SC28-6456-0

Program Product

IBM OS Full American National Standard COBOL Compiler and Library, Version 4, Programmer's Guide

Program Numbers: 5734-CB2
5734-LM2

This publication describes how to compile an American National Standard COBOL X3.23-1968 program using Version 4 of the IBM Operating System Full American National Standard COBOL compiler. It also discusses how to link edit and execute or load the program under control of the IBM Operating System. There is a description of the output of each of these steps, i.e., compile, link edit, load, and execute. In addition, there is an explanation of the features of the compiler and available options of the operating system.

IBM

First Edition (May 1972)

This edition corresponds to Version 4 of the IBM OS Full American National Standard COBOL Compiler.

Changes are periodically made to the information herein; any such changes will be reported in subsequent revisions or Technical Newsletters. Before using this publication in connection with the operation of IBM systems, refer to the latest SRL Newsletter, Order No. GN20-0360, for editions that are applicable and current.

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

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Programming Publications, 1271 Avenue of the Americas, New York, New York 10020.

The purpose of this publication is to enable programmers to compile, link-edit, and execute, or compile and load Full American National Standard COBOL Compiler and Library, Version 4, programs under control of the IBM Operating System. The COBOL language is described in the publication IBM OS Full American National Standard COBOL, Order No. GC28-6396, which is a corequisite to this publication.

Programmers who are familiar with the operating system and wish to know how to run COBOL programs should read "Job Control Statements" and "Data Set Requirements" under "Job Control Procedures," and "Output." These chapters provide information about the preparation of COBOL programs for processing by the operating system.

Programmers who are unfamiliar with the concepts of OS should read "Introduction," "Job Control Procedures," "Checklist for Job Control Procedures," and "Using Cataloged Procedures" in addition to the sections listed above.

The chapters "Program Checkout" and "Programming Techniques" are of special interest, since they contain information about debugging and efficient programming. Other chapters discuss optional features of the language and the operating system. Some chapters include introductory information about features of the operating system that are described in detail in other publications.

The chief features available with this compiler are Optimized Object Code, COBOL Teleprocessing, and Advanced Symbolic Debugging capabilities. With the Teleprocessing Feature, the user can write device-independent message-processing programs using COBOL language statements to send and receive messages over a communications network. The Optimized Object Code Feature allows for a considerable reduction in object-time code. Advanced Symbolic Debugging -- incorporating symbolic dumping capabilities, a flow trace of a user-specified number of procedures, and the number of the source statement causing abnormal termination -- can result in a marked saving of debugging time.

Additional features of this compiler provide for a syntax-only compilation, significantly reducing compilation time; the sharing of reentrant COBOL object-time library subroutines by multiple regions/partitions; dynamic invocation and release of user subprograms; and manipulation of data to separate contiguous data into multiple logical subfields or to concatenate two or more subfields into a single field.

The machine configuration required for system operations is described in the chapter "Machine Considerations."

Wider and more detailed discussions of the operating system are given in the following publications:

IBM OS Job Control Language Reference, Order No. GC28-6704

IBM OS Job Control Language Charts, Order No. GC28-6632

IBM OS System Programmer's Guide, Order No. GC28-6550

IBM OS Linkage Editor and Loader, Order No. GC28-6538

IBM OS Supervisor Services, Order No. GC28-6646

IBM OS Data Management Services, Order No. GC26-3746

IBM OS Supervisor and Data Management Macro Instructions, Order No. GC28-6647

IBM OS Sort/Merge, Order No. GC28-6543

IBM OS Sort/Merge Programmer's Guide, Order No. SC33-4007

IBM OS Utilities, Order No. GC28-6586

IBM OS System Generation, Order No. GC28-6554

IBM OS Programmer's Guide to Debugging, Order No. GC28-6670

IBM OS Storage Estimates, Order No. GC28-6551

IBM OS Messages and Codes, Order No. GC28-6631

Diagnostic messages, together with their problem determination documentation can be found in the following publication:

IBM OS Full American National Standard COBOL, Version 4 Messages, Order No. SC28-6457

Information on installing the compiler and using it under the Time Sharing Option (TSO) of the IBM Operating System can be found in the following Program Product publications:

IBM OS Full American National Standard COBOL Compiler and Library, Version 4, Installation Reference Material, Order No. SC28-6458

IBM OS (TSO): COBOL Prompter Installation Reference Material, Order No. SC28-6434

The COBOL teleprocessing user must write a message control program (MCP) to handle messages transmitted between remote stations and the central computer before they can be processed by a COBOL program. General telecommunications access method (TCAM) information, as well as specific guidelines for creating an MCP, can be found in the following publications:

IBM OS Telecommunications Access Method (TCAM) Concepts and Facilities, Order No. GC30-2022.

IBM OS Telecommunications Access Method (TCAM) Programmer's Guide and Reference Material, Order No. GC30-2024.

An American National Standard COBOL program can be processed by the IBM Operating System. The operating system consists of a number of processing programs and a control program.

The processing programs include the COBOL compiler, service programs, and any user-written programs.

The control program supervises the execution or loading of the processing programs; controls the location, storage, and retrieval of data; and schedules jobs for continuous processing.

A request to the operating system for facilities and scheduling of program execution is called a job. For example, a job could consist of compiling a program by utilizing the COBOL compiler. A job consists of one or more job steps, each of which specifies execution of a program. The programmer can make requests to the operating system by using job control statements.

Each job is headed by a JOB statement that identifies the job. Each job step is headed by an EXEC statement that describes the job step and calls for execution. Included in each job step are data definition (DD) statements, which describe data sets and request allocation of input/output devices.

The data processed by execution of any processing program must be in the form of a data set. A data set is a named, organized collection of one or more records that are logically related. Information in a data set may or may not be restricted to a specific type, purpose, or storage medium. A data set may be, for example, a source program, a library of subroutines, or a group of data records that is to be processed by a COBOL program.

A data set resides in one or more volumes. A volume is a unit of external storage that is accessible to an input/output device. For example, a volume may be a reel of tape or it may be a mass storage device.

To facilitate retrieval of a data set, the serial number of the volume upon which it resides can be entered, along with the data set name, in the system catalog of data sets. The catalog itself is a data set residing on one or more mass storage devices. It is organized into indexes that relate each data set name to its location--the volume in which it resides and its position within the volume. Only the data set name and DISP parameter need be specified to identify a cataloged data set to the system.

The catalog is originally created by a utility program. Once the catalog exists, any data set residing on either a mass storage device or a magnetic tape volume can be cataloged automatically by use of a catalog subparameter in a DD statement that refers to the data set.

Several input/output devices grouped together and given a single name when the system is generated constitute a device class. Each device class can be referred to by a collective name. For example, one device class called SYSDA could consist of all the mass storage devices in the installation; another called SYSSQ could consist of all the mass storage devices and tape devices.

EXECUTING A COBOL PROGRAM

Four basic operations are performed to execute a COBOL program:

- Compilation
- Linkage editing
- Loading
- Execution

COMPILATION

Compilation is the process of translating a COBOL source program into a series of instructions comprehensible to the computer, i.e., machine language. In operating system terminology, the input (source program) to the compiler is called the source module. The output (compiled source program) from the compiler is called the object module.

LINKAGE EDITING

The linkage editor is a service program that prepares object modules for execution. It can also be used to combine two or more separately compiled object modules into a format suitable for execution as a single program. The executable output of the linkage editor is called a load module, which must always be stored as a member of a partitioned data set.

In addition to processing object modules, the linkage editor can combine previously edited load modules, with or without one or more object modules, to form one load module.

During the process of linkage editing, external references between different modules are resolved.

LOADING

The Loader is a service program that processes COBOL object and load modules, resolves any references to subprograms, and executes the loaded module. All these functions are performed in one step. The Loader cannot produce load modules for a program library.

For detailed information on the Loader, see the publication IBM OS Linkage Editor and Loader, where a discussion of invoking the Loader can be found in "Using the Cataloged Procedures."

EXECUTION

Actual execution is under supervision of the control program, which obtains a load module from a library, loads it into main storage, and initiates execution of the machine language instructions contained in the load module.

OPERATING SYSTEM ENVIRONMENTS

The IBM Operating System offers two control programs. These are Multiprogramming with a Fixed Number of Tasks (MFT) and Multiprogramming with a Variable Number of Tasks (MVT).

MULTIPROGRAMMING WITH A FIXED NUMBER OF TASKS

The multiprogramming with a fixed number of tasks (MFT) control program divides storage into a number of discrete areas called partitions. Job steps are directed to these partitions using a priority scheduling system; that is, jobs are not executed as encountered in the job stream but according to a priority code. The MFT control program provides for:

- Priority scheduling of jobs using the class code
- Concurrent scheduling and execution of up to 15 separately protected jobs
- Reading one or more input streams

For further information about the various optional features of the MFT control program, see the publication IBM OS Storage Estimates.

MULTIPROGRAMMING WITH A VARIABLE NUMBER OF TASKS

The multiprogramming with a variable number of tasks (MVT) control program divides storage into areas called regions. Like MFT, the MVT control program uses a priority scheduling system and provides for concurrent execution of up to 15 jobs. In addition, the MVT control program provides for assignment of storage regions on a variable basis according to a region code.

Communication between the COBOL programmer and the job scheduler is effected through nine job control statements (hereinafter called control statements):

1. Job Statement
2. Execute Statement
3. Data Definition Statement
4. PROC Statement
5. PEND Statement
6. Command Statement
7. Delimiter Statement
8. Null Statement
9. Comment Statement

Parameters coded in these control statements aid the job scheduler in regulating the execution of jobs and job steps, retrieving and disposing of data, allocating input/output resources, and communicating with the operator.

The job statement (hereinafter called the JOB statement) marks the beginning of a job and, when jobs are stacked in the input stream, marks the end of the control statements for the preceding job. It may contain accounting information for use by an installation's accounting routines, give conditions for early termination of the job, and regulate the display of job scheduler messages. With priority schedulers, additional parameters are used to assign job priority, to request a specific class for job scheduler messages, to specify the amount of main storage to be allocated to the job, and to hold a job for later execution.

The execute statement (or EXEC statement) marks the beginning of a job step and identifies the program to be executed or the cataloged procedure to be used. It may also provide job step accounting information, give conditions for bypassing the job step, and pass control information to a processing program. With priority schedulers, additional parameters assign a time limit for the execution of the job step and specify the amount of main storage to be allocated.

The data definition statement (or DD statement) describes a data set and requests the allocation of input/output resources. The DD statement parameters identify the data set, give volume and unit information and disposition, and describe the labels and physical attributes of the data set.

The PROC statement appears as the first control statement in a cataloged procedure or an in-stream procedure and is used to assign default values to symbolic parameters defined in the procedure.

The PEND statement appears as the last control statement in an in-stream procedure and marks the end of the in-stream procedure. For further information about in-stream procedures, refer to the topic "Testing a Procedure as an In-Stream Procedure" in the chapter "Using the Cataloged Procedures."

The command statement is used by the operator to enter commands through the input stream. Commands can activate or deactivate system input and output units, request printouts and displays, and perform a number of other operator functions.

The delimiter statement and the null statement are markers in an input stream. The delimiter statement is used, when data is included in the input stream, to separate the data from subsequent control statements. The null statement can be used to mark the end of the control statements for certain jobs.

The comment statement can be inserted before or after any control statement and can contain any information deemed helpful by the person who codes the control statements. Comments can be coded in columns 4 through 80. The comment cannot be continued onto another statement. If the comment statement appears on a system output listing, it can be identified by the appearance of asterisks in columns 1 through 3.

The sequence of control statements required to specify a job is called a job control procedure.

For example, the job control procedure shown in Figure 1 could be placed in the input stream to compile a COBOL source module.

```
//JOB1       JOB
//STEP1      EXEC  PGM=IKFCBL00,PARM=DECK
//SYSUT1     DD    DSNAME=&&UT1,UNIT=SYSDA,SPACE=(TRK,(40))
//SYSUT2     DD    DSNAME=&&UT2,UNIT=SYSSQ,SPACE=(TRK,(40))
//SYSUT3     DD    DSNAME=&&UT3,UNIT=SYSSQ,SPACE=(TRK,(40))
//SYSUT4     DD    DSNAME=&&UT4,UNIT=SYSSQ,SPACE=(TRK,(40))
//SYSPRINT   DD    SYSOUT=A
//SYSPUNCH   DD    SYSOUT=B
//SYSIN      DD    *
   (source deck)
/*
```

Figure 1. Job Control Procedure

In the illustration, JOB1 is the name of the job. The JOB statement indicates the beginning of a job.

STEP1 is the name of the single job step in the job. The EXEC statement specifies that the IBM OS Full American National Standard COBOL Compiler (IKFCBL00) is to execute the job. The statement also specifies that a card deck of the object module is to be produced (PARM=DECK).

Note: Under MVT a REGION parameter is also required.

The SYSUT1, SYSUT2, SYSUT3, SYSUT4, and SYSUT5 (if the SYMDMP option is specified in the PARM parameter of the EXEC card) DD statements define utility data sets used by the compiler to process the source module. The names of the data sets defined by SYSUT1, SYSUT2, SYSUT3, SYSUT4, and SYSUT5 are &&UT1, &&UT2, &&UT3, &&UT4, and &&UT5, respectively. SYSUT1 must be on a mass storage device (UNIT=SYSDA). The system will allocate 40 tracks of space to SYSUT1 [SPACE=(TRK,(40))]. The other three utility data sets are assigned either to any available tape, in which case the SPACE parameter is ignored, or to a mass storage unit (UNIT=SYSSQ).

The SYSPRINT DD statement defines the data set that is to be printed. SYSOUT=A is the standard designation for data sets whose destination is the system output device, usually indicating that the data set is to be listed on a printer.

The SYSPUNCH DD statement defines the data set that is to be punched. SYSOUT=B designates a card punch.

The SYSIN DD statement defines the data set (in this case, the source module) that is to be used as input to the job step. The asterisk (*) indicates that the input data set follows in the input stream.

The delimiter (/*) statement separates data from subsequent control statements in the input stream.

Output from this job step includes any diagnostic messages associated with the compilation. They are printed in the data set specified by SYSPRINT.

Note: SYSDA, SYSQ, A, and B are IBM-specified device class names. If they are to be used, they must be incorporated at system generation time. If SYSOUT=B is to be used, the unit name SYSCP must be specified at system generation.

To avoid rewriting these statements, and the possibility of error, the programmer may place frequently used procedures on a system library called the procedure library. A procedure contained in the procedure library is called a cataloged procedure. A cataloged procedure can be called for execution by placing in the input stream a simple procedure that may require only the JOB and EXEC statements.

If slightly modified, the procedure in the previous example can be cataloged, i.e., placed in the procedure library. For example, if it were cataloged and given the name CATPROC, it could be called for execution by placing the statements shown in Figure 2 in the input stream.

```
//JOB2          JOB
//STEPA         EXEC   PROC=CATPROC
//STEP1.SYSIN DD       *
   (source deck)
/*
```

Figure 2. Catalog Procedure

In Figure 2, JOB2 is the name of the job. STEPA is the name of the single job step.

The EXEC statement calls the cataloged procedure containing STEP1 to execute the job step (PROC=CATPROC).

A procedure can be tested before it is placed in the procedure library by converting it into an _in-stream procedure_. An in-stream procedure can be executed any number of times during a job. For further information about in-stream procedures, refer to the topic "Testing a Procedure as an In-Stream Procedure" in "Using the Cataloged Procedures."

"User File Processing" and "Appendix I: Checklist for Job Control Procedures" explain, with numerous examples, the preparation of job control procedures. "Data Set Requirements" describes required and optional data sets for compilation, linkage editing, and execution time job steps. The chapter "Using Cataloged Procedures" provides information about using and modifying cataloged procedures.

The section "Control Statements," below, shows the format and use of the parameters and subparameters that can be specified for each job control statement. Some parameters of the statements are described only briefly. For further information, see the publication _IBM OS Job Control Language Reference_. The syntactic format descriptions in this chapter can be used as a reference for the exact format and for the use of each parameter.

CONTROL STATEMENTS

The COBOL programmer uses the control statements shown in Table 1 to compile, linkage edit, and execute programs.

Table 1. 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 load module or cataloged procedure 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 must follow each data set that appears in the input stream, e.g., after a COBOL source module punched deck.
comment	Contains miscellaneous remarks and notes written by the programmer; it may appear anywhere in the job stream after the JOB statement.

JOB MANAGEMENT

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

The _job scheduler_ is a set of routines that reads input streams, analyzes control statements, allocates input/output resources, issues diagnostic messages to the programmer, and schedules job flow through the system.

The _master scheduler_ is a set of routines that accepts operator commands and acts as the operator's agent within the system. It relays system messages to the operator, performs system functions at his request, and responds to his inquiries regarding the status of a job or of the system. The master scheduler also relays all communication between a processing program and the operator.

Priority schedulers process complete jobs according to their relative priority, and available system resources.

PREPARING CONTROL STATEMENTS

Except for the comment statement, control statements are identified by the initial characters // or /* in card columns 1 and 2. The comment statement is identified by the initial characters //* in columns 1 through 3. Control statements may contain four fields: name, operation, operand, and comment, as shown in Figure 3.

Statement	Columns 1	2	3	4 (Fields)
Job	/	/	name	JOB operand[1] comments[1]
Execute	/	/	name[1]	EXEC operand comments[1]
Data Definition	/	/	name[1]	DD operand comments[1]
Procedure	/	/	name[1]	PROC operand comments[1]
Command	/	/		operation(command) operand comments[1]
Delimiter	/	*		comments[1]
Null	/	/		
Comment	/	/	*	comments
Pend	/	/	name[1]	PEND

[1]Optional.

Figure 3. General Format of Control Statements

Name Field

The name contains from one through eight alphanumeric characters, the first of which must be alphabetic. The name begins in card column 3. It is followed by one or more blanks. The name is used, as follows:

- To identify the control statement to the operating system
- To enable other control statements in the job to refer to information contained in the named statement
- To relate DD statements to files named in a COBOL source program

Operation Field

The operation field is preceded and followed by one or more blanks. It may contain one of the following operation codes:

JOB
EXEC
DD
PROC
PEND

If the statement is a delimiter statement, there is no operation field and comments may start after one blank.

Operand Field

The operand field is preceded and followed by one or more blanks and may continue through column 71 and onto one or more continuation cards. It contains the parameters or subparameters that give required and optional information to the operating system. Parameters and subparameters are separated by commas. A blank in the operand field causes the system to treat the remaining data on the card as a comment. There are two types of parameters: positional and keyword (Figures 4, 5, and 7).

Positional Parameters: Positional parameters are the first parameters in the operand field, and they must appear in the specified sequence. If a positional parameter is omitted and other positional parameters follow, the omission must be indicated by a comma. If other positional parameters do not follow, no comma is needed.

Keyword Parameters: A keyword parameter may be placed anywhere in the operand field following 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 subparameter list, it must be enclosed in parentheses or single quotation marks; the subparameters in the list must be separated by commas. Keyword parameters may appear in any sequence.

Subparameters are either positional or keyword. Positional and keyword subparameters for job control statements are shown in Figures 4, 5, and 7. Positional subparameters appear first in the parameter and must be in the specified sequence. If a positional subparameter is omitted and other positional subparameters follow, a comma must indicate the omission.

Comments Field

Optional comments must be separated from the last parameter (or the /* in a delimiter statement) by one or more blanks and may appear in the remaining columns up to and including column 71. An optional comment may be continued onto one or more continuation cards. Comments can contain blanks.

Note: Comments in the optional comments field follow different procedures from those on the comment statement.

CONVENTIONS FOR CHARACTER DELIMITERS

Commas, parentheses, and blanks are interpreted as character delimiters. If they are not intended by the programmer to be used as delimiters, the fields in which they appear must be enclosed in single quotation marks, indicating that the enclosed information is to be treated as a single field. When an apostrophe (or a single quotation mark, since the same character is used for either) is to be contained within such a field, it must be shown as two consecutive single quotation marks (5-8 punch), not as a double quotation mark (7-8 punch). For example,

    Wm. O'Connor

should be shown as

    'Wm. O''Connor'

This convention applies to three fields: programmer's name in the JOB statement, information in the PARM parameter of the EXEC statement, and accounting information in the JOB and EXEC statements.

RULES FOR CONTINUING CONTROL STATEMENTS

Except for the comment statement, 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 a parameter is to be placed on separate cards, the operating system continuation conventions must be used. To continue an operand field:

1. Interrupt the field at the end of a complete parameter or subparameter, including the comma that follows it, at or before column 71.

2. Include comments by following the interrupted field with at least one blank.

3. Optionally, code any nonblank character in column 72. If a character is not coded in column 72, the job scheduler treats the next statement as a continuation statement as long as the conventions outlined in items 4 and 5 are observed.

