, and the PARM parameter that is passed to the procedure step is inserted.

A maximum of 100 characters may be written between parentheses that enclose the list of options. If apostrophes enclose the option list, the PARM parameter must be coded on one card.

The format for compiler options and those linkage editor options most applicable to a FORTRAN program is shown in Figure 9.

```
Compiler:

 {PARM          }    {nnnnK  }                                      {,SOURCE  }
 {PARM.procstep}='[SIZE={yyyyyyy}][,LINELNG=zzz][,NAME=xxxxxx]      {,NOSOURCE}

                     {,DECK  } {,MAP  } {,LOAD  } {,BCD   } {,SPACE} {,ADJUST  }   '¹
                     {,NODECK} {,NOMAP} {,NOLOAD} {,EBCDIC} {,PRFRM} {,NOADJUST}


Linkage Editor:

 {PARM          }   [MAP ] [,LET ]
 {PARM.procstep}=( [XREF] [,XCAL] [,NCAL][,LIST][,OVLY])¹

¹The subparameters (options) are keyword subparameters.
```
Figure 9. Compiler and Linkage Editor Options

Detailed information concerning compiler and linkage editor options is given in the section "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. If the EXEC statement invokes a cataloged procedure, this COND parameter replaces all COND parameters in each step of the procedure.

COND.procstep
: states conditions for bypassing the execution of a specific cataloged procedure step "procstep". The specified COND parameter replaces all COND parameters in that procedure step.

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.GO=((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" or "stepname.procstep" is not given, "code" is compared to all codes issued by previous job or procedure steps.

Accounting Information

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

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

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

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

Setting Job Step Time Limits (TIME)

(Used by Priority Schedulers Only)

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

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

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

TIME.procstep
: assigns a time limit for a single step of a cataloged procedure. This parameter overrides, for the named step, any TIME parameter which is present. As many parameters of this form as there are steps in the cataloged procedure being executed can be written.

Specifying Main Storage Requirements for a Job Step (REGION)

(Used by Priority Schedulers Only)

The REGION parameter may be specified in the JOB statement, in which case it overrides the REGION parameters specified in the EXEC statements and applies to all steps of the job. However, if it is desired to allot to each job step only as much storage as it requires, the REGION parameter should be omitted from the JOB statement, and the EXEC statements should contain a REGION parameter specifying the amount of main storage to be allocated to the associated job step. If the REGION parameter is omitted from both JOB and EXEC statements, the default region size (as established in the cataloged procedure for the input reader) is assumed. The size is specified in the form "nnnnnK" where "nnnnn" is the number of 1024-byte areas to be allocated to the job step; e.g., REGION=50K.

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

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

For a discussion of the region size required for FORTRAN jobs, see "Specifying Main Storage Requirements for a Job (REGION)."

DATA DEFINITION (DD) STATEMENT

The DD statement (Figure 10) describes data sets. The DD statement can contain the following information:

1. Name of the DD statement.
2. Name of the data set to be processed.
3. Type and number of I/O devices for the data set.
4. Volume(s) on which the data set resides.
5. Amount and type of space allocated on a direct access volume.
6. Label information for the data set.
7. The status of the data set before execution of the step and the disposition of the data set after execution of the step.
8. Allocation of data sets to facilitate 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 program.
  3. To identify this DD statement to other control statements in the input stream.

```
Name                      Operation  Operand¹

                                     Positional Parameter

//⎧ddname          ⎫²      DD        ⎡*    ⎤⁴
  ⎪procstep.ddname ⎪                 ⎢DATA ⎥
  ⎨JOBLIB³         ⎬                 ⎣DUMMY⎦
  ⎪STEPLIB         ⎪
  ⎪SYSABEND        ⎪                 Keyword Parameters⁵
  ⎩SYSUDUMP        ⎭
                                      DDNAME=ddname
                                     ⎡         ⎧dsname                        ⎫⎤
                                     ⎢         ⎪dsname(element)               ⎪⎥
                                     ⎢         ⎪*.ddname                      ⎪⎥
                                     ⎢DSNAME=  ⎨*.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-no.])⁶ ⁷         ⎦

                                     [SEP=(subparameter-list)]
```

¹To allow a programmer flexibility in the use of the DD statement, all parameters are optional; however, a DD statement with a blank operand field is invalid.
²The name field must be blank when concatenating data sets.
³The JOBLIB statement precedes any EXEC statements in the job. See the discussion concerning JOBLIB under "Name Field" in this section.
⁴If either DATA or * is specified, keyword parameters cannot be specified.
⁵If "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.
⁶This form of the parameter is used only with priority schedulers.
⁷If program-name and form-no. are omitted, the delimiting parentheses can be omitted.

• Figure 10. Data Definition (DD) Statement

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

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

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

2. A DD statement that is to be added to the DD statements in the procedure step.

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

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

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

For a complete discussion of JOBLIB and STEPLIB DD statements, see the <u>Job Control Language</u> publication.

<u>Note</u>: Data sets with records of different formats, or data sets that reside on different types of devices, should not be concatenated.

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

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

  A complete discussion of SYSABEND and SYSUDUMP DD statements, including an example of use, appears in the <u>Job Control Language</u> publication.

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.
- 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 are to reside on magnetic tape or direct access volumes.
- Optimize I/O operations.

The following text describes the DD statement parameters that apply to processing unit record data sets and retrieving data sets created in previous job steps, or data sets created and cataloged in previous jobs (see Figure 11). The method of retrieving uncataloged data sets created in previous jobs is also discussed in this section. Parameters shown in Figure 10 and not mentioned in this section are used to create data sets, retrieve uncataloged data sets, and optimize I/O operations in job steps. These parameters are discussed in the sections "Creating Data Sets" and "Programming Considerations."

```
 {*   }1
 {DATA}

          /dsname                          \
          |dsname(element)                 |
          |*.ddname                        |
 DSNAME= <*.stepname.ddname                >
          |*.stepname.procstep.ddname      |
          |&name                           |
          \&name(element)                  /

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

          {PRTSP=0}
          {PRTSP=1}
          {PRTSP=2}
 DCB=(    {PRTSP=3}                              )

          {{MODE=E}{,STACK=1}}
          {{MODE=C}{,STACK=2}}
          subparameter-list4

 SYSOUT=A
 SYSOUT=B
 SYSOUT=(x[,program-name][,form-no.])5 6

                   [,DELETE  ]7    [,DELETE  ]
            {OLD}  [,KEEP    ]     [,KEEP    ]
 DISP=(     {NEW}  [,PASS    ]     [,CATLG   ]   )8
            {MOD}  [,CATLG   ]     [,UNCATLG ]
            {SHR}  [,UNCATLG ]

 LABEL=(subparameter-list) 4

 VOLUME=(subparameter-list)4
```

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

Figure 11. DD Statement Parameters

Unit Record Parameters

The UNIT, DCB, and SYSOUT parameters are used for unit record data sets; the * or DATA parameters designate that the data set for this job step follows in the input stream. Examples of DD statements for unit record data sets are shown in Figure 12.

Specifying Data in the Input Stream:

*
: indicates that a data set immediately follows this DD statement in the input stream. This parameter is used to specify a source module deck, object module deck, or data in the input stream. 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 in the input stream 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 // 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.

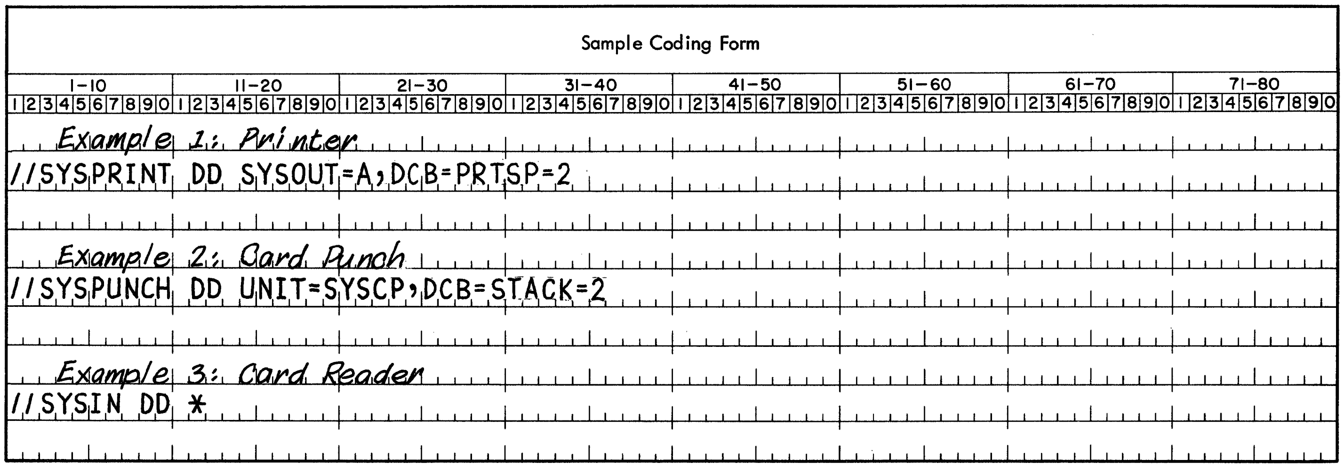
UNIT Parameter:

UNIT=(name[,{n|P}])
: specifies an address of an I/O device, a type of I/O device, or class of I/O devices, and the number of I/O devices assigned to a data set. When the system is generated, the "name" is assigned by the operating system or the installation. (See the section "System Generation Macro-Instructions" in the publication System Generation.) 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. The control program assigns as many devices as there are volumes indicated in the index and the label field of the cataloged data set.


Sample Coding Form

```
  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 12. Examples of Unit Record DD Statements

DCB Parameter:

DCB=PRTSP={0|1|2|3}
: specifies line spacing for the printer. The digits 0, 1, 2, and 3 specify no space, single space, double space, and triple space, respectively. The carriage control character in a FORTRAN record causes spacing before the line is printed. The PRTSP parameter causes spacing after the line is printed. A default value of 0 applies only to a FORTRAN program.

$DCB=\left(\begin{Bmatrix}\underline{MODE=E}\\MODE=C\end{Bmatrix}\begin{Bmatrix}\underline{,STACK=1}\\,STACK=2\end{Bmatrix}\right)$
: specify options for the card-read-punch. The MODE subparameter indicates whether the card is transmitted in column binary or EBCDIC mode; C specifies column binary, and E specifies EBCDIC.

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

Routing a Data Set to an Output Stream (SYSOUT)

Through the use of the SYSOUT parameter, output data sets can be routed to a system output stream and handled in much the same way as system messages

SYSOUT=A
: can be used with sequential schedulers to indicate that the data set is to be written on the system printer output device No parameter other than the DCB parameter has any meaning when SYSOUT=A is used. With systems providing multiprogramming with a fixed number of tasks, the processing program that writes the data must be in the lowest priority partition.

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

SYSOUT=(x[,program-name][,form-no.])
: is used with priority schedulers. When priority schedulers are used, the data set is normally written on an intermediate direct access device during program execution and later routed through an output stream to a system output device. The character "x" can be alphabetic or numeric, specifying the system output class. Output writers route data from the output classes to system output devices. The DD statement for this data set can also include a unit specification describing the intermediate direct access device and an estimate of the space required. If these parameters are omitted, the job scheduler provides default values as the job is read and processed.

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

If the output data set is to be printed or punched on a specific type of output form, a 4-digit "form-no." should be specified. This form number is used to instruct the operator, in a message issued at the time the data set is to be printed, of the form to be used.

Retrieving Previously Created Data Sets

If a data set on a magnetic tape or a direct access volume is created with standard labels and cataloged in a previous job or job step, all information for the data set such as device, volume, space, etc., is stored in the catalog and labels. This information need not be repeated in other DD statements. To retrieve the data set, the name (DSNAME) and disposition (DISP) of the data set must be specified.

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

If the data set was created with standard labels in a previous job but not cataloged, information concerning the data set, such as space, record format, etc., is stored in the labels. The volume and device information is not stored. To retrieve the data set, the name (DSNAME), if the data set is named, disposition (DISP), volume (VOLUME), and device (UNIT) must be specified.

If a data set created without standard labels in a previous job is retrieved, the LABEL and DCB parameters must be specified. The VOLUME, LABEL, and DCB parameters are discussed in the section "Creating Data Sets."

Examples of the use of DD statements to retrieve previously created data sets are shown in Figure 13.

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

Specifying a Cataloged Data Set by Name:

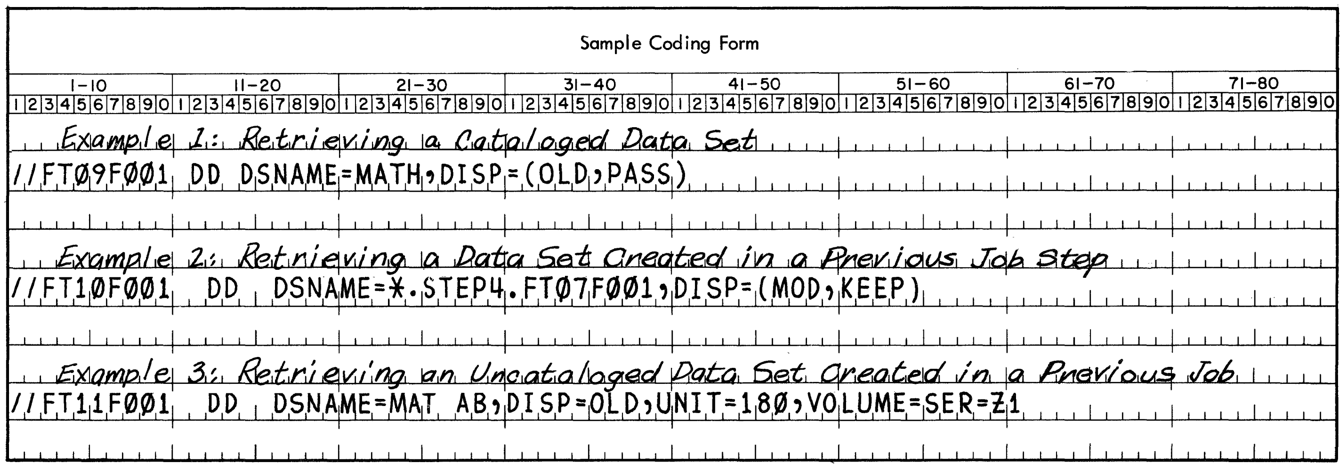
DSNAME=dsname
: the fully qualified name of the data set is indicated by "dsname." If the data set was previously created and cataloged, the control program uses the "dsname" to search the catalog, find the data set, and instruct the operator to mount the required volumes.

Specifying a Generation Data Group or PDS:

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

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

indicates the third most recent member of the generation data group FIRING. If "element" is a name, a member of a partitioned data set is indicated.


Sample Coding Form

```
  Example 1: Retrieving a Cataloged Data Set
//FTØ9FØØ1 DD DSNAME=MATH,DISP=(OLD,PASS)

  Example 2: Retrieving a Data Set Created in a Previous Job Step
//FT1ØFØØ1  DD  DSNAME=*.STEP4.FTØ7FØØ1,DISP=(MOD,KEEP)

  Example 3: Retrieving an Uncataloged Data Set Created in a Previous Job
//FT11FØØ1  DD  DSNAME=MAT AB,DISP=OLD,UNIT=18Ø,VOLUME=SER=Z1
```

Figure 13. Retrieving Previously Created Data Sets

Before any generation data set can be specified in the DSNAME parameter, the name for the generation data group must be inserted in the catalog index. The name of the generation data group is inserted by use of a utility program described in the section "Modifying System Control Data" in the *Utilities* publication.

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

Referring to a Data Set in the Current Job Step:

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

Referring to a Data Set in a Previous Job Step:

DSNAME=*.stepname.ddname
: indicates a data set defined in a DD statement in a previous job step in this job. The * indicates the current job, and "stepname" is the name of a previous job step. The name of the data set is copied from the DSNAME parameter in the DD statement named "ddname." For example, in the following control statements, the DD statement FT08F001 in job step S2 indicates that the data set name (TIME) is copied from the DD statement FT07F001 in job step S1.

```
//LAUNCH   JOB
//JOBLIB   DD DSNAME=FIRING,DISP=(OLD,PASS)
//S0 EXEC PGM=ROCKET
//FT05F001 DD DSNAME=RATES(+1),DISP=OLD
//FT07F001 DD DSNAME=TIME,DISP=(OLD,PASS)
//S2 EXEC PGM=DISTANCE
//FT08F001 DD DSNAME=*..FT07F001,DISP=OLD
//FT01F001 DD *
          .
          .
          .
```

Referring to a Data Set in a Cataloged Procedure:

DSNAME=*.stepname.procstep.ddname
: indicates a data set defined in a cataloged procedure invoked by a previous job step in this job. The * indicates the current job; "stepname" is the name of a previous job step; "procstep" is the name of a step in the cataloged procedure; and "ddname" is the name of the DD statement defining the data set.

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 may be accessed in following job steps by &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 object modules that will be link edited 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.

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

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

The first subparameter indicates when the data set was created.

NEW
: indicates that the data set is created in this step.

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

MOD
: indicates that the data set was created in a previous job or job step, and records are to 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 a data set specified as MOD does not exist, the specification is assumed to be NEW

SHR
: indicates that the data set resides on a direct-access volume and is used as input to a job whose operations do not

prevent simultaneous use of the data set as input to another job. This parameter has meaning only in a multiprogramming environment for existing data sets. If it is omitted in a multiprogramming environment, the data set is considered unusable by any other concurrently operating job. If it is coded in other than a multiprogramming environment, the system assumes that the disposition of the data set is OLD.

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

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

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

PASS
indicates that the data set is referred to in a later job step. When a subsequent reference to the data set is made, its PASS status lapses unless another PASS is issued. The final disposition of the data set should be stated in the last job step that uses the data set. When a data set is in PASS status, the operating system attempts to keep the volume(s) for the data set mounted. 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.

When a data set is concatenated with the system library through use of the JOBLIB DD statement, PASS assumes a different meaning. Without PASS in the JOBLIB statement, the concatenation is only in effect for the first job step. If PASS is specified, the concatenation is in effect for the entire 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 the data set to be removed from the catalog at the end of the job step. UNCATLG does not imply DELETE.

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 (NEW), it is deleted at the end of the current job step. If the data set existed before this job (MOD or OLD), it exists after the end of the job.

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

<u>Notes</u>:

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

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

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

DELIMITER STATEMENT

The delimiter statement (see Figure 14) is used to separate data from subsequent control statements in the input stream, and is placed after each data set in the input stream.

Name	
/*	

Figure 14. Delimiter Statement

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

COMMENT STATEMENT

The comment statement (see Figure 14.1) is used to enter any information considered helpful by the programmer. It can be inserted before or after any control statement. Comments can be coded in columns 4 through 80. The comments cannot be continued onto another statement. (If the comment statement appears on a system output listing, it can be identified by the appearance of *** in columns 1 through 3.)

Name	
//*	

Figure 14.1. Comment Statement

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

JOB PROCESSING

To execute a FORTRAN program, three steps are required -- compiling, link editing, and executing. Using cataloged procedures to make these steps easier is discussed in this section.

For each of the three steps involved in processing, ddnames and device names are specified by the operating system. These ddnames, options for the compiler and linkage editor, batched compilation, and specifying additional libraries for the linkage editor are discussed in this section.

USING CATALOGED PROCEDURES

Because writing job control statements can become time-consuming, IBM supplies four cataloged procedures to aid in the compiling, link editing, and executing of FORTRAN E programs. Each procedure requires a

```
//procstep.SYSIN DD
```

statement indicating the location of a source module or object module to the control program. In addition, a DD statement GO.SYSIN can be used to define data in the input stream for a procedure step that executes a load module. The job control statements needed to invoke the procedures, and deck structures used with the procedures are described in the following text.

Compile

The name of the cataloged procedure for compilation is FORTEC. It is invoked by the name FORTEC as the first parameter in an EXEC statement.

(The cataloged procedure, FORTEC, consists of the control statements shown in Figure 48 in "Cataloged Procedures.")

With the procedure FORTFC, a DD statement FORT.SYSIN indicating the location of the source module must be supplied. Figure 15 shows control statements that can be used to invoke the procedure.

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

Figure 15. Invoking the Cataloged Procedure FORTEC

Single Compile: A sample deck structure to compile a single source module is shown in Figure 16.

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

Figure 16. Compiling a Single Source Module

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 last statement in the source module.

Batched Compile: A sample deck structure to batch compile is shown in Figure 17.

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

Figure 17. Compiling Several Source Modules

If 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 a FORTRAN source module and link edit the resulting object module is named FORTECL. It is invoked by the name FORTECL as the first parameter in an EXEC statement.

(The cataloged procedure FORTECL consists of the job control statements shown in Figure 49 in "Cataloged Procedures".)

With the procedure FORTECL, a DD statement FORT.SYSIN must be supplied to indicate the location of the source module. This cataloged procedure writes the resulting load module in the FORTRAN library (SYS1.FORTLIB); however, an overriding DD statement

```
//LKED.SYSLMOD DD DSNAME=SYS1.FORTLIB(name)
```

can be supplied to name the resulting load module. Figure 18 shows control statements that can be used to invoke the procedure.

```
//jobname JOB
// EXEC FORTECL
//FORT.SYSIN DD *
┌─────────────────────────────────────────┐
│          FORTRAN Source Module          │
└─────────────────────────────────────────┘
/*
//LKED.SYSLMOD DD DSNAME=SYS1.FORTLIB(name)
```

Figure 18. Invoking the Cataloged Procedure FORTECL

Again the source module is defined as data in the input stream. Note that the DD statement LKED.SYSLMOD must follow the delimiter statement for the source modules in the input stream.

Batch Compile and Link Edit: A sample deck structure to batch compile several source modules and link edit the resulting object modules is shown in Figure 19. The resulting load module is placed in the FORTRAN library and assigned the name CHEM.

```
//JOBCLE JOB 012,'E .SMITH'
// EXEC FORTECL
//FORT.SYSIN DD *
┌─────────────────────────────────────────┐
│       First FORTRAN Source Module       │
└─────────────────────────────────────────┘
                    .
                    .
                    .
┌─────────────────────────────────────────┐
│       Last FORTRAN Source Module        │
└─────────────────────────────────────────┘
/*
//LKED.SYSLMOD DD DSNAME=SYS1.FORTLIB(CHEM)
```

Figure 19. Compiling and Link Editing Several Source Modules

Single Compile and Link Edit: A sample deck structure to compile and link edit a single source module, placing it in the FORTRAN library, and assigning the resulting module the name XYZ is shown in Figure 20. The source module is read from the cataloged sequential data set SOMOD.

```
//COMPLED JOB 527,'JOHN BROWN'
// EXEC FORTECL
//FORT.SYSIN DD DSNAME=SOMOD,DISP=OLD
//LKED.SYSLMOD DD DSNAME=SYS1.FORTLIB(XYZ)
```

Figure 20. Compiling and Link Editing a Source Module Residing in a Cataloged Data Set

Because the source modules reside in a cataloged data set, the delimiter statement is omitted.

Link Edit and Execute

The cataloged procedure to link edit FORTRAN object modules and execute the resulting load module is named FORTELG. It is invoked by the name FORTELG as the first parameter in an EXEC statement.

(The cataloged procedure, FORTELG, consists of the control statements shown in Figure 50 in "Cataloged Procedures").

With the procedure FORTELG, a DD statement LKED.SYSIN, which indicates the location of the object module, must be supplied.

Three data sets are defined by DD statements in the cataloged procedure for use during execution of the load module. If the programmer intends to use these DD statements, he can use data set reference numbers one, two, and three in the following way:

1 - the data set defined by the DD statement GO.SYSIN (used primarily to read data from the input stream)

2 - card output

3 - printed output

Any of the DD statements for these data set reference numbers may be overridden, as shown in "Cataloged Procedures".

Figure 21 shows control statements that can be used to invoke the FORTELG cataloged procedure.

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

Figure 21. Invoking the Cataloged Procedure FORTELG

Link Edit: 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 22.

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

Figure 22. Link Edit and Fxecute

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.

A sample deck structure is shown in Figure 23 for object modules that are members of the cataloged sequential data set, OBJMODS, that resides on a tape volume. In addition a data set in the input stream is processed using data set reference number 1.

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

Figure 23. Link Edit and Execute (Object Modules in a Cataloged Data Set)

Compile, Link Edit, and Execute

The fourth cataloged procedure, FORTECLG, passes a source module through three procedure steps - compile, link edit, and execute. The cataloged procedure is invoked by the name FORTECLG as the first parameter in an EXEC statement.

(The cataloged procedure, FORTECLG consists of the control statements shown in Figure 51 in "Cataloged Procedures.")

The SYSIN data set (source module) must be defined to the compiler. Figure 24 shows statements that can be used to invoke the procedure FORTECLG.

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

Figure 24. Invoking the Cataloged Procedure FORTECLG

Single Compile, Link Edit, and Execute: Figure 25 shows a sample deck structure to compile, link edit, and execute a single source module.

```
//JOBSCLG JOB 00,TJONES,MSGLEVEL=1
//EXECC EXEC FORTECLG
//FORT.SYSIN DD *
┌──────────────────────────────────────────┐
│          FORTRAN Source Module           │
└──────────────────────────────────────────┘
/*
```

Figure 25. Single Compile, Link Edit, and Execute

Batched Compile, Link Edit, and Execute: Figure 26 shows a sample deck structure to batch compile, link edit, and execute. The source modules are placed in the input stream along with a data set that is read using data set reference number 1 in the load module.