4. Code the identifying characters // in columns 1 and 2 of the following card or card image.

5. Continue the interrupted operand beginning in any column from 4 through 16.

Comments other than those on a comment statement 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.

Any control statements in the input stream that the job scheduler considers to contain only continued comments will print on a system output listing with a //* in columns 1 through 3. Comments written on a comment statement cannot be continued.

NOTATION FOR DESCRIBING JOB CONTROL STATEMENTS

The notation used in this publication to define the syntax of job control statements is as follows:

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

Name	Symbol	Purpose
hyphen	-	Joins lower-case letters, words, and symbols to form a single variable
"or" symbol	|	Indicates alternatives
braces	{ }	Indicate that the enclosed is a group of related items, only one of which is required
brackets	[ ]	Indicate that the enclosed are optional items. Brackets are also used with alternatives to indicate that a default is assumed if no alternative is listed
ellipsis	...	Indicates that the preceding item or group of items can be repeated
superscript	1 2 3	Indicates a footnote reference

2. Stacked items, enclosed in either brackets or braces, represent alternative items. No more than one of the stacked items can be written by the programmer.

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

Name	Symbol
single quotation mark	'
asterisk	*
comma	,
equal sign	=
parentheses	( )
period	.
slash	/

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

   Note: Many of these defaults can be changed at system generation time.

5. Lower-case letters, words, and symbols appearing in a control statement definition represent variables for which specific information is substituted in the actual statement.

6. Blanks are used in Figures 4, 5, 6, and 7 to improve the readability of control statement definitions. In actual statements, blanks would be interpreted as delimiters.

Name	Operation	Operand
//jobname	JOB	**Positional Parameters** [([account-number] [,accounting-information])[1] [2] [3]] [,programmer-name][4] [5] **Keyword Parameters** [MSGLEVEL=(x,y)][6] [TIME=(minutes,seconds)] [CLASS=jobclass] [COND=((code,operator) [,(code,operator)]...[7])[8]] [PRTY=job priority] [MSGCLASS=classname] [REGION=(nnnnnxK[,nnnnnyK])] [ROLL=(x,y)] [TYPRUN=HOLD] [RD=request] [RESTART=({ * / stepname / stepname.procstepname } [,checkid])]

[1]If the information specified (account-number and/or accounting-information) contains blanks, parentheses, or equal signs, the information must be delimited by single quotation marks 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 quotation marks is 142.
[4]If programmer-name contains any special characters other than the period, it must be enclosed within single quotation marks.
[5]The maximum number of characters allowed for programmer-name is 20.
[6]x = 0, 1, or 2 is the JCL message.
y = 0 or 1 is the allocation message level.
Note that the value 1 may be used in place of (1,1).
[7]The maximum number of repetitions allowed is 7.
[8]If only one test is specified, the outer pair of parentheses may be omitted.

Figure 4. JOB Statement

JOB STATEMENT

The JOB statement is the first statement in the sequence of control statements that describe a job. The JOB statement can contain the following information:

1. Name of the job.
2. Accounting information relative to the job.
3. Programmer's name.
4. Indication of whether or not the job control statements are to be printed on the system output listing.
5. Conditions for terminating the execution of the job.
6. For priority scheduling systems: job priority assignment, job scheduler message class, and for the MVT environment, main storage region size.

Figure 4 is a general format of the JOB statement.

Identifying the Job (jobname)

The jobname identifies the job to the job scheduler. It must satisfy the positional, length, and content requirements for a name field. No two jobs being handled by a priority scheduler should have the same jobname.

Supplying Job Accounting Information

For job accounting purposes, the JOB statement can be used to supply information to an installation's accounting procedures. To supply job accounting information, code the positional parameter first in the operand field.

```
(acct#,additional accounting information)
```

Replace the term "acct#" with the account number to which the job is charged; replace the term "additional accounting information" with other items required by an installation's accounting routines. As a system generation option with sequential schedulers, the account number can be established as a required subparameter. With priority schedulers, the requirement can be established with a cataloged procedure for the input reader. Otherwise, the account number is considered optional.

Notes:

- Subparameters of additional accounting information must be separated by commas.
- The number of characters in the account number and additional accounting information must not exceed a total of 142.
- If the list contains only an account number, the programmer need not code the parentheses.
- If the list does not contain an account number, the programmer must indicate its absence by coding a comma preceding the additional accounting information.
- If the account number or any subparameter of additional accounting information contains any special character (except hyphens), the programmer must enclose the number or subparameter in apostrophes (5-8 punch). The apostrophes are not passed as part of the information.

Reference:

- To write an accounting routine that processes job accounting information, see the section "Adding an Accounting Routine to the Control Program" of the publication IBM OS System Programmer's Guide.

Identifying the Programmer

The person responsible for a job codes his name or identification in the JOB statement, following the job accounting information. This positional parameter is also passed to an installation's routines. As a system generation option with sequential schedulers, the programmer's name can be established as a required parameter. With priority schedulers, the requirement can be established with a cataloged procedure for the input reader. Otherwise, this parameter is considered optional.

Notes:

- The number of characters in the name cannot exceed 20.
- If the name contains special characters other than periods, it must be enclosed in apostrophes. If the special characters include apostrophes, each must be shown as two consecutive apostrophes, e.g., 'T.O''NEILL'.
- If the job accounting information is not coded, the programmer must indicate its absence by coding a comma preceding the programmer-name.
- If neither job accounting information nor programmer-name is present, the programmer need not code commas to indicate their absence.

Reference:

- To write a routine that processes the programmer's name, see the section "Adding an Accounting Routine to the Control Program" of the publication IBM OS System Programmer's Guide.

Displaying All Control Statements, Allocation, and Termination Messages (MSGLEVEL)

The MSGLEVEL parameter indicates whether or not the programmer wants control statements and/or allocation and termination messages to appear in his output listing. To receive this output, code the keyword parameter in the operand field of the JOB statement.

```
MSGLEVEL=(x,y)
```

The letter "x" represents a job control language message code and can be assigned the value 0, 1, or 2. When x = 0 is specified, only the JOB statement, incorrect control statements, and associated diagnostic messages are displayed. When x = 1 is specified, input statements, cataloged procedure statements, and symbolic substitution of parameters are displayed. When x = 2 is specified, only input statements are displayed.

The letter "y" represents an allocation message code and can be assigned the value 0 or 1. When y = 0 is specified, no allocation, termination, or recovery messages are displayed, unless an ABEND occurs during problem program execution. If an ABEND occurs, termination messages are displayed. When y = 1 is specified, all allocation, termination, and recovery messages are displayed.

Notes:

- If the value 1 is selected for both codes, the value may be specified once without the parentheses; i.e., MSGLEVEL=1 is the same as MSGLEVEL=(1,1).
- The default values are taken from the reader procedure.
- If an error occurs on a control statement that is continued onto one or more cards, only one of the continuation cards is printed with the diagnostic messages.

Specifying Conditions for Job Termination (COND)

To eliminate unnecessary use of computing time, the programmer might want to base the continuation of a job on the successful completion of one or more of its job steps. At the completion of each job step, the processing program passes a number to the job scheduler as a return code. The COND parameter provides the means to test each return code as many as eight times. If any one of the tests is satisfied, subsequent steps are bypassed and the job is terminated.

To specify conditions for job termination, code the keyword parameter in the operand field of the JOB statement.

```
COND=((code,operator),..,(code,operator))
```

See the COND parameter on the EXEC statement for a discussion of the operator values and the codes issued by the compiler and linkage editor at the end of a job step.

Note:

- The subparameters EVEN and ONLY cannot be specified as part of the COND parameter on the JOB statement.

Requesting Restart for a Job (RD)

The restart facilities are used in order to minimize the time lost in reprocessing a job that abnormally terminates. These facilities permit execution of jobs that abnormally terminate to be automatically restarted.

Execution of a job can be automatically restarted at the beginning of the job step that abnormally terminated (step restart) or within the step (checkpoint restart). In order for checkpoint restart to occur, the CHKPT macro instruction must have been executed in the processing program prior to abnormal termination. The CHKPT macro instruction is activated by the COBOL source language RERUN clause. The RD parameter specifies that step restart can occur or that the action of the CHKPT macro instruction is to be suppressed.

To request that step restart be permitted or to request that the action of the RERUN clause be suppressed, code the keyword parameter in the operand field of the JOB statement.

```
RD=request
```

Replace the word "request" with:

- R -- to permit automatic step restart
- NC -- to suppress the action of the CHKPT macro instruction and not to permit automatic restart
- NR -- to request that the CHKPT macro instruction be allowed to establish a checkpoint, but not to permit automatic restart
- RNC -- to permit step restart and to suppress the action of the CHKPT macro instruction

Each of these requests is described in greater detail in the following paragraphs.

RD=R: If the processing programs used by the job do not include any CHKPT macro instructions, RD=R allows execution to be resumed at the beginning of the step that causes abnormal termination. If any of the programs do include one or more CHKPT macro instructions, step restart can occur if a step abnormally terminates before execution of a CHKPT macro instruction; thereafter, checkpoint restart can occur.

RD=NC or RD=RNC: RD=NC or RD=RNC should be specified to suppress the action of all CHKPT macro instructions included in the programs. When RD=NC is specified, neither step restart nor checkpoint restart can occur. When RD=RNC is specified, step restart can occur.

RD=NR: RD=NR permits a CHKPT macro instruction to establish a checkpoint, but does not permit automatic restart. Instead, at a later time, the job can be resubmitted and execution can begin at a specific checkpoint. (Resubmitting a job for restart is discussed later.)

Before automatic step restart occurs, all data sets in the restart step with a status of OLD or MOD, and all data sets being passed to steps following the restart step, are kept. All data sets in the restart step with a status of NEW are deleted. Before automatic checkpoint restart occurs, all data sets currently in use by the job are kept.

If the RD parameter is omitted and no checkpoints are taken, automatic restart cannot occur. If the RD parameter is omitted but one or more checkpoints are taken, automatic checkpoint restart can occur.

Notes:

- When using a system with MVT or MFT, restart can occur only if MSGLEVEL=1 is coded on the JOB statement.
- If step restart is requested, each step must be assigned a unique step name.
- If no RERUN clause is specified in the user's program, no checkpoints are written regardless of the disposition of the RD parameter.

Reference:

- For detailed information on the checkpoint/restart facilities, see the publication IBM OS Supervisor Services.

Resubmitting a Job for Restart (RESTART)

The restart facilities can be used if the job is abnormally terminated and the programmer wants to resubmit the job for execution. These facilities reduce the time required to execute the job since execution of the job is resumed, not repeated.

Execution of a resubmitted job can be restarted at the beginning of a step (step restart) or within a step (checkpoint restart). In order for checkpoint restart to occur, a program must previously have had a checkpoint record written. The RESTART parameter specifies where execution is to be restarted.

If execution is to be restarted at a particular job step, code the keyword parameter in the operand field of the JOB statement before resubmitting the job.

```
RESTART=stepname
```

Replace the word "stepname" with the name of the step at which execution is to be restarted. Replace stepname with an asterisk (*) if execution is to be restarted at the first job step.

If execution is to be restarted at a particular checkpoint within a particular job step, code the keyword parameter in the operand field of the JOB statement before resubmitting the job.

```
RESTART=(stepname,checkid)
```

Replace the word stepname with the name of the step in which execution is to be restarted. Replace the term "checkid" with the 1- to 16-character name that identifies the checkpoint within the step.

If execution is to be restarted at a checkpoint, the resubmitted job must include an additional DD statement. This DD statement defines the checkpoint data set and has the ddname SYSCHK. Do not include a SYSCHK DD statement if step restart is to be performed.

If the RESTART parameter is not specified on the JOB statement of the resubmitted job, execution is repeated.

Notes:

- If execution is to be restarted at or within a cataloged procedure step, give both the name of the step that invokes the procedure and the procedure step name, as below.

```
RESTART=stepname.procstepname
```

- If step restart is performed, generation data sets that were created and cataloged in steps preceding the restarted step must not be referred to in the restart step or in steps following the restart step by means of the same relative generation numbers that were used to create them. For example, a generation data set assigned a generation number of +1, would be referred to as 0 in the restart step or steps following the restart step.

- Backward references cannot be made to steps that precede the restart step using the following keyword parameters: PGM, COND, SUBALLOC, and VOLUME=REF, unless in the last case the referenced statement includes VOLUME=SER=(ser#).

Reference:

- For detailed information on the checkpoint/restart facilities, see the publication IBM System/360 Operating System: Supervisor Services.

PRIORITY SCHEDULING JOB PARAMETERS

Setting Job Time Limits (TIME)

To assign a limit to the computing time used by a job, code the keyword parameter in the operand field.

```
TIME=(minutes,seconds)
```

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 to represent the maximum time for execution of a job.

Notes:

- The number of minutes cannot exceed 1439 and the number of seconds cannot exceed 59. If the job is not completed in this time it is terminated.

- If the job requires use of the system for more than 24 hours (1439 minutes) specify TIME=1440. This number suppresses job timing.

- If the time limit is given in minutes only, the parentheses need not be coded; e.g., TIME=5.

- If the time limit is given in seconds, the comma must be coded to indicate the absence of minutes; e.g., TIME=(,45).

- If the TIME parameter is omitted, the default job time is assumed.

Assigning a Job Class (CLASS)

To assign a job class to a job, code the keyword parameter in the operand field of the JOB statement.

```
CLASS=jobclass
```

Replace the term "jobclass" with an alphabetic character A through O. The use of this parameter and the meaning of the character A through O are to be determined by each installation.

If the CLASS parameter is omitted, or CLASS=A is coded, the default job class of A is assigned to the job.

Note:

- If an installation provides time-slicing facilities in a system with MFT, the CLASS parameter can be used to make the job part of the group of jobs to be time-sliced. Time-slicing permits the processing of tasks of equal priority so that each is executed for its specified period of time. At system generation, a group of contiguous partitions are selected to be used for time-slicing, and each partition is assigned at least one job class. If the job is to be time-sliced, specify a class that was assigned only to the partitions selected for time-slicing.

Assigning Job Priority (PRTY)

To assign a priority other than the default job priority (as established in the input reader procedure), code the keyword parameter in the operand field of the JOB statement.

```
PRTY=nn
```

Replace the letters "nn" with a decimal number from 0 through 13 (the highest priority number is 13).

If an installation provides time-slicing facilities in a system with MVT, the PRTY parameter can be used to make the job part of a group of jobs to be time-sliced. At system generation, the priority of the time-sliced group is selected. If the job priority number specified corresponds with the priority number selected for time-slicing, then the job will be time-sliced.

If the PRTY parameter is omitted, the default job priority is assigned to the job.

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

Requesting a Message Class (MSGCLASS)

With the 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 parameter allows routing of all messages issued by the job scheduler to an output class other than the normal message class, A.

To choose such a class, code the keyword parameter in the operand field of the JOB statement.

```
MSGCLASS=x
```

Replace the letter "x" with an alphabetic (A-Z) or numeric (0-9) character. An output writer, which is assigned to process this class, will transfer this data to a specific device.

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

Reference:

- For a more detailed discussion of output classes, see the publication IBM OS Operator's Reference, Order No. GC28-6691.

Specifying Main Storage Requirements for a Job (REGION) (MVT only)

For jobs that require an unusual amount of main storage, the JOB statement provides the REGION parameter. The REGION parameter specifies:

- The maximum amount of main storage to be allocated to the job. This amount must include the size of those components required by the user's program that are not resident in main storage.
- The amount of main storage to be allocated to the job, and the storage hierarchy or hierarchies in which the space is to be allocated. This request should be made only if main storage hierarchy support has been specified during system generation. If an IBM 2361 Core Storage, Model 1 or 2, is present in the system, processor storage is referred to as hierarchy 0 and 2361 Core Storage is referred to as hierarchy 1. If 2361 Core Storage is not present but main storage hierarchy support was specified in system generation, a two-part region is established in processor storage when a region is defined to exist in two hierarchies. The two parts are not necessarily contiguous.

To specify a region size, code the keyword parameter in the operand field of the JOB statement.

```
REGION=(nnnnnxK[,nnnnnyK])
```

To request the maximum amount of main storage required by the job, the term "nnnnnx" should be replaced with the number of 1024-byte areas allocated to the job, e.g., REGION=52K. This number can range from 1 to 5 digits but cannot exceed 16383.

To request a region size and the hierarchy desired, the term nnnnnx is replaced with the number of contiguous 1024-byte areas to be allocated to the job in hierarchy 0; the term "nnnnny" is replaced with the number of contiguous 1024-byte areas to be allocated in hierarchy 1, e.g., REGION=(60K,200K). When only processor storage is used to include hierarchies 0 and 1, the combined values of nnnnnx and nnnnny cannot exceed 16383. If 2361 Core Storage is present, nnnnnx cannot exceed 16383 and, for a 2361 Model 1, nnnnny cannot exceed 1024, or 2048 for a 2361 Model 2. Each value specified should be an even number. (If an odd number is specified, the system treats it as the next higher even number.)

If storage is requested only in hierarchy 1, a comma must be coded to indicate the absence of the first subparameter, e.g., REGION=(,200K). If storage is requested only in hierarchy 0, or if hierarchy support is not present, the parentheses need not be coded, e.g., REGION=70K.

If the REGION parameter is omitted or if a region size smaller than the default region size is requested, it is assumed that the default value is that established by the input reader procedure.

Notes:

- Region sizes for each job step can be coded by specifying the REGION parameter in the EXEC statement for each job step. However, if a REGION parameter is present in the JOB statement, it overrides REGION parameters in EXEC statements.

- If main storage hierarchy support is not included but regions are requested in both hierarchies, the region sizes are combined and an attempt is made to allocate a single region from processor storage. If a region is requested entirely from hierarchy 1, an attempt is made to allocate the region from processor storage.

- For information on storage requirements to be considered when specifying a region size, see the publication IBM OS Storage Estimates.

Holding a Job for Later Execution

To temporarily prevent a job from being selected for processing, code the keyword parameter in the operand field of the JOB statement.

TYPRUN=HOLD

The job is then held until a RELEASE command is issued by the operator. This specification is particularly useful when one job must be run after another job has terminated.

Specifying Additional Storage (ROLL) (MVT only)

To allocate additional main storage to a job step whose own region does not contain any more available space, code the keyword parameter in the operand field of the JOB statement.

ROLL=(x,y)

In order to allocate this additional space to a job step, another job step may have to be rolled out, i.e., temporarily transferred to secondary storage. When x is replaced with YES, each of the programmer's job steps can be rolled out; when x is replaced with NO, the job steps cannot be rolled out. When y is replaced with YES, each job step can cause rollout; when y is replaced with NO, the job steps cannot cause rollout. If additional main storage is required for the job's steps, YES must be specified for y. If this parameter is omitted, ROLL=(YES,NO) is assumed. ROLL parameters can also be coded in EXEC statements, but are superseded by a ROLL parameter coded in the JOB statement.

EXEC STATEMENT

The EXEC statement defines a job step and calls for its execution. It contains the following information:

1. The name of a load module or the name of a cataloged procedure that contains the name of a load module that is to be executed. The load module can be the COBOL compiler, the linkage editor, the loader, or any COBOL program in load module form.
2. Accounting information for this job step.
3. Conditions for bypassing the execution of this job step.
4. For priority scheduling systems: computing time for a job step or cataloged procedure step, and main storage region size.

5. Compiler, linkage editor, or loader options chosen for the job step.

Figure 5 is the general format of the EXEC statement.

Note: If the information specified is normally delimited by parentheses but contains blanks, parentheses, or equal signs, it must be delimited by single quotation marks instead of parentheses.

Identifying the Step (stepname)

The stepname identifies a job step within a job. It must satisfy the positional, length, and content requirements for a name field. The programmer must specify a stepname if later control statements refer to the step or if the step is going to be part of a cataloged procedure. Each stepname in a job or procedure must be unique.

POSITIONAL PARAMETERS

Identifying the Program (PGM) or Procedure (PROC)

The EXEC statement identifies the program to be executed in the job step with the PGM parameter. To specify the COBOL compiler, code the positional parameter in the first position of the operand field of the EXEC statement.

```
PGM=IKFCBL00
```

It indicates that the COBOL compiler is the processing program to be executed in the job step.

To specify the linkage editor, code the positional parameter in the first position of the operand field of the EXEC statement.

```
PGM=IEWL
```

This indicates that the linkage editor is the processing program to be executed in the job step.

The PGM parameter depends upon the type of library in which the program resides. If the job step uses a cataloged procedure, the EXEC statement identifies it with the PROC parameter, in place of the PGM parameter.

1. Temporary libraries are temporary partitioned data sets created to store a program until it is used in a later job step of the same job. This type of library is particularly useful for storing the program output of a linkage editor run until it is executed in a later job step. To execute a program from a temporary library, code the positional parameter in the first position of the operand field of the EXEC statement.

   ```
   PGM=*.stepname.ddname
   ```

   The asterisk (*) indicates the current job step. Replace the terms stepname and ddname with the names of the job step and the DD statement within the procedure step, respectively, in which the temporary library is created.

   If the temporary library is created in a catalogued procedure step, in order to call it in a later job step outside the procedure, give both the name of the job step that calls the procedure and the procedure stepname by coding the positional parameter in the first position of the operand field of the EXEC statement.

   ```
   PGM=*.stepname.procstepname.ddname
   ```

2. The system library is a partitioned data set named SYS1.LINKLIB that contains nonresident control program routines, and processor programs. To execute a program that resides in the system library, code the positional parameter in the first position of the operand field.

   ```
   PGM=progname
   ```

   Replace the term progname with the member name or alias associated with this program. This same keyword parameter can be used to execute a program that resides in a private library. Private libraries are made available to a job with a special DD statement (see "Additional DD Statement Facilities").

```
Name              Operation   Operand

                              Positional Parameters

//[stepname]1     EXEC        ( PGM=progname                           )
                              ) PGM=*.stepname.ddname                  (
                              < PROCC=procname                         >
                              ( procname                               )
                              ( PGM=*.stepname.procstep.ddname         )

                              Keyword Parameters

                              [{ACCT2         }                            3 4 5 ]
                              [{ACCT.procstep } = (accounting-information)       ]

                              [{COND2         }                                        6 7 ]
                              [{COND.procstep } = ((code,operator[,stepname[.procstep]])...) ]

                              [{PARM2         }                  3 8 9 ]
                              [{PARM.procstep } = (option[,option]...) ]

                              [{TIME          }                     ]
                              [{TIME.procstep } = (minutes,seconds) ]

                              [{REGION          }                      ]
                              [{REGION.procstep } = nnnnnxK[,nnnnnyK]  ]

                              [{ROLL          }         ]
                              [{ROLL.procstep } = (x,y) ]

                              [{RD          }           ]
                              [{RD.procstep } = request ]

                              [{DPRTY          }                      ]
                              [{DPRTY.procstep } = (value 1, value 2) ]
```

1Stepname is required when information from this control statement is referred to in a later job step.
2If this format is selected, it may be repeated in the EXEC statement once for each step in the cataloged procedure.
3If the information specified contains any special characters except hyphens, it must be delimited by single quotation marks instead of parentheses.
4If accounting-information contains any special characters except hyphens, it must be delimited by single quotation marks.
5The maximum number of characters allowed between the delimiting quotation marks or parentheses is 142.
6The maximum number of repetitions allowed is 7.
7If only one test is specified, the outer pair of parentheses may be omitted.
8If the only special character contained in the value is a comma, the value may be enclosed in quotation marks.
9The maximum number of characters allowed between the delimiting quotation marks or parentheses is 100.

Figure 5. EXEC Statement

3. Instead of executing a particular program, a job step may use a cataloged procedure. A cataloged procedure can contain control statements for several steps, each of which executes a particular program. Cataloged procedures are members of a library named SYS1.PROCLIB. To request a cataloged procedure, code the positional parameter in the first position of the operand field of the EXEC statement.

```
PROC=procname
```

Replace the term procname with the unqualified name of the cataloged procedure (see "Using the DD Statement" for a discussion of qualified names).

Note: A procedure may be tested before it is placed in the procedure library by converting it into an in-stream procedure and placing it within the job step itself. In-stream procedures are discussed in the section, "Testing a Procedure as an In-Stream Procedure" in the chapter "Using the Cataloged Procedures."

KEYWORD PARAMETERS

Specifying Job Step Accounting Information (ACCT)

When executing a multistep job, or a job that uses cataloged procedures, the programmer can use this parameter so that jobsteps are charged to separate accounting areas. To specify items of accounting information to the installation accounting routines for this job step, code the keyword parameter in the operand field of the EXEC statement.

```
ACCT=(accounting information)
```

Replace the term "accounting information" with one or more subparameters separated by commas. If both the JOB and EXEC statements contain accounting information, the installation accounting routines decide how the accounting information shall be used for the job step.

To pass accounting information to a step within a cataloged procedure, code the keyword parameter in the operand field of the EXEC statement.

```
ACCT.procstep=(accounting information)
```

Procstep is the name of the step in the cataloged procedure. This specification overrides the ACCT parameter in the named procedure step, if one is present.

Specifying Conditions for Bypassing or Executing the Job Step (COND)

The execution of certain job steps is based on the success or failure of preceding steps. The COND parameter provides the means to:

- Make as many as eight tests on return codes issued by preceding job steps or cataloged procedure steps, which were completed normally. If any one of the tests is satisfied, the job step is bypassed.
- Specify that the job step is to be executed even if one or more of the preceding job steps abnormally terminated or only if one or more of the preceding job steps abnormally terminated.

To specify conditions for bypassing a job step, code the keyword parameter in the operand field of the EXEC statement.

```
COND=((code,operator,[stepname]),...,
           (code,operator,[stepname]))
```

The term "code" may be replaced by a decimal numeral to be compared with the job step return code. The return codes for both the compiler and the linkage editor are:

00 Normal conclusion

04 Warning messages have been listed, but program is executable.

08 Error messages have been listed; execution may fail.

12 Severe errors have occurred; execution is impossible.

16 Terminal errors have occurred; execution of the processor has been terminated.

The compiler issues a return code of 16 when any of the following are detected:

- BASIS member-name is specified and no member of that name is found

- COPY member-name is specified and no SYSLIB statement is included

- Required device not available

- Not enough core storage is available for the tables required for compilation

- A table exceeded its maximum size

- A permanent input/output error has been encountered on an external device

The return codes have a correlation with the severity level of the error messages. With linkage editor messages, for example, the rightmost digit of the message number states the severity level; this number is multiplied by 4 to get the appropriate return code. With the COBOL compiler, 04, 08, 12, and 16 are equal to the severity flags: W, C, E, and D, respectively.

The term "operator" specifies the test to be made of the relation between the programmer-specified code and the job step return code. Replace the term operator with one of the following:

GT (greater than)
GE (greater than or equal to)
EQ (equal to)
LT (less than)
LE (less than or equal to)
NE (not equal to)

The term "stepname" identifies the previously executed job step that issued the return code to be tested and is replaced by the name of that preceding job step. If stepname is not specified, code is compared to the return codes issued by all preceding steps in the job.

Replace the term stepname with the name of the preceding job step that issues the return code to be tested.

If the programmer codes

```
COND=((4,GT,STEP1),(8,EQ,STEP2))
```

the statement is interpreted as: "If 4 is greater than the return code issued by STEP1, or if STEP2 issues a return code of 8, this job step bypassed."

Notes:

- If only one test is made, the programmer need not code the outer parentheses, e.g., COND=(12,EQ,STEPX).

- If each return code test is made on all preceding steps, the programmer need not code the terms stepname, e.g., COND=((4,GT),(8,EQ)).

- When the return code is issued by a cataloged procedure step, the programmer may want to test it in a later job step outside of the procedure. In order to test it, give both the name of the job step that calls the procedure and the procedure stepname, e.g., COND=((code,operator, stepname.procstep),...).

Abnormal termination of a job step normally causes subsequent steps to be bypassed and the job to be terminated. By means of the COND parameter, however, the programmer can specify execution of a job step after one or more preceding job steps have abnormally terminated. For the COND parameter, a job step is considered to terminate abnormally if a failure occurs within the user's program once it has received control. (If a job step is abnormally terminated during scheduling because of failures such as job control language errors or inability to allocate space, the remainder of the job steps are bypassed, whether or not a condition for executing a later job step was specified.)

To specify the condition for executing a job step, code the keyword parameter in the operand field of the EXEC statement.

$$\text{COND=}\left\{\begin{array}{l}\text{EVEN}\\ \text{ONLY}\end{array}\right\}$$

The EVEN or ONLY subparameters are mutually exclusive. The subparameter selected can be coded in combination with up to seven return code tests, and can appear before, between, or after return code tests, e.g.,

```
COND=(EVEN,(4,GT,STEP3))

COND=((8,GE,STEP1),(16,GE),ONLY)
```

The EVEN subparameter causes the step to be executed even when one or more of the preceding job steps have abnormally terminated. However, if any return code tests specified in this job step are satisfied, the step is bypassed. The ONLY subparameter causes the step to be executed only when one or more of the preceding job steps have abnormally terminated. However, if any return code tests specified in this job step are satisfied, the step is bypassed.

When a job step abnormally terminates, the COND parameter on the EXEC statement of the next step is scanned for the EVEN or ONLY subparameter. If neither is specified, the job step is bypassed and the EXEC statement of the next step is scanned for the EVEN or ONLY subparameter. If EVEN or ONLY is specified, return code tests, if any, are made on all previous steps specified that executed and did not abnormally terminate. If any one of these tests is satisfied, the step is bypassed. Otherwise, the job step is executed.

If the programmer codes

```
COND=EVEN
```

the statement is interpreted as: "Execute this step even if one or more of the preceding steps abnormally terminated during execution." If COND=ONLY is coded, it is interpreted as: "Execute this step only if one or more of the preceding steps abnormally terminated during execution."

If the COND parameter is omitted, no return code tests are made and the step will be bypassed when any of the preceding job steps abnormally terminate.

Notes:

- When a job step that contains the EVEN or ONLY subparameter refers to a data set that was to be created or cataloged in a preceding step, the data set will not exist if the step creating it was bypassed.
- When a jobstep that contains the EVEN or ONLY subparameter refers to a data set that was to be created or cataloged in a preceding step, the data set may be incomplete if the step creating it abnormally terminated.
- When the job step uses a cataloged procedure, the programmer can establish return code tests and the EVEN or ONLY subparameter for a procedure step by including, as part of the keyword COND, the procedure stepname, e.g., COND.procstepname. This specification overrides the COND parameter in the named procedure step if one is present. The programmer can code as many parameters of this form as there are steps in the cataloged procedure.
- To establish one set of return code tests and the EVEN or ONLY subparameter for all steps in a procedure, code the COND parameter without a procedure stepname. This specification replaces all COND parameters in the procedure if any are present.

Job steps following a step that abnormally terminates are normally bypassed. If a job step is to be executed even if a preceding step abnormally terminates, specify this condition, along with up to seven return code tests:

```
//STEP3 EXEC PGM=CONVERT,                X
//   COND=(EVEN,(4,EQ,STEP1)),...
```

Here, the step is executed if the return code test is not satisfied, even if one or more of the preceding job steps abnormally terminated. If a job step is to execute only when one or more of the preceding steps abnornally terminate, replace EVEN in the above example with ONLY.

If the EXEC statement calls a cataloged procedure, the programmer can establish return code tests and the EVEN or ONLY subparameter for a procedure step by coding the COND parameter followed by the name of the procedure step to which it applies:

```
//STEP4 EXEC ANALYSIS,COND.              X
// REDUCE=((16,EQ,STEP4.LOOKUP),ONLY),...
```

Here, the cataloged procedure step named REDUCE will be executed only if a preceding job step has abnormally terminated and the procedure step named LOOKUP does not issue a return code of 16. The programmer can code as many COND parameters of this type as there are steps in the procedure.

Passing Information to the Processing Program (PARM)

For processing programs that require control information at the time they are executed, the EXEC statement provides the PARM parameter. To pass information to the program, code the keyword parameter in the operand field.

```
PARM=(option[,option]...)
```

This will pass options to the compiler, linkage editor, loader, or object program when any one of them is called by the PGM parameter in the EXEC statement or to the first step in a cataloged procedure.

To pass options to a compiler, the linkage editor, loader, or the execution step within the named cataloged procedure step, code the keyword parameter in the operand field.

```
PARM.procstep=(option[,option]...)
```

Any PARM parameter already appearing 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 the parentheses or single quotation marks that enclose the list of options. The COBOL compiler selects the valid options of the PARM field for processing by looking for three significant characters of each key option word. When the keyword is identified, it is checked for the presence or absence of the prefix NO, as appropriate. The programmer can make the most efficient use of the option field by using the significant characters instead of the entire option. Table 2 lists the significant characters for each option (see "Options for the Compiler" for an explanation of each).

Table 2. Significant Characters for Various Options

Option	Significant Characters
LINECNT	CNT
SEQ	SEQ
FLAGE(W)	LAG,LAGW
SIZE	SIZ
BUF	BUF
SOURCE	SOU
DECK	DEC
LOAD	LOA
SPACE	ACE
DMAP	DMA
PMAP	PMA
SUPMAP	SUP
CLIST	CLI
TRUNC	TRU
APOST	APO
QUOTE	QUO
XREF	XRE
BATCH	BAT
NAME	NAM
SXREF	SXR
STATE	STA
TERM	TER
NUM	NUM
FLOW	FLO
LIB	LIB
SYMDMP	SYM
OPTIMIZE	OPT
SYNTAX	SYN
CSYNTAX	CSY
RESIDENT	RES
DYNAM	DYN
SYSx	SYS
VERB	VER
ZWB	ZWB

Options for the Compiler

The IBM-supplied default options indicated by an underscore in the following discussion can be changed when the compiler is installed. The format of the PARM parameter is illustrated in Figure 6.

Notes:

- When a subparameter contains an equal sign, the entire information field of the PARM parameter must be enclosed by single quotation marks instead of parentheses, e.g., PARM='SIZE=160000,PMAP'.

- When an option and its default (such as XREF and NOXREF) are both specified, the last encountered option is generally the one assumed. (Exceptions to this rule are cited in the option descriptions.) Accordingly, the programmer may change one of the many options without repunching the entire EXEC card.

SIZE=yyyyyyy
indicates the amount of main storage, in bytes, available for compilation (see "Machine Considerations").

BUF=yyyyyy
indicates the amount of main storage to be allocated to buffers. If both SIZE and BUF are specified, the amount allocated to buffers is included in the amount of main storage available for compilation (see "Appendix D: Compiler Optimization" for information about how buffer size is determined).

Note: The SIZE and BUF compile-time parameters can be given in multiples of K, where K=1024 decimal bytes. For example, 80K is 81,920 decimal bytes.

SOURCE
NOSOURCE
indicates whether or not the source module is to be listed.

CLIST
NOCLIST
indicates whether or not a condensed listing is to be produced. If specified, the procedure portion of the listing will contain generated card numbers (unless the NUM option is in effect), verb references, and the location of the first instruction generated for each verb. Global tables, literal pools, register assignments, and information about the Working-Storage Section are also provided. CLIST and PMAP are mutually exclusive options.

Note: In nonsegmented programs, verbs are listed in source order. In segmented programs, the root segment is last. (For programs run with the OPTIMIZE option the root segment is first, followed by the individual segments in order of ascending priority.)

DMAP
NODMAP
indicates whether or not a glossary is to be listed. Global tables, literal pools, register assignments, and information about the Working-Storage Section are also provided.

PMAP
NOPMAP
indicates whether or not register assignments, global tables, literal pools, information about the Working-Storage Section, and an assembler-language expansion of the source modules are to be listed. CLIST and PMAP are mutually exclusive options.

Note: If any one of the options CLIST, DMAP, and PMAP is specified, the compiler will produce a message giving the hexadecimal length and starting address of the Working Storage Section. For an illustration of the use of these options, see the "Output" section.

VERB
NOVERB
indicates whether procedure-names and verb-names are to be listed with the associated code on the object-program listing. VERB has meaning only if PMAP or CLIST is in effect. NOVERB yields more efficient compilation.

LOAD
NOLOAD
indicates whether or not the object module is to be placed on a mass storage device or a tape volume so that the module can be used as input to the linkage editor. If the LOAD option is used, a SYSLIN DD statement must be specified.

DECK
NODECK
indicates whether or not the object module is to be punched. If the DECK option is used, a SYSPUNCH DD statement must be specified.

SEQ
NOSEQ
indicates whether or not the compiler is to check the sequence of the source module statements. If the statements are not in sequence, a message is printed.

Note: For examples of what the SOURCE, DMAP, PMAP, and SEQ options produce, see "Output."

LINECNT=nn
indicates the number of lines to be printed on each page of the compilation source card listing. The number specified by nn must be a 2-digit integer from 01 to 99. If the LINECNT option is omitted, 60 lines are printed on each page of the output listing.

Note: The compiler allows for headings three lines of what the user has specified. (For example, if nn=55 is specified, then 52 lines are printed on each page of the output listing.)

ZWB
NOZWB
indicates whether or not the compiler generates code to strip the sign from a signed external decimal field when comparing this field to an alphanumeric field. If ZWB is specified, the signed external decimal field is moved to an intermediate field, in which its sign is removed, before it is compared to the alphanumeric field. ZWB complies with the ANS standard; NOZWB should be used when, for example, input numeric fields are to be compared with SPACES.

Note: The default option cannot be changed when the compiler is installed.

FLAGW
FLAGE
indicates the type of messages that are to be listed for the compilation. FLAGW indicates that all warning and diagnostic messages are to be listed. FLAGE indicates that all diagnostic messages are to be listed, but that the warning messages are not to be listed.

SUPMAP
NOSUPMAP
indicates whether or not the object code listing, and object module and link edit decks are to be suppressed if an E-level or D-level message is generated by the compiler.

SPACE1
SPACE2
SPACE3
indicates the type of spacing that is to be used on the source card listing generated when SOURCE is specified. SPACE1 specifies single spacing, SPACE2 specifies double spacing, and SPACE3 specifies triple spacing.

TRUNC
NOTRUNC
applies to movement of COMPUTATIONAL arithmetic fields. If TRUNC (standard truncation) is specified and the number of digits in the sending field is greater than the number of digits in the receiving field, the arithmetic item is truncated to the number of digits specified in the PICTURE clause of the receiving field when moved. If NOTRUNC is specified, movement of the item is dependent on the size of the field (halfword, fullword).

QUOTE
APOST
indicates to the compiler that either the double quote (") or the apostrophe (') is acceptable as the character to delineate literals and to use that character in the generation of figurative constants.

STATE
NOSTATE
indicates whether or not the number of the COBOL statement being executed at the time of an abnormal termination is desired. STATE identifies the number of the statement and the number of the verb being executed. If the STATE option is used, a SYSDBOUT DD statement must be specified at execution time for the output data set on which the statement number message can be written. For more information, see "Debugging Facilities" in the chapter "Program Checkout."

FLOW[=nn]
NOFLOW
indicates whether or not a formatted trace is desired for a variable number of procedures executed before an abnormal termination. The number of procedures traced is specified by nn, where nn may be any integer value from one to 99. FLOW[=nn] must be specified at compile time to generate the necessary trace linkage; however, specifying nn may be deferred until execution time. If nn is omitted, thedefault value is employed. This value is either 99 or that specified at program product installation. Specifying NOFLOW at compile time precludes specification of the Flow Trace option at execution time. A SYSDBOUT DD statement must be included for the output data set on which the trace can be written. See "Options for Execution" for more information.

SYMDMP
NOSYMDMP
requests a formatted dump of the data area of the object program at abnormal termination. With this option, the programmer may request dynamic dumps of specified data-names at strategic points during program execution.

Notes:

1. If the SYMDMP option is in effect, the SYSUT5 data set must be specified.

2. If the BATCH option is requested, symbolic debugging is rejected.

3. Specification of the SYMDMP option automatically yields the OPTIMIZE feature, discussed below, and rejects the STATE option because SYMDMP output includes STATE output at abnormal termination.

For a discussion of the FLOW, STATE, and SYMDMP options, and their value to the COBOL programmer, see the chapter entitled "Symbolic Debugging Features." A SYSDBOUT, SYSDBG, and debug file DD codes are required at execution time.

OPTIMIZE
<u>NOOPTIMIZE</u>
causes optimized object code to be generated by the compiler, considerably reducing the use of object program main storage. In general, the greater the number of COBOL Procedure Division source statements, the greater the percentage of reduction in the amount of main storage required.

<u>Note</u>: The optimizer feature is automatically in effect when the SYMDMP feature is specified.

SYNTAX
CSYNTAX
<u>NOSYNTAX</u>
indicates whether the source text is to be scanned for syntax errors only and appropriate error message are to be generated. For conditional syntax checking (CSYNTAX), a full compilation is produced so long as no messages exceed the W or C level. If one or more E-level or higher severity messages are produced, the compiler generates the messages but does not generate object text.