```
 //JOBBCLG JOB 00,JBOND,MSGLEVEL=1
 //EXECCLG EXEC FORTECLG
 //FORT.SYSIN DD *
┌─────────────────────────────────────────┐
│       First FORTRAN Source Module       │
└─────────────────────────────────────────┘
                     .
                     .
                     .
┌─────────────────────────────────────────┐
│       Last FORTRAN Source Module        │
└─────────────────────────────────────────┘
 /*
 //GO.SYSIN DD *
┌─────────────────────────────────────────┐
│                  Data                   │
└─────────────────────────────────────────┘
 /*
```

Figure 26. Batched Compile, Link Edit, and Execute

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 transmitted to main storage by I/O operations under control of FORMAT statements, one character indicated by the FORMAT statement is contained in one byte.

When data is read into main storage, it is translated into internal format. A real constant or variable, or an integer constant or variable occupies one word (four bytes). A double-precision constant or variable occupies a double-word (eight bytes). For I/O operations not under FORMAT control, variables and constants are read from and written on the volume in the internal format.

COMPILER PROCESSING

The names for DD statements (ddnames) relate I/O statements in the compiler to 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 IEJFAAA0. If the compiler is to be executed without using the supplied cataloged procedures in a job step, the EXEC statement parameter

PGM=IEJFAAA0

must be used.

Compiler ddnames

The compiler can use six 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, four of these data sets are necessary -- SYSIN, SYSPRINT, SYSUT1, and SYSUT2, along with the direct-access volume(s) that contains the operating system. With these four data sets, only a listing is generated by the compiler. Two optional data sets are provided for writing the object module: the SYSPUNCH data set is intended for punching the object module and the SYSLIN data set is intended for writing the object module on a magnetic tape or a direct access volume.

For the DD statement SYSIN or SYSPRINT, an intermediate storage device may be specified instead of the card reader or printer. The intermediate storage device usually is magnetic tape or a direct access device.

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

Table 2. Compiler ddnames

ddname	Function	Device Requirements	Record Length[1]
SYSIN	Reading the source module	card reader, direct access, or magnetic tape	80
SYSPRINT	Writing the storage map, listing, and messages	printer, direct access, or magnetic tape	121
SYSPUNCH	Punching the object module deck	card punch, direct access, or magnetic tape	80
SYSLIN	Output data set for the object module, used as input to linkage editor	card punch, direct access, or magnetic tape	80
SYSUT1	Work data sets used by the compiler for compilation	direct access or magnetic tape	Determined by the compiler during compilation. Not specified by the programmer.
SYSUT2			

[1]The maximum number of records per block for the SYSIN, SYSPRINT, and SYSPUNCH data sets is determined by device type (see Table 12). The maximum number of records per block for the SYSLIN data set is either 1, 5, or 40, depending on which linkage editor is used to read the data set.

The following features of the compiler can be used only if the PRFRM compiler option is specified. For a more detailed description of the SPACE/PRFRM option, see "Compiler Options" in this section.

If the PRFRM compiler option is specified in the EXEC statement, the FORTRAN compiler can read or write blocked records for SYSIN, SYSPUNCH, SYSPRINT, and SYSLIN. Blocked records are grouped before they are written on a volume; the entire group is then written together, instead of writing each record individually. (Blocking for SYSUT1 and SYSUT2 is determined by the compiler; the programmer cannot specify blocking for these data sets.) Figure 27 illustrates blocked records.

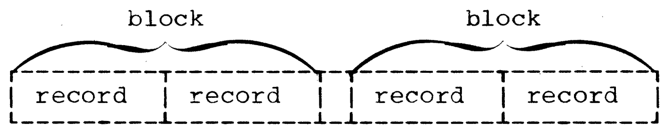


Figure 27. Blocked Records

Blocking saves space on the volume and increases the efficiency of the compiler because fewer I/O operations are performed. The programmer specifies whether records are blocked by the BLKSIZE subparameter in the DCB parameter of the DD statement (see "Creating Data Sets"). Records can be blocked only if they are read from or written on a direct access or magnetic tape volume. The SYSLIN data set should be blocked only if the object module is to be used as input to either of the linkage editor programs IEWLF440 or IEWLF880. Table 2 shows the record length and maximum number of records per block for each data set.

If the SPACE compiler option is specified, other data sets cannot be concatenated with the SYSIN data set. If the PRFRM compiler option is specified, other data sets can be concatenated with the SYSIN data set.

If the SPACE compiler option is specified, the SYSPRINT, SYSPUNCH, and SYSLIN data sets must be sequential data sets; only the SYSIN data set can be read as a member of a PDS. However, if the PRFRM compiler option is specified, the SYSPRINT, SYSPUNCH, and SYSLIN data sets can be written as members of partitioned data sets.

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.

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

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

Table 4. Correspondence Between Compiler ddnames and Device Classes

ddname	Possible Device Classes
SYSIN	SYSSQ, the input stream device (specified by DD * or DD DATA), or a device specified as the card reader
SYSPRINT	A, SYSSQ
SYSPUNCH	SYSCP[1], SYSSQ, SYSDA
SYSUT1	SYSSQ, SYSDA
SYSUT2	SYSSQ, SYSDA
SYSLIN	SYSSQ, SYSDA, SYSCP[1]
[1]Both the SYSPUNCH and SYSLIN data sets cannot be written on the SYSCP device class in the same job step.	

Compiler Options

Options (Figure 28) may be passed to the compiler through the PARM parameter in the EXEC statement. The following information may be specified:

1 Amount of main storage allocated to the compiler for this compilation

2. Maximum length of a FORTRAN record written under FORMAT control.

3. Name assigned to the program.

4. Whether the source program is coded in Binary Coded Decimal (BCD) or Extended Binary Coded Decimal Interchange Code (EBCDIC).

5. Whether a list of source statements is printed.

6. Whether an object module is punched.

7. Whether a map of the object module is printed.

8. Whether the compiler writes the object module on an output data set that resides on a direct access or tape volume.

9. Whether any additional main storage is used either to compile a larger source module or to increase the speed of compilation.

10. Whether the source statements contain embedded blanks in variable names, statement numbers, constants and reserved words, whether meaningful blanks are not inserted between names and reserved words, and whether FORTRAN keywords are used as variable names in the source program.

There is no specified order for compiler options.

Figure 28 shows the compiler options. For most options, a default for the option is underlined. If an alternative is not underlined, the default is indicated in the explanation of that option. The defaults indicated in this publication are the standard defaults for FORTRAN(E). However, when the operating system is generated, the installation can change the defaults for compiler options. For more information about changing the defaults for compiler options, see the section "System Generation Macro-Instructions" in the System Generation publication. Before using any of the default options, the programmer should determine the defaults for his installation. For purposes of illustration, this publication assumes that the defaults chosen by the installation are the standard defaults

```
┌──────────────────────────────────────────────────────────────────────────────────────┐
│{PARM          }  '    {nnnnK  }                                   {,BCD   }  {,SOURCE  }│
│{PARM.procstep}=  SIZE={yyyyyyy}  [,LINELNG=zzz] [,NAME=xxxxxx]  {,EBCDIC}  {,NOSOURCE}│
│                                                                                       │
│                  {,DECK  } {,MAP  } {,LOAD  } {,SPACE} {,ADJUST  } '1,2,3              │
│                  {,NODECK} {,NOMAP} {,NOLOAD} {,PRFRM} {,NOADJUST}                    │
├──────────────────────────────────────────────────────────────────────────────────────┤
│1If the information specified contains blanks, parentheses, or equal signs, it must not│
│ be delimited by parentheses but by apostrophes.                                       │
│2If only one option is specified and it does not contain any blanks, parentheses, or  │
│ equal signs, the delimiting parentheses or apostrophes may be omitted.                │
│3The maximum number of characters allowed between delimiting parentheses is 100. If   │
│ the option list is enclosed in apostrophes, however, the PARM parameter must be coded │
│ on one card.                                                                          │
└──────────────────────────────────────────────────────────────────────────────────────┘
```

Figure 28. Compiler Options

SIZE=yyyyyyy or SIZE=nnnnK: The SIZE option indicates the amount of main storage available for the compilation. The programmer specifies a number yyyyyyy, (yyyyyyy ≥ 15360) or nnnnK (K=1024 and 15≤nnnn≤9999). If the option is not specified or the number specified is less than 15,360 bytes, the compiler assumes 15,360. If the number specified is greater than the amount available, processing continues, provided the amount available is at least 15,360 bytes when the SPACE option is specified, or at least 19,456 bytes when the PRFRM option is specified. This figure assumes no blocking. If the input is blocked (e.g., by an input reader), a figure that is 160 times the blocking factor in bytes must be added to the 19,456-byte specification in the SIZE option. (See "SPACE or PRFRM.")

LINELNG=zzz: The LINELNG option indicates the maximum length of a FORTRAN record written under control of a FORMAT statement. The specified number zzz (0<zzz<256) represents the maximum length of a FORTRAN record. During compilation, the length of all records is calculated using the coded information in the FORMAT statement. If the record length exceeds zzz, a warning is issued by the compiler. If this option is not specified, zzz is assumed to be 132. For example, assume that 144 positions are specified in the LINELNG option and the following source statements are compiled:

```
    WRITE(7,10) POINT,ALPHA,I,J,K,L
            .
            .
            .
10  FORMAT(2F30.8,4I30)
```

A warning is issued because the record length indicated by the FORMAT statement is 180, and the LINELNG parameter indicates a maximum length of 144.

NAME=xxxxxx: The NAME option specifies the name (xxxxxx) assigned to the module by the programmer, where xxxxxx consists of one to six alphameric characters, the first of which is alphabetic. If NAME is not specified, the compiler assumes either the name MAIN for a main program or the name of the subprogram specified in the SUBROUTINE or FUNCTION statement for subprograms. If there is a conflict between the name given to the subprogram in the first statement of the source module and the name specified in the NAME option, the name specified in the SUBROUTINE or FUNCTION statement takes precedence. The name appears in the source listing, storage map, and object module.

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.

Note: The compilers do 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. Therefore, programs keypunched in BCD, should be carefully screened in order to avoid errors relating to literal data and print control characters.

SOURCE or NOSOURCE: The SOURCE option specifies that the source listing is written on the data set specified by the SYSPRINT DD statement. The NOSOURCE option indicates that 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 the compiled source module (i.e., the object module) is written on the data set specified by the SYSPUNCH DD statement. NODECK specifies that no object module is written. A description of the object module is given in the section "System Output."

MAP or NOMAP: The MAP option specifies that a storage map of the object module is written on the data set specified by the SYSPRINT DD statement; the option NOMAP specifies that no map is written A description of the map is given in the section "System Output."

LOAD or NOLOAD: The LOAD option indicates that the object module is written on the data set specified by the SYSLIN DD statement. This option must be used if a

cataloged procedure to compile, link edit, and execute is used. A description of the object module is given in the section "System Output".

The NOLOAD option indicates that the object module is not written on the SYSLIN data set. When NOLOAD is specified, the compiler automatically returns a condition code of 12. This option must not be used if a cataloged procedure to compile, link edit, and execute is used. If NOLOAD and DECK are specified, the SYSPUNCH data set may be used as input to the linkage editor.

If the LOAD and DECK options are specified, the object module is written on the two data sets, indicated by the SYSLIN and SYSPUNCH DD statements.

SPACE or PRFRM: When the PRFRM option is specified, the size of a source module is limited. (See Table 14.) By specifying the SPACE option and more than 15360 bytes in the SIZE option, the limit for the size of the source module is increased.

The PRFRM option indicates that excess main storage is allocated for faster compilations rather than larger source modules. The PRFRM option must be specified if any of the compiler data sets SYSIN, SYSPRINT, or SYSPUNCH are allocated to non-unit-record devices (e.g., priority schedulers). To block records for the compiler data sets SYSIN, SYSPRINT, SYSPUNCH, and SYSLIN, or to write the SYSPRINT, SYSPUNCH, and SYSLIN data sets as members of partitioned data sets, the PRFRM option must be specified. Other data sets can be concatenated with the SYSIN data set only if the PRFRM option is specified. (Note: Only data sets that reside on the same type of device can be concatenated.)

To ensure that these options improve the operation of the compiler, at least 19456 bytes should be allocated to the compiler in the SIZE option. If less than 19456 bytes are specified or if less than 19456 bytes are available and the PRFRM option is specified, processing continues using the SPACE option and the amount of storage available. If blocked input and output is specified with the PRFRM option, the SIZE option must specify enough storage to contain blocked records. Any storage not used by the PRFRM option is used to compile a bigger source module and increase the size of the buffers which decreases the number of I/O operations and increases the speed of the compiler.

ADJUST or NOADJUST: The ADJUST option indicates that the source module contains embedded blanks, contains no meaningful blanks, and uses keywords as variable names in the source statements. With the ADJUST option, the source statement can contain embedded blanks. For example, the source statements

```
F O R MA T (1H , I10)
DELTA T=T /INC
```

are valid. With the ADJUST option, the source statement need not contain meaningful blanks. For example, the source statements

```
DOUBLEPRECISIONFUNCTIONDPROD(X,Y)
DIMENSIONABC(10)
```

are valid. With the ADJUST option, the source can contain FORTRAN keywords (GO, DO, IF, READ, FIND, WRITE, etc.) used as variable names. For example, the source statements

```
IF(IF) 20,30,40
READ=A+B+C
```

are valid.

If NOADJUST is specified, the source module must not contain embedded blanks, must contain meaningful blanks, and must not contain FORTRAN keywords used as variable names. However, with the NOADJUST compiler option, source modules are compiled faster. For example, the previous source statements must be written as follows to make them acceptable to the compiler when the NOADJUST option is used.

```
FORMAT(1H ,I10)
DELTAT=T/INC
DOUBLE PRECISION FUNCTION DPROD(X,Y)
DIMENSION ABC(10)
IF(IFX)20,30,40
READX=A+B+C
```

Multiple Compilation Within a Job Step

Several compilations may be performed within one job step, if the conditions shown in Table 5 are met.

Table 5. Conditions for Multiple Compilation

Option	Input Stream Device	Source Modules Reside On
SPACE	card reader	input stream
	tape	input stream
		card reader
PRFRM	card reader	input stream, tape, card reader, direct access
	tape	

The compiler recognizes the FORTRAN END statement in a source deck, compiles the program, and determines if another source program follows the END statement. If there is another source program, another compilation is initiated (see Figure 29).

```
//JOBRA JOB  ,'RBLACK'
//STEP1 EXEC FORTEC
//FORT.SYSIN DD *
   1 READ (9,10)A,B,C
       .
       .
       .
     END
     SUBROUTINE CALC
       .
       .
       .
     END
/*
```

Figure 29. 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. These options are then used for all compilations in the batched compilation.

A main program compiled first in a multiple compilation is given the name specified in the NAME option. Any subprogram in a multiple compilation is given the name of the subprogram in the first card of the source subprogram. For example, in the multiple compilation,

```
//MULTCOMP JOB  ,'FRANK KELLY'
// EXEC FORTEC,PARM.FORT='NAME=GAMMA'
//FORT.SYSIN DD *
      SUBROUTINE ALPHA
        .
        .
        .
      END
      FUNCTION BETA(X,Y,Z)
        .
        .
        .
      END
/*
```

the first module is given the name ALPHA and the second is given the name BETA.

Any main program after the first program is given the name MAIN. Moreover, if the NAME option is not specified and the first module is a main program, the first program is also given the name MAIN. For example, in the multiple compilation,

```
//MULCOM     JOB
// EXEC   FORTEC
//FORT.SYSIN DD *
       READ(1,10)ALP,BETA
         .
         .
         .
       END
       SUBROUTINE INVERT(A,B)
         .
         .
         .
       END
       READ(5)P,Q,R
         .
         .
         .
       END
/*
```

both the first and third programs are given the name MAIN. The second program is assigned the name INVERT.

When a multiple compilation is performed, the SYSLIN or SYSPUNCH data set contains all the object modules because only one SYSLIN DD statement may be supplied for compiler output. The object modules are placed sequentially on the volume.

Object Module 1	Object Module 2	...

LINKAGE EDITOR PROCESSING

The linkage editor processes 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

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

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

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

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

PGM=IEWL

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

Linkage Editor Input and Output

There are two types of input to the linkage editor: primary and secondary. Primary input consists of a sequential data set that contains object modules and linkage editor control statements. Any external references among object modules in the primary input are resolved by the linkage editor as the primary input is processed. Furthermore, the primary input contains references to the secondary input. These references are linkage editor control statements and/or FORTRAN external references in the modules.

Secondary input resolves these 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 that omit the ddname, the automatic call library can be concatenated with other partitioned data sets. Three types of additional input may be specified by the programmer:

- An object module used as the main program in the load module being constructed. This object module, which can be accompanied by linkage editor control statements, is either a member of a PDS or is a sequential data set. The first record in the primary input 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 to include the module.

- 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 a PDS. A linkage editor LIBRARY control statement that defines the data set to the linkage editor must be given to include modules from the data set in the load module.

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 output of the linkage editor consists of the load module, module map, and error messages. The load module is always placed in a PDS. Error messages and the optional module map are written on an intermediate storage device or a printer. In addition, a work data set is required by the linkage editor to do its processing. Figure 30 shows the I/O flow in linkage editor processing.

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 (e.g., 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

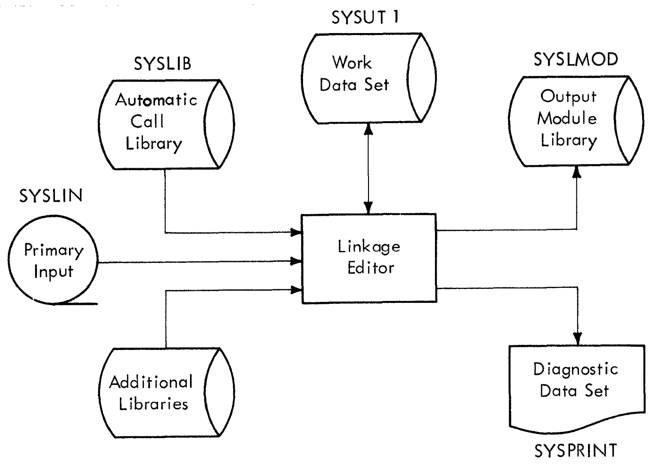


Figure 30. Linkage Editor Input and Output

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. (The other data sets are partitioned or sequential.) 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.

In addition, if one of the higher level linkage editors (program name: IEWLF440 or IEWLF880) is used, the SYSLIN data set can contain blocked records. The linkage editor can then accept a blocked SYSLIN data set that is created by the compiler. The record length for the SYSLIN data set is 80 bytes. With the linkage editor IEWLF440 the maximum number of records per block is 5. With IEWLF880, the maximum number of records per block is 40.

The device classes used by the compiler (see Table 3) <u>must</u> 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 specified by neither INCLUDE nor 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 (PDS) or a sequential data set. The "ddname" specifies a DD statement that defines either a PDS containing object modules and control statements or just load modules, or defines a sequential data set containing object modules and linkage editor control statements. The "member name" is the name of a member of a PDS and is not used when a sequential data set is specified.

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

LIBRARY Statement:

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

The LIBRARY statement is used to include members of additional libraries. The "ddname" must be the name of a DD statement that specifies a PDS that contains either object modules and linkage editor control statements, or just load modules. The "member name" is an external reference that is unresolved after primary input 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 31 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.

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 the SYSLIN data set or modules identified by INCLUDE statements.
2. Modules identified by the LIBRARY statement.
3. Modules appearing in the SYSLIB data set.

For example, if a module named SIN appears both in a module identified in a 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.

```
//JOBX     JOB
//STEP1    EXEC      PGM=IEJFAAA0,PARM='NAME=MAIN,LOAD'
                               .
                               .
                               .
//SYSLIN   DD        DSNAME=&GOFILE,DISP=(,PASS),UNIT=SYSSQ
//SYSIN    DD        *
           Source module for MAIN
/*
//STEP2    EXEC      PGM=IEJFAAA0,PARM='NAME=SUB1,LOAD'
                               .
                               .
                               .
//SYSLIN   DD        DSNAME=&SUBPROG1,DISP=(,PASS),UNIT=SYSSQ
//SYSIN    DD        *
           Source module for SUB1
/*
//STEP3    EXEC      PGM=IEJFAAA0,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 31. Linkage Editor Example Using INCLUDE and LIBRARY Statements

Multiple Link Editing Within a Step

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

Operation	Operand
NAME	member-name[(R)]

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

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

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

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

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

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

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

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

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

Other Linkage Editor Control Statements

In addition to the LIBRARY, INCLUDE, and NAME 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 section "Specifying Additional Processing" in 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 FORTRAN programming are:

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

                 [,LIST][,OVLY])
```

MAP or XREF: The MAP option instructs 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. If neither option is specified, neither the map nor the cross-reference listing is generated. Descriptions of the map and cross-reference listing are given in "System Output."

LET or XCAL: The LET option instructs linkage editor to mark the load module ready for execution even though error conditions were found. The XCAL option informs the linkage editor to mark the load module executable even though valid exclusive branches are made between modules that overlay each other.

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.

When an object module will be link edited again prior to its use in execution and that module contains either

1. An input/output statement (READ, WRITE, BACKSPACE, REWIND, END FILE),

2. A STOP/PAUSE statement,

3. Any service subprogram (SLITE, SLITET, OVERFL, DVCHK, EXIT, DUMP, PDUMP), or

4. Any one of the following library subprograms

DEXP	DLOG	DLOG10	DSIN
DCOS	DSQRT	DTANH	EXP
ALOG	ALOG10	SIN	COS
SQRT	TANH		

NCAL must be specified. An I/O statement, a STOP or PAUSE statement, any service subprogram, or any of the above library subprograms require FORTRAN load module execution routines. These routines are inserted by the linkage editor, and must be inserted only once in any load module. When the final linkage editor processing for the module is performed, NCAL should not be specified and the load module execution routines will be inserted.

LIST: The LIST option indicates that linkage editor control statements are listed in card-image format on the diagnostic output data set.

OVLY: The OVLY option indicates to the linkage editor that an overlay structure is to be constructed by the linkage editor. This option must be used if an OVERLAY linkage editor control statement is used. If an OVERLAY statement is not used, the OVLY option is ignored. For more information about overlay structures see the Linkage Editor publication.

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 adhere to 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, indicating the name of the private library, must be supplied to concatenate the private library with the system library. For example, assume that the load modules CALC and ALGBRA in the PDS MATH and the load module MATRIX in the PDS MATRICES are executed in the following job:

```
//JOBN JOB 00,JOHNSMITH
//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 either sequential or direct and must be defined by DD statements. The correspondence between a data set reference number and a DD statement is made by a ddname.

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

```
FTxxFyyy
```

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

Data Set Reference Number (xx): When the system is generated, the upper limit for data set reference numbers 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 refers to sequential data sets that are written using the same data set reference number.

For sequential or partitioned data sets, the first FORTRAN sequence number is always 001. This sequence number changes only when an END FILE statement is executed and the program later executes a READ or WRITE statement using the same data set reference number. For example, the following statements, executed in the order shown, cause the FORTRAN sequence number to change.

```
WRITE(10,5)A,B,C
      .
      .
END FILE 10
      .
      .
      .
WRITE(10,5)X,Y,Z
```

For the first WRITE, a DD statement identified by the ddname FT10F001 defines the data set. For the second WRITE, a DD statement identified by the ddname FT10F002 defines the data set.

For direct data sets, the FORTRAN sequence number is always 001. Attempting to execute an END FILE statement for a direct data set is ignored.

A DD statement with the required ddname must be supplied every time the WRITE, END FILE, READ/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
   ENDFILE 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
   ENDFILE 17
     .
     .
     .
```

If the preceding instructions are used to write a tape, the output tape has the appearance shown in Figure 32.

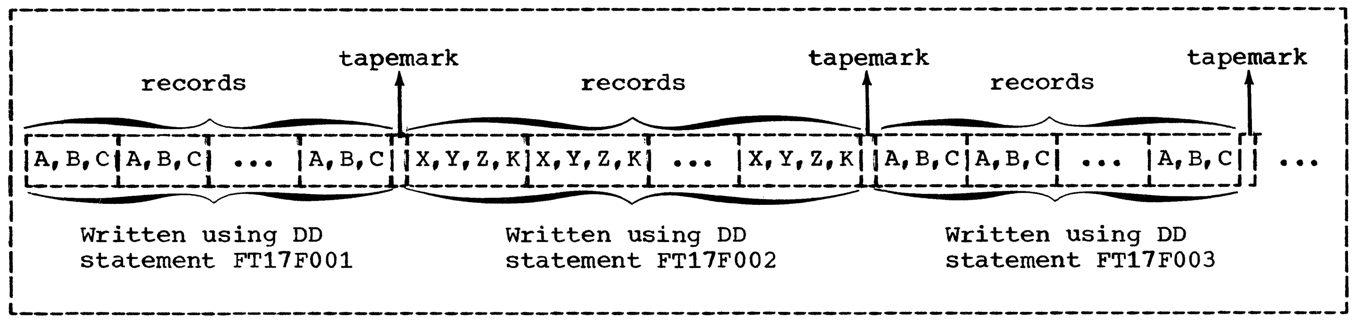

records
tapemark
records
tapemark
records
tapemark
A,B,C A,B,C ... A,B,C
X,Y,Z,K X,Y,Z,K ... X,Y,Z,K
A,B,C A,B,C ... A,B,C ...
Written using DD statement FT17F001
Written using DD statement FT17F002
Written using DD statement FT17F003

Figure 32. Tape Output for Several Data Sets Using Same Data Set Reference Number

Retrieving Data Sets Written with Varying FORTRAN Sequence Numbers

Retrieving the data sets shown in Figure 32 depends on when the data set was created and whether it was cataloged when it was created. There are four distinct conditions:

1. The data set is created in the job step in which it is retrieved.
2. The data set is created in one job step and retrieved in another job step, both steps in the same job.
3. The data set was created and cataloged in a previous job.
4. The data set was created in a previous job, but was not cataloged.

To retrieve the data sets shown in Figure 32, the data set sequence numbers in the LABEL parameter must be supplied in DD statements used to write the data sets. The LABEL parameter is described in detail in the section "Creating Data Sets."

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

The "data-set-sequence-number" indicates the position of the data set on a sequential volume. This sequence number is cataloged along with the remainder of the information in the DD statement. 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 32 is read in the same job step in which it is created, an END FILE statement and then a REWIND statement must be issued after the last WRITE instruction. The FORTRAN sequence number is incremented by the execution of the END FILE statement 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. For example, the following DD statements are used to write the three data sets shown in Figure 32 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)
```

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 created 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
```

If the data sets shown in Figure 32 are cataloged for later reading, the following DD statements should be used to write the data sets:

```
//FT17F001 DD DSNAME=N1,LABEL=(1,NL),          X
//     DISP=(,CATLG)
//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 only information necessary to retrieve the data sets is the DSNAME and the DISP parameters. (The data set sequence number is stored in the catalog and is accessible to the control program.) For example, if data set reference number 10 is used to retrieve the data set N1, the following DD statement is used to retrieve the data set:

```
//FT10F001 DD DSNAME=N1,DISP=OLD
```

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

Assume the data sets shown in Figure 32 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 to retrieve the data set:

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

REWIND and BACKSPACE Statements

The REWIND and BACKSPACE statements force execution of positioning operations for sequential data sets by the control program. For direct access data sets, REWIND and BACKSPACE operations are ignored.

The REWIND statement instructs the control program to position the volume on the device so that the next record read or written is the first record transmitted for that data set reference number on that volume, irrespective of data set sequence numbers. The space acquired dynamically for I/O buffers for a data set is released as part of the REWIND operation. For this reason, a program that uses many data sets may conserve main storage by issuing REWIND statements after processing is completed.

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

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, DUMP, and PDUMP 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 FORTLIB macro instruction in the section "System Generation Macro-Instructions.") If this data set is not defined and an error condition is encountered during the execution of the job step, the job step is terminated and a condition code of 16 is issued.

Execution Device Classes

For load module execution, the programmer can use the same names assigned to device classes used by the compiler (shown in Table 3). However, additional names for specific devices and device classes can be assigned by the installation where the system is generated. The programmer can choose which device to use for his data sets, and can specify the name of that device or class of devices to which that device belongs in the UNIT parameter of the DD statement.

However, a direct access device must be used for a data set which is defined (by the DEFINE FILE statement) as a direct access data set in the FORTRAN program.

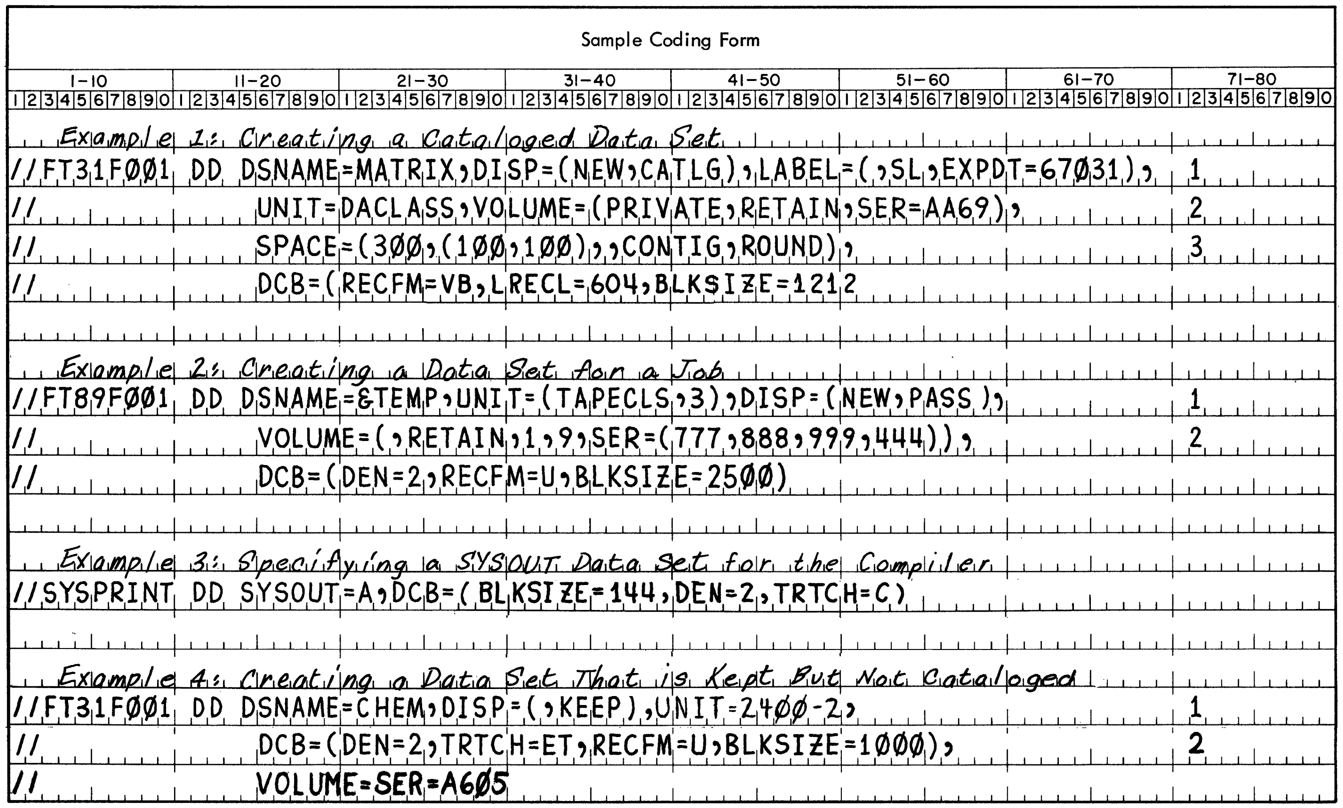
CREATING DATA SETS

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."

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

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

DSNAME -	name of the data set
UNIT -	class and number of devices used for the data set
VOLUME -	volume on which the data set resides
LABEL -	label specification
DISP -	the status of the data set at the beginning of the step and the disposition of the data set after the completion of the step
SYSOUT -	ultimate device for printer data sets
DCB -	tape density, record format, record length



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
//FT31FØØ1 DD DSNAME=MATRIX,DISP=(NEW,CATLG),LABEL=(,SL,EXPDT=67Ø31),    1
//             UNIT=DACLASS,VOLUME=(PRIVATE,RETAIN,SER=AA69),            2
//             SPACE=(3ØØ,(1ØØ,1ØØ),,CONTIG,ROUND),                      3
//             DCB=(RECFM=VB,LRECL=604,BLKSIZE=1212

  Example 2: Creating a Data Set for a Job
//FT89FØØ1 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=25ØØ)

  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
//FT31FØØ1 DD DSNAME=CHEM,DISP=(,KEEP),UNIT=24ØØ-2,                      1
//             DCB=(DEN=2,TRTCH=ET,RECFM=U,BLKSIZE=1ØØØ),                2
//             VOLUME=SER=A6Ø5
```

Figure 33. Examples of DD Statements for Creating Data Sets

```
 (         (dsname          ) )
 ( DSNAME= (dsname(element) ) )
 (         (&name           ) )
 (         (&name(element)  ) )
 ( DUMMY                      )
 ( DDNAME=ddname              )

UNIT=(name[,{n|P}¹])²
                                                                              [,SER=(volume-serial-number[,volume-serial-number]...)³]
VOLUME=([PRIVATE][,RETAIN][,volume-sequence-number][,volume-count]            [      (dsname                        )                 ])⁴
                                                                              [,REF= (*.ddname                      )                 ]
                                                                              [      (*.stepname.ddname             )                 ]
                                                                              [      (*.stepname.procstep.ddname    )                 ]

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

                                     (,NL )  [,IN ]  [,EXPDT=yyddd]
LABEL=([data-set-sequence-number]   (,SL )  [,OUT]  [,RETPD=xxxx ])⁸
                                     (,BLP)

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

      [dsname                     ] [     (0)] [       (C )] [        (1)¹⁰]              [        ({F|U}[A|M][T][,BLKSIZE=xxxx]               )]
DCB=( [*.ddname                   ] [,DEN=(1)] [,TRTCH=(E )] [,BUFNO=(2)   ] [,OPTCD=C]   [,RECFM=(V[A|M][T],LREC=xxxx,BLKSIZE=xxxx          )])¹¹
      [*.stepname.ddname          ] [     (2)] [       (T )]                              [        ({F|V}B[A|M][T],LRECL=xxxx,BLKSIZE=xxxx     )]
      [*.stepname.procstep.ddname ] [     (3)] [       (ET)]                              [                                                    ]
                                                                                          [,BLKSIZE=xxxx¹²                                     ]
```

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."
¹⁰BUFNO 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 34. DD Parameters for Creating Data Sets

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 may be read as input to a FORTRAN object program or created as output from a FORTRAN object program, but only when the member name and either LABEL=(,,,IN) or LABEL=(,,,OUT) are specified in the associated DD statement.

{DSNAME=&name }
{DSNAME=&name(element)}
specify data sets that are temporarily created for the execution of a single job or job step.

DUMMY
is specified in the DD statement to inhibit write operations specified for the data set. The 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.) The FORTRAN program-

mer should not specify DUMMY for a data set that is to be read; an end of data set condition results, and the execution of the load module is terminated.

DDNAME=ddname
: indicates a pseudo data set that will assume the characteristics specified in a subsequent DD statement "ddname." The DD statement identified by "ddname" then loses its identity; that is, the statement 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 job step or procedure step, but no two uses can refer to the same "ddname." The DDNAME parameter is used mainly for cataloged procedures (as shown in Figure 50 in the section "Cataloged Procedures").

SPECIFYING I/O DEVICES

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

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

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

{n|P}
: specifies the number of devices allocated to the data set.

SPECIFYING VOLUMES

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

```
VOLUME=([PRIVATE][,RETAIN]

        [,volume-sequence-number]

        [,volume-count]

        ┌,SER=(volume-serial-number                  ┐
        │     [,volume-serial-number]...)            │
        │                                            │
        │      ⎧dsname                         ⎫     │
        │,REF= ⎨*.ddname                       ⎬     │ )
        │      ⎪*.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.

RETAIN
: indicates that this volume is to remain mounted after the job step is completed. (Unless RETAIN is specified, the volume is dismounted after its last use in the job step.) Volumes are retained so that data may be transmitted to or from the data set, or so that other data sets may reside on the volume. If the data set requires more than one volume, only the last volume is retained; the other volumes are dismounted when the end of the 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 number that specifies the sequence number of a selected volume at which processing is to begin. All volumes whose sequence numbers precede the specified number are omitted from processing. Specification of the volume-sequence-number is useful only when the programmer is reading or writing a multi-volume cataloged data set.

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 which volumes are used for the data set by specifying the volume serial number for each specific volume. (The volume serial number is assigned and placed on the volume when the volume is made ready for use by the installation.) A volume serial number consists of one to six alphameric characters. If it contains fewer 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. A volume serial number need not be specified for an output data set.

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.

When REF is specified and the data set resides on a tape volume, 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 may be omitted.

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

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

<u>SPECIFYING SPACE ON DIRECT ACCESS VOLUMES</u>

The programmer indicates, in the SPACE parameter, the space to be allocated on a volume to a direct access data set.

$$\text{SPACE=(}\left\{\begin{array}{l}\text{TRK}\\ \text{CYL}\\ \text{average-record-length}\end{array}\right\}\text{,}$$

```
        (primary-quantity

        [,secondary-quantity]

        [,directory-quantity])
```

$$\text{[,RLSE]}\left[\begin{array}{l}\text{,MXIG}\\ \text{,ALX}\\ \text{,CONTIG}\end{array}\right]\text{[,ROUND])}$$

specifies space on a direct access volume. Although SPACE has no meaning for tape volumes, if a sequential data set is assigned to a device class that contains both direct access devices and tape devices, SPACE should be specified. 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.
5. Whether space is to begin on a cylinder boundary.

$$\left\{\begin{array}{l}\text{TRK}\\ \text{CYL}\\ \text{average-record-length}\end{array}\right\}$$

: specify the units of measurement in which storage is assigned. The units may be tracks (TRK), cylinders (CYL), or records (average record length expressed in decimal numbers).

(primary-quantity[,secondary-quantity]
[,directory-quantity])
: specify the amount of space allocated for the data set. The "primary quantity" indicates the number of records, tracks, or cylinders allocated when the job step begins. The "secondary quantity" indicates the amount of space to be allocated each time previously allocated space is exhausted. The operating system can allocate additional space specified in the secondary quantity 15 times. The "directory quantity" is used only when

writing a PDS, and it specifies the number of 256-byte records to reserve for the directory of the PDS.

For example, by specifying:

SPACE=(120,(400,100))

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

By specifying the following, 20 cylinders are allocated to the data set:

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

When previously allocated space is exhausted, two additional cylinders are allocated. In addition, space is reserved for five records in the directory of a PDS.

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

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

$\begin{bmatrix}\text{MXIG}\\\text{ALX}\\\text{CONTIG}\end{bmatrix}$
: specify the format of the space allocated to the data set. MXIG requests the largest single block of storage that is greater than or equal to the space requested in the "primary quantity." ALX requests up to five contiguous blocks of storage, each block greater than the "primary quantity." 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: The SPACE parameter in the DD statement must be used if a data set might be written on a direct access device. For the compiler, the programmer should allow 150 characters per source statement in the "primary quantity" for each data set except SYSPRINT. For SYSPRINT, he should allow approximately 220 characters per source statement.

LABEL INFORMATION

If the programmer wishes to catalog a data set so that he can refer to it without repeating information (record type, record length, number of buffers, etc.) that was supplied when the data set was created, he must specify certain information in the LABEL parameter. If the parameter is omitted and the data set is passed, the label information is retrieved from data set labels stored with the data set.

$$\text{LABEL=([data-set-sequence-number]}\left\{\begin{matrix}\text{,NL}\\\text{,}\underline{\text{SL}}\\\text{,BLP}\end{matrix}\right\}\begin{bmatrix}\text{,IN}\\\text{,OUT}\end{bmatrix}\begin{bmatrix}\text{,EXPDT=yyddd}\\\text{,RETPD=xxxx}\end{bmatrix}\text{)}$$

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

$\left\{\begin{matrix}\text{NL}\\ \underline{\text{SL}}\\ \text{BLP}\end{matrix}\right\}$
specify data set label information SL indicates standard labels which contain information such as record format, buffer length, dates, density, and identifiers for the data set. NL indicates no labels. BLP indicates that label processing is to be bypassed.

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

$\begin{bmatrix}\text{IN}\\ \text{OUT}\end{bmatrix}$
are used to control data sets that are to be processed as input or output only. A form of read/write protection is offered by these subparameters.

For input data sets, the IN subparameter allows:

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

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

IN
specifies that the data set is to be processed for input only. IN will be recognized only if the first input/output operation specifying the data set is a READ. If the first operation is not a READ, the IN subparameter has no effect and both READ/WRITE operations are allowed. When the first operation is a READ, any subsequent WRITE issued to the data set will be treated as an error, and the job will be terminated. Additionally, the specification of IN permits the reading of a password-protected data set (if the correct password is supplied), and avoids the need of operator intervention when reading a data set protected by either a high expiration date or the absence of a write-ring.

OUT
specifies that the data set defined by the DD statement is to be recognized only if the first input/output operation specifying the data set is a WRITE. If the first operation is not WRITE, the OUT subparameter has no effect and both READ/WRITE operations are allowed. However, the creation of a member of a partitioned data set is not allowed when the first operation is READ, even though the OUT subparameter was specified. When the first operation is a WRITE, any subsequent READ issued to the data set will be treated as an error, and the job will be terminated. OUT must be specified to create a member of a partitioned data set.

$\begin{bmatrix}\text{EXPDT=yyddd}\\ \text{RETPD=xxxx}\end{bmatrix}$
specify 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 by the DELETE subparameter in the DISP parameter. 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 using previously created data sets.

WRITING A UNIT RECORD DATA SET ON AN INTERMEDIATE DEVICE

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

SYSOUT=A
indicates that the ultimate destination for printed output data sets is the printer.

Note: For SYSOUT data sets, if the DEN subparameter is specified, only DEN=2 can be specified.

DCB PARAMETER

For the compiler or linkage editor, the length of a block can be specified. For load module execution, the FORTRAN programmer may specify record formats, record lengths, and the number of buffers for sequentially organized data sets that reside on magnetic tape or direct access volumes. For direct access organized data sets, only the number of buffers can be specified; any other specifications are ignored. 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}]
                 ┌                                    ┐
                 │          ⎧{F|U}[A|M][T]        ⎫   │
                 │          ⎪ [,BLKSIZE=xxxx]     ⎪   │
[,OPTCD=C]       │,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 retrieve DCB parameter information from previously created data sets. The DCB information specified for the data set referred to by this subparameter is copied by the control program for use in processing the data set defined by the DD statement in which this subparameter appears. Any subparameters in the DCB parameter that follow this subparameter override any copied DCB subparameters.

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

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

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

*.stepname.procstep.ddname
: indicates that the DCB parameter in the DD statement "ddname" is 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 tape volumes.

<u>DENSITY</u>: Density is only specified for data sets residing on magnetic tape volumes.

DEN={0|1|2|3|}

Table 10 correlates density with the numbers 0, 1, 2, and 3.

Table 10. DEN Values for Model 2400

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

<u>Note</u>: If SYSOUT=A is specified, DEN=2 is the only DEN option that may be specified.

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 or EBCDIC to BCD is required

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

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. 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 V, F, U, and B represent

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)

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

B - blocked records

Note: For blocked compiler and linkage editor I/O, RECFM should not be specified.

The character A indicates the use of the FORTRAN carriage control characters; the character M indicates the use of machine code control characters.

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 associated with each, 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: 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, BLOCK LENGTH, AND NUMBER OF BUFFERS FOR SEQUENTIAL DATA SETS

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

BLKSIZE=xxxx

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

For unblocked records used by the compiler or linkage editor, the values for BLKSIZE and LRECL are permanently specified.

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

BLKSIZE=xxxx

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

LRECL=xxxx

Buffer length is specified by

BLKSIZE=xxxx

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

LRECL=xxxx

Block length and buffer length are specified by

BLKSIZE=xxxx

Undefined records cannot be blocked.

Table 11 is a summary of the specifications made by the programmer for record types and blocking in FORTRAN processing.

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

BUFNO=x (x=1 or 2)

FORTRAN Records and Logical Records for Sequential Data Sets

In FORTRAN, records for sequential data sets are defined by specifications in FORMAT statements and by READ/WRITE lists. A record defined by a specification in a FORMAT statement is a FORTRAN record (see the section "Input/Output Statements" in the Basic 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.

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

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 is specified in the BLKSIZE subparameter. The information coded in a FORMAT statement indicates the FORTRAN record length (in bytes).

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

Example: Assume BLKSIZE=44

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

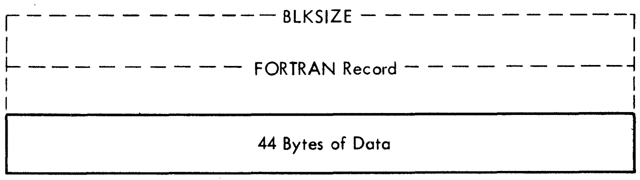


Figure 35. FORTRAN Record (FORMAT Control) Fixed-Length Specification

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

Table 11. Specifications Made by the FORTRAN Programmer for Record Types and Blocking

Step	Blocked or Unblocked	Record Type	RECFM Specification	Record Length	Buffer Length
Compiler or Linkage Editor	Unblocked	Fixed-Length	not specified[1]	not specified[1]	not specified[1]
	Blocked	Fixed-Length	RECFM=FB[2]	not specified[1]	BLKSIZE=xxxx
Load Module Execution	Unblocked	Fixed-Length	RECFM=F[3]	BLKSIZE=xxxx[3]	BLKSIZE=xxxx
		Variable-Length	RECFM=V	LRECL=xxxx	
		Undefined	RECFM=U	BLKSIZE=xxxx	
	Blocked	Fixed-Length	RECFM=FB	LRECL=xxxx	
		Variable-Length	RECFM=VB		
		Undefined	Blocked undefined records are not permitted		

[1]Permanently specified by the compiler and cannot be altered.
[2]For SYSPRINT or other written output, RECFM=FBA under the sequential scheduler, and RECFM=FM under the priority scheduler.
[3]Not specified for direct access data sets.

Example: Assume BLKSIZE=56

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

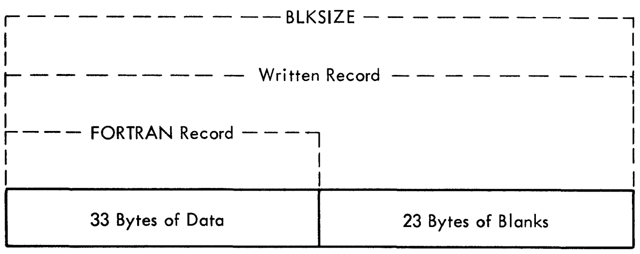


Figure 36. FORTRAN Record (FORMAT Control) With Fixed-Length Specification and FORTRAN Record Length Less Than BLKSIZE

Variable-Length Records: For unblocked variable-length records written under FORMAT control, LRECL is specified as four greater than the maximum FORTRAN record length and BLKSIZE as four greater than LRECL. These extra eight bytes are required for the 4-byte block control word (BCW) and the 4-byte segment control word (SCW), as shown in Figure 32. The BCW (see Figure 37) contains the length of the block; the SCW (see Figure 38) contains the length of the record segment; i.e., the data length plus four bytes for the SCW.

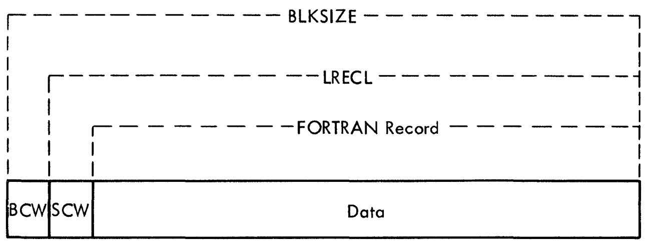


Figure 37. FORTRAN Record (FORMAT Control) Variable-Length Specification

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

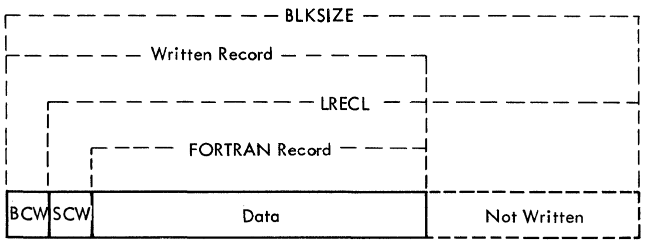


Figure 38. FORTRAN Record (FORMAT Control) With Variable-Length Specification and the FORTRAN Record Length Less Than (LRECL-4)

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

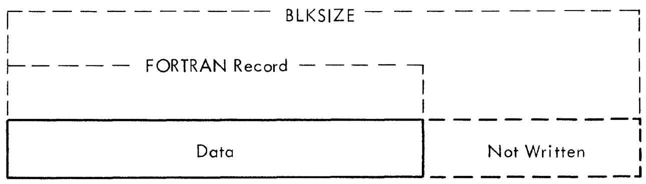


Figure 39. FORTRAN Record (FORMAT Control) With Undefined Specification and the FORTRAN Record Length Less Than BLKSIZE

BLOCKED RECORDS, 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 40).

Example: Assume BLKSIZE=48 and LRECL=24

```
10  FORMAT(I8,F16.4)

20  FORMAT(I12)

        .

        .

        .

    WRITE(13,10)N,B

        .

        .

        .

    WRITE(13,20)K
```

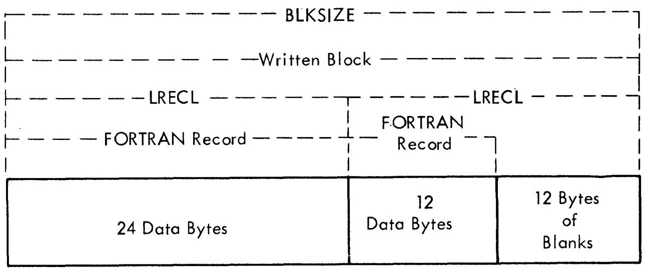


Figure 40. Fixed-Length Blocked Records Written Under FORMAT Control

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

If a WRITE statement 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 37); 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 statement (see Figure 41). If another WRITE statement is not executed before the job step is terminated, then the filled portion of the buffer is written.

<u>Example:</u> Assume BLKSIZE=28 and LRECL=12

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

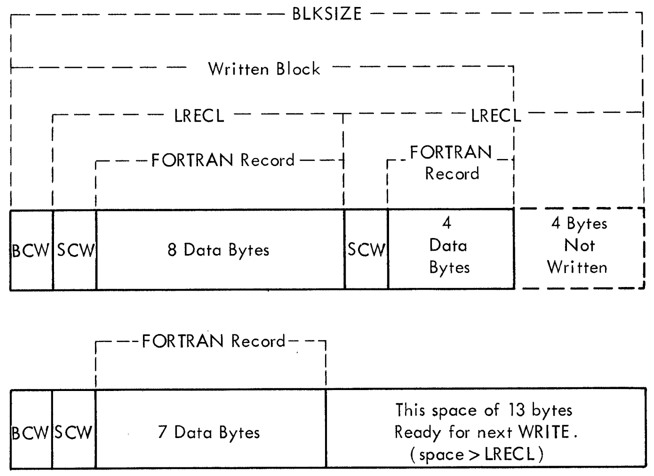


Figure 41. Variable-Length Blocked Records Written Under FORMAT Control

<u>NO FORMAT CONTROL:</u> 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 without FORMAT control have the following properties:

- The length of the logical record is controlled by the type and number of variables in the I/O list of its 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.

The format of a BCW is given in Figure 42.

block-length	reserved
2 bytes	2 bytes

Figure 42. 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 43.

segment-length	code	reserved
2 bytes	1 byte	1 byte

Figure 43. Format of a Segment Control Word

where:

segment-length
: is a binary count of the number of bytes in the SCW (four bytes) plus the number of bytes in the data portion of the segment following the SCW. (The permissible range is from 4 to 32,763 bytes.)

code
: indicates the position of the segment with respect to the other segments (if

any) of the record. Bits 0 through 5 are reserved for system use and are set to 0. Bits 6 and 7 contain the codes:

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

reserved
is a byte of zeros reserved for system use.

Unblocked records: For unblocked records written without FORMAT control, the value of BLKSIZE is equal to LRECL+4. (The four additional bytes are for the BCW.)

If the logical record length is less than or equal to LRECL-4, the logical record comprises one record segment. Hence, for the associated READ or WRITE statement, one record segment, i.e., one block, is transmitted (see Figure 44). Note that the unused portion of the block is not transmitted.

If the logical record length is greater than LRECL-4, the logical record comprises N record segments, where:

N=logical record length/LRECL-4.

Hence, for the associated READ or WRITE statement, N record segments, i.e., N blocks, are transmitted (see Figure 45).

Example 1: Assume BLKSIZE=28 and LRECL=24

WRITE(18)Q,R

where:
Q and R are real *8 variables.

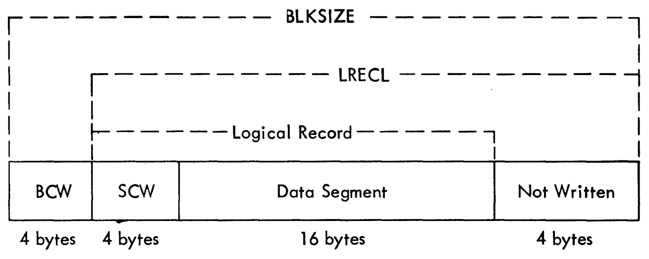


Figure 44. Variable-length Unblocked Records, No FORMAT Control, One Record Segment

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.

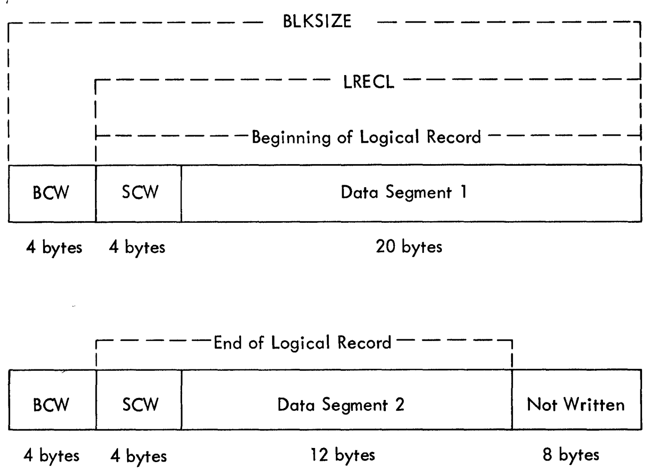


Figure 45. Variable-length Unblocked Records, No FORMAT Control, Two Record Segments

Blocked Records: For blocked records written without FORMAT control, each block, except the last, is composed of at least M record segments, where:

M = 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 the associated READ or WRITE statement, one block, i.e., M record segments, is transmitted.

If the logical record length is greater than LRECL-4, the logical record comprises N record segments, where:

N=logical record length/LRECL-4.

Hence, for the associated READ or WRITE statement, N record segments (i e., as many blocks of M segments each as are needed to make up N segments) are transmitted. The unused portion of the last block is held for the next READ or WRITE (see Figure 46).

Example: Assume BLKSIZE=28 and LRECL=12

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

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

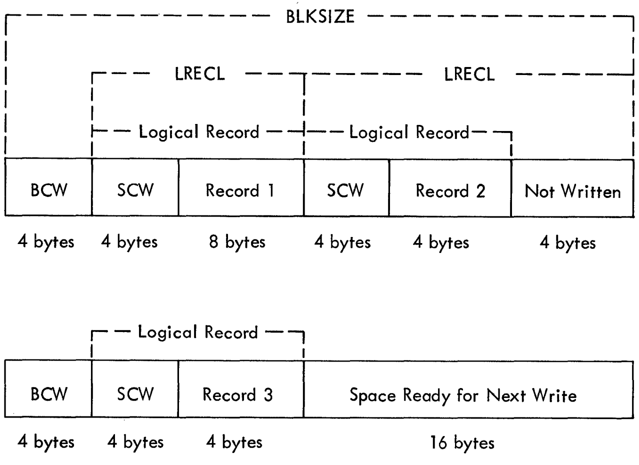


Figure 46. Variable-length, Blocked Records, No FORMAT Control

BACKSPACE Operations

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

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

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

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 following statement sets the record length and buffer length permanently at 152 bytes:

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

The direct access data set defined by this DEFINE FILE statement contains 1000 fixed-length unblocked records. Each record is 152 bytes long, and is written under FORMAT control.

The only DCB parameter that can be supplied for direct access data sets is the number of buffers:

```
BUFNO=x
```

where:
x (1 or 2) 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 without FORMAT control, the records must be specified as fixed-length and unblocked. These records do not contain a block control word or a segment control word.

Spanning Considerations

For records written without FORMAT control, the input/output list may exceed the logical record length (i.e., block size). In this case a new block is started on output, and the next block is processed on input. If it is shorter than the record length, the remaining portion of the record is padded with zeros (see Figure 47).

Note that the DEFINE FILE field _r_ (_r_=152 in the preceding example) specifies the maximum size of each record in a data set. It is only when this size is exceeded by the I/O list that spanning occurs. Although FORTRAN allows the creation and retrieval of such records, this feature is not supported by other processors. Thus, data sets containing such records cannot be processed in other than FORTRAN jobs.

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

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

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

where:

DP1 and DP2 are double precision variables.

R1 and R2 are real variables.

IX is an integer variable that contains the record position.

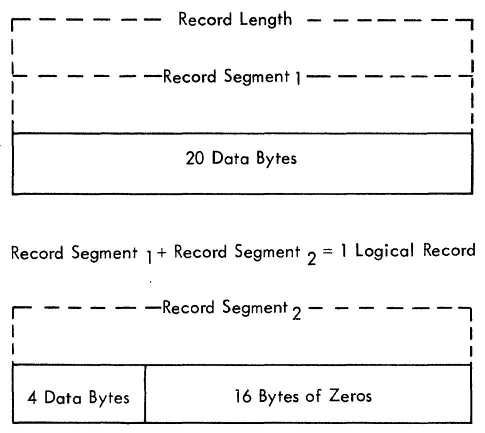


Figure 47. Logical Record (No FORMAT Control) for Direct Access

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

DCB RANGES AND ASSUMPTIONS

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

Data Set	LRECL Value
SYSPRINT	121
SYSIN	80
SYSPUNCH	80
SYSLIN	80

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

The BLKSIZE value must be an integral multiple of the corresponding LRECL value shown above. The maximum BLKSIZE value is limited only by the type of input/output device (see Table 12), except that for SYSLIN the maximum BLKSIZE value is 400 with linkage editor IEWLE440.

For load module execution, specifications depend on record type. For F type records, the BLKSIZE value must be an integral multiple of the LRECL value; for V type records, BLKSIZE must be specified as 4 + n x LRECL (where n is the number of records in the block); for U type records, no blocking is permitted. Note, too, that the BLKSIZE and LRECL range is limited only by the type of device used to directly write the data set (see Table 12). Load module DCB parameter default values are shown in Table 13.

Table 12. BLKSIZE Ranges: Device Considerations

Device Type	BLKSIZE Ranges	
	F and U Record Type	V Record Type
Card Reader	1≤x≤80	9≤x≤80
Card Punch	1≤x≤81	9≤x≤89
Printer:		
120 Spaces	1≤x≤121	9≤x≤129
132 Spaces	1≤x≤133	9≤x≤141
144 Spaces	1≤x≤145	9≤x≤153
Magnetic Tape	18≤x≤32,000	
Direct Access:	Without Track Overflow[1]	With Track Overflow[1]
2301	1≤x≤20,483	1≤x≤32,763
2302	1≤x≤4984	1≤x≤32,763
2303	1≤x≤4892	1≤x≤32,763
2311	1≤x≤3625	1≤x≤32,763
2314	1≤x≤7294	1≤x≤32,763
[1]If RECFM=V, the minimum block size is 9.		

Table 13. Load Module DCB Parameter Default Values

Data Set Reference Number	ddname	Sequential Data Sets[1]		Direct Access Data Sets	
		RECFM	Default LRECL or BLKSIZE	Default RECFM	Default LRECL or BLKSIZE
1	FT01F001	F	80	F	The value specified as the maximum size of a record in the DEFINE FILE statement.
2	FT02F001	F	80	F	
3	FT03F001	UA[2]	133	FA[2]	
4	FT04F001	U	800	F	
.	.	.	.	.	
.	.	.	.	.	
.	.	.	.	.	
99	FT99F001	U	800	F	

[1]If the records have no FORMAT control, the default RECFM is V and the default LRECL is 4 less than BLKSIZE, where the default BLKSIZE is as specified in this table.
[2]The first character in each record is assumed to be a carriage control character.

This section contains figures showing the job control statements used in the FORTRAN IV (E) cataloged procedures and a brief description of each procedure. This section also describes how to override statements and parameters in any cataloged procedure. (The use of cataloged procedures is discussed in "Job Processing.") The SPACE parameter shown in these cataloged procedures is written for use with IBM 2311 disk storage drive.

Compile

The cataloged procedure for compilation (FORTEC) is shown in Figure 48.

The EXEC statement that invokes the FORTRAN E compiler is named FORT; the EXEC statement indicates that the operating system is to execute the program IEJFAAA0 [the name for the FORTRAN (E) compiler]. Compiler options are not explicitly supplied with the procedure: default options are assumed. The programmer can override these default options by using an EXEC statement which includes the options he desires.

Compile and Link Edit

The cataloged procedure (FORTECL) to compile a source module and link edit the resulting object module into the FORTRAN library (FORTLIB) is shown in Figure 49.

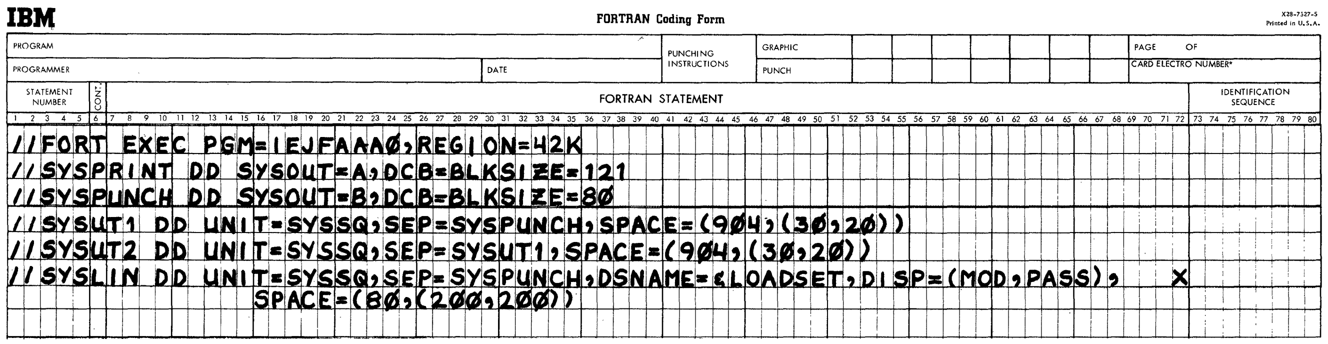
The EXEC statement named FORT instructs the control program to execute the FORTRAN (E) compiler. Again, no compiler options are specified; default options are assumed.