Notes:

1. When the SYNTAX option is in effect, all of the following compile-time options are suppressed:

LOAD	PMAP	FLOW
XREF	DECK	STATE
SXREF	SYMDMP	NAME
CLIST	TRUNC	RESIDENT
NOSUPMAP	OPTIMIZE	

2. If both SYNTAX and OPTIMIZE are specified, no object code is produced.

3. Unconditional syntax checking is assumed if all of the following compile-time options are specified:

NOLOAD	NOCLIST	SUPMAP
NOXREF	NOPMAP	NODECK
NOSXREF		

NUM
<u>NONUM</u>
indicates whether or not line numbers have been recorded in the input and, rather than compiler-generated source numbers, should be used in error messages, as well as in PMAP, CLIST, STATE, XREF, SXREF, and FLOW. NONUM indicates that the compiler-generated numbers should be used in error messages as well as in PMAP, CLIST, STATE, XREF, SXREF, and FLOW.

<u>Note</u>: If when the NUM option is in effect the compiler discovers a non-numeric character in a line number or if ascending numeric sequence is broken and one or more of the debugging options are in effect, the compiler invalidates the number. The compiler then takes the last valid card number in sequence, adds a 1 to that number and begins generating card numbers from that point. The increment is 1. Six digits is the maximum sequence number. The card that follows 999999 will be flagged and NUM, SYMDMP, and test cancelled. STATE and FLOW will not be cancelled.

XREF
<u>NOXREF</u>
indicates whether or not a cross-reference listing is produced. If XREF is specified, an unsorted listing is produced with data-names and procedure-names appearing in two parts in source order.

SXREF
<u>NOSXREF</u>
indicates whether or not a sorted cross-reference listing is produced. If SXREF is specified, a sorted listing is produced with data-names and procedure-names in alphanumeric order.

<u>Note</u>: XREF and SXREF are mutually exclusive.

LIB
NOLIB
indicates whether or not a COPY and/or a BASIS request will be part of the COBOL source input stream. If no library facilities are to be used, the specification of NOLIB will save compilation time, because it avoids the opening of the SYSLIB data set.

BATCH
NOBATCH
indicates whether or not multiple programs and/or subprograms are to be compiled with a single invocation of the compiler. In the BATCH environment all compiler options specified on the EXEC card, plus all default options, will apply to every program in the batch unless specific options are overridden on the CBL card, which must be included for each program. See "Batch Compilation" for more information on batch compilations and the CBL card.

NAME
NONAME
indicates whether or not programs in a batch compilation environment will be link-edited into one or more load modules. If NAME is specified, each succeeding program in the batch will be link-edited into a separate load module. This option will remain in effect for the entire compilation unless NONAME is specified on the CBL card for an individual program. If NONAME is specified on the CBL card, no name will be generated for this compilation. Names for the load modules will be formed according to the rules for forming module names from the PROGRAM-ID. See "Batch Compilation" for more details on batch compilation and the CBL card.

Note: If the BATCH option is not specified, NONAME will be in effect.

RESIDENT
NORESIDENT
requests the COBOL Library Management feature. When one program in a given region/partition requests the RESIDENT option, the main program and all subprograms in that region/partition should also request it.

Note: The RESIDENT option is automatically in effect when the DYNAM option is invoked.

DYNAM
NODYNAM
causes subprograms invoked through the CALL literal statement to be dynamically loaded and through the CANCEL statement to be dynamically deleted at object time (instead of link-edited with the calling program into a single load module).

Note: When both NORESIDENT and NODYNAM are either specified or implied by default, and a CALL identifier statement occurs in the source statement being compiled, the COBOL Library Management Facility option (RESIDENT) is automatically in effect. A printed statement of this is given in the compiler output. (For a discussion of the COBOL Library Management Facility, see the section "Sharing COBOL Library Subroutines" in the "Libraries" chapter.)

SYST
SYSx
indicates whether SYSOUT or SYSOUx, where x must be alphanumeric (that is, 0-9 or A-Z except for T), is the ddname of the file to be used for debug output and for data when SYSOUT is specified, either implicitly or explicitly, in a DISPLAY statement. The specification in the program that is first to access the file is chosen.

Options for Use Only Under TSO

In addition to the preceding compiler options, the following options are designed for use with the Time Sharing Option (TSO). Time Sharing provides the COBOL programmer with facilities for entering, compiling, and testing programs at his terminal. (For further information on the Time Sharing Option, see the Program Product publication *IBM OS (TSO): COBOL Prompter User's Guide and Reference*.) These options are listed in Figure 6, where:

$$\text{PRINT} \begin{Bmatrix} (*) \\ (\text{dsname}) \end{Bmatrix}$$
NOPRINT
indicates whether or not the program listing is to be suppressed, placed on the output data set specified by dsname, or displayed at the terminal. If PRINT is specified, the listing will include page headings, line numbers of the statements in error, message identification numbers, severity levels, and message texts (as well as any other output requested by SOURCE, CLIST, DMAP, PMAP, XREF, or SXREF). If (*) is specified instead of data-set name, the printed output

is sent to the terminal. If PRINT alone is specified, a listing data set is created on secondary storage and named according to standard data set naming conventions. NOPRINT specifies that no listing is to be printed. If neither PRINT nor NOPRINT is specified and any one or more of the options SOURCE, CLIST, DMAP, XREF, or PMAP are specified, PRINT is the default. Otherwise, NOPRINT is the default. If PRINT is specified in a non-TSO environment, it is ignored.

TERM
<u>NOTERM</u>
indicates whether or not progress and diagnostic messages are to be printed on the SYSTERM terminal data set. The severity level of the messages may be controlled by the FLAG option. If PRINT (*) is specified, then NOTERM is the default, to ensure that messages appear only once. If TERM is specified in a non-TSO environment, the output that normally goes to the SYSTERM DD data set is written on the SYSTERM file if a SYSTERM DD card has been included. If there is no SYSTERM DD card, a warning message is issued.

Options for the Linkage Editor

MAP
indicates that a map of the load module is to be listed. If MAP is specified, XREF cannot be specified, but both can be omitted.

XREF
indicates that a cross-reference list and a module map are to be listed. If XREF is specified, MAP cannot be specified.

LIST
indicates that any linkage editor control statements associated with the job step are to be listed.

OVLY
indicates that the load module is to be in the format of an overlay structure. This option is required when the COBOL Segmentation feature is used.

The format of the PARM parameter is illustrated in Figure 6. For examples of what the MAP, XREF, and LIST options produce, see "Output." Linkage editor control statements and overlay structures are explained in "Calling and Called Programs." There are other PARM options for linkage editor processing that describe additional processing options and special attributes of the load module (see the publication <u>IBM OS Linkage Editor and Loader</u>).

Options for the Loader

MAP
<u>NOMAP</u>
indicates whether or not a map of the loaded module is to be produced that lists external names and their absolute addresses on the SYSPRINT data set. If the SYSPRINT DD statement is not used in the input deck, this option is ignored. An example of a module map is shown in "Output."

```
Compiler:
 {PARM         }  =([SIZE=yyyyyyy] [,BUF=yyyyyy]  [,SOURCE  ]  [,DMAP  ]  [,PMAP  ]
 {PARM.procstep}                                  [,NOSOURCE]  [,NODMAP]  [,NOPMAP]

                  [,SUPMAP  ]  [,LOAD  ]  [,DECK  ]  [,SEQ  ]  [,LINECNT=nn]
                  [,NOSUPMAP]  [,NOLOAD]  [,NODECK]  [,NOSEQ]

                  [,TRUNC  ]  [,CLIST  ]  [,FLAGW]  [,QUOTE]
                  [,NOTRUNC]  [,NOCLIST]  [,FLAGE]  [,APOST]

                  [,SPACE1]  [,STATE  ]  [,XREF  ]  [,SXREF  ]  ,NAME
                  [,SPACE2]  [,NOSTATE]  [,NOXREF]  [,NOSXREF]  ,NONAME
                  [,SPACE3]

                  [,BATCH  ]  [,FLOW[=nn]]  [,TERM  ]⁴  [,PRINT {(*)     }]⁴
                  [,NOBATCH]  [,NOFLOW   ]  [,NOTERM]    [       {(dsname)}]
                                                         [,NOPRINT         ]

                  [,SYMDMP  ]  [,OPTIMIZE  ]  [,SYNTAX  ]
                  [,NOSYMDMP]  [,NOOPTIMIZE]  [,CSYNTAX ]
                                              [,NOSYNTAX]

                  [,RESIDENT  ]  [,DYNAM  ]  [,VERB  ]  [,ZWB  ]  [,SYST]¹ ² ³
                  [,NORESIDENT]  [,NODYNAM]  [,NOVERB]  [,NOZWB]  [,SYSx]

Linkage Editor:
 {PARM         }  = ( [{MAP }] [,LIST] [,OVLY])
 {PARM.procstep}      [{XREF}]

Loader:
 {PARM         }  = ( [MAP  ]  [,RES  ]  [,CALL  ]  [,LET  ]  [,SIZE=100K ]
 {PARM.procstep}      [NOMAP]  [,NORES]  [,NOCALL]  [,NOLET]  [,SIZE=size]

                  [,EP=name]  [,PRINT  ]  )
                              [,NOPRINT]
Execution:
 {PARM         }  =([user parameters]  [,FLOW[=nn]]  )
 {PARM.procstep}                       [,NOFLOW   ]
```

[1]If the information specified contains any special characters, it must be delimited by single quotation marks instead of parentheses.
[2]If the only special character contained in the value is a comma, the value may be enclosed in parentheses or quotation marks.
[3]The maximum number of characters allowed between the delimiting quotation marks or parentheses is 100.
[4]These options should be used in the Time Sharing environment only.
[5]TSO-only format.

Figure 6. Compiler, Linkage Editor, and Loader PARM Options

RES
NORES
indicates whether or not an automatic search of the link pack area queue is to be made. This search is always made after processing the primary input (SYSLIN), and before searching the SYLIB data set. When the RES option is specified, the CALL option is automatically set.

CALL
NOCALL (NCAL)
indicates whether or not an automatic search of the SYSLIB data set is to be made. If the SYSLIB DD statement is not used in the input deck, this option is ignored. The NOCALL option causes an automatic NORES.

LET
NOLET
indicates whether or not the loader will try to execute the object program when a severity level 2 error condition is found.

SIZE=100K
SIZE=size
specifies the size, in bytes, of dynamic main storage that can be used by the loader. This storage must be large enough to accommodate the object program.

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

PRINT
NOPRINT
indicates whether or not diagnostic messages are to be produced on the SYSLOUT data set.

The format of the PARM parameter is illustrated in Figure 6. The default options, indicated by an underscore, can be changed at system generation with the LOADER macro instruction.

Options for Execution

Note: The programmer may want to include additional user parameters in the PARM field for the execution step of his job. These parameters are discussed below.

FLOW[=nn]
NOFLOW
If the FLOW option is specified at compile time for a trace of procedure names, at execution time a value for nn may be specified that overrides any value set at compile time. If FLOW is requested at compile time with no value for nn, a value should be specified at execution time. A default of 99 is assumed for nn if it is not specified at either step and FLOW is in effect; otherwise, nn is as previously specified. When specified at execution time, FLOW must be the last option in the PARM field. (The format of the PARM parameter is illustrated in Figure 6.)

The FLOW trace may be suppressed at execution time by specifying NOFLOW. FLOW cannot be specified as an option for execution if it is not specified at compile time or if NOFLOW is in effect by default. See the sections "Debugging Facilities" and "Options for the Compiler" for additional information.

Requesting Restart for a Job Step (RD)

The restart facilities can be used in order to minimize the time lost in reprocessing a job that abnormally terminates. These facilities permit the automatic restart of jobs that were abnormally terminated during execution.

The programmer uses this parameter to tell the operating system: (1) whether or not to take checkpoints during execution of a program, and (2) whether or not to restart a program that has been interrupted.

A checkpoint is taken by periodically recording the contents of storage and registers during execution of a program. The RERUN clause in the COBOL language facilitates taking checkpoint readings. Checkpoints are recorded onto a checkpoint data set.

Execution of a job can be automatically restarted at the beginning of a job step that abnormally terminated (step restart) or within the step (checkpoint restart). In order for checkpoint restart to occur, a checkpoint must have been taken in the processing program prior to abnormal termination. The RD parameter specifies that step restart can occur or that the action of the CHKPT macro instruction is to be suppressed.

To request that step restart be permitted or to request that the action of the CHKPT macro instruction be suppressed in a particular step, code the keyword parameter in the operand field of the EXEC statement.

```
RD=request
```

Replace the word "request" with:

R -- to permit automatic step restart. The programmer must specify at least one RERUN clause in order to take checkpoints.

NC -- to suppress the action of the CHKPT macro instruction and to prevent automatic restart. No checkpoints are taken; no RERUN clause in the COBOL program is necessary.

NR -- to request that the CHKPT macro instruction be allowed to establish a checkpoint, but to prevent automatic restart. The programmer must specify at least one RERUN clause in order to take checkpoints.

RNC -- to permit step restart and to suppress the action of the CHKPT macro instruction. No checkpoints are taken; no RERUN clause in the COBOL program is necessary.

Each request is described in greater detail in the following paragraphs.

RD=R: If the processing programs used by this step do not include a RERUN statement, RD=R allows execution to be resumed at the beginning of this step if it abnormally terminates. If any of these programs do include one or more CHKPT macro instructions (through the use of the RERUN clause), step restart can occur if this step abnormally terminates before execution of a CHKPT macro instruction; thereafter, checkpoint restart can occur.

RD=NC or RD=RNC: RD=NC or RD=RNC should be specified to suppress the action of all CHKPT macro instructions included in the programs used by this step. When RD=NC is specified, neither step restart nor checkpoint restart can occur. When RD=RNC is specified, step restart can occur.

RD=NR: RD=NR permits a CHKPT macro instruction to establish a checkpoint, but does not permit automatic restarts. However, a resubmitted job could have execution start at a specific checkpoint.

Before automatic step restart occurs, all data sets in the restart step with a status of OLD or MOD, and all data sets being passed to steps following the restart step, are kept. All data sets in the restart step with a status of NEW are deleted. Before automatic checkpoint restart occurs, all data sets currently in use by the job are kept.

If the RD parameter is omitted and no CHKPT macro instructions are executed, automatic restart cannot occur. If the RD parameter is omitted but one or more CHKPT macro instructions are executed, automatic checkpoint restart can occur.

Notes:

- If the RD parameter is specified on the JOB statement, RD parameters on the job's EXEC statements are ignored.
- Restart can occur only if MSGLEVEL=1 is coded on the JOB statement.
- If step restart is requested for this step, assign the step a unique step name.
- When this job step uses a cataloged procedure, make restart request for a single procedure step by including, as part of the RD parameter, the procedure stepname, i.e., RD.procstepname. This specification overrides the RD parameter in the named procedure step if one is present. Code as many parameters of this form as there are steps in the cataloged procedure.
- To specify a restart request for an entire cataloged procedure, code the RD parameter without a procedure stepname. This specification overrides all RD parameters in the procedure if any are present.
- If no RERUN clause is specified in the user's program, no checkpoints are written, regardless of the disposition of the RD parameter.

Reference:

- For detailed information on the checkpoint/restart facilities, see the publication IBM OS Supervisor Services.

Priority Scheduling EXEC Parameters

Establishing a Dispatching Priority (DPRTY) (MVT only)

The DPRTY parameter allows the programmer to assign to a job step, a dispatching priority different from the priority of the job. The dispatching priority determines in what sequence tasks use main storage and computing time. To assign a dispatching priority to a job step, code the keyword parameter in the operand field of the EXEC statement.

```
DPRTY=(value 1,value 2)
```

Both "value 1" and "value 2" should be replaced with a number from 0 through 15. "Value 1" represents an internal priority value. "Value 2" added to "value 1" represents the dispatching priority. The higher numbers represent higher priorities. A default value of 0 is assumed if no number is assigned to "value 1." A default value of 11 is assumed if no number is assigned to "value 2."

Notes:

- Whenever possible, avoid assigning a number of 15 to "value 1." This number is used for certain system tasks.
- If "value 1" is omitted, the comma must be coded before "value 2" to indicate the absence of "value 1," e.g., DPRTY=(,14).
- If "value 2" is omitted, the parentheses need not be coded, e.g., DPRTY=12.
- On an MVT system with time-slicing facilities, the DPRTY parameter can be used to make a job step part of a group of job steps to be time-sliced. The priorities of the time-sliced groups are selected at system generation. To cause the job step to be time-sliced, assign to "value 1" a number that corresponds to a priority number selected for time-slicing. "Value 2" is either omitted or assigned a value of 11.
- When the step uses a cataloged procedure, a dispatching priority can be assigned to a single procedure step by including the procedure step name in the DPRTY parameter, i.e., DPRTY.procstepname=(value 1, value 2). This parameter may be used for each step in the cataloged procedure.
- To assign a single dispatching priority to an entire cataloged procedure, code the DPRTY parameter without a procedure step name. This specification overrides all DPRTY parameters in the procedure if there are any.

Setting Job Step Time Limits (TIME)

To assign a limit to the computing time used by a single job step, a cataloged procedure, or a cataloged procedure step, code the keyword parameter in the operand field of the EXEC statement.

```
TIME=(minutes,seconds)
```

Such an assignment is useful in a multiprogramming environment where more than one job has access to the computing system. Minutes and seconds represent the maximum number of minutes and seconds allotted for execution of the job step.

Notes:

- If the job step requires use of the system for 24 hours (1440 minutes) or longer, the programmer should specify. TIME=1440. Using this number suppresses timing. The number of seconds cannot exceed 59.
- If the time limit is given in minutes only, the parentheses need not be coded; e.g., TIME=5.
- If the time limit is given in seconds, the comma must be coded to indicate the absence of minutes; e.g., TIME=(,45).
- When the job step uses a cataloged procedure, a time limit for a single procedure step can be set by qualifying the keyword TIME with the procedure step name; i.e., TIME.procstep=(minutes,seconds). This specification overrides the TIME parameter in the named procedure step if one is present. As many parameters of this form can be coded as there are steps in the cataloged procedure.
- To set a time limit for an entire procedure, the TIME keyword is left unqualified. This specification overrides all TIME parameters in the procedure if any are present.
- If this parameter is omitted, the standard job step time limit is assigned.

Specifying Main Storage Requirements for a Job Step (REGION)
(MVT only)

The REGION parameter permits the programmer to specify the size of the main storage region to be allocated to the associated job step. The REGION parameter specifies:

- The maximum amount of main storage to be allocated to the job. This amount must include the size of those components required by the user's program that are not resident in main storage.
- The amount of main storage to be allocated to the job, and the storage hierarchy or hierarchies in which the space is to be allocated. This request should be made only if main storage hierarchy support has been specified during system generation. If an IBM 2361 Core Storage, Model 1 or 2, is present in the system, processor storage is referred to as hierarchy 0 and 2361 Core Storage is referred to as hierarchy 1. If 2361 Core Storage is not present but main storage hierarchy support was specified in system generation, a two-part region is established in processor storage when a region is defined to exist in two hierarchies. The two parts are not necessarily contiguous.

To specify a region size, code the keyword parameter in the operand field of the EXEC statement.

```
REGION=(nnnnnxK[,nnnnnyK])
```

To request the maximum amount of main storage required by the job, replace the term "nnnnnx" with the maximum number of contiguous 1024-byte areas allocated to the job step, e.g., REGION=52K. This number can range from 1 to 5 digits but must not exceed 16383.

To request a region size and the hierarchy desired, the term "nnnnnx" is replaced with the number of contiguous 1024-byte areas to be allocated to the job in hierarchy 0; the term "nnnnny" is replaced with the number of contiguous 1024-byte areas to be allocated in hierarchy 1, e.g., REGION=(60K,200K). When only processor storage is used to include hierarchies 0 and 1, the combined values of nnnnnx and nnnnny cannot exceed 16383. If 2361 Core Storage is present, nnnnnx cannot exceed 16383 and, for a 2361 Model 1, nnnnny cannot exceed 1024, or 2048 for a 2361 Model 2. Each value specified should be an even number. (If an odd number is specified, the system treats it as the next higher even number.)

If storage is requested only in hierarchy 1, a comma must be coded to indicate the absence of the first subparameter, e.g., REGION=(,200K). If storage is requested only in hierarchy 0, or if hierarchy support is not present, the parentheses need not be coded, e.g., REGION=70K.

If the REGION parameter is omitted or if a region size smaller than the default region size is requested, it is assumed that the default value is that established by the input reader procedure.

Notes:

- Region sizes for each job step can be coded by specifying the REGION parameter in the EXEC statement for each job step. However, if a REGION parameter is present in the JOB statement, it overrides REGION parameters in EXEC statements.
- If main storage hierarchy support is not included but regions are requested in both hierarchies, the region sizes are combined and an attempt is made to allocate a single region from processor storage. If a region is requested entirely from hierarchy 1, an attempt is made to allocate the region from processor storage.
- For information on storage requirements to be considered when specifying a region size, see the publication IBM OS Storage Estimates.