The EXEC statement named LKED instructs the control program to execute the program IEWL (the alias for the highest level linkage editor in the installation's operating system). This statement also specifies the XREF, LIST, LET, and NCAL linkage editor options. The NCAL option instructs the linkage editor not to resolve any external references in the FORTRAN library. This means that the resulting load module must be processed by the linkage editor again before the module can be executed unless NCAL is overridden.

Link Edit and Execute

The cataloged procedure to link edit FORTRAN object modules and execute the resulting load modules (FORTELG) is shown in Figure 50.

The EXEC statement that executes the linkage editor is named LKED and specifies that the operating system is to execute the program IEWL, the alias for the highest level linkage editor. This statement also specifies the XREF, LIST, and LET options for the linkage editor. The programmer can override these options by using an EXEC statement in the input stream.


IBM FORTRAN Coding Form

```
//FORT EXEC PGM=IEJFAAAØ,REGION=42K
//SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=121
//SYSPUNCH DD SYSOUT=B,DCB=BLKSIZE=8Ø
//SYSUT1 DD UNIT=SYSSQ,SEP=SYSPUNCH,SPACE=(9Ø4,(3Ø,2Ø))
//SYSUT2 DD UNIT=SYSSQ,SEP=SYSUT1,SPACE=(9Ø4,(3Ø,2Ø))
//SYSLIN DD UNIT=SYSSQ,SEP=SYSPUNCH,DSNAME=&LOADSET,DISP=(MOD,PASS),   X
               SPACE=(8Ø,(2ØØ,2ØØ))
```

Figure 48. Compile Cataloged Procedure (FORTEC)

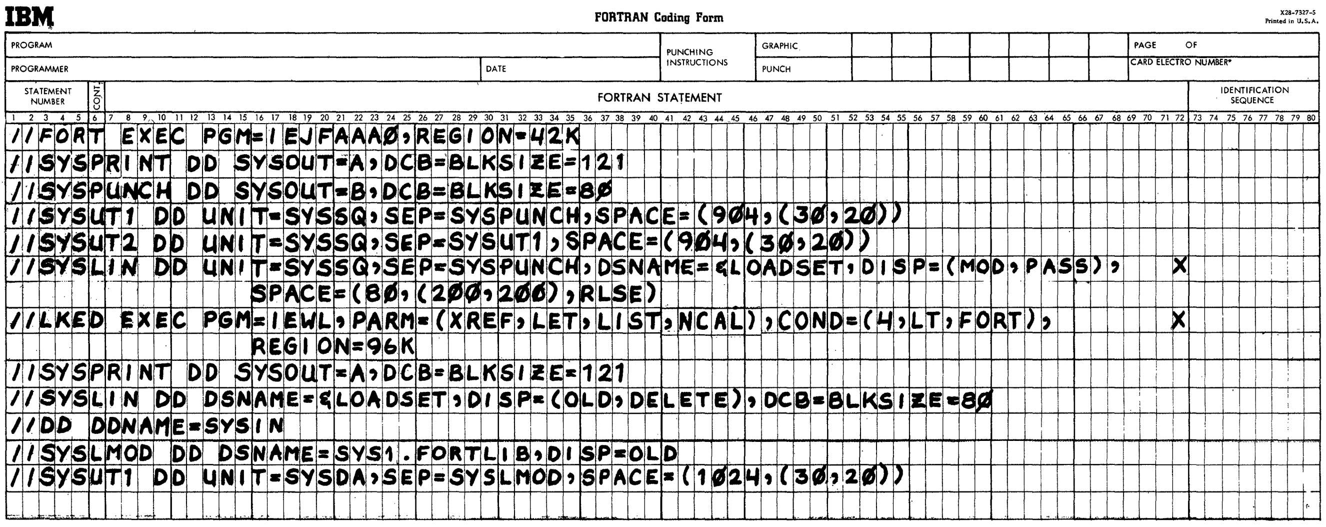
The EXEC statement named GO executes the load module produced by the linkage editor procedure step. The PGM parameter specifies that the operating system is to execute the data set defined by the DD statement SYSLMOD in the procedure step LKED.

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 factor, the number and record sizes of data sets used, the number and sizes of library subprograms invoked, and the sizes of other execution time routines required by the program. If the default REGION is not large enough for program execution, REGION.GO must be used to specify a REGION parameter on the program's EXEC statement.

A listing of the execution time routines required for various input/output, interruption, and error procedures is contained in the publication IBM System/360 FORTRAN IV Library Subprograms. It also lists the sizes of both the execution time routines and the mathematical subprograms.

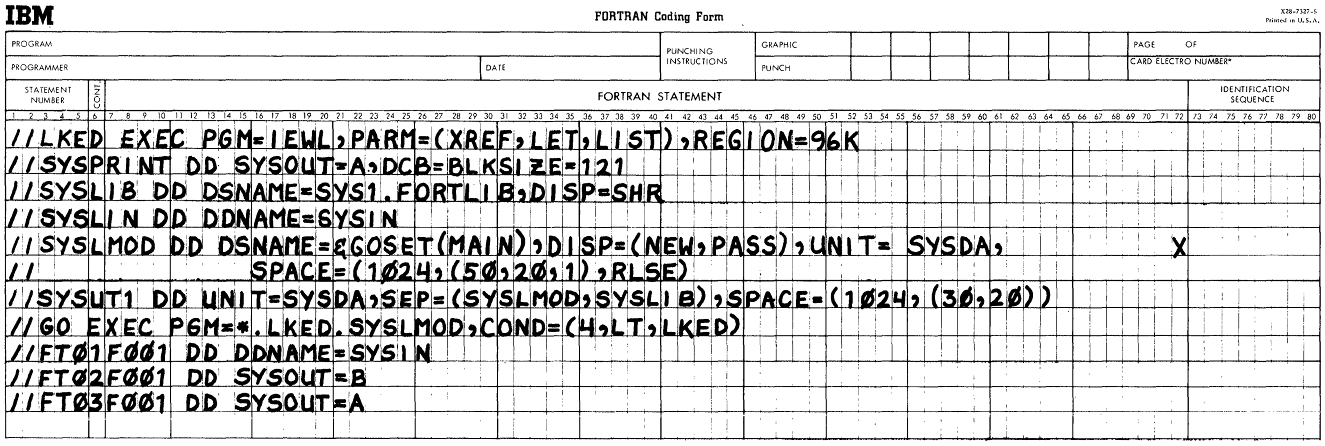
The following is an example of using a REGION.GO specification to indicate the main storage requirements for the execute step of a FORTRAN program.

```
//EXAMPLE1 JOB ACCOUNT1,'JOHN SMITH',                X
                MSGLEVEL=1
// EXEC FORTECLG,PARM.FORT=DECK,                     X
                REGION.GO=60K
//FORT.SYSIN DD *
                .
                .
                .
      FORTRAN SOURCE SYMBOLIC DECKS
                .
                .
                .
/*
//LKED.SYSIN DD *
                .
                .
                .
      PREVIOUSLY COMPILED OR ASSEMBLED
        OBJECT DECKS
                .
                .
                .
/*
//GO.SYSIN DD *
                .
                .
                .
      INPUT DATA
                .
                .
                .
/*
```



```
//FORT EXEC PGM=IEJFAAAØ,REGION=42K
//SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=121
//SYSPUNCH DD SYSOUT=B,DCB=BLKSIZE=8Ø
//SYSUT1 DD UNIT=SYSSQ,SEP=SYSPUNCH,SPACE=(9Ø4,(3Ø,2Ø))
//SYSUT2 DD UNIT=SYSSQ,SEP=SYSUT1,SPACE=(9Ø4,(3Ø,2Ø))
//SYSLIN DD UNIT=SYSSQ,SEP=SYSPUNCH,DSNAME=&LOADSET,DISP=(MOD,PASS),   X
               SPACE=(8Ø,(2ØØ,2ØØ),RLSE)
//LKED EXEC PGM=IEWL,PARM=(XREF,LET,LIST,NCAL),COND=(4,LT,FORT),       X
               REGION=96K
//SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=121
//SYSLIN DD DSNAME=&LOADSET,DISP=(OLD,DELETE),DCB=BLKSIZE=8Ø
//DD DDNAME=SYSIN
//SYSLMOD DD DSNAME=SYS1.FORTLIB,DISP=OLD
//SYSUT1 DD UNIT=SYSDA,SEP=SYSLMOD,SPACE=(1Ø24,(3Ø,2Ø))
```

Figure 49. Compile and Link Edit Cataloged Procedure (FORTECL)



```
//LKED EXEC PGM=IEWL,PARM=(XREF,LET,LIST),REGION=96K
//SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=121
//SYSLIB DD DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSLIN DD DDNAME=SYSIN
//SYSLMOD DD DSNAME=&GOSET(MAIN),DISP=(NEW,PASS),UNIT= SYSDA,              X
//             SPACE=(1Ø24,(5Ø,2Ø,1),RLSE)
//SYSUT1 DD UNIT=SYSDA,SEP=(SYSLMOD,SYSLIB),SPACE=(1Ø24,(3Ø,2Ø))
//GO EXEC PGM=*.LKED.SYSLMOD,COND=(4,LT,LKED)
//FTØ1FØØ1 DD DDNAME=SYSIN
//FTØ2FØØ1 DD SYSOUT=B
//FTØ3FØØ1 DD SYSOUT=A
```

Figure 50. Link Edit and Fxecute Cataloged Procedure (FORTELG)

Compile, Link Edit, and Execute

The cataloged procedure (FORTECLG) to compile, link edit, and execute FORTRAN source modules is shown in Figure 51.

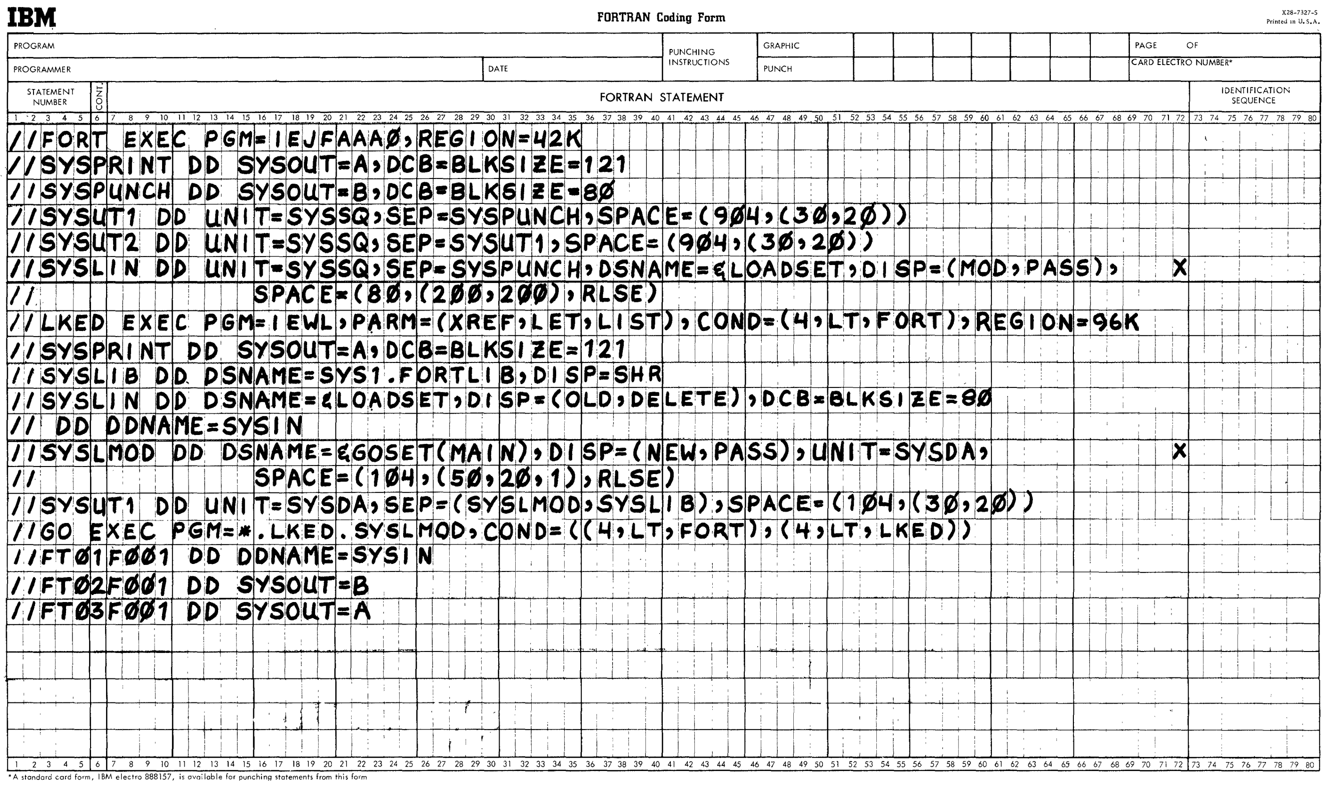
The cataloged procedure FORTECLG consists of the statements in the FORTEC and FORTELG procedures, with one exception: the DD statement SYSLIN (in the compiler procedure step FORT) defines the output of the compiler, and the statement SYSLIN (in the linkage editor procedure step LKED) identifies this data set as the primary input to the linkage editor. The programmer does not have to define the linkage editor input as he did with the procedure FORTELG, but he must define the data set SYSIN for the compiler so that the source module can be read. He may also define a data set which becomes part of the primary input by using a DD statement LKED.SYSIN which is concatenated with object module. This data set is concatenated with the data set containing the output of the compiler.

USER AND MODIFIED CATALOGED PROCEDURES

The programmer can write his own cataloged procedures and tailor them to the facilities in his installation. He can also permanently modify the IBM-supplied cataloged procedures. For information about adding and permanently modifying cataloged procedures, see the section "Cataloged Procedures" in the publication System Programmer's Guide.

If during system generation, the G or H level library option was specified in the FORTLIB macro-instruction, the FORTRAN (E) cataloged procedures must be permanently modified to correspond to the FORTRAN (G) or (H) cataloged procedures. The FORTLIB macro-instruction is described in the publication IFM System/360 Operating System, System Generation. Further modifications to cataloged procedure may be necessary as described in the Job Control Language publication.

If the E level library option was specified, but the value of the OBJERR parameter of the FORTLIB macro-instruction was omitted or specified as something other than 03, the following DD card must be added to the FORTRAN E cataloged procedure either to modify them permanently or to override them at execution time.


IBM FORTRAN Coding Form X28-7327-5 Printed in U.S.A.

PROGRAM | PROGRAMMER | DATE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE OF | CARD ELECTRO NUMBER* | STATEMENT NUMBER | CONT. | FORTRAN STATEMENT | IDENTIFICATION SEQUENCE

```
//FORT EXEC PGM=IEJFAAAØ,REGION=42K
//SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=121
//SYSPUNCH DD SYSOUT=B,DCB=BLKSIZE=8Ø
//SYSUT1 DD UNIT=SYSSQ,SEP=SYSPUNCH,SPACE=(9Ø4,(3Ø,2Ø))
//SYSUT2 DD UNIT=SYSSQ,SEP=SYSUT1,SPACE=(9Ø4,(3Ø,2Ø))
//SYSLIN DD UNIT=SYSSQ,SEP=SYSPUNCH,DSNAME=&LOADSET,DISP=(MOD,PASS),    X
//             SPACE=(8Ø,(2ØØ,2ØØ),RLSE)
//LKED EXEC PGM=IEWL,PARM=(XREF,LET,LIST),COND=(4,LT,FORT),REGION=96K
//SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=121
//SYSLIB DD DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSLIN DD DSNAME=&LOADSET,DISP=(OLD,DELETE),DCB=BLKSIZE=8Ø
// DD DDNAME=SYSIN
//SYSLMOD DD DSNAME=&GOSET(MAIN),DISP=(NEW,PASS),UNIT=SYSDA,             X
//             SPACE=(1Ø4,(5Ø,2Ø,1),RLSE)
//SYSUT1 DD UNIT=SYSDA,SEP=(SYSLMOD,SYSLIB),SPACE=(1Ø4,(3Ø,2Ø))
//GO EXEC PGM=*.LKED.SYSLMOD,COND=((4,LT,FORT),(4,LT,LKED))
//FTØ1FØØ1 DD DDNAME=SYSIN
//FTØ2FØØ1 DD SYSOUT=B
//FTØ3FØØ1 DD SYSOUT=A
```

*A standard card form, IBM electro 888157, is available for punching statements from this form

Figure 51. Compile, Link Edit, and Execute Cataloged Procedure (FORTECLG)

```
//GO.FTxxF001 DD SYSOUT=A
```

where:

<u>xx</u> (2 digits) is the unit specified. (See Figures 50 and 51.)

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 current execution of the procedure. Overriding is only temporary; that is, the parameters added or modified are in effect only for that execution.

If the same cataloged procedure is executed by two different job steps in the same job, the overriding parameters or statements supplied for the first execution are not carried over for the second execution of the procedure. For example, consider these job control statements:

```
//JOB1 JOB MSGLEVEL=1
//STEP1 EXEC FORTEC,PARM.FORT='SIZE=22K'
          .
          .
          .
//STEP2 EXEC FORTEC
          .
          .
          .
```

When the procedure is executed in the first step STEP1, the compiler is allocated 22K bytes. However, when the procedure FORTEC is executed in the second step, the SIZE option reverts to the default option (15K) because the overriding parameter only affects the current execution of the cataloged procedure.

The following text discusses the techniques used to override cataloged procedures.

Overriding Parameters in the EXEC Statement

Two forms of keyword parameters ("keyword" and "keyword.procstep") in the EXEC statement 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, specify accounting information, or he can state different conditions for bypassing a job step for an execution of a cataloged procedure.

Note: When the PARM parameter is overridden, all options stated in the EXEC statement in the procedure step are deleted, and the overriding PARM parameter is substituted.

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

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

The PARM options apply to the procedure step FORT.

Example 2: Assume the cataloged procedure FORTECL is used to compile and link edit a program. The programmer wants to specify the ADJUST option for the compiler because his source module contains embedded blanks and FORTRAN keywords used as variable names. Furthermore, he wants to remove the NCAL linkage editor option because he does not want to make another pass through the linkage editor prior to using the load module in execution. The following EXEC statement can be used to add the ADJUST option to the compiler procedure step (FORT), and remove the NCAL option from the linkage editor procedure step (LKED).

```
//CL EXEC FORTECL,PARM.FORT=ADJUST,             X
//     PARM.LKED=(XREF,LIST,LET)
```

Example 3: Assume the cataloged procedure FORTELG is used to link edit and execute a module. Furthermore, the MAP linkage editor option overrides XREF, LET, and LIST in the linkage editor step and the COND parameter is changed for bypassing the execution of the load module. The following EXEC statement adds and overrides parameters in the procedure.

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

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

Example 4: Assume a source module is compiled, link edited, and executed using the cataloged procedure FORTECLG. Furthermore, the compiler option SIZE and the linkage editor option MAP 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 FORTECLG,                          X
//    PARM.FORT='SIZE=22000',                   X
//    PARM.LKED=MAP,                            X
//    ACCT.GO=506
```

Overriding and Adding DD Statements

A DD statement with the name "procstep.ddname" is used to override parameters in DD statements in cataloged procedures or to add DD statements to cataloged procedures. The "procstep" 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 DD statement in the input stream 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. Only parameters that must be changed are specified in the overriding DD statement.

If the programmer wants to delete a keyword parameter in a DD statement, he supplies an overriding DD statement that contains that keyword, followed by an equal sign, followed by a comma.

```
keyword=,
```

For example, if the SYSOUT parameter is to be deleted from the SYSPRINT data set and the data set is to be written on the device PRINT in the cataloged procedure FORTEC, the following DD statement is used:

```
//FORT.SYSPRINT DD SYSOUT=,UNIT=PRINT
```

If more than one DD statement is modified, the overriding DD statements must be in the same order that the DD statements appear in the cataloged procedure. Any DD statements that are added to the procedure must follow overriding DD statements.

When the procedures FORTEC, FORTECL, and FORTECLG are used, a DD statement FORT.SYSIN must be added to define the SYSIN data set to the compile step in the procedures (see Figures 15, 18, and 24). When the procedure FORTFLG is used, a DD statement LKED.SYSIN must be added to define the SYSLIN data set (see Figure 21).

When the procedures FORTELG, FORTECL, and FORTECLG 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.

If the programmer is using the procedure FORTECL and he does not supply an overriding DD statement assigning the resulting load module to a private PDS, he must supply an overriding DD statement

```
//LKED.SYSLMOD DD DSNAME SYS1.FORTLIB(name)
```

to name the load module before he places it in the FORTRAN library (SYS1.FORTLIB). This procedure can be a powerful tool for adding modules to the FORTRAN library and replacing load modules in the FORTRAN library.

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 DD statements.

<u>Example 1:</u> The cataloged procedure FORTECLG is used to compile, link edit, and execute a FORTRAN program. Since the operating system for this installation contains the highest level linkage editor, blocking can be specified for the SYSLIN data set. In addition, the SYSPRINT data set for the compiler is blocked. The PRFRM and SIZE compiler options are specified in the PARM parameter, along with the BLKSIZE LRECL subparameter in the DCB parameter for these data sets.

During load module execution, the programmer wants the data set identified by ddname FT03F001 to be written on the device class TAPE, instead of treating this data set as a SYSOUT data set and writing it on device class A. To do this the SYSOUT keyword parameter must be deleted from the SYSPRINT DD statement in the procedure step FORT, and a UNIT parameter must be supplied. 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 FORTECLG,                            X
//    PARM.FORT='PRFRM,SIZE=22K'
//FORT.SYSPRINT DD DCB=BLKSIZE=968
//FORT.SYSLIN DD DCB=BLKSIZE=800
//FORT.SYSIN DD *
┌────────────────────────────────────────┐
│          FORTRAN Source Module         │
└────────────────────────────────────────┘
/*
//LKED.SYSLIN DD DCB=BLKSIZE=800
//GO.FT03F001 DD SYSOUT=,,UNIT=TAPE
//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
```

<u>Example 2:</u> Assume that DCB information is added to the DD statement identified by ddname FT03F001, and that a sequential data set that contains blocked records and

resides on a direct access volume is created and cataloged, using data set reference number 2. The following statements override statements FT02F001 and FT03F001 in the procedure and indicate the location of the object module.

```
//JOB2 JOB
//STEP1 EXEC FORTELG
//LKED.SYSIN DD *
┌─────────────────────────────────────────┐
│          FORTRAN Object Module          │
└─────────────────────────────────────────┘
/*
//GO.FT02F001 DD DSNAME=FIRING,               X
//    UNIT=SYSDA,DISP=(NEW,CATLG),            X
//    SPACE=(100,(2000,200),,,ROUND),         X
//    VOLUME=(PRIVATE,SER=207H),              X
//    DCB=(RECFM=VB,BLKSIZE=2416,             X
//    LRECL=804)
//GO.FT03F001 DD DCB=(RECFM=F,                X
//    BLKSIZE=50)
```

Example 3: Assume the cataloged procedure FORTECL is used to compile and link edit a module, DER, which is added to the FORTRAN library. The following job control statements can be used to add the module to the FORTRAN library.

```
//ADDMDL JOB 427,'R.WHITE'
//CL EXEC FORTECL,PARM.FORT='NAME=DER'
//FORT.SYSIN DD *
┌─────────────────────────────────────────┐
│          FORTRAN Source Module          │
└─────────────────────────────────────────┘
/*
//LKED.SYSLMOD DD DSNAME=SYS1.FORTLIB(DER)
```

After the procedure has been executed, DER can be used to resolve external references made in FORTRAN source modules to the name DER.

Example 4: Assume the cataloged procedure FORTECL is used to replace the library function SQRT in the FORTRAN library. The following job control statements can be used to replace the SQRT function in the FORTRAN library.

```
//REPLAC JOB ,'JIM JONES'
// EXEC FORT FORTECL,PARM='NAME=SQRT,MAP'
//FORT.SYSIN DD *
┌─────────────────────────────────────────┐
│          FORTRAN Source Module          │
└─────────────────────────────────────────┘
/*
//LKED.SYSLMOD DD DSNAME=SYS1.FORTLIB(SQRT)
```

After the execution of the cataloged procedure, the new module SQRT is used to resolve any external references made to the name SQRT. The IBM-supplied library subprogram is no longer used.

Example 5: Assume the cataloged procedure FORTEC is used to compile a source module STARS. The resulting object module STARS is to be written in the PDS SCIENCE. The SYSPRINT data set is written on the PDS PRINT and assigned the member name STARS. The following job control statements can be used to write this output in the partitioned data sets.

```
//JOB2 JOB ,JIM
//STEP EXEC FORTEC
//FORT.SYSPRINT DD DSNAME=PRINT(STARS)      X
//    DISP=OLD
//FORT.SYSLIN DD DSNAME=SCIENC(STARS)       X
//    DISP=OLD
```

Example 6: Assume the link edit and execute cataloged procedure (FORTELG) 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. The parameters not overridden in the SYSLMOD DD statement are copied and used to write the SYSLMOD data set.

```
//JOB3 JOB
//STEP1 EXEC FORTELG
//LKED.SYSLMOD DD DSNAME=MATH(DERIV),       X
//    DISP=(MOD,PASS)
//LKED.SYSIN DD *
┌─────────────────────────────────────────┐
│          FORTRAN Object Module          │
└─────────────────────────────────────────┘
/*
```

Example 7: Assume the compile, link edit, and execute cataloged procedure (FORTECLG) 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 1 in the source module.

```
//JOBCLG JOB 00,'J.DAVID',MSGLEVEL=1
//EXECCLGX EXEC FORTECLG
//FORT.SYSIN DD *
┌──────────────────────────────────────────┐
│       FORTRAN Source Module MAIN         │
├──────────────────────────────────────────┤
│       FORTRAN Source Module SUB1         │
├──────────────────────────────────────────┤
│                   .                      │
│                   .                      │
│                   .                      │
├──────────────────────────────────────────┤
│       FORTRAN Source Module SUBN         │
└──────────────────────────────────────────┘
/*
//LKED.ADDLIB DD DSNAME=MYLIB,DISP=OLD
//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.

MINIMUM SYSTEM REQUIREMENTS FOR THE FORTRAN COMPILER

IBM System/360 Operating System operates in a device-independent environment. In particular, the FORTRAN compiler may operate with any combination of devices (shown in Table 3); however, there are certain requirements.

The FORTRAN (E) compiler requires at least a System/360 Model 30 with 32K bytes of storage, and the standard instruction set with the floating-point option. At least 15,360 bytes should be allocated in the SIZE compiler option. If less than 15,360 bytes is specified, the compiler assumes the design point value 15360.

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.

If the data sets identified by the DD statements SYSUT1, SYSUT2, and SYSLIN are to reside on direct access volumes, another direct access device should be made available to the compiler for more efficient compilation.

When a DD statement specifies that a data set resides on a tape volume, there must be one tape device available in the installation for that data set.

SOURCE PROGRAM CONSIDERATIONS

Facilities are available in the FORTRAN language that enable a programmer to optimize compilation and execution speed and to reduce the size of the object module.

Initialization

The programmer should initialize all variables that are not initialized by arithmetic statements in his program. Operating System/360 may place a load module anywhere in available main storage; the value of a variable cannot be guaranteed until the programmer has given that variable a value by an assignment statement. For example, in the subprogram

```
SUBROUTINE ALPHA(X,Y,Z)
A=B+2.0
   .
   .
   .
```

the result A may contain any value, because B was not initialized. If the programmer expects B to be zero, he should initialize B as shown in the following statements:

```
SUBROUTINE ALPHA(X,Y,Z)
B=0.0
A=B+2.0
   .
   .
   .
```

Coding the Source Program

The ADJUST compiler option permits the programmer to insert embedded blanks, eliminate meaningful blanks, and use FORTRAN keywords as variable names in his program. In order to decrease compilation time, the NOADJUST option may be specified. However, the program to be processed by the NOADJUST option must be such that there are no embedded blanks, meaningful blanks are placed where required, and there are no uses of FORTRAN keywords or variable or array names in the source program.

Arithmetic Statements

The use of multiplication instead of the exponential operation is recommended when the exponent is a small integer. For example, the statement

```
VOL=(4.*R*R*R)/3.
```

is more efficient than the statement

```
VOL=(4.*R**3)/3.
```

because the exponential operation requires a library subprogram. When multiplication is used, storage is conserved and both compiler and linkage editor processing time are decreased.

To calculate the square root, the square root library subprogram should be used instead of the exponential function. For example, the statement

```
HYPOT=SQRT(A*A+B*B)
```

is more accurate than the statement

```
HYPOT=(A*A+B*B)**0.5
```

because the SQRT function is more accurate than the exponential function.

The mixed mode arithmetic expression is provided to reduce errors because of unintentional use of different modes in arithmetic statements. However, when mixed mode arithmetic statements are used, extra instructions are generated. For example, in the statement

```
A=A+1
```

an in-line subprogram is generated to perform the operation indicated. Both main storage and execution time would be saved by using the statement

```
A=A+1.0
```

IF Statement

An arithmetic IF statement lists three statement numbers. One of the listed numbers should immediately follow the IF statement to eliminate unnecessary branching in the load module. For example, the coding represented by the statements

```
    IF (A-B)20,30,30
30  A=0.0
       .
       .
       .
20  B=0.0
       .
       .
       .
```

is more efficient than coding represented by the statements

```
    IF(A-B) 20,30,30
10  X=2.+Y
       .
       .
       .
30  A=0.0
       .
       .
       .
20  B=0.0
```

DO Loop Considerations

Values for expressions that remain constant within a DO loop should be calculated before entry into the loop, instead of calculating the expression each time through the loop. For example, in the statements

```
    DO     10 I=1,100
    X(I)=2.0*(G+ALPHA)+Y(I)
10  CONTINUE
```

the expression 2.0*(G+ALPHA) must be calculated each time the DO loop is executed. For greater efficiency, the following statements should be substituted

```
    BETA=2.0*(G+ALPHA)
    DO 10 I=1,100
    X(I)=BETA+Y(I)
10  CONTINUE
```

The execution time is decreased, because the expression 2.0*(G+ALPHA) is calculated only once.

Any subscripts that remain constant within the range of a DO loop should not be used in the DO loop. For example, in the statements

```
    DO  10 I=1,50
    X(I)=Y(I)+Z(J)
       .
       .
       .
10  CONTINUE
```

a subscript calculation for Z(J) is performed each time the DO loop is executed, even though Z(J) remains constant for each execution of the loop.

By substituting the statements

```
    B=Z(J)
    DO 10 I=1,50
    X(I)=Y(I)+B
         .
         .
         .
10  CONTINUE
```

only one subscript calculation is made for Z(J) and execution time is decreased.

Intricate subscript calculation within the range of a DO should be avoided. For example, in the statements

```
    DO 10 I=1,10
5   X(3*I+4)=Y(3*I+4)+B
         .
         .
         .
10  CONTINUE
```

two intricate subscript calculations are made each time statement 5 is executed. The DO loop should be rewritten as shown in the statements

```
    DO 10 I=7,34,3
5   X(I)=Y(I)+B
         .
         .
         .
10  CONTINUE
```

to reduce the subscript calculation to simpler terms and allow faster execution of the DO loop.

READ/WRITE Statements

To read or write an array, an implied DO in a READ/WRITE statement should be used instead of a DO loop. For example, five FORTRAN records, each containing two values, are written by the statements

```
10  FORMAT (F20.5,I10)
    DO 15 I=1,5
15  WRITE(5,10)A(I),J(I)
```

In the statements

```
10  FORMAT (5(F20.5,I10))
    WRITE(5,10)(A(I),J(I),I=1,5)
```

only one FORTRAN record containing ten values is written. The use of an implied DO saves load module execution time and space on the volume

Extra subscript calculation within the range of an implied DO should be avoided. This is the same consideration shown in regard to the DO loop. For example, if the statements

```
2   FORMAT('0',10F12.6)
         .
         .
         .
    READ(1,2)(A(I),I=4,31,3)
```

are substituted for the statements

```
2   FORMAT('0',10F12.6)
    READ(1,2)A(3*I+1),I=1,10)
```

the intricacy of the subscript calculation is reduced and the load module execution time is reduced.

Program Structure

Better efficiency in load module execution is achieved when storage for a main program or each subprogram (excluding COMMON) is less than 12K bytes. A program that exceeds 12K bytes may be segmented into a group of subprograms and one main program.

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 variable in COMMON only once, and fewer instructions are needed to cross-reference the variables between programs.

Statement Numbers and Names

For its internal use, the compiler places statement numbers and names used for variables, arrays, and subprograms in two tables. Each table is divided into several strings and is searched many times during compilation. If the number of entries in each string is approximately equal, the average time required to find a name or a statement number is reduced.

STATEMENT NUMBERS: Statement numbers are assigned to five strings in the statement number table; assignment is made according to the last digit in the statement number. Statement numbers ending in 0 or 1 are placed in the first string; those ending in 2 or 3 are placed in the second; those ending in 4 or 5 are placed in the third; etc. Statement numbers should be evenly distributed in the strings to decrease compilation time.

For example, using 100 statement numbers that end only in 0 or 5 is inefficient, because two long strings of 50 entries each are created in the statement number table. If these 100 statement numbers were distributed equally in strings, that is, 10 statement numbers ending in 0, 10 ending in 1, etc., five strings of 20 entries each would be created. The time used to compile the source program is decreased because excessive time is not spent searching long strings.

NAMES: Names used in the program are assigned to six strings; assignment is made according to the length of the name. Names that are one character long are placed in the first string; names two characters long are placed in the second string, etc. For faster compilation, the names should be distributed equally among the six strings. For example, if there are 26 names of one character each in a program, one long string is created. For greater efficiency, the names should be distributed equally to make six strings, each containing four or five names.

Use of Embedded Blanks in FORTRAN Programs

To improve the readability of a source program, the programmer may use any number of blanks when writing FORTRAN statements. Except for literal data, in which blanks are retained as coded in the source statement, blanks are normally ignored by the compiler. Thus, the statement DO 25 J = 10 is the equivalent of DO25J=10. Both statements are syntactically correct assignment statements and are executed as such. (A value of 10 is assigned to the variable DO25J.) Neither statement will cause an error message.

Use of DUMP and PDUMP

Three facts are pertinent when the subroutines DUMP and PDUMP are used:

1. Under the operating system, a program can be loaded into different areas of main storage for different executions.

2. The compiler assigns locations to variables and arrays in COMMON in the same order that the programmer specified in a COMMON statement.

3. The compiler assigns locations in the object module to variables and arrays that are not in COMMON by name length and the order in which they are encountered.

The following text shows several examples of how to write statements that use DUMP and PDUMP.

If a series of variables and arrays that reside in COMMON are to be dumped, only the first and last variables to be dumped should be listed as arguments for the subroutine For example, if COMMON is defined as:

```
COMMON A,B,C(20),I(10),D
```

the following statement can be used to dump the variable B and the arrays C and I in hexadecimal format and terminate execution after the dump.

```
CALL DUMP (B,I(10),0)
```

If the variables and arrays are not in COMMON, a set of arguments should be listed for each name that is to be dumped. For example, if COMMON is defined as:

```
COMMON A,B,C(20),I(10),D
```

and the array X is defined as:

```
DIMENSION X(25)
```

and a variable Y is defined in the module, the following statement should be used to dump B, C, I, Y, and X in real format without terminating execution:

```
CALL PDUMP(B,I(10),5,X(1),X(25),5,Y,Y,5)
```

If the statement

```
CALL PDUMP(B,I(10),5,X(1),Y,5)
```

is used, the COMMON area is dumped correctly, but all main storage between X(1) and Y is dumped.

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

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

and the call to SUBI within the source module is:

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

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

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

Object Time Input/Output Efficiency

FORTRAN processing time can be appreciably reduced by the use of programming techniques that result in greater data transfer efficiency. Such techniques are particularly important in executing programs that require substantial input/output operations. Discussed below are four programming areas in which the correct choice of programming method can increase FORTRAN processing speed.

READ/WRITE TYPE: The unformatted form of the READ and WRITE statement provides the fastest data transfer rate. Therefore, for most efficient processing, the unformatted form should be used to transfer information to or from an intermediate data set -- a data set that is written out during a program, not examined by the programmer, and then read back for additional processing later in the program or in another program. Thus, for an intermediate data set, statement 11 in the following example is preferable to statement 9.

```
      COMMON A(10), B(10)
      DIMENSION D(20)
      EQUIVALENCE (A(1), D(1))

  9   WRITE(10, 10)A, B
 10   FORMAT (10E13.3/)
 11   WRITE (9) D
```

IMPLIED DO: Array notation is far more efficient than the indexing capability of an implied DO in an I/O list. Thus, for efficiency, the statement WRITE (9) A (where A is an array name) is preferable to WRITE (9) (A(I), I=1, 13).

EQUIVALENCE STATEMENT: In FORTRAN, on input, data is taken from a record and placed into storage locations that are not necessarily contiguous. On output, data is normally gathered from diverse storage locations. Input/output operations, however, can be made more efficient by storing and retrieving data from contiguous locations.

To construct an efficient READ or WRITE statement for an I/O list consisting of many variables, use a COMMON or named COMMON statement to force all the variables in the list to be allocated contiguous storage space. Next, use an EQUIVALENCE statement to define a single dimensioned variable that is the same length as the list of variables. Finally, use a WRITE on the single dimensioned variable using array notation. The following example illustrates this technique:

```
COMMON/LISTA/A(10),B(8), C, D, I, K, L(10)
REAL*8 B
COMPLEX*16 LIST(10)
EQUIVALENCE (A(1), LIST(1))

WRITE(9)LIST
```

BACKSPACE STATEMENT: Use of the BACKSPACE statement is not recommended if efficient processing is desired.

Data Definition Considerations

The DCB parameter of the DD statement allows for the redefinition of many data set characteristics at execution time. Those specifications that most concern the FORTRAN programmer are discussed in the following text. For a full description of the DCB parameter, see the publication IBM System/360 Operating System Supervisor and Data Management Macro-Instructions, Form C28-6647.

BLKSIZE: The BLKSIZE subparameter specifies the buffer size to be used; the maximum is 32K bytes. As a general rule for tape, a larger block size is more efficient. On disk, track capacity is the most efficient block size. The block size specified should be large enough to hold the largest logical record produced. No spanning of a logical record into physical records will then occur.

BUFNO: The BUFNO subparameter specifies the number of buffers to be used. If a value of 1 is specified for BUFNO, single buffering is provided. If either no value or any value other than 1 is specified, double buffering, which offers an overlap advantage, is provided.

RECFM: The RECFM subparameter specifies both record format and the use of blocking. When records are blocked, fewer I/O requests are made to a device during the processing of logical records; I/O processing speeds are thereby increased. In general, large blocking factors improve performance. (See "Record Format" for additional information.)

OPTCD: OPTCD=C requests the use of chained scheduling, a feature that results in the decrease of I/O transfer time. Chained scheduling is put into effect only when an I/O request is received before a previous I/O request has ended. For this reason, it is difficult to predict when chained scheduling will be effective. However, the use of chained scheduling will provide a performance improvement in the formatting that is done with a new direct access data set. For sequential data sets, the user may wish to measure the effect before selecting chained scheduling for production runs.

Direct Access Programming

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

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

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

Each of the master records should contain a unique identification that distinguishes this record from any other master record. Detail records used to update the masters should contain an identification field that identifies a detail record with a master record. For example, astronomers might have assigned unique numbers to some stars and 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. Any detail record used to update a master record for a star would have to contain the same number as the star.

A FORTRAN program indicates which record to FIND, READ or WRITE by its record position within the data set. The ideal situation would be to use the unique record identification as the record position. However, in most cases this is impractical. The solution to this problem is a randomizing technique. A randomizing technique is a function which operates on the identification field and converts it to a record position. For example, if 6-digit numbers are assigned to each star, the randomizing technique may truncate the last two digits of the number assigned to the star and use the remaining four digits as a record position. For example, star number 383320 would be assigned position 3833. Another example of a randomizing technique would be a mathematical operation performed on the identification number, such as squaring the identification number and truncating the first four digits and the last four digits of the result. Then the record for star number 383320 is assigned record position 3422. There is no general randomizing technique for all sets of identification numbers. The programmer must devise his own technique for a given set of identification numbers

Two problems arise when randomizing techniques are used. The first problem is that there may be a lot of space wasted on the volume. The solution in this instance must be developed within the randomizing

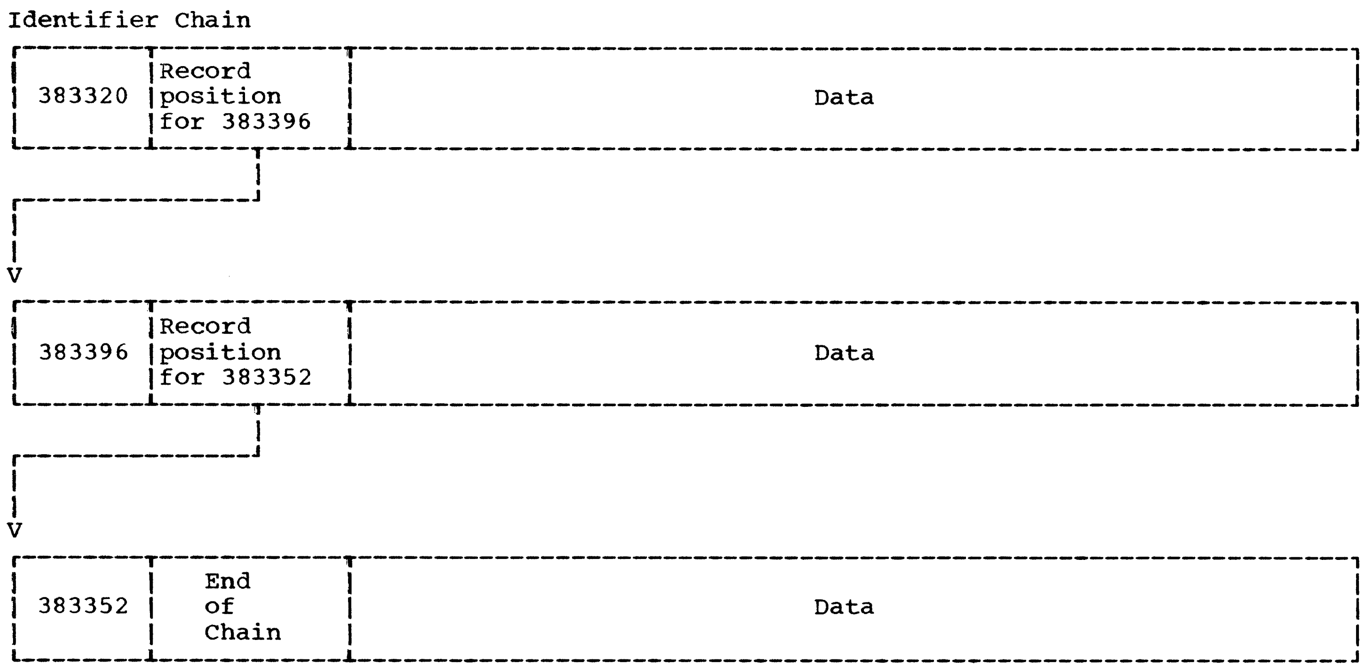


Figure 52. Record Chaining

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 1000 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 <u>synonyms</u>. 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.

<u>Chaining</u> 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 52 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).

In addition, 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 <u>Supervisor and Data Management Services</u> publication, Form C28-6646.

Figure 53 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 4 in Appendix A shows a program and job control statements used to update a direct access data set.

Direct Access Programming Considerations

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

Space must be allocated in the SPACE parameter of the DD statement for a data set written on a direct access volume. For direct access data sets, the space allocated in the SPACE parameter should be consistent with the record length and number of records specified in the DEFINE FILE statement in the FORTRAN program. For example, in the DEFINE FILE statement

```
DEFINE FILE 8(1000,40,E,I)
```

the number of records is specified as 1000 and the record length is specified as 40. When this program is executed the DD statement for this data set should contain the SPACE parameter

```
SPACE (40,(1000))
```

indicating that space is allocated for 1000 records, and 40 bytes for each record.

The DEFINE FILE statement for a data set must be in a source module in the root segment (i.e., it cannot be overlaid), but does not have to be in the same source module in which I/O operations occur. For example, the DEFINE FILE statement can be given in a main program with a subprogram performing the I/O operations on the data set. However, if an associated variable defined in the main program is to be used by a subprogram, it must be passed to the subprogram in COMMON. Since an associated variable is updated by input/output operations, the subprogram cannot get to the updated value to make use of it in its operations unless the associated variable is in COMMON.

The FIND statement permits record retrieval to occur concurrently with computation or I/O operations performed on different data sets. By using the FIND statement, load module execution time can be decreased. For example, the statements

```
10  A=SQRT(X)
    .
    .
    .
52  E=ALPHA+BETA*SIN(Y)
64  WRITE(9)A,B,C,D,E
76  READ(8'101)X,Y
```

are inefficient because computations are performed between statements 10 and 52 and an I/O operation is performed on another data set while record number 101 could be retrieved. If the following statements are substituted, the execution of this module becomes more efficient because record number 101 is retrieved during computation and I/O operations on other data sets:

```
 5  FIND(8'101)
10  A=SQRT(X)
    .
    .
    .
52  E=ALPHA+BETA*SIN(Y)
64  WRITE(9)A,B,C,D,E
76  READ(8'101)X,Y
```

How Arguments Are Passed

Although the programmer cannot alter the method for passing arguments to a subprogram, knowing how arguments are passed may be valuable when he debugs his program. A main program passes arguments to a subprogram in two ways.

The first method is called "call by value." When this method is used, the main program moves the value currently residing in the argument in the main program into the location assigned to the argument in the subprogram. When the subprogram returns to the main program, the value of the argument in the subprogram is moved to the location of the argument in the main program. In FORTRAN (E), only variables are passed using "call by value."

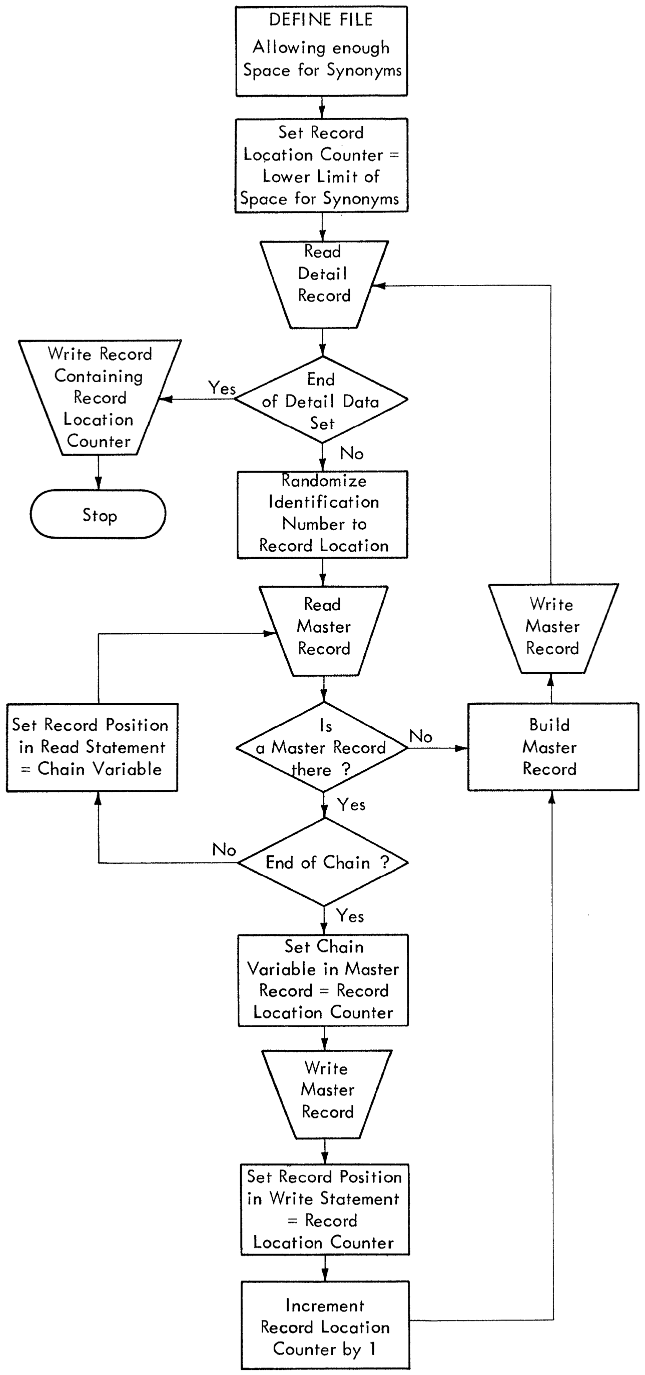

DEFINE FILE Allowing enough Space for Synonyms
Set Record Location Counter = Lower Limit of Space for Synonyms
Read Detail Record
End of Detail Data Set
Yes
Write Record Containing Record Location Counter
Stop
No
Randomize Identification Number to Record Location
Read Master Record
Write Master Record
Set Record Position in Read Statement = Chain Variable
Is a Master Record there ?
No
Build Master Record
Yes
No
End of Chain ?
Yes
Set Chain Variable in Master Record = Record Location Counter
Write Master Record
Set Record Position in Write Statement = Record Location Counter
Increment Record Location Counter by 1

Figure 53. Writing a Direct Access Data Set for the First Time

The second method is called "call by name." When this method is used, the main program moves an address into the location assigned to the argument in the subprogram. In FORTRAN (E), arrays and subprograms, used as arguments, are passed using "call by name." The main program moves the address of the first element in the array into the subprogram or moves the address of the entry point of the subprogram, used as an argument, into the subprogram.

DD STATEMENT CONSIDERATIONS

Several DD statement parameters and subparameters are provided for I/O optimization (see Figure 54). 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 decreases the time required for I/O operations.

I/O Device Optimization

UNIT subparameters can be specified for device optimization.

VOLUME MOUNTING AND DEVICE SEPARATION:

UNIT=(name$\begin{bmatrix},n\\,P\end{bmatrix}$[,DEFER]

[,SEP=(ddname[,ddname]...)])

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

DEFER
: indicates that the volume(s) for the data set need not be mounted until needed. The control program notifies the operator when to mount the volume.

```
SEP=(ddname[,ddname]...¹) ²

      {(name[, n|P ³][,DEFER][,SEP=(ddname[,ddname]...¹) ²]⁴ ⁵)⁶}
UNIT= {AFF=ddname                                             }

 /SPACE=(ABSTR,(quantity,beginning-address[,directory-quantity]))              \
|                                                                                 |
| SPLIT=(n [{,CYL                   },(primary-quantity[,secondary-quantity])] ) |
|          [{,average-record-length }                                         ]   |
|                                                                                 |
|             {TRK                  }                                             |
| SUBALLOC=( {CYL                  },(primary-quantity[,secondary-quantity]     |
|             {average-record-length}                                             |
|                                           {ddname                 }             |
 \                  [,directory-quantity]),{stepname.ddname        })            /
                                           {stepname.procstep.ddname}
```

¹The maximum number of repetitions allowed is 7.
²If only one "ddname" is specified, the delimiting parentheses may be omitted.
³If neither "n" nor "P" is specified, 1 is assumed.
⁴This subparameter is applicable only for direct-access devices.
⁵This subparameter is the only keyword subparameter shown in this figure. All the remaining subparameters shown in the UNIT, SPACE, SPLIT, and SUBALLOC parameters are positional subparameters.
⁶If only "name" is specified, the delimiting parentheses may be omitted.

Figure 54. DD Statement Parameters for Optimization

SEP=(ddname[,ddname]...)
: is used when a data set is not to be 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. This parameter is meaningful only for data sets residing on direct access volumes. The SEP subparameter in the UNIT parameter provides for device separation, while the SEP parameter provides for channel spearation.

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 the AFF subparameter 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 discussed in "Creating Data Sets" is used to allocate space on a volume. Another form of the SPACE parameter may also 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 that data set did not use.

SPACE BEGINNING AT A SPECIFIED ADDRESS:

SPACE=(ABSTR,quantity,beginning-address
    [,directory-quantity])

: specifies space beginning at a particular address on a direct access volume. The "quantity" is the number of tracks allocated to the data set. The "beginning address" is the track address on a direct access volume where the space begins. If the data set is a partitioned data set, the "directory quantity" specifies the number of records allocated to the directory.

SPLITTING THE USE OF CYLINDERS AMONG DATA SETS: If several sequential data sets in a step use the same direct access volume, 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. Direct access and partitioned data sets cannot be created using the SPLIT parameter.

```
          ┌ ┌,CYL                   ┐
SPLIT=(n  │ └,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 the current step or previous steps may not have used all the space allocated to them by a DD statement. The SUBALLOC parameter may be substituted for the SPACE parameter to permit a new data set to use this unused space.

```
          ┌TRK                   ┐
SUBALLOC=(┤CYL                   ├,
          └average-record-length ┘

          (primary-quantity,

          [,secondary-quantity]

          [,directory-quantity]),

          ┌ddname                    ┐
          ┤stepname.ddname           ├)
          └stepname.procstep.ddname  ┘
```

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

Priority Scheduler Considerations for Cataloged Procedures

If, during system generation, the installation selects a priority scheduler and an operating system that provides multiprogramming with a variable number of tasks (MVT), the following information must be taken into consideration when writing FORTRAN programs.

1. The PRFRM option must be in effect for the compile step, either by default or by explicit request in the compiler job step PARM field. Similarly, the SIZE allocation must be at least 19,456 bytes. This figure assumes no blocking. If the input is blocked (e.g., by an input reader), a figure that is 160 times the blocking factor must be added to the 19,456-byte specification in the SIZE option. For a compile step, REGION must be at least 16K bytes greater than the compiler SIZE specified in the PARM field (or default SIZE).

2. If the default or cataloged REGION parameter is not sufficient for a particular compile, link edit, or execute job step, an adequate REGION must be specified on the appropriate JOB or EXEC card. For a link edit step, the specification for REGION depends on which linkage editor is used (see "Linkage Editor Restrictions").

Linkage Editor	REGION
IEWLE150	24K
IEWLE180	26K
IEWLE440	54K

3. DCB BLKSIZE parameters must be specified on the DD cards for SYSLIN, SYSPUNCH, and SYSPRINT. For SYSPRINT, this block size is 121; for SYSPUNCH it is 80. The blocking factor determines the specification for SYSLIN, but it must be specified as a multiple of 80. If this figure is added to the SIZE specification, it must also be added to that for REGION.

4. Compiler data sets handled by output writers cannot be blocked (e.g., SYSPRINT and SYSPUNCH).

5. All FORTRAN programs, upon completion of processing, must use either a STOP statement or a CALL statement using the entry name EXIT. The omission of STOP or CALL EXIT in a FORTRAN (E) program will result in failure to clear buffers for all FORTRAN data sets used in the execution of the job;

therefore, with blocked output, a block of print lines could be lost.

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 publication IBM System/360 FORTRAN IV Library Subprograms, Form C28-6596. 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 section "Moving and Copying Data" in the Utilities publication.

When the FORTRAN library is changed either 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 to store the resultant load module back into the FORTRAN library. When the facilities of the linkage editor are used to provide an overlay structure or to replace a single control section, care should be taken not to mix FORTRAN library modules that are at diverse operating system levels. This warning applies especially to the library modules created when FORTRAN E selects the extended error message facility at system generation time and uses the FORTRAN G or H library.

COMPILER RESTRICTIONS

Table 14 shows the average number of source statements that can be compiled by the FORTRAN compiler with regard to the SPACE and PRFRM option and the size (in bytes) of the Dictionary and the Overflow Table used by the compiler. The Dictionary and the Overflow Table are used by the compiler to contain information concerning variables, arrays, subscripts, functions, data set reference numbers, statement numbers, etc.

The Dictionary and the Overflow Table size in bytes, S, required to compile a number of source statements, X, is approximately

$$S=10X+500$$

The following is a list of compiler restrictions:

- The maximum number of variables that may be equated in EQUIVALENCE statements is approximately 100. For compilations in which the largest unused portion of the Dictionary and the Overflow Table exceeds 800 bytes, the maximum becomes the number of bytes in this segment divided by 8. For example, if the compiler allocates 5500 bytes to the Dictionary and the Overflow Table, and 3100 are used, then the maximum becomes 2400/8=300.
- The maximum number of names for variables and arrays that may appear in an I/O list is approximately 250.
- The maximum number of arguments in a subprogram call or subprogram definition is 48.
- The maximum level of nesting for DO loops is 25.
- The maximum number of statement numbers in a computed GO TO statement is approximately 250.
- The maximum number of records allowed in a direct access data set is 2^{24} (2^{24}=16,777,216).
- The maximum size of an array is 131,071 bytes.
- The maximum total program and data size (including COMMON) is 196,608 bytes.
- The total number of statement numbers referred to (excluding FORMAT statement numbers), DO statements, and statement functions cannot exceed 1000.
- The number of arguments in a statement function cannot exceed 15.

Table 14. Source Module Size Restrictions

SIZE Option		Average Number of Source Statements that Can be Compiled		Dictionary and Overflow Table Size (in bytes)		Intermediate Text Capacity (in bytes)		
						SPACE	PRFRM	
SPACE	PRFRM[1]	SPACE	PRFRM	SPACE	PRFRM	Buffer Size	Buffer Size	In Storage[2]
15K	19K	170	170	2216	2216	104	96	0
44K	48K	2500	1980	25512	20328	1704	1696	5184
86K	90K	6500	6500	65536	65536	1704	1696	8104
200K	204K	6500	6500	65536	65536	1704	1696	119720

[1]If blocked input and output are used, the value of the expression [2*(BLKSIZE)] must be added for each data set that contains blocked records to the number shown under the PRFRM option.
[2]The figures under "In Storage" indicate how many additional bytes are available for retaining the intermediate text in main storage before the text has to be written on external storage. If the storage required for intermediate text does not exceed this figure, no I/O operations are performed for the intermediate text.

LINKAGE EDITOR RESTRICTIONS

The maximum number of load modules and object modules that can be processed by each linkage editor varies according to the number of bytes of main storage reserved for linkage editor operations. This maximum is shown below for each of the three linkage editors.

Linkage Editor Name	Bytes Reserved for Operation	Maximum
IEWLE150	15K	119
	18K	229
IEWLE180	18K	75
	20K	140
IEWLE440	44K	349
	88K	1250

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

FORTRAN LOAD MODULE RESTRICTIONS

The following is a list of FORTRAN load module restrictions:

- A FORTRAN load module cannot read past the end of a data set.
- For the EXP and DEXP library functions, the argument cannot exceed 174.673.
- For the SIN and COS library functions, the absolute value of the argument cannot exceed $2^{18}\cdot\pi$ ($2^{18}\cdot\pi$=.82354966406249996D+06).
- For the DSIN and DCOS library functions, the absolute value of the argument cannot exceed $2^{50}\cdot\pi$($2^{50}\cdot\pi$=.35371188737802239D+16).
- The minimum record length for records written on a magnetic tape volume is 18. The minimum record length for records read from a magnetic tape volume is 12.
- A data set reference number cannot exceed the maximum data set reference number specified by the installation when the system is generated.