Specifying Additional Main Storage for a Job Step (ROLL)
(MVT only)

To allocate additional main storage to a job step whose own region does not contain any more available space, code the keyword parameter in the operand field of the EXEC statement.

```
ROLL=(x,y)
```

In order to allocate this additional space to a job step, another job step may have to be rolled out, i.e., temporarily transferred to secondary storage. When x is replaced with YES, the job step can be rolled out; when x is replaced with NO, the job step cannot be rolled out. When y is replaced with YES, the job step can cause rollout; when y is replaced with NO, the job step cannot cause rollout. (If additional main storage is required for the job step, YES must be specified for y.) If this parameter is omitted, ROLL=(YES,NO) is assumed.

Notes:

- If the ROLL parameter is specified in the JOB statement, the ROLL parameter in the EXEC statements is ignored.

- When a job step uses a cataloged procedure, it can be indicated whether or not a single procedure step has the ability to be rolled out and to cause rollout of another job step. To indicate this, the procedure stepname, i.e., ROLL.procstepname, is included as part of the ROLL parameter. This specification overrides the ROLL parameter in the named procedure step, if one is present. As many parameters of this form can be coded as there are steps in the cataloged procedure.

- To indicate whether or not all of the steps of a cataloged procedure have the ability to be rolled out and to cause rollout of other job steps, the ROLL parameter can be coded without a procedure stepname. This specification overrides all ROLL parameters in the procedure, if any are present.

DD STATEMENT

The data definition (DD) statement identifies each data set that is to be used in a job step, and it furnishes information about the data set. The DD statement specifies input/output facilities required for using the data set; it also establishes a logical relationship between the data set and input/output references in the program named in the EXEC statement for the job step.

Figure 7 is a general format of the DD statement.

Parameters used most frequently for COBOL programs are discussed in detail. The other parameters (e.g., SEP and AFF) are mentioned briefly. For further information, see the publication IBM OS Job Control Language Reference.

Name	Operation	Operand
// {ddname / procstep.ddname} [1]	DD	(see below and next page)

Operand[2]

```
        Positional Parameters

  [ *     ]  3
  [ DATA  ]
  [ DUMMY ]

        Keyword Parameters  4 5

 [DDNAME=ddname]
 [                   { dsname                        } ]  11
 [                   { dsname(element)               } ]
 [ {DSNAME}          { *.ddname                      } ]
 [ {      }    =     { *.stepname.ddname             } ]
 [ {DSN   }          { *.stepname.procstep.ddname    } ]
 [                   { &&name                        } ]
 [                   { &&name(element)               } ]
 [QNAME=processname]
 [        [ dsname                     ]                           ]  6
 [ DCB=(  [   *.ddname                 ]   [,subparameter-list])   ]
 [        [ *.stepname.ddname          ]                           ]
 [        [ *.stepname.procstep.ddname ]                           ]
 [ SEP=(subparameter list) 7 ]  10
 [ AFF=ddname                ]

          Positional Subparameters    Keyword Subparameters

 [ UNIT=(name[,[n/P][,DEFER]][,SEP=(list of up to 8 ddnames)]) 8 ]  10 12
 [ UNIT=(AFF=ddname)                                             ]
                               Positional Subparameters

            { TRK                   }
 SPACE=(    { CYL                   }    ,(primary-quantity[,secondary-quantity],
            { average-record-length }

                                                      [,MXLG  ]
                  [directory- or index-quantity])[,RLSE]  [,ALX   ]  [, ROUND])
                                                      [,CONTIG]

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

              { CYL                   }
 SPLIT=(n,    {                       }  ,(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 }
```

Figure 7. The DD Statement (Part 1 of 2)

```
Operand² (cont.)

                Positional Subparameters
 { VOLUME }
 { VOL    }     =([PRIVATE],[RETAIN],[volume sequence number],[volume count])

                Keyword Subparameters

 [ ,SER=(volume-serial-number[volume-serial-number]⁹...)         ]
 [                                                               ]
 [            { dsname                          }                ]
 [ ,REF=      { *.ddname                        }                ]
 [            { *.stepname.ddname               }                ]
 [            { *.stepname.procstep.ddname      }                ]

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

 [           [NEW]      [,DELETE  ]   [,DELETE  ]      ]
 [           [OLD]      [,KEEP    ]   [,KEEP    ]      ]
 [ DISP=(    [SHR]      [,PASS    ]   [,CATLG   ]   )  ]
 [           [MOD]      [,CATLG   ]   [,UNCATLG ]      ]
 [                      [,UNCATLG ]                    ]
 [                                                     ]
 [ SYSOUT=classname                                    ]
 [ SYSOUT=(x[,program-name][,form-no.])                ]
```

[1]The name field must be blank when concatenating data sets.
[2]All parameters are optional to allow a programmer flexibility in the use of the DD statement; however, a DD statement with a blank operand field is meaningless.
[3]If the positional parameter is specified, keyword parameters other than DCB cannot be specified.
[4]If subparameter-list consists of only <u>one</u> subparameter and no leading comma (indicating the omission of a positional subparameter) is required, the delimiting parentheses may be omitted.
[5]If subparameter-list is omitted, the entire parameter must be omitted.
[6]See "User-Defined Files" for the applicable subparameters.
[7]See the publication <u>IBM OS Job Control Language Reference</u>.
[8]If only name is specified, the delimiting parentheses may be omitted.
[9]If only one volume-serial-number is specified, the delimiting parentheses may be omitted.
[10]The SEP and AFF parameters should not be confused with the SEP and AFF subparameters of the UNIT parameter.
[11]The value specified may contain special characters if the value is enclosed in apostrophes. If the only special character used is the hyphen, the value need not be enclosed in apostrophes. If DSNAME is a qualified name, it may contain periods without being enclosed in apostrophes.
[12]The unit address may contain a slash, and the unit type number may contain a hyphen, without being enclosed in apostrophes, e.g., UNIT=293/5,UNIT=2400-2.
[13]The QNAME= parameter is used in COBOL teleprocessing and must be the name of a TCAM destination queue.

Figure 7. The DD Statement (Part 2 of 2)

Name Field

ddname (Identifying the DD Statement)
is used:

- To identify data sets defined by this DD statement to the compiler or linkage editor (see "Compiler Data Set Requirements" and "Linkage Editor Data Set Requirements").

- To relate the data sets defined in this DD statement to a file described in a COBOL source program (see "User-Defined Files").

- To identify this DD statement to other control statements in the input stream.

procstep.ddname
is used to alter or add DD statements in cataloged procedures. The step in the cataloged procedure is identified by procstep. The ddname identifies either one of the following:

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

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

Operand Field

* (Defining Data in an Input Stream)
indicates that data immediately follows this DD statement in the input stream. This parameter is used to specify a source deck or data in the input stream. If the EXEC statement specifies execution of a program, only one data set may be placed in the input stream. The end of the data set must be indicated by a delimiter statement. The data cannot contain // or /* in the first two characters of any record. The DD * statement must be the last DD statement of the job step. In MVT, for a step with a single input stream data set, DD * and a /* statement are not required. The system will supply both if missing. The default DDNAME will be SYSIN.

DATA (Defining Data in an Input Stream)
also indicates a source deck or data in the input stream. If the EXEC statement specifies execution of a program, only one data set may be placed in the input stream. The end of the data set must be indicated by a delimeter statement. The data cannot contain /* in the first two characters of any record. The DD DATA statement must be the last DD statement of the job step. // may appear in the first and second positions in the record, for example, when the data consists of control statements of a procedure that is to be cataloged.

DUMMY (Bypassing Input/Output Operations on the Data Set)
allows the user's processing program to operate without performing input/output operations on the data set. The DUMMY parameter is valid only for sequential data sets to which reference is made by the basic sequential or queued sequential file processing techniques. If the DUMMY parameter is specified, a read request results in an end of data set exit. A write request is recognized, but no data is transmitted. No device allocation, external storage allocation, or cataloging takes place for dummy data sets.

In multiprogramming environments, data in the input stream is temporarily transferred to a direct-access device for later high-speed retrieval. Normally, the reader procedure assigns a blocking factor for the data when it is placed on the direct-access device. The programmer may assign his own values through use of the BLKSIZE parameter of the DCB parameter. He may also indicate the number of buffers to be assigned to transmitting the data, through use of the BUFNO parameter. For example, he may assign the following:

```
DCB=(BLKSIZE=800,BUFNO=2)
```

If the programmer omits these parameters or assigns values greater than the capacity of the input reader, it is assumed that the established default values for the reader are in effect.

DDNAME Parameter (Postponing the Definition of a Data Set)
defines a pseudo data set that will assume the characteristics of a real data set if a subsequent DD statement of the step is labeled with the specified ddname. When the DDNAME parameter is specified, it must be the first parameter in the operand. All other parameters are ignored and should be omitted when the DDNAME parameter appears (see "Using the Cataloged Procedures").

DDNAME Subparameter

ddname

names a DD statement that, if present, supplies the attributes of the data set. If it is not present, the statement is ignored.

DSNAME Parameter (Identifying the Data Set)

allows the programmer to specify the name of the data set to be created or to refer to a previously created data set. Various types of names can be specified (see "Using the DD Statement" for a discussion of the various names) as follows:

- Fully qualified names: For data sets to be retrieved from or stored in the system catalog.
- Generation data group names: For an entire generation data group, or any single generation thereof.
- Simple names: For data sets that are not cataloged.
- Reference names: For data sets whose names are given in the DSNAME parameter of another DD statement in the same job.
- Temporary names: For temporary data sets that are to be named for the duration of one job only.

If the DSNAME parameter is omitted, the operating system assigns a unique name to the data set. (This parameter should be supplied for all except temporary data sets to allow future referencing of the data set.) DSNAME may be coded DSN.

DSNAME Subparameters

dsname

specifies the fully qualified name of a data set. This is the name under which the data set can be cataloged or otherwise identified on the volume.

dsname(element)

specifies a particular generation of a generated data group, a member of a partitioned data set, or an area of an indexed data set. To indicate a generation of a generated data group, the element is a zero or a signed integer. To indicate a member of a partitioned data set, the element is a name. To indicate an area of an indexed data set, the element is PRIME, OVFLOW, or INDEX (see "Using the DD Statement" for information about generation data groups and examples of partitioned data sets).

*.ddname

indicates that the DSNAME parameter (only) is to be copied from a preceding DD statement in the current job step.

*.stepname.ddname

indicates that the DSNAME parameter (only) is to be copied from the DD statement, ddname, that occurred in a previous step, stepname, in the current job. If this form of the subparameter appears in a DD statement of a cataloged procedure, stepname refers to a previous step of the procedure, or, if no such step is found, to a previous step of the current job.

*.stepname.procstep.ddname

indicates that the DSNAME parameter (only) is to be copied from a DD statement in a cataloged procedure. The EXEC statement that called for execution of the procedure, as well as the step and DD statement of the procedure, must be identified.

&&name

allows the programmer to supply a temporary name for a data set that is to be deleted at the end of the job. The operating system substitutes a unique symbol for this subparameter. The programmer can use the temporary name in other steps to refer to the data set. The same symbol is substituted for each recurrence of this name within the job. Upon completion of the job, the name is dissociated from the data set. The same temporary name can be used in other jobs without ambiguity.

&&name(element)

allows the programmer to supply a name for a member of a temporary partitioned data set that will be deleted at the end of the step.

QNAME Parameter (Defining the Data to be Accessed by TCAM)

specifies the name of a TPROCESS macro that defines a destination queue for messages that are to be processed by an application program and creates a process entry for the queue in the Terminal Table (see the section "Defining Terminal and Line Control Areas" in the chapter entitled "Using the Teleprocessing Feature").

Note: The DCB parameter is the only parameter that can be coded on a DD statement with the QNAME parameter. The only operands that may be specified as subparameters are BLKSIZE, BUFL, LRECL, OPTCD, and RECFM.

DCB Parameter (Describing the Attributes of the Data Set)
: allows the programmer to specify at execution time, rather than at compilation time, information for completing the data control block associated with the data set (see "Execution Time Data Set Requirements" and "Additional File Processing Information" for further information about the data control block and DCB subparameters).

The first subparameter of this parameter may be used to copy DCB attributes from the data set label of a cataloged data set or from a preceding DD statement (see the publication IBM OS Supervisor and Data Management Macro Instructions for detailed information about the DCB subparameter).

SEP and AFF Parameters (Optimizing Channel Usage)
: allow the programmer to optimize the use of channels among groups of data sets. SEP indicates channel separation and AFF indicates channel affinity.

If neither parameter is supplied, any available channel, consistent with the UNIT parameter requirement, is assigned. The affinity parameter groups two or more data sets so that they can be separated from another data set requesting channel separation. For indexed sequential data sets these parameters are written in the same way as those for any data set. They can be used in succeeding DD statements to refer to the first DD statement defining an indexed sequential data set. However, the second and third DD statements cannot request separation from or affinity to one another because they are unnamed. Thus, to establish channel separation and affinity for all of the areas, the name subparameter of the UNIT parameter must be used to request specific devices on specific channels.

UNIT Parameter (Requesting a Unit)
: specifies the quantity and types of input/output devices to be allocated for use by the data set.

If the UNIT parameter is not specified in the current DD statement, there are several ways in which the unit information may be inferred by the system:

- If the current data set has already been created and it is either being passed to the current step, or if it has been cataloged, any unit name specified in this DD statement is ignored.
- If the REF subparameter of the VOLUME parameter is specified, the current data set is given affinity with the data set referred to; that data set's defining DD statement provides the unit information.
- If the current data set is to operate in the split cylinder mode with a previously defined data set, it will reside on the unit specified in the DD statement for the previous data set.
- If the current data set is to use space suballocated from that assigned to a previously defined data set, it will reside on the same unit as the data set from which the space is obtained.
- If the current data set is assigned to the standard output class (SYSOUT is specified), it is written on the unit specified by the operator for class A.

If the current data set is in the input stream (defined by a DD * or DD DATA statement), the DD statement defining the data set should not contain a UNIT parameter.

If this parameter specifies a mass storage device for a data set being created, it is also necessary to reserve the space the data set will occupy, using another parameter of the DD statement. Depending on the way in which the space will be used, the SPACE, SPLIT, or SUBALLOC parameter can be specified. These parameters are discussed under individual headings.

If the UNIT parameter specifies a tape device, no SPACE, SPLIT, or SUBALLOC parameters are required.

The UNIT parameter must be specified if VOLUME=SER is specified in the DD statement.

UNIT Subparameters:

name
: specifies the name of an input/output device, a single cell within a data cell drive, a device class name, or any meaningful combination of input/output devices specified by an installation. (Mass storage devices and magnetic tape devices can be combined. No other device type combination is allowed.) Names and device classes are defined at system generation time. The device class names that are required for IBM cataloged procedures and are normally used by most installations are shown in Figure 8. These names can be specified by the installation at system generation time.

The block size specified in the source program (in the BLOCK CONTAINS clause or in the record description) must not exceed the maximum block size permitted for the device. For example, the maximum block size for the IBM 2311 is 3625 characters, and the maximum block size for the IBM 2400 series is 32,760 characters.

Note: When device-independence is specified by use of UT as the device class in the ASSIGN statement in the Environment Division, the device chosen by the system will be dependent on the DD statement. Therefore, if the user's installation has both an IBM 2311 and an IBM 2302 that may be used as utility devices, the user should write

BLOCK CONTAINS 3625 CHARACTERS

(or any number smaller than 3625) to ensure that the block can be contained on one track.

n
: specifies the number of devices to be allocated to the data set. If this parameter is omitted, 1 is assumed.

P
: specifies parallel mount.

DEFER
: indicates deferred mounting. Deferred mounting cannot be specified for a new output data set on a mass storage device or for an indexed data set.

SEP=(list of up to eight ddnames)
: specifies unit separation.

AFF=ddname
: specifies unit affinity.

Class Name	Class Functions	Device Type
SYSSQ	writing reading	mass storage magnetic tape
SYSDA	writing reading	mass storage

Figure 8. Device Class Names Required for IBM-Supplied Cataloged Procedures

SPACE Parameter (Allocating Mass Storage Space)
: specifies space to be allocated in a mass storage volume. Although SPACE has no meaning for tape volumes, if a data set is assigned to a device class that contains both mass storage devices and tape devices, SPACE should be specified.

Two forms of the SPACE parameter may be used, with or without absolute track address (ABSTR). The ABSTR parameter requests that allocation begin at a specific address.

SPACE Subparameters:

$\left\{\begin{array}{l}\text{ABSTR}\\\text{TRK}\\\text{CYL}\\\text{average-record-length}\end{array}\right\}$
: specifies the unit of measurement in which storage is to be assigned. The units may be tracks (ABSTR or TRK), cylinders (CYL), or records (average-record-length, expressed as a decimal number). In addition, the ABSTR subparameter indicates that the allocated space is to begin at a specific track address. If the specified tracks are already allocated to another data set, they will not be reallocated to this data set.

Note: For indexed data sets, only the CYL or ABSTR subparameter is permitted. When an indexed data set is defined by more than one DD statement, all must specify either CYL or ABSTR; if some statements contain CYL and others ABSTR, the job will be abnormally terminated.

(primary-quantity[,secondary-quantity] [,directory- or index-quantity])
: specifies the amount of space to be allocated for the data set. The

primary quantity indicates the number of records, tracks, or cylinders to be allocated when the job step begins. For indexed data sets, this subparameter specifies the number of cylinders for the prime, overflow, or index area (see "Execution Time Data Set Requirements"). The secondary quantity indicates how much additional space is to be allocated each time previously allocated space is exhausted. This subparameter must not be specified when defining an indexed data set. If a secondary quantity is specified for a sequential data set, the program may receive control when additional space cannot allocated to write a record. The directory quantity is used when initially creating a partitioned data set (PDS), and it specifies the number of 256-byte records to be reserved for the directory of the PDS. It can also specify the number of cylinders to be allocated for an index area embedded within the prime area when a new indexed data set is being defined (see the publication IBM OS Job Control Language Reference).

Note: The directory contains the name and the relative position, within the data set, for each member of a partitioned data set. The name requires eight bytes, the location four bytes. Up to 62 additional bytes can be used for additional information. For a directory of a partitioned data set that contains load modules, the minimum directory requirement for each member is 34 bytes.

RLSE
: indicates that all unused external storage assigned to this data set is to be released when processing of the data set is completed.

{ MXIG | ALX | CONTIG }
: qualifies the request for the space to be 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 the allocation of additional tracks in the volume. The operating system will allocate tracks in up to five blocks of storage, each block equal to or greater than the primary quantity. CONTIG requests that the space indicated in the primary quantity be contiguous.

If this 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. It can be used only when average record length is specified as the first subparameter.

quantity
: specifies the number of tracks to be allocated. For an indexed data set, this quantity must be equivalent to an integral number of cylinders; it specifies the space for the prime, overflow, or index area (see "Execution Time Data Set Requirements").

beginning address
: specifies the relative number of the track desired, where the first track of a volume is defined as 0. (Track 0 cannot be requested.) The number is automatically converted to an address based on the particular device assigned. For an indexed data set this number must indicate the beginning of a cylinder.

directory quantity
: defines the number of 256-byte records to be allocated for the directory of a new partitioned data set. It also specifies the number of tracks to be allocated for an index area embedded within the prime area when a new indexed data set is being defined. In the latter case, the number of tracks must be equivalent to an integral number of cylinders (see the publication IBM OS Job Control Language Reference).

SPLIT Parameter (Allocating Mass Storage Space)
: is specified when other data sets in the job step require space in the same mass storage volume, and the user wishes to minimize access-arm movement by sharing cylinders with the other data sets. The device is then said to be operating in a split cylinder mode. In this mode, two or more data sets are stored so that portions of each occupy tracks within every allocated cylinder.

Note: SPLIT should not be used when one of the data sets is an indexed data set.

SPLIT Subparameters:

n
: indicates the number of tracks per cylinder to be used for this data set if CYL is specified. If the average record length is specified, n is the percentage of the tracks per cylinder to be used for this data set.

{CYL | average-record-length}
: indicates the units in which the space requirements are expressed in the next subparameter. The units may be cylinders (CYL) or physical records (in which case the average record length in bytes is specified as a decimal number not exceeding 65,535). If the average record length is given, and the data set is defined to have a key, the key length must be given in the DCB parameter of this DD statement.

primary-quantity
: defines the number of cylinders or space for records to be allocated to the entire group of data sets.

secondary-quantity
: defines the number of cylinders or space for records to be allocated each time the space allocated to any of the data sets in the group has been exhausted and more data is to be written. This quantity will not be split.