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 storage map showing the location of variables and constants in the object module, and an object module card deck. Source module diagnostic messages are also produced during compilation.

Source Listing

If the SOURCE option is specified or assumed, the source listing is written by the output device specified in the SYSPRINT DD statement. A sequential internal statement number:

S.nnnn (1≤nnnn≤9999)

is assigned to each source statement. (Comments and continuation cards are not assigned an internal statement number.) The internal statement number is then used in the diagnostic messages to indicate erroneous statements in the source program. An example of a source program listing is shown in Figure 55. This printout is the source listing of Sample Program 1 shown in "Appendix D" of the publication _Basic FORTRAN IV Language_.

Storage Map

If the MAP option is specified, a storage map of the object module is written on the data set specified by the SYSPRINT DD card. The storage map gives a listing of:

1. The relative addresses and names of all variables, including subprogram names and in-line subprogram names.
2. The relative addresses and names of all external references, including all subprograms, except in-line subprograms.
3. All user-specified literal constants and their relative addresses.
4. All compiler-generated constants and their relative addresses.
5. A branch list consisting of all statement numbers referred to and their relative addresses.

An example of a map printout is shown in Figure 56. This printout is the source module map of Sample Program 1 shown in "Appendix D" of the publication _Basic FORTRAN IV Language_.

Object Module Card Deck

The compiler produces an object module. This module is composed of four types of card images -- TXT, RLD, ESD, and END. If the DECK option is specified, the object module is written on the device specified

```
            C     PRIME NUMBER PROBLEM
S.0001      100 WRITE (6,8)
S.0002        8 FORMAT (52H FOLLOWING IS A LIST OF PRIME NUMBERS FROM 1 TO 1000/
               119X,1H1/19X,1H2/19X,1H3)
S.0003      101 I=5
S.0004        3 A=I
S.0005      102 A=SQRT(A)
S.0006      103 J=A
S.0007      104 DO 1 K=3,J,2
S.0008      105 L=I/K
S.0009      106 IF(L*K-I)1,2,4
S.0010        1 CONTINUE
S.0011      107 WRITE (6,5)I
S.0012        5 FORMAT (I20)
S.0013        2 I=I+2
S.0014      108 IF(1000-I)7,4,3
S.0015        4 WRITE (6,9)
S.0016        9 FORMAT (14H PROGRAM ERROR)
S.0017        7 WRITE (6,6)
S.0018        6 FORMAT (31H THIS IS THE END OF THE PROGRAM)
S.0019      109 STOP
S.0020          END
```

Figure 55. Source Module Listing

```
                                 STORAGE MAP      VARIABLES  (TAGSO C=COMMON, E=EQUIVALENCE)

NAME     TAG            REL ADR    NAME      TAG          REL ADR    NAME      TAG        REL ADR    NAME      TAG          REL ADR

I                       000154     A                      000158     J                    00015C     K                      000160
L                       000164

                                                 EXTERNAL REFERENCES

NAME                    REL ADR    NAME                   REL ADR    NAME                 REL ADR    NAME                   REL ADR

SQRT                    000168

                                                      CONSTANTS

NAME                    REL ADR    NAME                   REL ADR    NAME                 REL ADR    NAME                   REL ADR

00000005                000184     00000002               000188     000003E8             00018C

                                             IMPLIED EXTERNAL REFERENCES

NAME                    REL ADR    NAME                   REL ADR    NAME                 REL ADR    NAME                   REL ADR

IBCOM#                  000220

STATEMENT NUMBER        REL ADR    STATEMENT NUMBER       REL ADR    STATEMENT NUMBER     REL ADR    STATEMENT NUMBER       REL ADR

    00100               000256         00100              000256          00008           000190          00101             00026C
    00003               000274         00102              000294          00103           0002A6          00104             0002C4
    00105               0002CC         00106              0002DC          00001           0002F4          00107             000308
    00005               0001DC         00002              000324          00108           000330          00004             000344
    00009               0001E0         00007              000358          00006           0001F4          00109             00036C
```

Figure 56. Storage Map

in the SYSPUNCH DD statement; if LOAD is specified, the module is written on the device specified in the SYSLIN DD statement. A functional description of these card images is given in the following paragraphs.

OBJECT MODULE CARD IMAGES: Every card image in the object module 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 77-80.

ESD Card: Three types of ESD card images are generated:

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

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

ESD, type 5 - contains information about the COMMON area.

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

RLD Card Image: An RLD card image is generated for external references indicated in the ESD, type 2 card images. To complete external references, the addresses in the RLD card image are matched with external symbols in the ESD card images by the linkage editor. When it has resolved external references, the storage at the address indicated in the RLD card image contains the address assigned to the subprogram indicated in the ESD, type 2 card image. RLD card images are also generated for a branch list produced for statement numbers, DO loops, and Statement Functions.

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

END Card Image: One END card image is generated for each compiled source module. This card indicates the end of the object module to the linkage editor, and contains the entry point (where control is given to begin execution of the module after it is link edited).

OBJECT MODULE DECK STRUCTURE: Figure 57 indicates the FORTRAN object module deck structure. If the object module is written on a device other than the card punch, the structure of the module is the same.

Source Module Diagnostics

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

Source Module Informative Messages: Source module messages tell the programmer or operator about the status of the compiler. A message is generated when the compilation

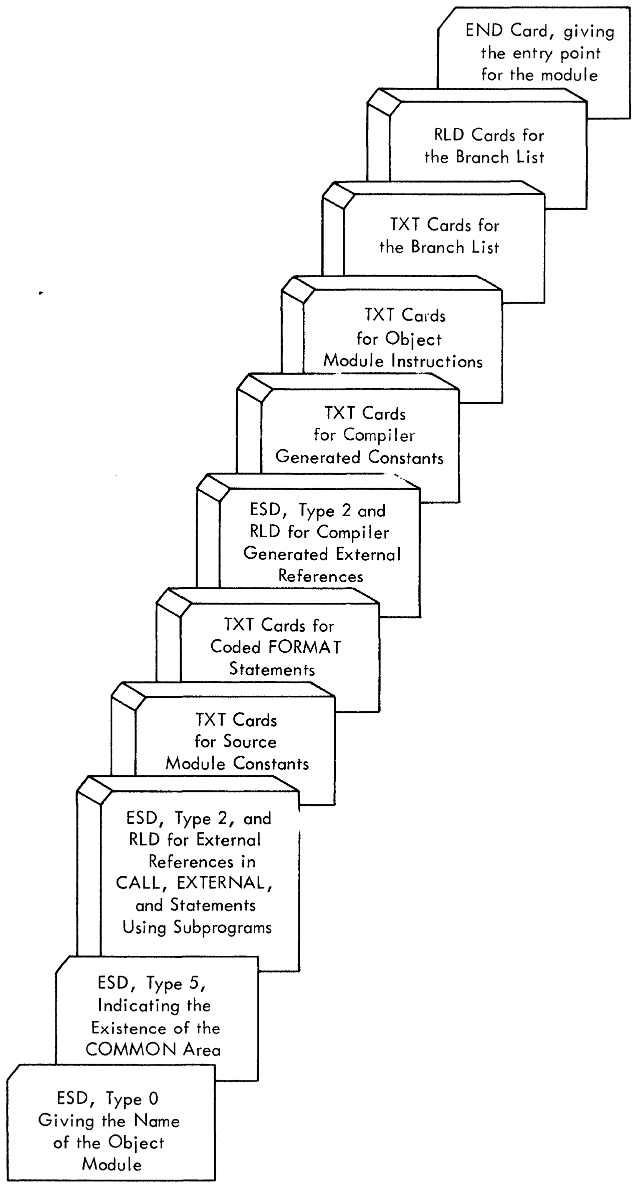

END Card, giving the entry point for the module
RLD Cards for the Branch List
TXT Cards for the Branch List
TXT Cards for Object Module Instructions
TXT Cards for Compiler Generated Constants
ESD, Type 2 and RLD for Compiler Generated External References
TXT Cards for Coded FORMAT Statements
TXT Cards for Source Module Constants
ESD, Type 2, and RLD for External References in CALL, EXTERNAL, and Statements Using Subprograms
ESD, Type 5, Indicating the Existence of the COMMON Area
ESD, Type 0 Giving the Name of the Object Module

Figure 57. Object Module Deck Structure

has begun, when the compiler options are processed, and at the end of compilation. 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 storage map. Figure 58 shows the format of each message as it is written on the data set specified by the SYSPRINT DD statement.

When error conditions cannot be ascribed to a single source statement, the error message contains an internal statement number S.0000. For example, in the FORTRAN statements

```
DOUBLE PRECISION DP
COMMON R,DP
```

the error message

```
     IEJ146I     S.0000 INCONSISTENT EQUATE
```

is issued, because a double-precision variable is not placed on the proper boundary. The error could be attributed to either FORTRAN statement, so the internal statement number S.0000 is assigned to the error message.

There are two types of error/warning messages: serious error messages and warning messages beginning with the word "WARNING". Serious error messages transmit a condition code of 8, 12, or 16. Warning messages transmit a condition code of 4.

MESSAGE NUMBER	STATEMENT NUMBER	DESCRIPTION
IEJnnnI	S.xxxx	message

nnn	is the message number
xxxx	is the internal statement number
message	is the actual message printed

Figure 58. Format of Diagnostic Messages

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 "Appendix D" of the publication Linkage Editor.

Module Map

The module map is written on the data set specified in the SYSPRINT DD statement for the linkage editor. To the linkage editor, each program (main or subprogram) and the COMMON area are control sections.

Each control section name is written along with origin and length of the control section. For a program, the name is listed; for 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 publication Linkage Editor).

The names and locations of each control section and 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 also listed.

Figure 59 shows a load module map for Sample Program 1 shown in Appendix D of the publication Basic FORTRAN IV Language.

Cross-Reference List

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

Figure 60 shows the cross-reference list and module map for Sample Program 1 shown in "Appendix D" of the publication Basic FORTRAN IV Language.

LOAD MODULE OUTPUT

The programmer defines the output data sets for load module execution in READ,

```
                                        ---- MODULE MAP ----

 CONTROL SECTION                        ENTRY

   NAME     ORIGIN  LENGTH                NAME    LOCATION     NAME    LOCATION     NAME    LOCATION     NAME    LOCATION

 MAIN          00     37A
 IHCSSQRT*    380      AC
                                        SQRT         380
 IHCFCOME*    430    1484
                                        IBCOM#       430     FDIOCS#      A60
 IHCFIOSH*   1888     C50
                                        FIOCS#      1888
 IHCUATBL*   2508     638

 ENTRY ADDRESS         00
 TOTAL LENGTH        2B40
```

Figure 59. Module Map

```
                                          ---- CROSS REFERENCE TABLE ----

   CONTROL SECTION                        ENTRY

     NAME      ORIGIN  LENGTH               NAME   LOCATION     NAME   LOCATION     NAME   LOCATION     NAME   LOCATION

   MAIN           00     37A
   IHCSSQRT*     380      AC
                                          SQRT         380
   IHCFCOME*     430    1484
                                          IBCOM#       430     FDIOCS#      A60
   IHCFIOSH*    1888     C50
                                          FIOCS#      1888
   IHCUATBL*    2508     638


   LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION

       168           SQRT            IHCSSQRT
       220           IBCOM#          IHCFCOME
       404           IBCOM#          IHCFCOME
       A5C           FIOCS#          IHCFIOSH
       CFC           FIOCS#          IHCFIOSH
      19DC           IHCUATBL        IHCUATBL
      19E8           IBCOM#          IHCFCOME
  ENTRY ADDRESS        00
  TOTAL LENGTH       2840
```

Figure 60. Linkage Editor Cross-Reference List

WRITE, and FORMAT statements. At execution time, FORTRAN load module diagnostics are generated in three forms - error code diagnostics, program interrupt messages, and operator messages. An error code indicates an input/output error or a misuse of a FORTRAN library function. A program interrupt message (which is a special form of an error code diagnostic) indicates a condition which System/360 cannot correct. An operator message is generated when a STOP or PAUSE is executed.

Error Code Diagnostics

When an error condition arises during execution of a FORTRAN load module, a message of the form

IHCxxxI [message text]

is printed. The error code is the number specified by the digits xxx. With some error code diagnostics, a "message text" is printed. The error code diagnostics are described in Appendix D.

The error code diagnostics are written on a data set specified by the programmer. (See "Job Processing.")

Program Interrupt Messages

Program interrupt messages containing the old program status word (PSW) are written when an exception occurs. Operator intervention is not required for any of these exceptions (interrupts), and execution is not terminated. The program interrupt messages are written on a data set specified by the programmer (see "Job Processing.") For a detailed description of these messages, see Appendix D.

ABEND Dump

If a program interrupt occurs that causes abnormal termination of a load module, an indicative dump is given (i.e., only the contents of significant registers, indicators, etc., are dumped). However, if a programmer adds the statement:

```
//GO.SYSABEND DD SYSOUT=A
```

to the execute step of a cataloged procedure, all main storage and registers are dumped. For information about interpreting the indicative and abnormal termination dumps, see Part V of 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.**

The following examples show several methods to process load modules. Example 1 consists of a single job step that uses blocked variable-length records as output in a matrix inversion application. Example 2 shows the rocket firing example used in the "Introduction" to show job and job step relationships. Example 3 uses a generation data group to report and forecast the weather. Example 4 shows a program to update a direct access data set that contains star master records.

Example 1

Problem Statement:
A previously created and cataloged data set SCIENCE.MATH.MATRICES contains 80 matrices. Each matrix is an array containing real variables. The size of the matrices varies from 2x2 to 15x15; the average size is 10x10. The matrices are inverted by a load module MATINV in the PDS MATPROGS. Each inverted matrix is written as a single record on the data set SCIENCE.MATH.INVMATRS. The first variable in each record denotes the size of the matrix. Each inverted matrix is printed.

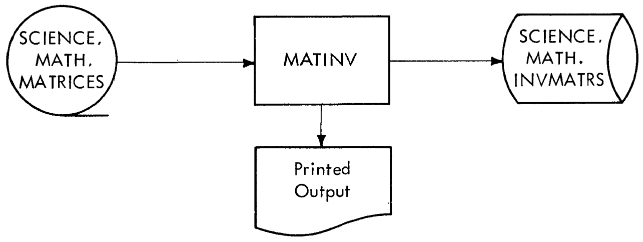


Figure 61. Input/Output Flow for Example 1

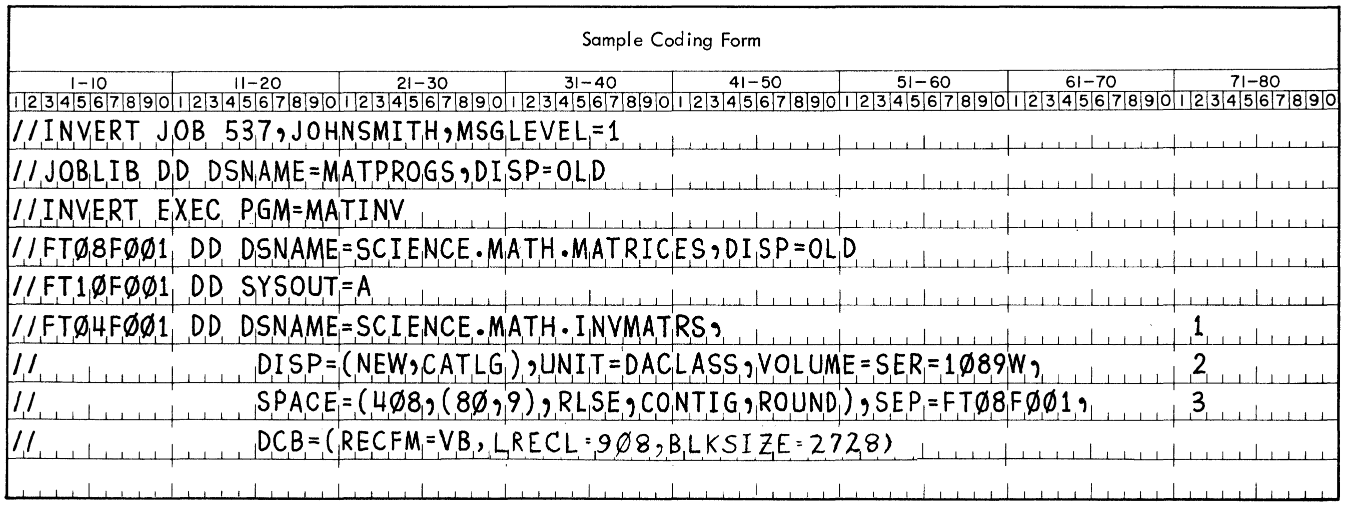
The I/O flow for the example is shown in Figure 61. The job control statements used to define this job are shown in Figure 62.

Explanation: The JOB statement identifies the programmer as JOHN SMITH and supplies the account number 537. The MSGLEVEL parameter indicates that both control statements and control statement diagnostic messages are printed on the console typewriter.

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

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

DD statement FT08F001 identifies the input data set, SCIENCE.MATH.MATRICES. (In the load module, data set reference number 8 is used to read the input data set.) Because this data set has been previously created and cataloged (DISP=OLD), no other information has to be supplied.



Sample Coding Form

```
//INVERT JOB 537,JOHNSMITH,MSGLEVEL=1
//JOBLIB DD DSNAME=MATPROGS,DISP=OLD
//INVERT EXEC PGM=MATINV
//FTØ8FØØ1 DD DSNAME=SCIENCE.MATH.MATRICES,DISP=OLD
//FT1ØFØØ1 DD SYSOUT=A
//FTØ4FØØ1 DD DSNAME=SCIENCE.MATH.INVMATRS,                            1
//             DISP=(NEW,CATLG),UNIT=DACLASS,VOLUME=SER=1Ø89W,         2
//             SPACE=(4Ø8,(8Ø,9),RLSE,CONTIG,ROUND),SEP=FTØ8FØØ1,      3
//             DCB=(RECFM=VB,LRECL=9Ø8,BLKSIZE=2728)
```

Figure 62. Job Control Statements for Example 1

DD statement FT10F001 identifies the printed output. (In the load module, data set reference number 10 is used for printed output.) The data set is written on the device class specified in the SYSOUT parameter. The records are then written on a device determined by the operator when the job is executed.

DD statement FT04F001 defines the output data set. (In the load module, data set reference number 4 is used to write the data set containing the inverted matrices.) Since the data set is created and cataloged in this job step, device, volume, space, record format, and length information are supplied.

The DSNAME parameter indicates that the data set is named SCIENCE.MATH.INVMATRS. The DISP parameter indicates that the data set is created (NEW) and cataloged (CATLG) in this job step. 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 (CONTIG); any unused space is released (RLSE), and allocation begins and ends on cylinder boundaries (ROUND).

The DCB parameter indicates blocked variable-length records (RECFM), because the size of matrices varies. The FORTRAN record length is 904, the maximum size of a FORTRAN record. The maximum size of a FORTRAN record in this data set is the maximum number of elements in a matrix multiplied by the number of bytes allocated for an element (900) plus 4 for the variable that indicates the size of the matrix. LRECL is specified as 908 (the FORTRAN record length plus 4 for the segment control word used by the operating system for a variable-length record). BLKSIZE is specified as 2728 (an integral multiple of LRECL plus 4 for the block control word used for blocked variable-length records).

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

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 63 shows the I/O flow for this example. It is the same as Figure 1 in the "Introduction" except that the data sets and programs have been assigned the names indicated in the preceding text. Figure 64 shows the job control statements used to process this job.

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

Explanation: The JOB statement indicates the programmer's name, JOHN SMITH, and that control statements and control statement errors are printed on the console typewriter (MSGLEVEL=1).

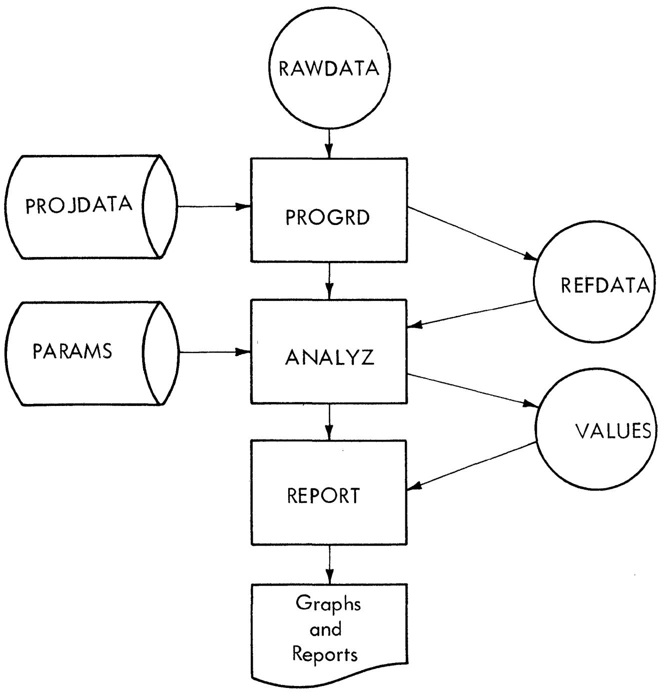
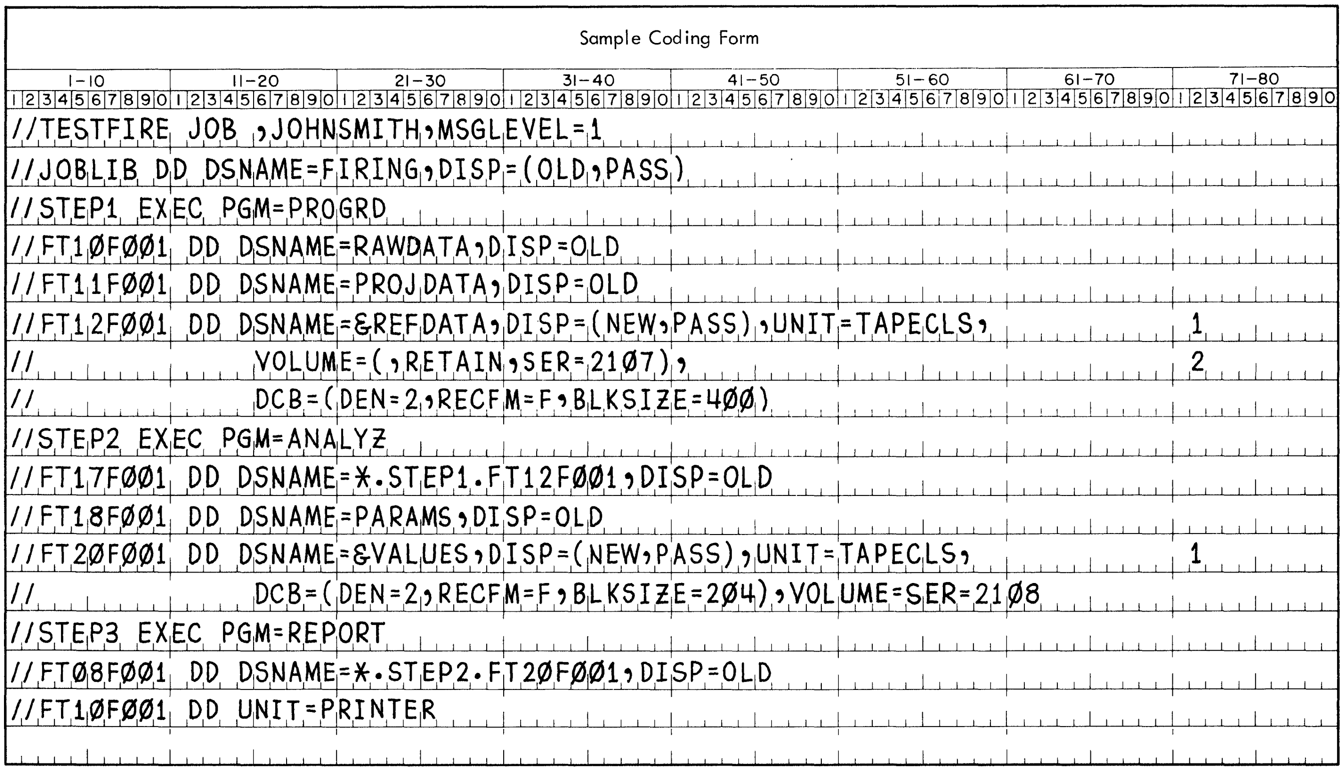


Figure 63. I/O Flow for Example 2


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

```
//TESTFIRE JOB ,JOHNSMITH,MSGLEVEL=1
//JOBLIB DD DSNAME=FIRING,DISP=(OLD,PASS)
//STEP1 EXEC PGM=PROGRD
//FT1ØFØØ1 DD DSNAME=RAWDATA,DISP=OLD
//FT11FØØ1 DD DSNAME=PROJDATA,DISP=OLD
//FT12FØØ1 DD DSNAME=&REFDATA,DISP=(NEW,PASS),UNIT=TAPECLS,            1
//             VOLUME=(,RETAIN,SER=21Ø7),                              2
//             DCB=(DEN=2,RECFM=F,BLKSIZE=4ØØ)
//STEP2 EXEC PGM=ANALYZ
//FT17FØØ1 DD DSNAME=*.STEP1.FT12FØØ1,DISP=OLD
//FT18FØØ1 DD DSNAME=PARAMS,DISP=OLD
//FT2ØFØØ1 DD DSNAME=&VALUES,DISP=(NEW,PASS),UNIT=TAPECLS,             1
//             DCB=(DEN=2,RECFM=F,BLKSIZE=2Ø4),VOLUME=SER=21Ø8
//STEP3 EXEC PGM=REPORT
//FTØ8FØØ1 DD DSNAME=*.STEP2.FT2ØFØØ1,DISP=OLD
//FT1ØFØØ1 DD UNIT=PRINTER
```

Figure 64. Job Control Statements for Example 2

The JOBLIB DD statement indicates that the PDS FIRING is concatenated with the system library.

The EXEC statement STFP1 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 created (NEW) and is passed (PASS). The data set is written using the device class TAPFCLS. 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 (DEN). The records are fixed-length (RECFM). The logical record length is 400; therefore, the buffer length (BLKSIZE) is specified as 400.

The EXEC statement STFP2 defines the second job step in the job and indicates that the load module ANALYZ is executed.

DD statement FT17F001 identifies the data set which contains refined data. The DSNAME parameter indicates that the data set was created using DD statement FT12F001 in job step STEP1. The DISP parameter indicates that the data set is deleted after execution of this job step. The DD statement FT18F001 identifies the previously created and cataloged data set PARAMS.

DD statement FT20F001 defines the temporary data set &VALUES containing the intermediate values. The DISP parameter indicates that the data set is created in this step and that it is passed. The data set is written on volume 2108 using one of the devices assigned to the class TAPECLS. The DCB parameter indicates high density (DEN), fixed-length records (RECFM). Each record is 204 characters long (BLKSIZE).

The EXEC statement STEP3 defines the third job step and indicates that the load module REPORT is executed.

DD statement FT08F001 identifies the data set containing intermediate values. The DSNAME parameter indicates that this data set is defined by the DD statement FT20F001 in job step STEP2.

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

Example 3

Problem Statement: A generation data group, WEATHER, is updated and then several of the data sets within the group are used to produce a forecast.

The load module FILECR in the PDS WTHRPR reads a card data set and creates a new generation data set. The new generation contains current data about weather conditions. FILECR also generates a weather report.

The load module FORCST in the PDS WTHRPR then uses the new generation along with three other generations of the group to forecast the weather. The weather forecast is written on the printer. Figure 65 shows the input/output flow for the job. Figure 66 shows the job control statements.

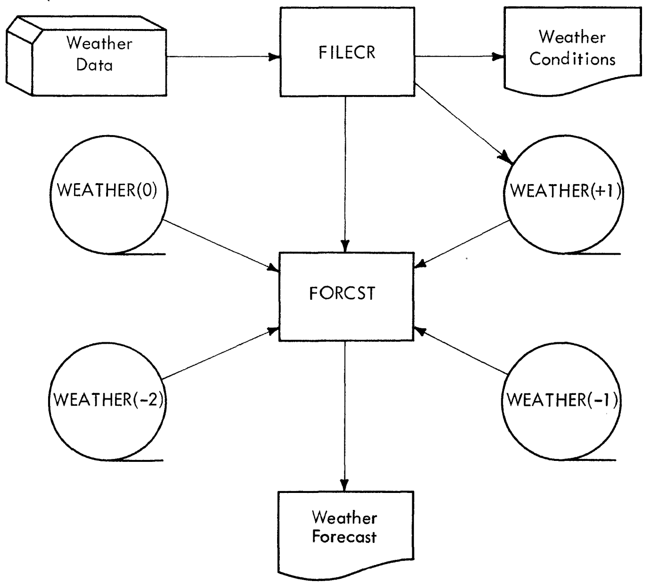


Figure 65. I/O Flow for Example 3

Explanation: The JOB statement defines the job WEATHRP to the operating system, and indicates that only control statement error messages are printed on the console typewriter. The JOBLIB DD statement indicates that the PDS WTHRPR is concatenated with the system library.

Sample Coding Form

```
//WEATHRP JOB MSGLEVEL=Ø
//JOBLIB DD DSNAME=WTHRPR,DISP=(OLD,PASS)
//CREATE EXEC PGM=FILECR
//FTØ8FØØ1 DD DSNAME=WEATHER(+1),UNIT=(HYPERT,,DEFER),                 1
//             VOLUME=(,RETAIN,SER=ØØ12),DISP=(NEW,CATLG),             2
//             LABEL=(,SL,RETPD=ØØ3Ø),DCB=(RECFM=F,BLKSIZE=4ØØ)
//FTØ3FØØ1 DD UNIT=PRINTER,DCB=PRTSP=1
//FTØ1FØØ1 DD *             WEATHER DATA FOLLOWS
        Weather Data
/*                          INDICATES END OF DATA
//FORECAST EXEC PGM=FORCST
//FT2ØFØØ1  DD  DSNAME=WEATHER(+1),DISP=OLD
//FT21FØØ1  DD  DSNAME=WEATHER(Ø),SEP=FT2ØFØØ1,DISP=OLD
//FT22FØØ1  DD  DSNAME=WEATHER(-1),DISP=OLD
//FT23FØØ1  DD  DSNAME=WEATHER(-2),DISP=OLD
//FT03FØØ1  DD  UNIT=PRINTER
```

Figure 66. Job Control Statements for Example 3

The EXEC statement defines the first step CREATE to the operating system and indicates the execution of the load module FILECR in the PDS WTHRPR.

DD statement FT08F001 defines the new member of the generation data group. A member of the class of devices HYPERT is used for the data set, and mounting of the volume is deferred (DEFER). The DISP parameter indicates a new data set (NEW) and that it is cataloged (CATLG). The label parameter indicates standard labels are written and the retention period is 30 days. The DCB parameter indicates fixed-length records (RECFM), each 400 characters long (BLKSIZE).

DD statement FT03F001 defines printed output. The DCB field indicates that the report is double-spaced. The SEP parameter indicates channel separation from the data set defined by DD statement FT08F001. DD statement FT01F001 indicates that the card data set is in the input stream.

The second job step is defined by the EXEC statement FORECAST, which indicates that the load module FORCST is to be executed. The DD statements for data set reference numbers 20 through 23 retrieve members of the generation data group WEATHER. DD statement FT03F001 indicates printed output for the weather forecast.

Example 4

A data set has been created that contains master records for an index of stars. Each star is identified by a unique six-digit star identification number. Each star is assigned a record position in the data set by truncating the last two digits in the star identification number. Because synonyms arise, records are chained.

The following conventions must be observed in processing this data set:

1. The star master record that contains the record location counter pointing to space reserved for chained records is assigned to record location 1.

2. A zero in the chain variable indicates that the end of a chain has been reached.

3. The first variable in each star master record is the star identification field; the second variable in each star master is the chain variable.

4. Each record contains six other variables that contain information about that star.

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.

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 53, except Figure 67 includes blocks that describe updating variables in master records already present in the data set. (Figure 53 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 4, whereas Figure 53 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; 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, W, 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, then the variables containing the chain record numbers are followed until the correct star master is found or a new star master is created.

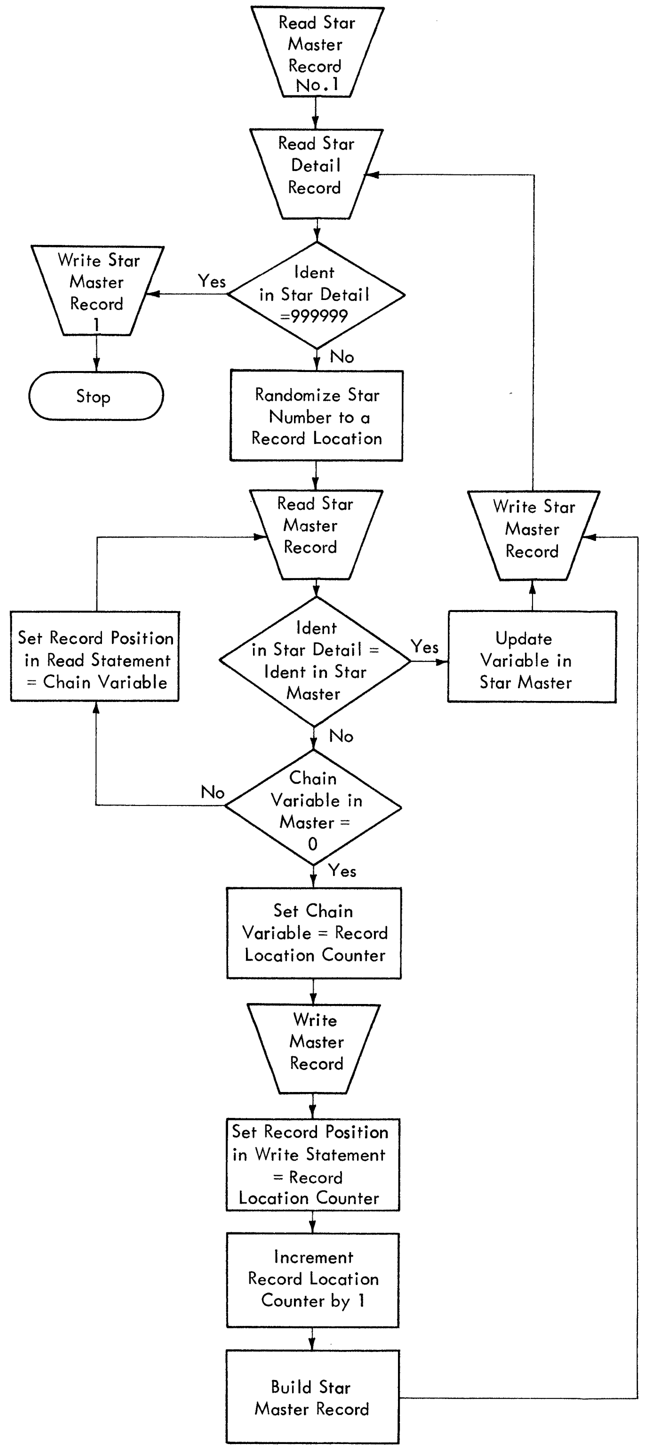
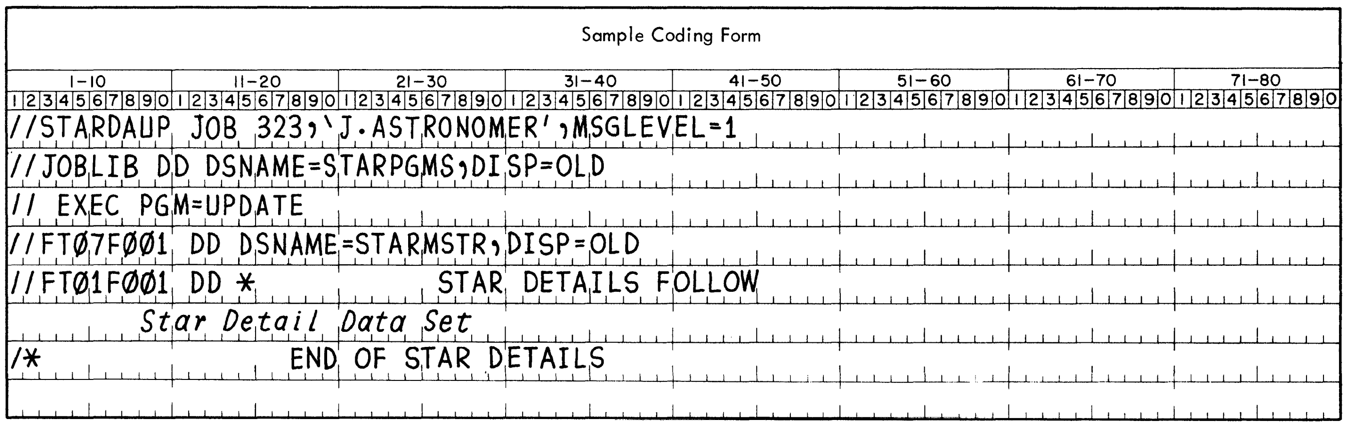

Read Star Master Record No.1
Read Star Detail Record
Write Star Master Record 1
Yes
Ident in Star Detail =999999
No
Stop
Randomize Star Number to a Record Location
Read Star Master Record
Write Star Master Record
Set Record Position in Read Statement = Chain Variable
Ident in Star Detail = Ident in Star Master
Yes
Update Variable in Star Master
No
No
Chain Variable in Master = 0
Yes
Set Chain Variable = Record Location Counter
Write Master Record
Set Record Position in Write Statement = Record Location Counter
Increment Record Location Counter by 1
Build Star Master Record

Figure 67. Block Diagram for Example 4

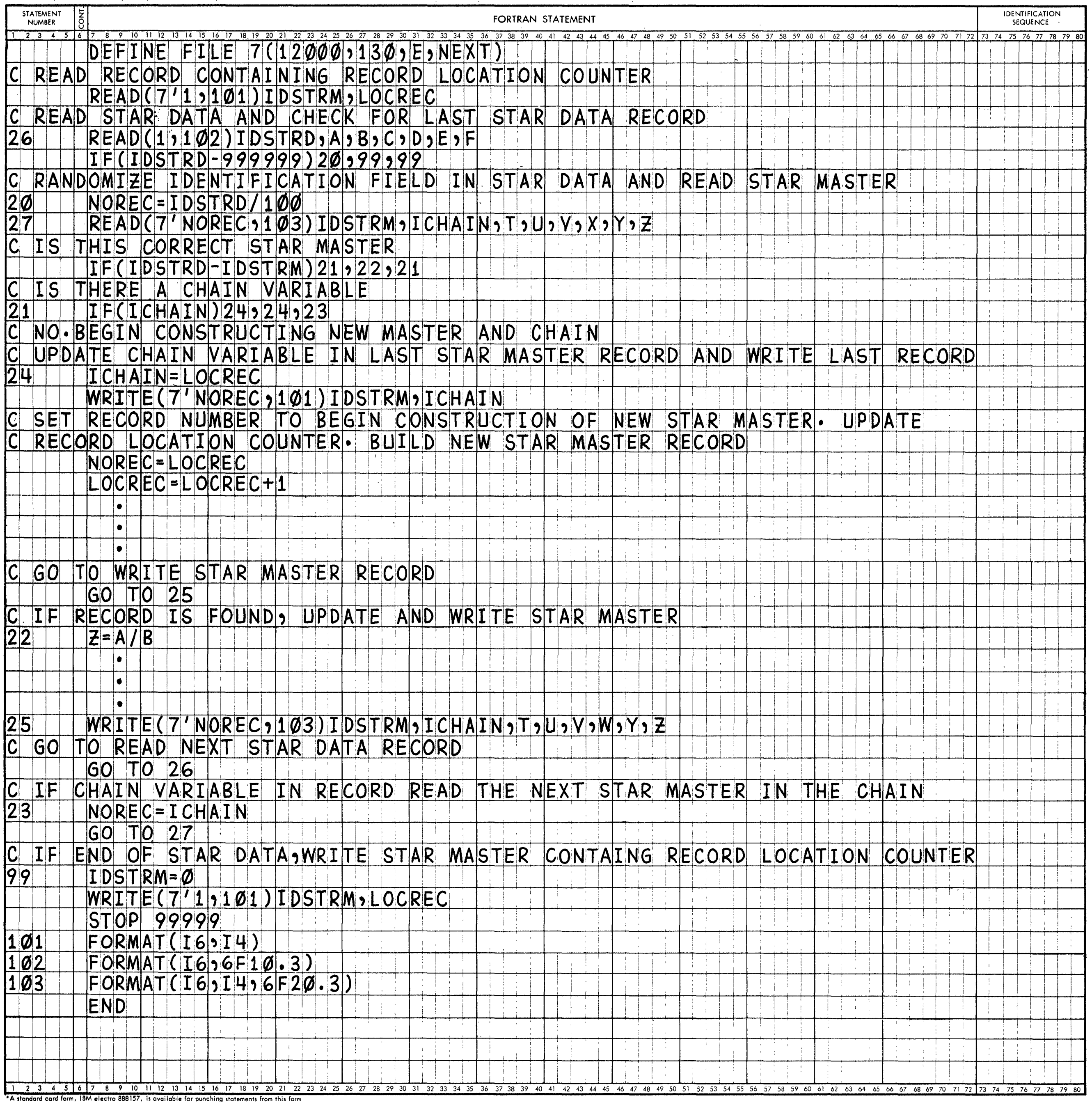
Job Control Statements: The program shown in Figure 69 is compiled and link edited, placing the load module in the PDS STARPGMS and assigning the load module the name UPDATE. The data set that contains the star master records was cataloged and assigned the name STARMSTR when it was created. Figure 68 shows the job control statements needed to execute the module UPDATE.



Sample Coding Form

```
//STARDAUP JOB 323,'J.ASTRONOMER',MSGLEVEL=1
//JOBLIB DD DSNAME=STARPGMS,DISP=OLD
// EXEC PGM=UPDATE
//FTØ7FØØ1 DD DSNAME=STARMSTR,DISP=OLD
//FTØ1FØØ1 DD *            STAR DETAILS FOLLOW
        Star Detail Data Set
/*                END OF STAR DETAILS
```

Figure 68. Job Control Statements for Example 4



STATEMENT NUMBER | CONT. | FORTRAN STATEMENT | IDENTIFICATION SEQUENCE

```
      DEFINE FILE 7(12ØØØ,13Ø,E,NEXT)
C READ RECORD CONTAINING RECORD LOCATION COUNTER
      READ(7'1,1Ø1)IDSTRM,LOCREC
C READ STAR DATA AND CHECK FOR LAST STAR DATA RECORD
26    READ(1,1Ø2)IDSTRD,A,B,C,D,E,F
      IF(IDSTRD-999999)2Ø,99,99
C RANDOMIZE IDENTIFICATION FIELD IN STAR DATA AND READ STAR MASTER
2Ø    NOREC=IDSTRD/1ØØ
27    READ(7'NOREC,1Ø3)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,1Ø1)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,1Ø3)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=Ø
      WRITE(7'1,1Ø1)IDSTRM,LOCREC
      STOP 99999
1Ø1   FORMAT(I6,I4)
1Ø2   FORMAT(I6,6F1Ø.3)
1Ø3   FORMAT(I6,I4,6F2Ø.3)
      END
```

*A standard card form, IBM electro 888157, is available for punching statements from this form

Figure 69. FORTRAN Coding for Example 4

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 publications Assembler Language and Assembler (E) Programmer's Guide or Assembler (F) 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. An argument list in which the addresses of the arguments are placed 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. Since the arguments are located in the main program, these addresses are locations within the main program. Each entry in the argument list is four bytes long 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 it is the last entry, the first (leftmost) bit in the entry contains a 1.

The address of the argument list is placed in general register 1 by the calling program.

Save Area

The calling program contains a save area in which the subprogram places information, such as the entry point for this program, an address to which the subprogram returns, general register contents, and addresses of save areas used by programs other than the subprogram. The amount of storage reserved by the calling program is 18 words. Figure 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.

FORTRAN main programs do not save floating-point registers. If these registers must be saved, the assembler language subprogram is responsible for saving and restoring these registers.

Calling Sequence

A calling sequence is generated to transmit control to the subprogram. The address of the save area in the calling program is placed in general register 13. The address of the parameter list is placed in general register 1, and the entry address for the subprogram is placed in general register 15. An instruction is then generated to branch to the address in the general register 15 and to save the return address in general register 14. Table 15 shows a summary of the use of general registers.

AREA------> (word 1)	This word is used by a FORTRAN-compiled routine to store its epilogue address and may not be used by the assembler language subprograms for any purpose.
AREA+4----> (word 2)	If this program which calls the assembler language subprogram is itself a subprogram, this location contains the address of the save area of the calling program. Otherwise this location is not used.
AREA+8----> (word 3)	The address of the save area of the subprogram called by this program.
AREA+12---> (word 4)	The contents of register 14; that is, the address to which the subprogram returns. If a subprogram returns to this program, the first byte of this location is set to ones, indicating that the called subprogram has returned control.
AREA+16---> (word 5)	The contents of register 15; that is, the address to which entry to the subprogram is made.
AREA+20---> (word 6)	The contents of register 0.
. . .	. . .
AREA+68---> (word 18)	The contents of register 12.

Figure 70. Save Area

Table 15. 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. (For subroutine subprograms, the result(s) is returned by the subprogram in a variable(s) passed to the subprogram by the programmer in his CALL statement.)
1	Argument List Register	Address of the argument list passed to the called subprogram.
13	Save Area Register	Address of the area reserved by the calling program in which the contents of 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 subprogram.

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 instruction that names an entry point for the subprogram.
2. An instruction(s) to save any 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.

Sharing Data in COMMON

With FORTRAN (E), general register 4 contains the address of the COMMON area. If the size of the COMMON area exceeds 4095 bytes, additional registers (e.g., register 5, 6, and 7) are assigned consecutively.

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 calls.

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 0 THROUGH 12.
	BALR	B,0	ESTABLISH BASE REGISTER (2≤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. Lowest Level Assembler Subprogram

4. Additional instructions in the return routine to retrieve entries in the save area.

Figure 72 shows the linkage conventions for an assembler subprogram that calls another assembler subprogram.

In-Line Argument List

The assembler programmer can establish an in-line argument list instead of an out-of-line list. In this case, he deletes the argument list shown in Figure 71 and substitutes the calling sequence shown in Figure 73 for that shown in Figure 71.

Name	Oper.	Operand	Comments
deckname	START	0	
	EXTRN	name_2	NAME OF THE SUBPROGRAM CALLED BY THIS SUBPROGRAM
	BC	15,m+1+4(15)	
	DC	X'm'	
	DC	CLm'name_1	
*		SAVE ROUTINE	
	STM	14,R,12(13)	THE CONTENTS OF REGISTERS 14, 15, AND 0 THROUGH R ARE
			STORED IN THE SAVE AREA OF THE CALLING PROGRAM. R IS ANY
			NUMBER FROM 2 THROUGH 12.
	BALR	B,0	ESTABLISH BASE REGISTER
	USING	*,B	
	LR	Q,13	LOADS REGISTER 13, WHICH POINTS TO THE SAVE AREA OF THE
*			CALLING PROGRAM, INTO ANY GENERAL REGISTER, Q, EXCEPT 0,
*			1, 13, and 15.
	LA	13,AREA	LOADS THE ADDRESS OF THIS PROGRAM'S SAVE AREA INTO
*			REGISTER 13.
	ST	13,8(0,Q)	STORES THE ADDRESS OF THIS PROGRAM'S SAVE AREA INTO THE
*			CALLING PROGRAM'S SAVE AREA
	ST	Q,4(0,13)	STORES THE ADDRESS OF THE PREVIOUS SAVE AREA (THE SAVE
*			AREA OF THE CALLING PROGRAM) INTO WORD 2 OF THIS PRO-
*			GRAM'S SAVE AREA
	BC	15,prob_1	
AREA	DS	18F	RESERVES 18 WORDS FOR THE SAVE AREA
*		END OF SAVE ROUTINE	
prob_1	user	written program statements	
	.		
	.		
	.		
*		CALLING SEQUENCE	
	LA	1,ARGLIST	LOAD ADDRESS OF ARGUMENT LIST
	L	15,ADCON	
	BALR	14,15	
	more	user written program statements	
	.		
	.		
	.		
*		RETURN ROUTINE	
	L	13,AREA+4	LOADS THE ADDRESS OF THE PREVIOUS SAVE AREA BACK INTO
*			REGISTER 13
	LM	2,R,28(13)	
	L	14,12(13)	LOADS THE RETURN ADDRESS INTO REGISTER 14.
	MVI	12(13),X'FF'	
	BCR	15,14	RETURN TO CALLING PROGRAM
*		END OF RETURN ROUTINE	
ADCON	DC	A(name_2)	
*		ARGUMENT LIST	
ARGLIST	DC	AL4(arg_1)	ADDRESS OF FIRST ARGUMENT
	.		
	.		
	.		
	DC	X'80'	INDICATE LAST ARGUMENT IN ARGUMENT LIST
	DC	AL3(arg_n)	ADDRESS OF LAST ARGUMENT

Figure 72. Higher Level Assembler Subprogram

Name	Oper.	Operand
ADCON	DC	A(prob1)
	.	
	.	
	.	
	LA	14,RETURN
	L	15,ADCON
	CNOP	2,4
	BALR	1,15
	DC	AL4(arg$_1$)
	.	
	.	
	.	
	DC	X'80'
	DC	AL3(arg$_n$)
RETURN	BC	0,X'isn'

Figure 73. In-Line Argument List

GETTING 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 double-precision 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 double-precision variable A to location VAR in the subprogram.

```
   L      Q,0(1)
   MVC    VAR(8),0(Q)
```

where:
Q is any general register.

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.

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 16 shows the general subscript format for arrays of 1, 2, and 3 dimensions.

Table 16. 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 real variable C(2,1,2) in the main program is to be moved to a location ARVAR in the subprogram. Using the formula for displacement of variables in a three-dimensional array, the displacement is calculated to be 28 and placed in the general register DISP. The following instructions can be used to move the variable

```
L       Q,8(1)
LE      S,0(Q,DISP)
STE     S,ARVAR
```

where:
    Q, R, and S are general registers.

Example: An assembler language subprogram is to be named ADDARR, and a real variable, an array, and an integer variable are to be passed as arguments to the subprogram. The statement

CALL ADDARR (X,Y,J)

is used to call the subprogram. Figure 74 shows the linkage used in the assembler subprogram.

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

Figure 74. Assembler Subprogram Example

Figure 75 shows a storage map for load module execution. The superscripts shown in the figure indicate one of the notes listed in this appendix.

<u>Note 1:</u> The COMMON area is inserted into the load module after the first object module that refers to it. For example, if the first object module does not refer to COMMON and the second object module does, the COMMON area follows the second object module.

<u>Note 2:</u> Buffers for direct access data sets are extended upward in main storage as buffers are needed.

<u>Note 3:</u> The order in which IOBs are placed in main storage is dependent on the sequence of I/O operations. The IOBs can be located anywhere in upper main storage.

<u>Note 4:</u> The routines for direct access I/O are loaded into main storage when a direct access data set is defined by a DEFINE FILE statement. The routines for sequential I/O are loaded into main storage when a sequential data set is first used.

<u>Note 5:</u> A DCB is allocated for every data set reference number used by the load module.

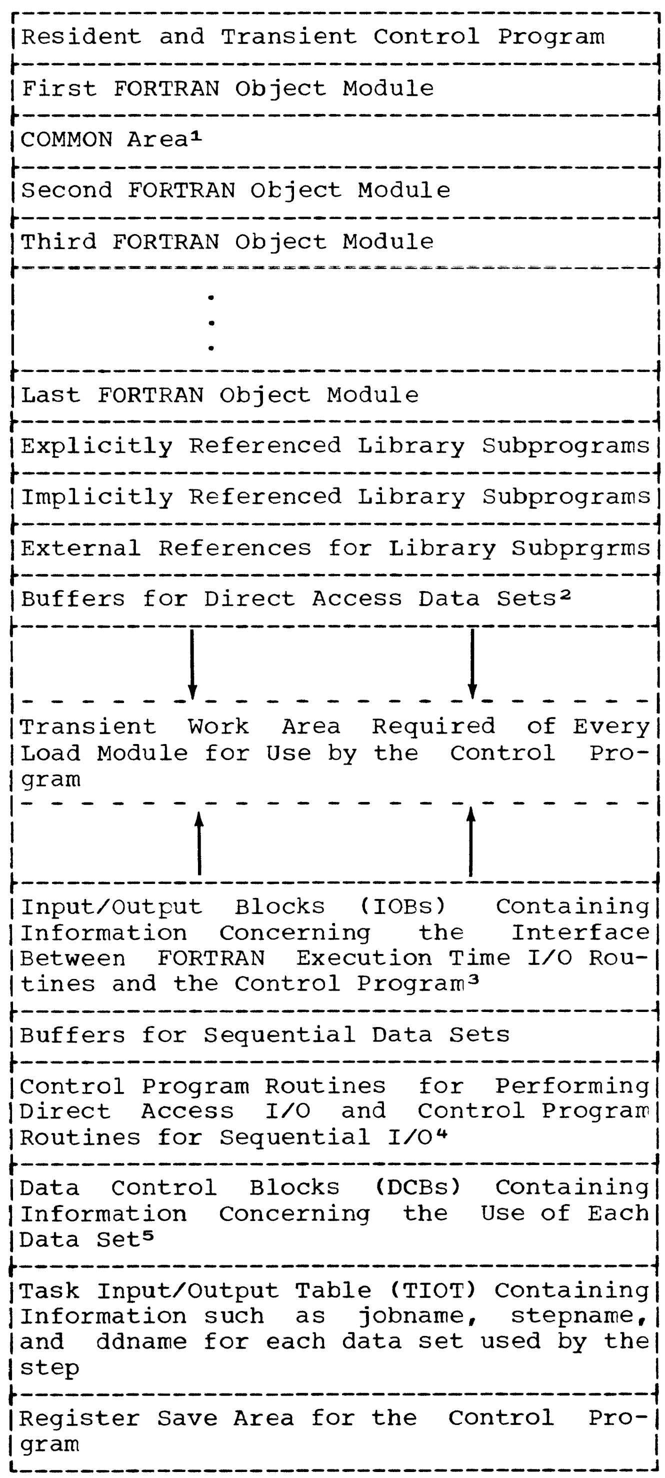

Resident and Transient Control Program
First FORTRAN Object Module
COMMON Area[1]
Second FORTRAN Object Module
Third FORTRAN Object Module
.
.
.
Last FORTRAN Object Module
Explicitly Referenced Library Subprograms
Implicitly Referenced Library Subprograms
External References for Library Subprgrms
Buffers for Direct Access Data Sets[2]
Transient Work Area Required of Every Load Module for Use by the Control Program
Input/Output Blocks (IOBs) Containing Information Concerning the Interface Between FORTRAN Execution Time I/O Routines and the Control Program[3]
Buffers for Sequential Data Sets
Control Program Routines for Performing Direct Access I/O and Control Program Routines for Sequential I/O[4]
Data Control Blocks (DCBs) Containing Information Concerning the Use of Each Data Set[5]
Task Input/Output Table (TIOT) Containing Information such as jobname, stepname, and ddname for each data set used by the step
Register Save Area for the Control Program

Figure 75. Load Module Execution Storage Map

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

Four informative messages are generated by the compiler to inform the programmer or operator of the status of the compilation. The message and any compiler action taken is shown.

```
LEVEL: rmthyr                IBM OS/360 BASIC
FORTRAN IV (E) COMPILATION      DATE: yy.ddd
```

Explanation: This message is generated at the beginning of every compilation. The level number (r) and date (mthyr) of the compiler is given by "rmthyr". The number of the day (ddd) in the year (yy) that the compilation takes place is given by "yy.ddd".

```
IEJ001I  COMPILER OPTIONS IN EFFECT:
         [SOURCE,][BCD,][MAP,][DECK,]
                            (SPACE,)
         [LOAD,][ADJUST,]   (PRFRM,)
         [NAME=xxxxxx,]SIZE=yyyyyyy,
         LINELNG=zzz
```

Explanation: This message occurs for every compiler job step. All bracketed options appear, if they are specified or assumed by default.

SIZE OF COMMON xxxxx PROGRAM yyyyy

Explanation: This message is generated before the end of every compilation. The number of bytes needed to contain the COMMON area is the decimal number xxxxx. The number of bytes needed to contain the program (instructions generated by the compiler, constants, and variables not assigned to COMMON, etc.) is the decimal number yyyyy.

END OF COMPILATION

Explanation: This message indicates that a compilation is successfully completed.

Compiler Action: If this message is not generated by the compiler, the compilation was terminated abnormally and a condition code of 16 was generated because of internal errors.

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. Unless otherwise specified, messages preceded by "WARNING" generate a condition code of 4; other messages generate a condition code of 8.

IEJ002I ONE OR MORE INVALID COMPILER OPTIONS IN 'PARM' FIELD OPTION(S) IGNORED

Explanation: An invalid compiler option is specified in the PARM field of the EXEC statement.

Compiler Action: The compilation proceeds using only the specified valid compiler options and defaulted options.

IEJ003I 'NAME' OPTION TOO LONG - TRUNCATED

Explanation: The name specified for the compiler option NAME=xxxxxx is longer than six characters.

Compiler Action: The characters beyond the sixth position are truncated and the compiler proceeds as if the truncated name had been specified.

IEJ004I MISSING OR ERRONEOUS DD STATEMENT

$$\text{FOR} \left\{\begin{array}{l}\text{SYSIN}\\ \text{SYSUT1}\\ \text{SYSUT2}\\ \text{SYSLIN}\\ \text{SYSPUNCH}\\ \text{xxxxxxxx}\end{array}\right\}$$

Explanation: A DD statement is not supplied or the ddname is mispunched for the DD statement indicated in the message. If another program passes control to the compiler, then a DD statement xxxxxxxx passed as a parameter is missing.

Compiler Action: A condition code of 16 is generated for the omission of SYSIN. The compilation is terminated. The compilation proceeds if any other ddnames are omitted. If SYSLIN is omitted, a condition code of 12 is generated and the compiler assumes NOLOAD, even though the programmer may have specified LOAD. For the omission of SYSPUNCH, the compiler assumes NODECK, even though DECK may have been specified.

IEJ005I NO INPUT FOUND

Explanation: A source module is nonexistent; that is a DD * statement defining the location of the source module for the compiler is immediately followed by a delimiter statement.

Compiler Action: A condition code of 16 is generated, and control is returned to the control program.

IEJ006I INSUFFICIENT STORAGE. SPACE OPTION IN EFFECT

Explanation: The PRFRM option is specified; however, there is not enough main storage available for the PRFRM option.

Compiler Action: The SPACE compiler option is assumed, and the compiler begins the compilation again.

IEJ007I INSUFFICIENT STORAGE FOR COMPILATION

Explanation: Blocked compiler I/O is specified with the PRFRM option; however, there is not enough main storage available for the PRFRM option.

Compiler Action: The compilation is terminated, and a condition code of 16 is issued.

IEJ008I INVALID BLKSIZE SPECIFICATION

Explanation: The BLKSIZE subparameter specified in a DD statement is not an integral multiple of the specified LRECL value for that data set.

Compiler Action: The compilation is terminated, and a condition code of 16 is issued.