A group of data sets that share cylinders in the same device is defined by a sequence of DD statements. The first statement in the sequence must specify all parameters except secondary quantity, which is optional. Each of the statements that follow the first statement must specify only n, the amount of space required.

SUBALLOC Parameter (Allocating Mass Storage Space)
: permits space to be obtained from another data set for which contiguous space was previously allocated. This enables data sets to be stored in a single volume. Space obtained through suballocation is removed from the original data set, and may not be further suballocated. The SUBALLOC parameter should not be used to obtain space for an indexed data set.

Except for the subparameters described below, the subparameters in the SUBALLOC parameter have the same meaning as those described in the SPACE parameter.

SUBALLOC Subparameters:

ddname
: indicates that space is to be suballocated from the data set defined by the DD statement, ddname, that appears in the current step.

stepname.ddname
: indicates that space is to be suballocated from the data set defined by the DD statement, ddname, occurring in a previous step, stepname. If this form of the subparameter appears in a DD statement in a cataloged procedure, stepname refers to a previous step of the procedure, or if no such step is found, to a previous step of the current job.

stepname.procstep.ddname
: indicates that space is to be suballocated from a data set defined in a cataloged procedure. The first term identifies the step that called for execution of the procedure, the second identifies the procedure step, and the third identifies the DD statement that originally requested space.

VOLUME (VOL) Parameter (Specifying Volume Information)
: specifies information about the volume(s) on which an input data set resides, or on which an output data set will reside. A volume can be a tape reel, or a mass storage device. Volumes can be used most efficiently if the programmer is familiar with the states a volume can assume. Volume states involve two criteria: the type of data set the programmer is defining and the manner in which the programmer requests a volume.

Data sets can be classified as one of two types, temporary or nontemporary. A temporary data set exists only for the duration of the step that creates it. A nontemporary data set can exist after the job is completed. The programmer indicates that a data set is temporary by coding:

• DSNAME=&&name

• No DSNAME parameter

• DISP=(NEW,DELETE), either explicitly or implied, e.g., DISP=(,DELETE)

• DSNAME=reference, referring to a DD statement that defines a temporary data set.

All other data sets are considered nontemporary. If the programmer attempts to keep or catalog a passed data set that was declared temporary, the system changes the disposition to PASS unless DEFER was specified in theUNIT parameter. Such a data set is deleted at the end of the job.

The manner in which the programmer requests a volume can be considered <u>specific</u> or <u>nonspecific</u>. A specific reference is implied whenever a volume with a specific serial number is requested. Any one of the following conditions denotes a specific volume reference:

• The data set is cataloged or passed from an earlier job step.

• VOLUME=SER is coded in the DD statement.

• VOLUME=REF is coded in the DD statement, referring to an earlier specific volume reference.

All other types of volume references are nonspecific. (Nonspecific references can be made only for new data sets, in which case the system assigns a suitable volume.)

The state of a volume determines when the volume will be demounted and what kinds of data sets can be assigned to it.

<u>Mass Storage Volumes</u>: Mass storage volumes differ from tape volumes in that they can be shared by two or more data sets processed concurrently by more than one job. Because of this difference, mass storage volumes can assume different volume states than tape volumes. The volume state is determined by one characteristic from each of the following groups:

Mount Characteristics	Allocation Characteristics
Permanently Resident	Public
Reserved	Private
Removable	Storage

<u>Permanently resident</u> volumes are always mounted. The permanently resident characteristic applies automatically to:

• All physically permanent volumes, such as 2301 Drum Storage.

• The volume from which the system is loaded (the IPL volume).

• The volume containing the system data sets SYS1.LINKLIB, SYS1.PROCLIB, and SYS1.SYSJOBQE.

• Other volumes can be designated as permanently resident in a special member of SYS1.PROCLIB named PRESRES.

Permanently resident volumes are always public. The <u>reserved</u> characteristic applies to volumes that remain mounted until the operator issues an UNLOAD command. They are reserved by a MOUNT command referring to the unit on which they are mounted or by a PRESRES entry. The <u>removable</u> characteristic applies to all volumes that are neither permanently resident nor reserved. Removable volumes do not have an allocation characteristic when they are not mounted. A reserved volume becomes removable after an UNLOAD command is issued for the unit on which it resides.

The allocation characteristics, public, private, and storage, indicate the availability status of a volume for assignment by the system to temporary data sets, and, if the volume is removable, when it is to be demounted. A <u>public</u> volume is used primarily for temporary data sets and, if it is permanently resident, for frequently used data sets. It must be requested by a specific volume reference if a data set is to be kept or cataloged on it. If a public volume is removable, it is demounted only when its unit is required by another volume. The programmer can change a public volume to private status by specifying VOLUME=PRIVATE. A <u>private</u> volume must be requested by a specific volume reference. A new data set can be assigned to a private volume by specifying VOLUME=PRIVATE. If the volume is reserved, it remains mounted until the operator issues an UNLOAD command for the unit on which it resides. If it is removable, it will be demounted after it is used, unless the programmer specifically requested that it be retained (VOLUME=,RETAIN) or passed (DISP=,PASS). Once a removable volume has been made private, it will ultimately be demounted. To use it as a public volume, it must be remounted. A storage volume is used as an extension of main storage, to keep or catalog nontemporary data sets having nonspecific volume requests. The programmer can assign the PRIVATE option to storage volumes.

Table 3 shows how mass storage volumes are assigned their mount and allocation characteristics.

Table 3. Mass Storage Volume States

Mount Characteristic	Allocation Characteristic		
	Public	Private	Storage
Permanently Resident	PRESRES or Default	PRESRES	PRESRES
Reserved	PRESRES or MOUNT command	PRESRES or MOUNT command	PRESRES or MOUNT command
Removable	Default	VOLUME= PRIVATE	na
na = Not applicable			

Magnetic Tape Volumes: The volume state of a reel of magnetic tape is also determined by a combination of mount and allocation characteristics:

Mount Characteristics	Allocation Characteristics
Reserved	Private
Removable	Scratch

The reserved-scratch combination is not a valid volume state. Reserved tape volumes assume their state when the operator issues a MOUNT command for the unit on which they reside. They remain mounted until the operator issues a corresponding UNLOAD command. Reserved tapes must be requested by a specific volume reference.

A removable tape volume is assigned the private characteristic when one of the following occurs:

- It is requested with a specific volume reference.
- It is requested for allocation to a nontemporary data set.
- The VOLUME parameter is coded with the PRIVATE option.

A removable-private volume is demounted after its last use in the job step, unless the programmer requests that it be retained.

All other tape volumes are assigned the removable-scratch state. The tape volumes remain mounted until their unit is required by another volume.

Volume Parameter Facilities: The facilities of the VOLUME parameter allow the programmer to:

- Request private volumes (PRIVATE)
- Request that private volumes remain mounted until the end of the job (RETAIN)
- Select volumes when the data set resides on more than one volume (volume-sequence-number)
- Request more than one nonspecific volume (volume-count)
- Identify specific volumes (SER and REF)

These facilities are all optional. The programmer can omit the VOLUME parameter when defining a new data set, in which case the system assigns a suitable public or scratch volume.

VOLUME Subparameters:

PRIVATE
indicates that the volume on which space is being allocated to the data set is to be made private. If the PRIVATE, SER, and REF subparameters are omitted for a new output data set, the system assigns the data set to any suitable public or scratch volume that is available.

RETAIN
indicates that this volume is to remain mounted after the job step is completed. Volumes are retained so that data may be transmitted to or from the data set, or so that other data sets may reside in the volume. If the data set requires more than one volume, only the last volume is retained; the other volumes are previously dismounted. Another job step indicates when to dismount the volume by omitting RETAIN. If each job step issues a RETAIN for the

volume, the retained status lapses when execution of the job is completed.

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

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

SER
: specifies one or more serial numbers for the volumes required by the data sets. A volume serial number consists of one to six alphanumeric characters. If it contains fewer than six characters, the serial number is left justified 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 is not required for new output data sets. Two volumes should not have the same serial number. When the SER parameter is included, the volume is treated as PRIVATE commencing with allocation for the current job step. If this subparameter is specified, the UNIT parameter must also be specified.

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 4 shows the data set references.

When the data set resides in a tape volume and REF is specified, the data set is placed in the same volume, immediately behind the data set referred to by this subparameter. When the REF subparameter is used, the UNIT and LABEL parameters, if supplied, are ignored.

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

<u>LABEL Parameter (Describing Data Set Label)</u>
: specifies information about the label or labels associated with the data set. If a data set is passed from a previous job step, label information is retained from the DD statement that specified DISP=(,PASS). A LABEL parameter, if specified in the DD statement receiving the passed data set, is ignored. If the LABEL parameter is omitted and the data set is not being passed, standard labeling is assumed. The operating system verifies mounting when the label parameter specifies standard labels (SL) or standard and user labels (SUL). Nonstandard labels can be specified only when installation-written routines to write and process nonstandard labels have been incorporated into the operating system (see "User Label Processing" and the publication <u>IBM System/360 Operating System: Systems Programmer's Guide</u> for information about writing these routines).

<u>LABEL Subparameters</u>:

data-set-sequence-number
: is a 4-digit number that identifies the relative location of the data set

Table 4. 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 cataloged procedure step procstep called in the job step stepname (see "Using the Cataloged Procedures")

with respect to the first data set in a tape volume. (For example, if there are three data sets in a magnetic tape volume, the third data set is identified by data set sequence number 0003.) If the data set sequence number is not specified, the operating system assumes that it is 0001. (This option should not be confused with the volume sequence number, which represents a particular volume for a data set.)

$\left\{\begin{matrix} \text{NL} \\ \underline{\text{SL}} \\ \text{NSL} \\ \text{SUL} \end{matrix}\right\}$
specifies the kind of label used for the data set. NL indicates no labels. SL indicates standard labels. NSL indicates nonstandard label. SUL indicates standard and user labels.

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

PASSWORD
indicates that the data set is to be made accessible only when the correct password is issued by the operator. The operating system assigns security protection to the data set. In order to retrieve the data set, the operator must issue the password on the console.

<u>DISP Parameter (Specifying Data Set Status and Disposition)</u>
describes the status of a data set and indicates what is to be done with it after its last use, or at the end of the job. The job scheduler executes the requested disposition functions at the completion of the associated job step. If the step is not executed because of an error found by the system before trying to initiate the step (e.g., an error in a job control language statement), the remaining statements are read and interpreted; however, none of the succeeding steps are executed, and the requested dispositions are not performed. This parameter can be omitted for data sets created and deleted during a single job step. Additional information about the relationship between the DISP parameter and the volume table of contents is contained in "Additional File Processing Information."

<u>DISP Subparameters</u>:

NEW
indicates that the data set is being generated in this step. If the status is omitted, the NEW subparameter is assumed.

OLD
indicates that the data set specified in the DSNAME parameter already exists.

SHR
has meaning only in a multiprogramming environment for existing data sets that reside on mass storage volumes. This subparameter indicates that the data set is part of a job in which operations do not prevent simultaneous use of the data set by another job. For a data set that is to be shared, the DD statement DISP parameter should be specified as DISP=SHR for every reference to the data set in a job. Unless this is done, the data set cannot be used by a concurrently operating job, and the job will have to wait until the particular file is free.

MOD
causes logical positioning after the last record in the data set. It indicates that the data set already exists and that it is to be added to, rather than read. When MOD is specified and neither the volume serial number is given nor the data set cataloged or passed from an earlier job step, MOD is ignored and NEW is assumed. If the volume serial number is given, it is assumed that the data set is on the specified volume.

DELETE
causes the space occupied by the data set to be released for other purposes at the end of the current step. If the data set is cataloged, and the catalog is used to locate it, reference to the data set is removed from the catalog. If it is on a mass storage device, all references are removed from the volume table of contents, and the device space is made available for use by other data sets. If the data set is on tape, the volume in which the data set resides is then available for use by other data sets.

KEEP
ensures that the data set remains intact until a DELETE parameter is exercised in either the current job or

some subsequent job. If the data set is on a mass storage device, it remains tabulated in the volume table of contents after completion of the job. When the volume containing the data set is to be dismounted, the operator is advised of the disposition.

PASS
: indicates that the data set is to be referred to in a later step of the current job, at which time its disposition may be determined. When a subsequent reference to this data set is encountered, its PASS status lapses unless another PASS is issued. The final disposition of the data set should be specified in the last DD statement referring to the data set within the current job.

  While a data set is in PASS status, the volume(s) on which it resides are, in effect, retained; that is, the system will attempt to avoid demounting them. If demounting is necessary, the system will ensure proper remounting, through operator messages. The unit name specified on the DD statement in the receiving step must be consistent with the unit name in the passing step.

CATLG
: causes the creation, at the end of the job step, of an index entry in the system catalog pointing to the data set. The data set can be referred to by name in subsequent jobs, without the need for volume serial number or device type information from the programmer. Cataloging also implies KEEP.

UNCATLG
: causes the index entry that points to this data set to be removed from the index structure at the end of this step. The data set is not deleted. If it is on a mass storage volume, reference to it remains in the volume table of contents.

  Note: The absence of DELETE, KEEP, PASS, CATLG, and UNCATLG indicates that no special action is to be taken to alter the permanent or temporary status of this data set. If the data set was created in this job, it will be deleted at the end of the current step. If the data set existed before this job, it will be kept.

The third subparameter indicates the disposition of the data set in the event the job step terminates abnormally. This is the conditional disposition subparameter. Explanations for DELETE, KEEP, CATLG, and UNCATLG are the same as those for normal termination. The following points should be noted when using the third subparameter.

- If a conditional disposition is not specified and the job step abnormally terminates, the requested disposition (the second subparameter) 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 at that time, all new data sets are deleted and all other data sets are kept.
- A conditional disposition other than DELETE for a temporary data set is invalid and the system assumes that it is DELETE.

SYSOUT Parameter (Routing Data Set through the Output Stream)
: schedules a printing or punching operation for the data set described by the DD statement.

SYSOUT Subparameters:

classname
: specifies the system output class on which the data set is to be written. A classname is an installation specified 1-character name designating the output class to which the data set is to be written. Each classname is related to a particular output unit. Valid values for the SYSOUT parameter are A through Z and 0 through 9. A is the standard output class. Both data sets and system messages can be routed through the same output stream when using a priority scheduler. In this case, the output class selected for the data sets must be the same output class as that selected for the MSGCLASS parameter in the JOB statement.

  Note: Classes 0 through 9 should not be used except in cases where the other classes are not sufficient. These classes are intended for future features of systems using priority schedulers.

(x[,program-name][,form-no])
is used for priority scheduling systems only. When priority schedulers are used, the data set is usually written on an intermediate mass storage device during program execution, and later routed through an output stream to a system output device. The x can be an alphabetic or numeric character 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 mass storage device and an estimate of the space required. If there is a special installation program to handle output operations, its program-name should be specified. Program-name is the member name of the program, which must reside in the system library. If the output data set is to be printed or punched on a specific type of output form, a 4-digit form number should be specified. Form-no. is used to instruct the operator of the form to be used in a message issued at the time the data set is to be printed.

Notes:

- If both the program-name and form-no. are omitted, the delimiting parentheses can be omitted.
- If the Direct SYSOUT Writer is used to write a data set, both the form-no. and program-name are ignored. All parameters on the DD statement, i.e., UNIT or SPACE, are also ignored.

ADDITIONAL DD STATEMENT FACILITIES

By specifying certain ddnames, the programmer can request the operating system to perform additional functions. The operating system recognizes these special-purpose ddnames:

- JOBLIB and STEPLIB to identify private user libraries
- SYSABEND and SYSUDUMP to identify data sets on which a dump may be written

JOBLIB AND STEPLIB DD STATEMENTS

The JOBLIB and STEPLIB DD statements are used to concatenate a user's private library with the system library (SYS1.LINKLIB). Use of JOBLIB results in the system library being combined with the private library for the duration of a job; use of STEPLIB, for the duration of a job step. During execution, the library indicated in these statements is scanned for a module before the system library is searched.

The JOBLIB DD statement must appear immediately after the JOB statement and its operand field must contain at least the DSNAME and DISP parameters. The DISP parameter must contain PASS as the second subparameter if the library is to be made available to later job steps. Only one JOBLIB statement may be specified for a job but more than one library may be specified on a JOBLIB statement. The JOBLIB statement is meant to concatenate existing private libraries with the system library. It need not be specified for load modules created in the job or for permanent members of the system library (see "Checklist for Job Control Statements" and "Libraries" for examples).

The STEPLIB DD statement may appear in any position among the DD statements for the job step. The library should be defined as OLD. If the library is to be passed to other job steps, the second subparameter of the DISP parameter should be coded PASS. A later job step may then refer to the library by coding its STEPLIB DD statement as follows:

```
//STEPLIB DD DSNAME=*.stepname.STEPLIB, X
//    DISP=(OLD,PASS)
```

The STEPLIB statement overrides the JOBLIB statement if both are present in a job step.

SYSABEND AND SYSUDUMP DD STATEMENTS

The ddnames SYSABEND or SYSUDUMP identify a data set on which an abnormal termination dump may be written. The dump is provided for job steps subject to abnormal termination.

The SYSABEND DD statement is used when the programmer wishes to include in his dump the problem program storage area, the system nucleus, and the trace table if the trace table option had been requested at system generation time.

The SYSUDUMP DD statement is used when the programmer wishes to include only the problem program storage area.

The programmer may rout the dump directly to an output writer by specifying the SYSOUT parameter on the DD statement. In a multiprogramming environment, the programmer may also define the intermediate direct-access device by specifying the UNIT and SPACE parameters.

PROC STATEMENT

The PROC statement may appear as the first control statement in a cataloged procedure and must appear as the first control statement in an in-stream procedure. The PROC statement must contain the term PROC in its operation field. For a cataloged procedure, the PROC statement assigns default values to symbolic parameters defined in the procedure; its operand field must contain symbolic parameters and their default values. The PROC statement marks the beginning of an in-stream procedure; its operand may contain symbolic parameters and their default values.

PEND STATEMENT

The PEND statement must appear as the last control statement in an in-stream procedure and marks the end of the in-stream procedure. It must contain the term PEND in the operation field. The PEND statement is not used for cataloged procedures. For further information about in-stream procedures, see "Testing a Procedure as an In-Stream Procedure" in "Using the Cataloged Procedures."

COMMAND STATEMENT

The operator issues commands to the system via the console or a command statement in the input stream. Commands can also be issued to the system via a command statement in the input stream. However, this should be avoided since commands are executed as they are read (except for SET and START in systems with PCP) and may not be synchronized with execution of job steps. Command statements must appear immediately before a JOB statement, an EXEC statement, a null statement, or another command statement.

The command statement contains identifying characters (//) in columns 1 and 2, a blank name field, a command, and, in most cases, an operand field. The operand field specifies the job name, unit name, or other information being considered.

Note: A command statement cannot be continued, it must be coded on one card or card image.

DELIMITER STATEMENT

The delimiter statement marks the end of a data set in the input stream. The identifying characters /* must be coded into columns 1 and 2, the other fields are left blank. Comments are coded as necessary.

Note: When using a system with MFT or MVT, the end of a data set need not be marked in an input stream that is defined by a DD * statement.

NULL STATEMENT

The null statement is used to mark the end of certain jobs in an input stream. If the last DD statement in a job defines data in an input stream, the null statement should be used to mark the end of the job so that the card reader is effectively closed. The identifying characters // are coded into columns 1 and 2, and all remaining columns are left blank.

COMMENT STATEMENT

The comment statement is used to enter any information considered helpful by the programmer. It may be inserted anywhere in the job control statement stream after the JOB Statement. (The comment statement contains a slash in columns 1 and 2, and an asterisk in column 3. The remainder of the card contains comments.) Comments are coded in columns 4 through 80, but a comment may not be continued onto another statement.

When the comment statement is printed on an output listing, it is identified by the appearance of asterisks in columns 1 through 3.

BATCH COMPILATION

The batch compile feature is used to compile multiple programs or subprograms with one invocation of the compiler. The object programs produced from the batch compilation may be link-edited into either one load module or separate load modules.

This feature must be requested at compile time by specification of BATCH in the PARM field or, if a cataloged procedure is used, in the PARM.COB field of the EXEC card. In the BATCH mode, all options specified on the EXEC card, as well as all default options, apply to every program in the batch unless specific options are overridden, via the CBL card, for an individual compilation.

The CBL card must be the first card in each program within a batch mode. The CBL card, used to specify additional compiler options or to change existing options for that individual program, has the following format:

```
CBL  [option 1]   [,option 2]
```

The letters CBL may appear in any three consecutive columns 1 through 72, and the option(s) specified may be any PARM compiler option(s) except SIZE, BUF, and BATCH, which are ignored if indicated.

Notes:

- A sequence number may appear in columns 1 through 6 of the CBL card.
- Any option given on the CBL card overrides options on the EXEC card for this compilation only, except where the option requires the use of a file desired in a subsequent compilation (for example, LOAD and SYSLIN). In such a case, the option must be specified either as a default or as an option on the EXEC card. This is not to imply that the option cannot be negated on any CBL card when it is not desired.
- If a CBL card is present and BATCH is not specified on the EXEC card, the CBL card is regarded as an invalid statement.
- If the compiler NAME option is specified on the CBL card, a linkage editor NAME control card is generated for this compilation, facilitating the link-editing of the program into a separate load module.
- The output of a batch compilation may be executed immediately only if it is made up of a single load module (for example, a main program and subprograms). In order for this load module to be executed, the member name specified at compile time must be specified at execution time.
- The batch option may be used in conjunction with BASIS. This facility provides the COBOL programmer with the ability to combine a (multiple) BASIS library member(s) and/or a (multiple) COBOL source program(s) with one invocation of the compiler.
- The BATCH option and the SYMDMP option are mutually exclusive.

When the batch option is used in combination with BASIS, the following rules apply:

1. All the BASIS library members to be compiled must be members of the partitioned data set(s) referred to by the SYSLIB DD data set name(s).
2. Each BASIS library member must contain only one source program.

Figure 9 shows that with one invocation of the COBUCL cataloged procedure (see the chapter "Using the Cataloged Procedures"), the programs COMPILE1, COMPILE2, and COMPILE3 are compiled and two load modules created as follows:

1. COMPILE1 and COMPILE2 are link-edited together to form one load module with the member name of COMPILE2, a typical called/calling situation. (For further discussion of articulation between COBOL programs, see the chapter "Called and Calling Programs".) In this case, the entry point of the load module is still the first program, COMPILE1.
2. COMPILE3 is link-edited to create the load module with the member name of COMPILE3.

Figure 10 shows that with one invocation of the COBUCL procedure the programs PROG1 and PROG2 and BASIS library members PAYROLL and PAYROLL2 are compiled and four load modules are created. (An example of how to execute load modules created with the BATCH feature using the procedure COBUCL is given in Figure 9.)

```
//jobname          JOB          1,BATCH,MSGLEVEL=1
//COMPILE          EXEC¹        COBUCL,PARM.COB='BATCH,NAME'
//COB.SYSIN        DD           *
 CBL NONAME
     ID DIVISION.
     PROGRAM-ID.   COMPILE1.
         .
         .
CBL NAME
     ID DIVISION.
     PROGRAM-ID.   COMPILE2.
         .
         .
CBL NAME
     ID DIVISION.
     PROGRAM-ID.   COMPILE3.
         .
         .
/*
//LKED.SYSLMOD     DD           DSN=BATCHRUN,SPACE=(TRK,(10,5,2)),....
/*
//COMPILE2         EXEC         PGM=COMPILE2
//STEPLIB²         DD           DSN=BATCHRUN²,DISP=SHR,....
//  (Cards needed to execute COMPILE1 and COMPILE2)
/*
//COMPILE3         EXEC         PGM=COMPILE3
//STEPLIB          DD           DSN=BATCHRUN,DISP=SHR,....
//  (Cards needed to execute COMPILE3)
/*
```

[1]In the compile step, no special JCL is needed for SYSLIN because the COBUCL cataloged procedure is used (see the chapter "Using The Cataloged Procedures").

[2]In the link-edit step, a partitioned data set is created with the DSN of BATCHRUN.

Figure 9. Example of a Batch Compilation

```
//jobname          JOB          1,BATBASIS,MSGLEVEL=1
//COMP             EXEC         COBUCL,PARM.COB='BATCH,NAME,LIB'
//COB.SYSLIB       DD           DSN=LIBPOS,...¹
//COB.SYSIN        DD           *
 CBL               NAME,NOLIB
                   IDENTIFICATION DIVISION.
                   PROGRAM-ID. PROG1.
 CBL               NAME,LIB
BASIS              PAYROLL²
 CBL               NAME,LIB
BASIS              PAYROLL2
 CBL               NAME,NOLIB
                   IDENTIFICATION DIVISION
                   PROGRAM-ID. PROG2.
                        .
                        .
/*
//LKED.SYSLMOD³    DD           DSN=BATCHBAS,SPACE=(TRK,(10,5,2)),...
/*
```

[1]This partitioned data set contains as separate members PAYROLL and PAYROLL2.

[2]Example as shown in Figures 76-78.

[3]The load modules of these four COBOL programs exist as separate members of a partitioned data set named BATCHBAS.

Figure 10. Creation of Four Load Modules with Programs PROG1 and PROG2 and BASIS Library Members PAYROLL and PAYROLL2

DATA SET REQUIREMENTS

COMPILER

Eleven data sets may be defined for a compilation job step; six of these (SYSUT1, SYSUT2, SYSUT3, SYSUT4, SYSIN, and SYSPRINT) are required. A seventh data set SYSUT5, is required if the SYMDMP option is invoked. The other three data sets (SYSLIN, SYSPUNCH, and SYSLIB) are optional.

For compiler data sets other than utility data sets, a logical record size can be specified by using the LRECL and BLKSIZE subparameters of the DCB parameter. The values specified must be permissible for the device on which the data set resides. LRECL equals the logical record size, and BLKSIZE equals LRECL multiplied by n, where n is equal to the blocking factor. If this information is not specified in the DD statement, it is assumed that the logical record sizes for the unblocked data sets have the following default values:

Unblocked Data Set	Default Value (bytes)
SYSIN	80
SYSLIN	80
SYSPUNCH	80
SYSLIB	80
SYSPRINT	121
SYSTERM	121

Note: When using the SYSUT1, SYSUT2, SYSUT3, SYSUT4, SYSUT5, SYSPRINT, SYSPUNCH, or SYSLIN data sets, the following should be considered: If the primary space allocated for the data set is insufficient when compiling large programs, an area of core storage may be used to complete processing. This area would be used for an extra data extent block (DEB) and would be in the middle of the compiler's required core. Therefore, enough contiguous space may not be available to load a compiler phase. Such a condition will result in an abnormal termination of the job. The programmer should therefore attempt to allocate sufficient primary space to eliminate the need for a secondary allocation of space. See the Program Product publication IBM OS Full American National Standard COBOL Compiler and Library, Version 3, Installation Reference Material for information on storage estimates for compile data sets.

The ddname that must be used in the DD statement describing the data set appears as the heading for each description that follows. Table 5 lists the function, device requirements, and allowable device classes for each data set. (See "Appendix D: Compiler Optimization" for further information on blocked compiler data sets other than utility data sets.)

SYSUT1, SYSUT2, SYSUT3, SYSUT4, SYSUT5

The DD statements using these ddnames define utility data sets that are used by the compiler when processing the source module. The data set defined by the SYSUT1 DD statement must be on a mass storage device. Except for SYSUT5, which is needed at execution time, these data sets are temporary and have no connection with any other job step. For example, the DD statement

```
//SYSUT1 DD UNIT=SYSDA,SPACE=(TRK,(40,10))
```

specifies that the data set is to be written on any available mass storage device, with a primary allocation of 40 tracks. Additional tracks, if required, are to be allocated in groups of 10. The data set is to be deleted at the end of the job step (by default).

SYSIN

The data set defined by the SYSIN DD statement contains the input for the compiler, i.e., the source module statements that are to be processed. The input/output device assigned to this data set can be either the device transmitting the input stream (the device designated as SYSIN at system generation time) or a device designated by the programmer. When using a cataloged procedure, the DD statement describing this data set usually appears in the input stream. For example,

```
//SYSIN    DD    *
```

specifies that the input data set follows in the input stream. If the asterisk or DATA convention is used, the SYSIN DD statement must be the last DD statement in the job step.

SYSPRINT

This data set is used by the compiler to produce a listing. Output may be directed to a printer, a mass storage device, or a magnetic-tape device. The listing will include the results of the default or specified options of the PARM parameter (i.e., diagnostic messages, the object code listing). For example, in the DD statement

```
//SYSPRINT  DD  SYSOUT=A
```

SYSOUT is the disposition for printer data sets, and A is the standard output class for printer data sets.

SYSPUNCH

The data set defined by the SYSPUNCH DD statement is used to punch an object module deck. This data set can be directed to a card punch, mass storage device, or magnetic tape. For example, in the DD statement

```
//SYSPUNCH  DD  SYSOUT=B
```

SYSOUT is the disposition for punch data sets, and B is the standard output class for punch data sets.

Note: The SYSPUNCH DD statement is not required if NODECK is in effect. SYSPUNCH may be either a sequential data set or a member of a PDS.

SYSLIN

The device defined by the SYSLIN DD statement is used by the compiler to store an object module. It may be on a mass storage or magnetic tape device. For example:

```
//SYSLIN DD  DSNAME=&&GOFILE,                 X
//             DISP=(MOD,PASS),               X
//             UNIT=SYSDA,                    X
//             SPACE=(TRK,(30,10))
```

The temporary name of the data set is GOFILE, the parameter DISP=(MOD,PASS) indicates that the data is to be created or added to in this job step and is to be passed to another job step, which may be the linkage editor step. The device to be assigned for storage is a mass storage device on which 30 tracks are initially allocated to the data set. If more space is needed, tracks are allocated 10 at a time.

Note: The SYSLIN DD statement is not required if NOLOAD is in effect. SYSLIN may be either a data set or a member of a PDS.

SYSLIB

The SYSLIB DD statement defines the library (PDS) that contains the data requested by a COPY statement (in the source module) or by a BASIS card in the input stream. Note that more than one partitioned data set may be used for the library function by concatenating them with SYSLIB (see "Libraries" for an example). Libraries must always be on mass storage devices. Only one SYSLIB statement may be used in a compilation job step. For example, in the DD statement

```
//SYSLIB  DD  DSNAME=USERLIB,DISP=OLD
```

the name of the library is USERLIB, and DISP=OLD indicates that the library has been created in a previous job and is cataloged, or has been created in a previous step in this job. No other information need be given if the specified library has been cataloged.

Note: The SYSLIB DD statement is not required if NOLIB is in effect.

Table 5. Data Sets Used for Compilation

ddname	Type	Function	Device Requirements	Allowable Device Classes
SYSIN (required)	Input/output	Reading the source program	Card reader Intermediate storage	SYSSQ, SYSDA, or the input stream device (specified by DD * or DD DATA)
SYSPRINT (required)		Writing the storage map, listings, and and messages	Printer Intermediate storage	SYSSQ, SYSDA, standard output class A
SYSPUNCH (optional)		Punching the object module deck	Card punch Mass storage Magnetic tape	SYSCP, SYSSQ, SYSDA, standard output class B
SYSLIN (optional)		Creating an object module data set as output from the compiler and input to the linkage editor	Mass storage Magnetic tape	SYSSQ, SYSDA
SYSUT1 (required)	Utility	Work data set needed by the compiler during compilation	Mass storage	SYSDA
SYSUT2 (required)		Work data set needed by the compiler during compilation	Mass storage Magnetic tape	SYSSQ, SYSDA
SYSUT3 (required)		Work data set needed by the compiler during compilation	Mass storage Magnetic tape	SYSSQ, SYSDA
SYSUT4 (required)		Work data set needed by the compiler during compilation	Mass storage Magnetic tape	SYSSQ, SYSDA
SYSUT5		Work data set needed when the SYMDMP option is in effect	Mass storage Magnetic tape	SYSSQ, SYSDA
SYSLIB (optional)	Library	Optional user source program library	Mass storage	SYSDA

Note: Once created, a SYSUT5 data set can be moved only to a device of the same class. That is, if the SYSUT5 data set is put on tape at compile time, that data set cannot be moved to a disk at execution time. The SYSUT5 data set must be unblocked.

LINKAGE EDITOR

Five data sets are required for linkage editor processing. Others may be necessary if secondary input is specified. In the following discussions, the ddname that must be used in the DD statement describing the data set appears as the heading for each description of the particular data set. For any user-defined data set, the ddname is defined by the programmer. Table 6 lists the function, device requirements, and allowable device classes for each data set.

SYSLIN

The SYSLIN DD statement defines the data set that is primary input to linkage editor processing. Normally this data set consists of the output from a previous compilation job step. The primary input may also be linkage editor control statements, such as the INCLUDE, LIBRARY, or OVERLAY statements (see "Calling and Called Programs"). The input device assigned to this data set is either the device transmitting the input stream, if

Table 6. Data Sets Used for Linkage Editing

ddname	Type	Function	Device Requirements	Allowable Device Classes
SYSLIN (required)	Input/ output	Primary input data, normally the output of the compiler	Mass storage Magnetic tape Card reader	SYSSQ, SYSDA, or the input stream device (specified by DD * or DD DATA)
SYSPRINT (required)		Diagnostic messages Informative messages Module map Cross-reference list	Printer Intermediate storage	SYSSQ, standard output class A
SYSLMOD (required)		Output data set for the load module	Mass storage	SYSDA
SYSUT1 (required)	Utility	Work data set	Mass storage	SYSDA
SYSLIB (required) for COBOL Library subroutines	Library	Automatic call library (SYS1.COBLIB is the name of the COBOL subroutine library)	Mass storage	SYSDA
User-specified (optional)	--	Additional object modules and load modules	Mass storage Magnetic tape	SYSDA, SYSSQ

the input is an object module deck, or a device designated by the programmer. However, the data set may simply be passed from the previous compilation job step. For example, in the DD statement

```
//SYSLIN    DD  DSNAME=*.STEPNAME.SYSLIN,   X
//              DISP=(OLD,DELETE)
```

the data set is defined in the SYSLIN DD statement contained in the compiler job step, STEPNAME. DISP=(OLD,DELETE) indicates that the data set was created in a previous job step and is to be deleted at the end of this job step.

SYSPRINT

The data set defined by the SYSPRINT DD statement is used by the linkage editor to produce a listing. For example:

```
//SYSPRINT  DD  SYSOUT=A
```

Output may be directed to a printer or to magnetic tape. The listing may include any options specified by the PARM parameter of the EXEC statement (a module map or cross reference list, diagnostic or informative messages, etc.).

SYSLMOD

The SYSLMOD DD statement defines the output data set, in this case the load module. The load module must be placed in a library as a named member. The library can be the Link Library (SYS1.LINKLIB) or a private user-defined library. Such libraries must always reside on a mass storage device, and space for the library is allocated when the library is created. For example, in the DD statement

```
//SYSLMOD DD DSNAME=SYS1.LINKLIB(MEMBER),X
//             DISP=OLD
```

the load module, MEMBER, is stored as a member of the link library. DISP=OLD indicates that the library is already created and additions are to be made to it.

```
//SYSLMOD   DD   DSNAME=LIB1(BALANCE),         X
//               DISP=(NEW,CATLG),             X
//               VOLUME=SER=111111,            X
//               SPACE=(TRK,(40,10,1)),        X
//               UNIT=SYSDA
```

The load module, BALANCE, is to be a member of a library, LIB1, which is to be created in this job step, with BALANCE as its first member. The mass storage volume to which it is directed is identified by the serial number, 111111. A primary quantity of 40 tracks is allocated to the library with an additional allocation for one 256-byte record to be used for the directory. If more space is needed for the library, tracks are added, 10 at a time. (However, no additional space can be allocated for the directory.)

Note: If the load module is placed in a private library, the JOBLIB DD statement must be specified in subsequent jobs that execute load modules from the library.

SYSUT1

The SYSUT1 DD statement defines a utility data set used by the linkage editor when processing object modules and load modules. The data set must be on a mass storage device. It is a temporary data set and has no connection with any other job step. For example:

```
//SYSUT1 DD UNIT=SYSDA,SPACE=(TRK,(40,10))
```

The data set is initially allocated 40 tracks on any available mass storage device. If more space is needed, tracks are added, 10 at a time. A temporary name is assigned to the data set for the job step.

SYSLIB

The SYSLIB DD statement assigns the named partitioned data set to the automatic call library from which modules may be automatically obtained by the linkage editor to resolve external references.

```
//SYSLIB  DD  DSNAME=SYS1.COBLIB,DISP=SHR
```

This statement assigns the COBOL subroutine library to the automatic call library. When there is a possibility that the compiler may have generated calls to any COBOL library subroutines, the SYSLIB statement must be specified (see "Appendix B: COBOL Library Subroutines" for a list of library subroutines, their functions, and entry points).

Note: The SYSLIB statement can also define a sequential data set (see "Libraries").

User-Specified Data Sets

Additional data sets may be defined for linkage editor processing. These data sets may be used as additional input sources of object modules or load modules. They may also be concatenated with the primary input data set or the automatic call library (see "Libraries").

LOADER

One data set (SYSLIN) is required for loader processing. Two are optional (SYSLIB, SYSLOUT). (These ddnames can be changed during system generation with the LOADER macro instruction.) In addition, any DD statements and data required by the loaded program must be included in the input deck.

In the following discussions, the default ddname for the DD statement describing the data set appears as the heading for each description of the particular data set.

SYSLIN

The SYSLIN DD statement defines the data set that is primary input to the loader. This input can be either object modules produced by the COBOL compiler or load modules produced by the linkage editor, or both. The loader allows both object module and load module concatenation on SYSLIN. The data sets defined by the SYSLIN DD statements can be either sequential data sets or members of a partitioned data set.

SYSLIB

The SYSLIB DD statement defines the data set containing IBM or user-written library routines to be included in the loaded program. The SYSLIB data set is searched when unresolved references remain after processing SYSLIN and, optionally, searching the link pack area of MVT or the resident reusable routine of MFT. The library may contain either object modules or load modules but not both. The data set defined by the SYSLIB DD statement must be a partitioned data set.

SYSLOUT

The SYSLOUT DD statement defines the data set used for error and warning messages and for an optional map of external references. The record format of SYSLOUT must be FA, FBA, or FBSA.

EXECUTION TIME DATA SETS

Any number of data sets may be used for execution time processing. These data sets, or files, are identified in the source program, and each must be described by a DD statement. The ddname is used to link the DD statement to the COBOL ASSIGN clause in the source program that specifies the ddname. DD statement requirements for the DISPLAY, ACCEPT, EXHIBIT, and TRACE statements are discussed in the following text. DD statements that specify COBOL debugging aids and an abnormal termination dump are also discussed. Use of either the Sort or the RERUN feature requires additional DD statements. For information about these statements, see "Using the Sort Feature" and "Using the Checkpoint/Restart Feature."

DISPLAY Statement

The DISPLAY statement requires an associated DD statement unless the data is to be displayed on the console. The DD statements needed for each form of the DISPLAY statement are as follows:

Example 1:

$$\text{DISPLAY} \left\{ \begin{array}{l} \text{identifier} \\ \text{literal} \end{array} \right\} \ldots \text{UPON SYSPUNCH}$$

//SYSPUNCH DD applicable parameters

It is assumed that SYSPUNCH is an unblocked data set that has a logical record length of 80 characters. For example:

```
//SYSPUNCH DD SYSOUT=B
```

However, the programmer can specify a blocked data set by using the subparameters of the DCB parameter as follows:

RECFM=FB,BLKSIZE=n*80

where:

n is the blocking factor

SYSPUNCH must be on a device where blocking is permitted. For example:

```
//SYSPUNCH    DD    UNIT=SYSSQ,              X
//                  DCB=(RECFM=FB,           X
//                  BLKSIZE=160),            X
//                  LABEL=(,NL)
```

When the UPON option is omitted, SYSOUT is the default option.

Example 2:

$$\text{DISPLAY} \left\{ \begin{array}{l} \text{identifier} \\ \text{literal} \end{array} \right\} \ldots$$

//SYSOUT DD applicable parameters

It is assumed that SYSOUT is an unblocked data set that has a line width of 121 characters (1-byte per control character).

For example:

```
//SYSOUT DD SYSOUT=A
```

However, the programmer can specify an alternate line width, recording mode, and/or a blocked data set by using the DCB parameter. To specify an alternate line width, the subparameters of the DCB parameter are used as follows:

LRECL=line width+1,BLKSIZE=LRECL value

To specify a blocked data set, the subparameters are used as follows:

RECFM=FBA,LRECL=line width+1,
BLKSIZE=n*(LRECL value),

where:

n is a blocking factor

SYSOUT must be on a device where blocking is permitted. The extra character in LRECL allows for the carriage control character. For example, to specify an alternate line width, the following SYSOUT statement can be used.

```
//SYSOUT       DD SYSOUT=A,DCB=(LRECL=133, X
//                BLKSIZE=133)
```

To specify a blocked data set, the following SYSOUT statement can be used.

```
//SYSOUT       DD    DSNAME=PRINTOUT,          X
//                   UNIT=SYSDA,...,           X
//                   DCB=(RECFM=FBA,           X
//                   LRECL=121,                X
//                   BLKSIZE=605),             X
//                   VOLUME=SER=111111
```

The DISPLAY statement can use a mnemonic-name rather than a system-name.