IEJ010I I/O ERROR, COMPILATION TERMINATED, xxx...xxx

Explanation: An irrecoverable input/output error was encountered during compilation, which makes continuation impossible. xxx...xxx is the character string formatted by the SYNADAF macro instruction. For an interpretation of this information, see IBM System/360 Operating System: Supervisor and Data Management Macro-Instructions, Form C28-6647.

Compiler Action: Compilation is terminated, and a condition code 16 is generated.

IEJ011I I/O ERROR, 'DECK' CANCELED, xxx...xxx

Explanation: An irrecoverable error was encountered during an output operation on the data set defined by SYSPUNCH. xxx...xxx is the character string formatted by the SYNADAF macro instruction. For an interpretation of this information, see IBM System/360 Operating System: Supervisor and Data Management Macro-Instructions, Form C28-6647.

Compiler Action: The DECK option is changed to NODECK, and compilation continues. In the case of multiple compilations, only the last (partial) deck punch need be discarded.

IEJ029I ARRAY MUST BE DIMENSIONED ON ITS FIRST AND ONLY ITS FIRST OCCURRENCE

Explanation: The dimension of an array must be given in either a DIMENSION, COMMON, or Explicit Specification statement prior to its use in any other statement and can never be redefined.

IEJ030I ILLEGAL USE OF FUNCTION NAME

Explanation: A function name may not appear in an EQUIVALENCE or COMMON statement.

IEJ031I EQUIVALENCE TABLE FULL

Explanation: There are too many equated variables.

User Response: See "Compiler Restrictions."

IEJ032I INTEGER TOO BIG

Explanation: Integer is larger than maximum size allowable (i.e., larger than $2^{31}-1$ or 2,147,483,647), or the number of records specified in a DEFINE FILE statement exceeds 2^{24} (16,777,116).

IEJ033I WARNING -- FIRST CARD IS CONTINUATION

Explanation: First non-comment statement was a continuation line (i.e., a nonzero character, other than a blank, was encountered in column 6.)

Compiler Action: The statement is processed as if it were the initial line of a statement.

IEJ034I SUBPROGRAM CARD NOT FIRST

Explanation: A FUNCTION or SUBROUTINE statement appears after the first statement in a program. For example, the first card in a subprogram (other than a comments card) is not a FUNCTION or SUBROUTINE statement.

IEJ035I ARGUMENT MISSING IN FUNCTION DEFINITION

Explanation: Function definition (either in Statement Function or FUNCTION header statement) must have at least one argument.

IEJ036I ILLEGAL CHARACTER

Explanation: Character is not recognizable.

Compiler Action: The character is taken to be a delimiter, which should be either
b - * . , + / = ( )
or Column 73, where b is a blank.

IEJ037I INVALID STATEMENT OR STATEMENT NUMBER

Explanation: For example, an equal sign is missing in a Statement Function Definition or an arithmetic statement. A left parenthesis is missing in an IF statement or an illegal delimiter precedes the statement.

IEJ038I SEQUENCE ERROR

Explanation: All specification statements (DIMENSION, EQUIVALENCE, REAL, INTEGER, DOUBLE PRECISION, COMMON) must precede all Statement Function Definition statements. All Statement Function Definition statements must precede all executable statements.

IEJ039I MORE THAN 6 CHARACTERS IN NAME

IEJ041I MULTI-DEFINED OR ILLEGAL NAME

Explanation: A name is defined more than once or an illegal name is used as a variable. For example, a real variable is redefined as an integer variable.

IEJ042I MULTI-DEFINED STATEMENT NUMBER

Explanation: This statement number has been used previously. Every statement number should be unique, and associated with only one statement in a program.

IEJ043I ILLEGAL USE OF RESERVED WORDS

Explanation: When NOADJUST is specified, a reserved word must not be used as a variable, array, or subprogram name.

IEJ044I TOO MANY DECIMAL POINTS

Explanation: Only one decimal point can appear in a real or double-precision number.

IEJ045I DECIMAL POINT AFTER E

Explanation: A decimal point has been found in the exponent part of a real or double-precision number.

IEJ046I TOO MANY E'S

Explanation: A second E has been found in a number (e.g., 2,7E2E2).

IEJ047I ILLEGAL NUMBER OR NAME

Explanation: Illegal use of a number. For example, in the statement, DIMENSION 5 (1,2), the number 5 is not a proper array name.

IEJ048I MORE THAN 3 DIMENSIONS

Explanation: Maximum number of dimensions permitted in an array is three.

IEJ049I DIMENSION ERROR

Explanation: Illegal delimiter or size of an array exceeds 131,071 bytes in a COMMON, DIMENSION, or Explicit Specification statement.

IEJ050I CANNOT EQUATE

Explanation: At least two variables or subscripted variables should appear in the parentheses of an EQUIVALENCE statement.

IEJ051I WARNING -- COMMA MISSING

Explanation: A required comma was not encountered.

Compiler Action: The statement is compiled as through a comma were there.

IEJ052I WRONG DIMENSION

Explanation: Number of subscripts in the variable used does not correspond to the number of subscripts in the array as defined in a COMMON, DIMENSION, or Explicit Specification statement.

IEJ053I SUBSCRIPT ERROR

Explanation: The subscript expression contains more than three subscripts, an illegal delimiter, or an illegal variable.

IEJ054I INVALID ARGUMENT IN ASF

Explanation: An illegal symbol appears as an argument in the Statement Function argument list. For example, SF(A,B,*,C) or SF(A,B,C) where C is an array.

IEJ055I INVALID ARGUMENT IN HEADER CARD

Explanation: An illegal variable or a multidefined variable appears in the function definition argument list.

IEJ056I ILLEGAL STATEMENT NUMBER FIELD

Explanation: Statement number list in a computed GO TO or in an arithmetic IF statement is invalid.

IEJ057I DATA SET REFERENCE NUMBER MISSING

Explanation: There is no data set reference number specified, for example, WRITE(,10), or the data set reference number is multiply defined.

IEJ058I LEFT PARENTHESIS MISSING AFTER R/W

Explanation: The left parenthesis in a READ or WRITE statement is missing. For example, in the statement: WRITE3,10), the left parenthesis before the 3 is missing.

IEJ060I ERROR IN VARIABLE

Explanation: Symbol in an EQUIVALENCE statement is not a variable, for example, EQUIVALENCE (10,B), or is a dummy variable.

IEJ061I WARNING -- STATEMENT CANNOT BE REACHED

Explanation: Statement following a GO TO, IF, RETURN, or STOP has no statement number.

IEJ063I EQUIVALENCE SUBSCRIPT ERROR

Explanation: There is an illegal delimiter or a missing subscript in an EQUIVALENCE subscript.

IEJ064I TOO MANY SYMBOLS AND STATEMENT NUMBERS

Explanation: The Dictionary and Overflow Table have overflowed, or the total number of statement numbers referred to (excluding FORMAT statement numbers), DO statements, and statement functions exceeds 1000.

User Response: Subdividing the program or reducing the number of symbols and statement numbers is necessary.

IEJ065I INVALID STATEMENT NUMBER OR PAUSE/STOP NUMBER

Explanation: Either there is an alphabetic or illegal character in the number, or there are more than five digits in the number.

IEJ066I BACKWARD DO LOOP

Explanation: The statement specified in the range of the DO statement may not precede the DO statement.

IEJ067I INVALID DATA SET CONTROL CHARACTER

Explanation: The FORMAT control specification in the DEFINE FILE statement is not L, E, or U.

IEJ068I ERROR IN EXPONENT

Explanation: An exponent is missing or is too large in a real or double-precision number.

IEJ069I TOO MANY ARGUMENTS IN ASF

Explanation: More than 15 arguments in Statement Function definition is not permitted.

IEJ070I INVALID FUNCTION NAME

Explanation: Invalid subprogram name in a FUNCTION or SUBROUTINE header statement.

IEJ071I ILLEGAL SUBROUTINE NAME

Explanation: Illegal delimiter or illegal subroutine name in a CALL statement.

IEJ072I ASF OUT OF SEQUENCE

Explanation: Statement Function statement is out of sequence or an array is not dimensioned prior to its first use.

IEJ073I TRANSFER TO NON-EXECUTABLE STATEMENT

Explanation: The statement number referred to by a GO TO, computed GO TO, or an arithmetic IF statement is a FORMAT or specification statement.

IEJ074I VARIABLE ALREADY IN COMMON

Explanation: A variable appears in COMMON more than once or an inconsistent equate was made (e.g., the statement COMMON (A,B,C,A) is illegal).

IEJ075I UNFINISHED STATEMENT

Explanation: A FORMAT statement is not finished.

IEJ076I PARENTHESIS ERROR

Explanation: A parenthesis is not closed or is missing. The parentheses are not nested properly.

Compiler Action: The compiler cannot assume their position.

IEJ077I ILLEGAL DELIMITER OR MISSING NAME

Explanation: An improper delimiter or illegal special character was encountered.

IEJ078I ILLEGAL END DO

Explanation: The last statement in the range of a DO loop cannot be a nonexecutable statement, Arithmetic IF, GO TO, PAUSE, RETURN, STOP, or another DO statement.

IEJ079I TYPE MUST BE INTEGER SCALAR

Explanation: The DO variable, computed GO TO variable, or the associated variable in a DEFINE FILE statement must be a non-subscripted integer variable.

IEJ080I COMMA MISSING

Explanation: A required comma was not encountered.

Compiler Action: The compiler cannot assume its position.

IEJ081I ILLEGAL FORMAT SPECIFICATION

Explanation: Illegal decimal point or a number is missing following decimal point.

IEJ082I INVALID NUMBER

Explanation: There is an error in an integer, real, or double-precision number.

IEJ083I ERROR IN INTEGER

Explanation: Number zero not allowed in most FORMAT specifications, a DIMENSION statement, or in a subscript.

IEJ084I MORE THAN 4 WARNINGS IN STATEMENT

Explanation: More than four warnings have been generated for a statement.

Compiler Action: The compilation of the statement is terminated.

IEJ085I THIS MESSAGE IS A COMPILER ERROR

Explanation: Compilers working text contains meaningless code.

Compiler Action: Compilation continues.

IEJ086I ILLEGAL BLANK

Explanation: When NOADJUST is specified, an illegal embedded blank is found in the FORMAT statement.

IEJ087I NUMBER MISSING

Explanation: A number is missing in E, F, T, A, I, D, or X conversion code or an illegal delimiter precedes the number.

IEJ088I NESTED PARENTHESIS

Explanation: Not more than one level of nested parenthesis is permitted in a FORMAT statement.

IEJ089I ILLEGAL DATA SET REFERENCE NUMBER

Explanation: Data set reference number must be an integer variable or a constant within the range 1 to 99.

IEJ090I APOSTROPHE NOT CLOSED

Explanation: An apostrophe was not found terminating the literal data in a FORMAT statement.

IEJ091I ILLEGAL SIGN

Explanation: A P Format code or a blank are the only legal delimiters following a plus or minus sign in a FORMAT statement, unless the sign appears in literal data.

IEJ092I ILLEGAL COMMA

Explanation: An erroneous comma appears in a FORMAT statement.

IEJ093I NOT IN INTEGER MODE

Explanation: The associated variable indicating the relative position of a record in a direct access FIND, READ, or WRITE statement is not an integer variable.

IEJ094I WARNING -- TOO MANY DECIMAL PLACES

Explanation: The number of decimal places must be less than the size of the entire number in a FORMAT statement. The size of the entire number is equal to the number of decimal places.

IEJ095I STATEMENT NUMBER REFERENCE NOT FORMAT STATEMENT

Explanation: The statement number referred to in a READ/WRITE statement is not that of a FORMAT statement.

IEJ096I ILLEGAL VARIABLE IN I/O LIST

Explanation: The use of subprogram names or constants are not allowed in an I/O list.

IEJ097I TOO MANY ELEMENTS IN LIST

Explanation: The list in the READ, WRITE, or Computed GO TO statement contains too many elements. There are approximately 250 variables permitted in a single list. The use of implied DO's in READ/WRITE statement decreases the number of variables permitted.

User Response: The statement should be divided into several statements.

IEJ098I NO CHARACTER BETWEEN APOSTROPHES

Explanation: An open apostrophe is immediately followed by a close apostrophe in the FORMAT statement. At least one character should appear within the apostrophes.

IEJ099I TOO MANY CHARACTERS BETWEEN APOSTROPHES

Explanation: The number of characters appearing within apostrophe in the FORMAT statement is too large for the compiler to handle. That is, not more than 255 characters should appear between apostrophes in a FORMAT statement.

IEJ100I ILLEGAL DO VARIABLE OR CONSTANT

Explanation: DO parameter must be a nonsubscripted integer variable, or integer constant.

IEJ123I FUNCTION NAME NOT ASSIGNED A VALUE

Explanation: The function name is not defined in its function subprogram (i.e., it does not appear on the left side of an equal sign, as a list item in a READ statement or as an argument in a CALL statement).

IEJ124I NOT IN INTEGER MODE

Explanation: The associated variable indicating the relative position of a record in a direct access FIND, READ, or WRITE statement is not an integer variable.

IEJ125I DO VARIABLE REDEFINED

Explanation: The same variable is used as the DO variable more than once in nested DO loops.

IEJ126I FUNCTION ARGUMENT MISSING

Explanation: An argument in a function reference is missing.

IEJ127I THIS MESSAGE IS A COMPILER ERROR

Explanation: Compiler's intermediate text contains meaningless code.

Compiler Action: Compilation is terminated.

IEJ128I INVALID CALL OR IF STATEMENT

IEJ129I MULTI-DEFINED NAME OR KEYWORD

Explanation: A name is redefined, or a keyword is used as a variable.

IEJ130I ILLEGAL ARGUMENT

Explanation: An illegal argument is specified for a call to a function or subroutine.

IEJ131I WRONG MODE

Explanation: The mode of the argument does not agree with the mode of the in-line function.

IEJ132I INCORRECT NESTING OF DO

Explanation: The last statement in the range of the DO loop nested within other DO loops exceeds the range of one or more of those DO loops.

IEJ133I ILLEGAL EQUAL SIGN

Explanation: Two equal signs appear in the same statement.

IEJ135I SUBSCRIPT OR ARRAY NOT ALLOWED

Explanation: A subscript or array is not allowed in Statement Function definition.

IEJ136I UNDEFINED STATEMENT NUMBER

Explanation: The statement number referred to does not exist in the program.

IEJ137I NAME MISSING OR ILLEGAL DELIMITER

Explanation: Illegal delimiter found. For example, X=A+*B. A variable or constant is missing between the two operators or one of the operators is superfluous.

IEJ139I WRONG NUMBER OF ARGUMENTS IN CALL

Explanation: The number of arguments in Statement Function reference or in an in-line function reference does not agree with the function definition.

IEJ140I TOO MANY PARAMETERS

Explanation: The maximum number of arguments allowed in a subprogram call or definition is 48.

IEJ141I ILLEGAL SUBPROGRAM NAME

Explanation: Name of a function or subroutine call is not defined as a function or subroutine subprogram name.

IEJ142I MORE THAN 25 LEVELS OF DO NESTING

Explanation: Not more than 25 DO loops may be nested.

IEJ143I INVALID RESULT FIELD

Explanation: The result field of an arithmetic statement is invalid.

IEJ144I ILLEGAL NUMBER OF STATEMENT NUMBERS

Explanation: An arithmetic IF statement must contain exactly three statement numbers.

IEJ145I PROGRAM TOO BIG

Explanation: The object module has exceeded the basic register range.

IEJ146I INCONSISTENT EQUATE

Explanation: For example, EQUIVALENCE (A(1),B),(A(2),C),(B,C) or a double-precision variable in COMMON is not on the proper boundary.

IEJ147I TWO VARIABLES IN COMMON ARE EQUATED

Explanation: Two or more variables equivalenced are in COMMON.

IEJ148I COMMON EXTENDED UPWARD

Explanation: An EQUIVALENCE statement cannot cause COMMON to be extended before the beginning of the COMMON area.

IEJ149I DUMMY ARRAY OR VARIABLE IN COMMON

Explanation: A dummy variable or array is not permitted in COMMON.

IEJ150I EQUATED NAME NOT A VARIABLE

Explanation: The equated name must be a variable.

IEJ158I WARNING--POSSIBLE MISSING DEFINE FILE STATEMENT

Explanation: There is no DEFINE FILE statement for a data set reference number specified in a direct access FIND, READ, or WRITE statement.

IEJ159I WARNING--LAST EXECUTABLE STATEMENT NOT RETURN, STOP, IF OR GO TO

Explanation: The last executable statement of a program should be a RETURN, STOP, IF, or an unconditional GO TO statement.

Compiler action: The compiler generates a RETURN before the END statement.

IEJ160I WARNING--STATEMENT CONTAINS SUPERFLUOUS INFORMATION

Explanation: The statement has been compiled but something superfluous exists at the end, e.g.,
REWIND I XYZ
XYZ is superfluous

IEJ161I WARNING--SUGGEST SUBDIVIDING PROGRAM

Explanation: Program causes use of spill base register producing inefficient object code

User Response: Subdivide program into a main program and a series of subprograms.

IEJ162I WARNING--BLANK CARD

Explanation: The card contains only a statement number.

Compiler Action: The card is ignored.

IEJ163I WARNING--TOO MANY DIGITS IN NUMBER

IEJ164I WARNING--STATEMENT NUMBER MISSING

Explanation: Format statement must have a statement number.

Compiler Action: The statement is ignored.

IEJ165I WARNING--UNREFERENCED FORMAT STATEMENT

Explanation: A FORMAT statement is not referred to by any other statement.

Compiler Action: The FORMAT statement is not processed.

IEJ166I WARNING--REDUNDANT COMMA

Explanation: There is a redundant comma in the statement.

Compiler Action: The redundant comma is ignored.

IEJ167I WARNING--LINE TOO LONG

Explanation: Record length indicated in the FORMAT statement exceeds length stated or assumed for the compiler option LINELNG.

IEJ168I WARNING--END CARD MISSING

Explanation: The end of the source program is reached and an END card is not there.

Compiler Action: Processing continues as if it were there.

IEJ169I WARNING--RIGHT PARENTHESIS MISSING

Explanation: The right parenthesis in the statement is missing.

Compiler Action: Processing continues as if it were there.

IEJ170I WARNING--ZERO OR NO COUNT IN X CONVERSION

Explanation: The number preceding the X format code is 0 or blank.

Compiler Action: Processing continues, ignoring the X format code.

IEJ171I WARNING--PARAMETERS MISSING

Explanation: There are no parameters following a left parenthesis or a comma.

IEJ172I WARNING--UNREFERENCED ASF ARGUMENT

Explanation: Argument of statement function not referred to in the arithmetic expression of the statement function.

IEJ173I WARNING--EXCESSIVE RIGHT PARENTHESIS

Compiler Action: The additional right parentheses in the statement are ignored and processing continues.

IEJ174I WARNING--ARRAY
USED AS SCALAR

Explanation: The name of the array is not followed by a subscript enclosed in parentheses.

IEJ175I WARNING--STATEMENT NUMBER ON DECLARATIVE STATEMENT

Explanation: The statement number associated with the declarative statement is superfluous.

LOAD MODULE EXECUTION DIAGNOSTIC MESSAGES

The load module produces three types of diagnostic messages:

- Operator messages.
- Execution error messages.
- Program interrupt messages.

Operator Messages

Operator messages for STOP and PAUSE are generated by FORTRAN load modules.

The message for a PAUSE is of the form

yy IHC001A PAUSE xxxxx

where:
yy is the identification number and
xxxxx is the number specified in the PAUSE source statement.

Explanation: A PAUSE is executed. The programmer should give instructions that indicate the action to be taken by the operator when the PAUSE is encountered.

User Response: To resume execution, the operator presses the REQUEST key. When the PROCEED light comes on, the operator types

REPLY yy,'Z'

where:
yy is the identification number and Z is any letter or number. To resume program execution the operator must press the alternate coding key and a numeric 5.

The message for a STOP statement is of the form

IHC002I STOP xxxxx

where:
xxxxx is the number specified in the STOP source statement.

User Response: None

Program Interrupt Messages

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

IHC210I PROGRAM INTERRUPT (P) - OLD PSW IS

$$\text{xxxxxxx}\begin{Bmatrix}9\\C\\D\\F\end{Bmatrix}\text{xxxxxxxx}$$

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 four characters in the PSW (i.e., 9, C, D, or F) represent the code number (in hexadecimal) associated with the type of interruption. The following text describes these interruptions.

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 would 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 would occur during execution of the statement

```
A = 1.0E+75 + 7.2E+75
```

When the interrupt occurs, the result register contains a floating-point number whose fraction and sign is correct. However, the number is not usable for further computation since its characteristic field no longer reflects the true exponent. The content of the result register as it existed when the interrupt occurred is printed following the program interrupt message with the format:

REGISTER CONTAINED hhhhhhhhhhhhhhhh

where:
hhhhhhhhhhhhhhhh is the floating-point number in hexadecimal notation.

Exponent overflow causes "exponent wraparound" -- i.e., the characteristic field represents an exponent that is 128 smaller than the correct one. Treating bits 1 to 7 (the exponent characteristic field) of the floating-point number as a binary integer, the true exponent (TE) may be computed, as follows:

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

Before program execution continues, the FORTRAN library sets the result register to the largest possible floating-point number that can be represented in short precision ($16^{63}*(1-16^{-6})$) or in long precision ($16^{63}*(1-16^{-14})$), but the sign of the result is not changed. The condition code is not altered.

Exponent-Underflow Exception: The exponent-underflow exception, assigned code number D, is recognized when the result of a floating-point addition, subtraction, multiplication, or division is less than 16^{-65} (approximately $5.4x10^{-79}$). An exponent-underflow exception would occur during execution of the statement:

```
A = 1.0E-50 * 1.0E-50
```

Although exponent underflows can be masked, 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 notations.

Exponent underflow causes "exponent wraparound" - i.e., the characteristic field represents an exponent that is 128 larger than the correct one. Treating bits 1 to 7 (the exponent characteristic field) of the floating-point number as a binary integer, the true exponent (TE) may be computed as follows:

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

Before program execution continues, the FORTRAN library sets the result register to a true zero of correct precision. If the interrupt resulted from a floating-point addition or subtraction operation, the condition code is set to zero to reflect the setting of the result register.

Note: The System/360 Operating System FORTRAN programmer who wishes to take advantage of the "exponent wraparound" feature and handle the interrupt in his own program must call an assembly language subroutine to issue a SPIE macro instruction which will override the FORTRAN interruption routine.

Floating-Point-Divide Exception: The floating-point-divide exception, assigned code number F, is recognized when division of a floating-point number by zero is attempted. A floating-point divide exception would occur during execution of the following statements:

```
B=0.0
A=1.0
C=A/B
```

Execution Error Messages

In the following text, the error codes are given with an explanation describing the type of error. Preceding the explanation, an abbreviated name is given indicating the origin of the error. Unless otherwise specified, a condition code of 16 is generated and the job step is terminated.

The abbreviated name for the origin of the error is:

IBC - IHCFCOME, IHCFIOSE, and IHCDIOSE routines (perform input/output conversions for FORTRAN load module execution and act as an interface between FORTRAN I/O statements and the control program).

LIB - SYS1.FORTLIB. In the explanation of the messages, the module name is given followed by the entry point name(s) enclosed in parentheses.

IBERR - IHCIBERR Routine (detects error conditions that arise because a load module is executed that has FORTRAN language errors indicated in diagnostic messages given when the source module was compiled).

LIB - SYS1.FORTLIB. In the explanation of the messages, the module name is given followed by the entry point name(s) enclosed in parentheses.

IBERR - IHCIBERR Routine (detects error conditions that arise because a load module is executed that has FORTRAN language errors indicated in diagnostic messages given when the source module was compiled).

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, under FORMAT control, a record that exceeds the BLKSIZE value.

IHC213I Explanation: IBC -- The input list in an input/output statement without a FORMAT specification is larger than the logical record.

IHC214I Explanation: FIOCS -- For records in sequential data sets written without FORMAT control, for which the RECFM subparameter must be V (variable), either U (undefined) or F (fixed) was specified.

IHC215I Explanation: IBC -- 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. xxx...xxx is the character string formatted by the SYNADAF macro instruction. For an interpretation of this information, see IBM System/360 Operating System Supervisor and Data Management Macro-Instructions, Form C28-6647.

IHC219I Explanation: FIOCS -- A data set is referred to in the load module, but no DD statement is supplied for it, or a DD statement has an erroneous ddname.

IHC220I Explanation: FIOCS -- A data set reference number exceeds the limit specified for data set reference numbers when this operating system was generated.

IHC230I SOURCE ERROR AT ISN xxxx - EXECUTION FAILED

Explanation: IBERR -- During load module execution, a source statement error is encountered. The internal statement number for the source statement is xxxx.

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 record read did not correspond to length of record specified in the 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=0 and J≤0 is an error.

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 variable or constant, and J represents an integer variable or constant, R=0 and J≤0 is an error.

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 double precision variable and J represents an integer variable or constant, D=0 and J≤0 is an error.

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 are real variables or real constants, R=0 and S≤0 is and error.

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 are double precision variables or double precision constants, D=0 and P≤0 is an error.

IHC251I SQRT NEGATIVE ARGUMENT=X.X

Explanation: LIB -- In the subprogram IHCSSQRT(SQRT), an argument less than 0 is an error.

IHC252I EXP ARG=X.X, GT 174.673

Explanation: LIB -- In the subprogram IHCSEXP(EXP), an argument greater than 174.673 is an error.

IHC253I ALOG-ALOG10 ARG=X.X LE ZERO

Explanation: LIB -- In the subprogram IHCSLOG(ALOG and ALOG10), an argument less than or equal to zero is an error. 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 greater than or equal to $2^{18} \cdot \pi$ is an error.
($2^{18} \cdot \pi$ =.82354966406249996D+06)

IHC261I DSQRT NEGATIVE ARGUMENT=X.X

Explanation: LIB -- In the subprogram IHCLSQRT(DSQRT), an argument less than 0 is an error.

IHC262I DEXP ARG=X.X, GT 174.673

Explanation: LIB -- In the subprogram IHCLEXP(DEXP), an argument greater than 174.673 is an error.

IHC263I DLOG-DLOG10 ARG=X.X, LE ZERO

Explanation: LIB -- In the subprogram IHCLLOG(DLOG and DLOG10), an argument less than or equal to zero is an error. Because the subprogram is called by an exponential subprogram, this message also indicates that an attempt has been made to raise a negative double precision base to a 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 an argument greater than or equal to $2^{50} \cdot \pi$ is an error.
(2^{50} π =.35371188737802239D+16)

C28-6603-2

Printed in U.S.A. C28-6603-2

IBM®

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]

IBM Technical Newsletter

File Number	S360-25
Re: Form No.	C28-6603-2
This Newsletter No.	N28-0580
Date	May 1, 1968
Previous Newsletter Nos.	N28-0578

IBM System/360 Operating System FORTRAN IV (E) Programmer's Guide

This Technical Newsletter amends the Systems Reference Library publication IBM System/360 Operating System: FORTRAN IV (E) Programmer's Guide, Form C28-6603-2.

In the referenced publication, replace the pages listed below with the corresponding pages attached to this newsletter:

Pages to be Removed	Pages to be Inserted
37,38	37,38,38.1
49 through 52	49 through 52
55,56	55,56,56.1
73 through 76	73 through 76
81 through 84	81 through 84
116.1,117,118	117,118

Changes to text are indicated by a vertical line to the left of the change; changes to illustrations are shown by the symbol (•) to the left of the caption.

Summary of Amendment

The changes in this newsletter correspond to Release 16 of the IBM System/360 Operating System.

This newsletter explains how the FORTRAN IV (E) user might possibly make use of the optional Extended Error Message facility. To do so, he must select the facility at system generation time and use the FORTRAN G or H library. In addition, this newsletter lists the text of messages produced when the mathematical library detects certain error conditions. There are also miscellaneous clarifications and corrections.

File this cover letter at the back of the publication. It will then serve as a record of the changes received and incorporated.

IBM Corporation, Programming Systems Publications, 1271 Avenue of the Americas, New York, N.Y. 10020

PRINTED IN U.S.A.

F6 MAR 1969

IBM / **Technical Newsletter**

File Number	S360-25
Re: Form No.	C28-6603-2
This Newsletter No.	N28-0586
Date	November 15, 1968
Previous Newsletter Nos.	N28-0578 (applies to -2) N28-0580 (applies to -2)

IBM System/360 Operating System Basic FORTRAN IV (E) Programmer's Guide

This Technical Newsletter, a part of Release 17 of the IBM System/360 Operating System, provides replacement pages for the publication IBM System/360 Operating System: FORTRAN IV (E) Programmer's Guide, Form C28-6603-2. These replacement pages remain in effect for subsequent releases unless specifically altered. Pages to be inserted and/or removed are listed below.

Pages

Cover, preface
11-16.1
19-22
25-32 (31.1 added)
37-38.1
47-56.1
57-58
61-62.1
75-76.1
77-78.2

Changes to the text, and small changes to illustrations, are indicated by a vertical line to the left of the change; changed or added illustrations are denoted by the symbol • to the left of the caption.

Summary of Amendments

This newsletter contains information describing two newly supported FORTRAN features: logical backspace and partitioned data set processing. New sections have also been added to explain how to use the comment statement, how to increase processing efficiency, and how to avoid operator intervention for file-protected tape volumes. Additional information relating to the DISP parameter of the DD statement is also included, and miscellaneous clarifications and corrections have been made throughout the publication.

File this cover letter at the back of the publication to provide a record of changes.

IBM Corporation, Programming Publications, 1271 Avenue of the Americas, New York, N.Y. 10020

PRINTED IN U.S.A.