Example 3:

```
         {identifier}
DISPLAY  {          }  ...UPON mnemonic-name
         {literal   }
```

where mnemonic-name is associated with the word SYSPUNCH or SYSOUT in the Environment Division.

```
   {SYSPUNCH}
// {        }  DD applicable parameters
   {SYSOUT  }
```

ACCEPT Statement

The ACCEPT statement requires an associated DD statement unless either the data is being accepted from the console or format 2 of the ACCEPT statement is used (making possible use of the options DATE, DAY, and TIME). The DD statements for each form of the ACCEPT statement are as follows:

Example 1:

```
     ACCEPT identifier
```

When the FROM option is omitted, SYSIN is the default option.

```
//SYSIN   DD   applicable parameters
```

Example 2:

```
     ACCEPT identifier FROM mnemonic-name
```

where mnemonic-name is associated with the word SYSIN in the Environment Division.

```
//SYSIN DD applicable parameters
```

It is assumed that SYSIN is an unblocked data set that has a logical record length of 80 characters. For example:

```
//SYSIN  DD  *
          (data)
/*
```

However, the programmer can specify a blocked data set by using the subparameters of the DCB parameter as follows:

```
RECFM=FB,BLKSIZE=n*80
```

where:

n is the blocking factor

SYSIN must be on a device where blocking is permitted. For example:

```
//SYSIN        DD    UNIT=2400,...,            X
//                   DCB=(RECFM=FB,            X
//                   BLKSIZE=160),             X
//                   LABEL=(,NL)
```

If a logical record length of other than 80 characters is desired, it must be specified in the LRECL field of the DCB parameter.

EXHIBIT or TRACE Statement

The EXHIBIT or TRACE statement requires a SYSOUT DD statement as discussed for DISPLAY.

Note: If the job step already includes a SYSOUT DD statement for some other use, another may not be inserted since all SYSOUT output from any source in the job step will be merged onto the one SYSOUT data set defined for that job step.

COBOL Debugging Aids

If one or more of the options FLOW, STATE, and SYMDMP is in effect, the following DD statement must be used:

```
//SYSDBOUT   DD   applicable parameters
```

If the output is routed through the output stream and written on a system output device, the following may be used:

```
//SYSDBOUT   DD   SYSOUT=A
```

The recording mode is FBA. The user can, however, specify a blocked data set and alternate recording mode by using the DCB subparameters.

Note: It is assumed that SYSDBOUT is an unblocked data set that has a line width of 121 bytes (one byte for a control character).

Abnormal Termination Dump

To obtain an operating system hexadecimal dump in case the job is abnormally terminated, one of the following DD statements must be used:

```
//SYSABEND DD applicable parameters.

//SYSUDUMP DD applicable parameters.
```

The dump provided when the SYSABEND DD statement is used includes the system nucleus, the 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 the program storage area. The applicable parameters are those for a standard sequential data set. If the dump is routed through the output stream and written on a system output device, the following DD statement may be used:

```
//SYSUDUMP DD SYSOUT=A
```

Note: If a COBOL program abnormally terminates, then a formatted dump is provided for all COBOL programs compiled with the SYMDMP option which could include the abnormally terminating program and its callers, up to and including the main program. The //SYSABEND or //SYSUDUMP DD card need not be included. For a discussion of the symbolic dumping option, as well as of other COBOL symbolic debugging options, see the chapter entitled "Symbolic Debugging Features."

COBOL Subroutine Library

The user should concatenate a library of selected COBOL object-time subroutines with the link library as soon as the compiler is installed. (For information on how this can be accomplished, see the section "Sharing COBOL Library Subroutines" in the chapter entitled "Libraries").

USER-DEFINED FILES

Files that are processed in a COBOL program must be described as data sets to the operating system. Whenever a file is specified in a program by the following statement:

```
SELECT [OPTIONAL] file-name
    ASSIGN TO system-name
```

this file must be described in an FD file-name entry and in a DD statement in the execution-time job step. The ddname in the DD statement is a portion of the system-name specified in the ASSIGN TO clause. In the system-name

```
UT-2400-S-TAXRATE
```

TAXRATE is the ddname portion of the system-name.

Note: The device-number specified in the system-name is ignored used by the compiler. Actual device allocation is a function of the DD statement.

FILE NAMES AND DATA SET NAMES

The terms "file" (COBOL usage) and "data set" (operating system usage) have essentially the same meaning. There may, however, be a difference between the file-name and the data set name. The data set name always represents a specific data set. The file-name can, at different times, represent different data sets. The DD statement allows a programmer to select, at the time his program is executed, the specific data set that is to be associated with a particular file-name. This facility can be especially powerful when applied to input data sets.

The file-name is a name known within the COBOL program. Changing a file-name requires changing input/output statements and recompiling the program. Changing a DD statement when a program is executed is a simple procedure.

As an example, consider a COBOL program that might be used in exactly the same way for several different master files. It might contain the clause

```
SELECT MASTER ASSIGN TO
    DA-2302-D-MASTERA... .
```

In that case, the following DD statements, used at different times, would assign the different named data sets to the program:

```
//MASTERA   DD  DSNAME=MASTER1,...
//MASTERA   DD  DSNAME=MASTER2,...
//MASTERA   DD  DSNAME=MASTER3,...
```

If the first DD statement appears in the job step that calls for execution of the program, any reference within the program to MASTER is a reference to the data set named MASTER1; if the second DD statement appears, the reference is to MASTER2; if the third, the reference is to MASTER3.

However, if a file-name within a program is always to be applicable to only a single data set, the names might be written as follows:

```
SELECT TAXRATE ASSIGN TO
    UT-2400-S-TAXRATE...
```

The applicable DD statement might be:

```
//TAXRATE   DD   DSNAME=TAXRATE,...
```

Of the names, the ddname portion of the system-name that appears in the ASSIGN clause and the ddname of the DD statement must always be the same. The file-name and the data set name may be the same, or they may be different. (Of course, the file-name in the SELECT sentence must be the same as the FD name.)

If two or more files on direct-access devices have the same ddname and are open at the same time (i.e., the output from the files is being merged into one data set), the files must have no conflicting attributes. The foregoing also applies to SYSOUT data sets if they are written on an intermediate direct-access device.

The use of the DISPLAY, EXHIBIT, or READY TRACE verbs causes the compiler to open its own file whose ddname is SYSOUT. If the programmer has also assigned one of his output files to SYSOUT, he must ensure that he has opened, written, and closed his file before the first execution of any of the previously mentioned verbs.

SPECIFYING INFORMATION ABOUT A FILE

Some of the information about the file must always be specified in the FD entry, SELECT sentence, APPLY, and other COBOL clauses. Other information must be specified in the DD statement. For example, the amount of space allocated for a mass storage output file must be specified in the DD statement by the SPACE, SPLIT, or SUBALLOC parameters. Certain characteristics of files cannot be expressed in the COBOL language, and may be specified on the DD statement for the file by the DCB parameter. This parameter allows the programmer to specify information for completing the data control block associated with the file (see "Additional File Processing Information" for a discussion of the data control block, and "Appendix C: Fields of the Data Control Block").

Each file used in the program must be referred to by a particular file processing technique. Four processing techniques are discussed in this publication. They are standard sequential (QSAM), direct (BSAM, BDAM), relative (BSAM, BDAM), and indexed (QISAM, BISAM).

A fifth processing technique, called partitioned data organization (BPAM), is discussed throughout the publication, when it is used for program storage.

A _partitioned_ data set (PDS) is composed of named, independent groups of sequential data, each of which is called a _member_. Each member has a simple name stored in a directory that is part of the data set and that contains the location of each member's starting point. Partitioned data sets are used to store programs, and are often referred to as _libraries_.

The full range of facilities available in BPAM are not available to the COBOL programmer. A partitioned data set may be referred to in COBOL only by treating it as a standard sequential data set.

FILE PROCESSING TECHNIQUES

DATA SET ORGANIZATION

A data set used by a COBOL program can have one of four types of organization: standard sequential, direct, relative, and indexed. The first type (sequential) may be on any input/output device. All other types must be on mass storage devices (see Figure 11 for information in determining the file processing technique to be used, according to data set organization).

1. A _standard sequential_ data set is one in which records are organized solely on the basis of their successive physical positions.

2. A _direct_ data set is one in which records are referred to by use of relative _track_ addressing. An ACTUAL KEY specifies the track relative to the first track allocated to the data set and identifies the record on the track.

3. A _relative_ data set is one in which records are referred to by use of relative _record_ addressing. A NOMINAL KEY identifies the record location relative to the first record in the data set.

4. An _indexed_ data set is one in which records are arranged on the tracks of a mass storage device so as to permit access in logical sequence (according to a key that is part of every record). A separate index or set of indexes maintained by the system indicates the location of each record. This permits random, as well as sequential, access to any record.

File Processing Requirements	ACCESS Clause and Organization Field (N) in System-name	Permissible Record Formats		Device Requirements	File Processing Technique
		Blocked	Unblocked		
Write, read, and update standard sequential file	ACCESS SEQUENTIAL or ACCESS clause is omitted N=S	F,V,S	F,V,U	Mass Storage Magnetic Tape Unit Record	QSAM
Write and read a mass storage file with relative record addressing	ACCESS SEQUENTIAL or omitted N=R		F	Mass Storage	BSAM
Read and update a mass storage file with relative record addressing	ACCESS RANDOM N=R		F	Mass Storage	BDAM
Create and read a mass storage file with relative track addressing	ACCESS SEQUENTIAL or omitted N=D		F,V,U,S	Mass Storage	BSAM
Create, read, update, and insert into a mass storage file with relative track addressing	ACCESS RANDOM N=D or W(REWRITE)		F,V,U,S	Mass Storage	BDAM
Create a mass storage file with indexed sequential organization	ACCESS SEQUENTIAL or omitted N=I	F	F	Mass Storage	QISAM
Read and update a mass storage file with indexed organization	ACCESS SEQUENTIAL or omitted N=I	F	F	Mass Storage	QISAM
Read, update, and insert into a mass storage file with indexed random organization	ACCESS RANDOM N=I	F	F	Mass Storage	BISAM

Figure 11. Determining the File Processing Technique

ACCESSING A STANDARD SEQUENTIAL FILE

A standard sequential file may only be accessed sequentially, i.e., records are read or written in the order in which they appear on the file. The file processing technique used to create and retrieve a standard sequential file is QSAM (Queued Sequential Access Method). Table 7 shows the COBOL clauses that may be used with these files. Special considerations for these clauses are as follows:

1. The RESERVE clause can be used to specify more buffer areas, allowing overlap of input/output operations with the processing of data. If this clause is not used, additional buffers may be specified by using the BUFNO option in the DD statement. If no additional buffer areas are specified, two buffers are reserved by the system. When the SAME AREA clause is specified for the file, the number of buffers used is determined from the RESERVE clause or if the RESERVE clause is not present, it is given a default of two. The BUFNO option in the DD statement is ignored if the SAME AREA clause is specified.

2. If the WRITE BEFORE/AFTER ADVANCING statement or the WRITE AFTER POSITIONING statement is used, the record size specified in the FD entry must allow for the carriage control or stacker select character, even though the character is not to be printed or punched. For example, if the record size specified in the FD entry is 121, the actual record is 121 characters; however, only 120 characters are printed or punched.

   Notes:

   - If the immediate destination of the record is a device that does not recognize a carriage control or stacker select character, the system assumes that the control character is the first character of the data. If the WRITE BEFORE/AFTER ADVANCING statement or the WRITE AFTER POSITIONING statement is not used, the first byte of the record is treated as data by the punch or printer.

   - The compiler may direct extra records, containing the appropriate control characters, to the file to effect printer spacing as specified in the WRITE BEFORE/AFTER ADVANCING statement. These extra records are for spacing purposes only and will not appear externally if the file is assigned to an online printer. However, if the file is assigned to a device that does not recognize the control characters (for example, a tape or a direct-access device), the extra records are written onto the file. These extra records are produced only if ADVANCING more than three lines is specified or if both the BEFORE and AFTER options are specified for a file.

3. If the input device is the card reader, RECORDING MODE IS F should be specified. If RECORDING MODE IS V or S is specified, the first 8 bytes of the record will be interpreted as the control bytes required for files with format V or S records.

4. If standard sequential files are on magnetic tape, the record block size should be at least 18 bytes. Records less than 18 bytes in length will be read with no problems, unless a parity check occurs. If a parity check occurs while reading a record less than 18 bytes, it will be treated as a noise record and skipped over.

5. The S (standard) option can be specified in the DCB RECFM subparameter for a fixed/blocked record data set with only standard blocks (i.e., having no truncated blocks or unfilled tracks within the data set, except for the last block of the last track). If a fixed/blocked data set is created through the use of an American National Standard COBOL F program, a truncated physical block may be written only by the executions of the CLOSE or CLOSE UNIT (or REEL) statement. Therefore, on a single volume data set, a COBOL-created fixed record set is standard except, possibly, when the data set is extended using DISP=MOD.

6. The T (TRACK OVERFLOW) option can be specified for the DCB RECFM subparameter of the DD statement for QSAM files with RECORDING MODE V, S, or F. Specification of the T option is equivalent to including the APPLY RECORD-OVERFLOW option in the source program, but use of the T option in the DD statement allows the user to make his selection at object time.

Table 7. COBOL Clause for Sequential File Processing

Data Management Techniques	Device Type	Access Method	KEY Clauses	OPEN Statement	Access Verbs	CLOSE Statement
QSAM	TAPE	SEQUENTIAL	NOT ALLOWED	INPUT [REVERSED NO REWIND] [LEAVE REREAD DISP]	READ [INTO] AT END	[REEL] [LOCK NO REWIND POSITIONING DISP]
				OUTPUT [NO REWIND LEAVE REREAD DISP]	WRITE [FROM] [{BEFORE AFTER} ADVANCING] [AFTER POSITIONING]	
QSAM	MASS STORAGE	SEQUENTIAL	NOT ALLOWED	INPUT	READ [INTO] AT END	[UNIT] [LOCK]
				OUTPUT	WRITE [FROM] INVALID KEY WRITE [FROM] [{BEFORE AFTER} ADVANCING] [AFTER POSITIONING]	
				I-O	READ [INTO] AT END WRITE [FROM] INVALID KEY REWRITE [FROM] INVALID KEY	[LOCK]

Figures 12 and 13 show the parameters in the DD statement that may be used with standard sequential files. All parameters except the DCB are described in "Job Control Procedures." Additional DCB subparameters not shown in the illustration are required for use with the Sort feature (see the chapter "Using the Sort Feature" for information on these parameters).

The DCB subparameters that can be specified in the DD statement for standard sequential files are as follows:

```
DCB=[DEN={0|1|2|3}]
    [,TRTCH={C|E|T|ET}]
    [,PRTSP={0|1|2|3}]
    [,MODE={C|E}]
    [,STACK={1|2}]
    [,OPTCD={W|C|WC|T|Q|Z}]
    [,BLKSIZE=integer]
    [,BUFNO=integer]
    [,EROPT={ACC|SKP|ABE}]
    [,RECFM={S|T}]
```

DEN={0|1|2|3}
: can be used with magnetic tape, and specifies a value for the tape recording density in bits per inch as listed in Table 8. If no value is specified, 800 bits-per-inch is assumed for 7-track tape, 800 bits-per-inch for 9-track tape without dual density and 1600 bits-per-inch for 9-track tape with dual density.

Table 8. DEN Values

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

TRTCH={C|E|T|ET}
is used with 7-track tape to specify the tape recording technique, as follows:

C - Specifies that the data-conversion feature is to be used; if data conversion is not available, only format F and format U records are supported by the control program.

E - Specifies that even parity is to be used; if omitted, odd parity is assumed.

T - Specifies that BCD to EBCDIC conversion is required.

ET- Specifies that even parity is to be used and BCD to EBCDIC conversion is required.

PRTSP={0|1|2|3}
specifies the line spacing on a printer as 0, 1, 2, or 3. If PRTSP is not specified, 1 is assumed.

The PRTSP subparameter is valid only if the unit specified for the file is a printer. It is not valid if the file is a report file, nor is it valid if the WRITE statement with the BEFORE/AFTER ADVANCING option or WRITE AFTER POSITIONING is specified in the COBOL source program. Single spacing always is assumed for a printer unless other information is supplied.

MODE={C|E}
can be used with a card reader, a card punch or a card-read punch and specifies the mode of operation as follows:

C - Specifies card image (column binary) mode.

E - Specifies EBCDIC code.

If this information is not supplied by any source, E is assumed.

STACK={1|2}
can be used with a card reader, a card punch, or a card-read punch, and it specifies which stacker bin is to receive the card. Either 1 or 2 is specified. If this information is not supplied by any source, 1 is assumed.

STACK should not be used when the WRITE statement with the AFTER ADVANCING or POSITIONING option is used to specify pocket selection.

OPTCD={W|C|WC|T|Q|Z}
requests an optional service provided by the system as follows:

W - To perform a write validity check (on mass storage devices only).

C - To process using the chained scheduling method (see the publication IBM OS Data Management Services).

WC- To perform a validity check and use chained scheduling.

T - To request user totaling facility.

Q - To translate to or from ASCII

Z - To request the search direct option

If this information is not supplied by any source, none of the services are provided, except in the case of the IBM 2321 mass storage device where OPTCD=W is specified by the operating system.

Note: If the validity check is specified, the system verifies that each record transferred from main storage to mass storage is written correctly. Standard recovery procedures are initiated if an error is detected.

BLKSIZE=integer
is used to specify the block size. This clause is used only when BLOCK CONTAINS 0 RECORDS was specified at compile time.

BUFNO=number of buffers
is used to specify the number of buffers to be assigned to the file when neither the RESERVE nor the SAME AREA clause is specified for the file in the source program. The maximum number is 255. However, the maximum number allowed for an installation is established when the program product is installed.

EROPT={ACC|SKP|ABE}
specifies the options to be executed if an error occurs in writing or reading a record as follows:

ACC - To accept the error block for processing.

SKP - To skip the error block.

ABE - To terminate the job.

There are two cases when the subparameter can be specified:

- If no error processing declarative (USE sentence) is specified, the option is taken immediately.
- If an error processing declarative is specified, the option is taken after the error declarative returns control via a normal exit (and only if that is the case).

If no option is specified, ABE is assumed.

RECFM={S|T}
specifies the option to be executed in processing the data set, as follows:

S - to expect the data set to consist of standard blocks; this option can be specified only if the RECORDING MODE is F.

T - to use the TRACK OVERFLOW option (this specification has the same effect as including the APPLY RECORD-OVERFLOW option in the source program).

	Device Type		
Parameter	Mass Storage	Magnetic Tape	Unit Record
DSNAME	as		
UNIT	as		
VOLUME	as		na
LABEL	SL SUL	SL NL NSL SUL	NL
SPACE	as	na	
SUBALLOC	as	na	
SPLIT	as	na	
DISP	{NEW / MOD}	{,KEEP / ,PASS / ,CATLG / ,DELETE}	SYSOUT=A,B...
DCB Device Dependent	OPTCD=W, WC	TRTCH, DEN	PRTSP, MODE, STACK
DCB General	OPTCD=C/T, BUFNO, BLKSIZE, EROPT=ABE RECFM={S|T}		EROPT=ACC (printer only) EROPT=ABE
as = Applicable subparameters na = Not applicable			

Figure 12. DD Statement Parameters Applicable to Standard Sequential OUTPUT Files

Parameter	Device Type: Mass Storage	Magnetic Tape	Unit Record
DSNAME	as		
UNIT	Not required if cataloged	Not required if cataloged	as
VOLUME	Not required if cataloged	Not required if cataloged	na
LABEL	SL SUL	SL NL NSL SUL	na
SPACE	na		
SUBALLOC	na		
SPLIT	na		
DISP	{OLD | SHR} {,KEEP | ,PASS | ,CATLG | ,UNCATLG | ,DELETE}		
DCB Device Dependent	--	TRTCH, DEN	MODE, STACK
DCB General	OPTCD=C/T, BLKSIZE, BUFNO, EROPT=ACC, SKP, ABE, RECFM={S|T}		

as = Applicable subparameters
na = Not applicable

Figure 13. DD Statement Parameters Applicable to Standard Sequential INPUT and I-O Files

SPECIFYING ASCII FILE PROCESSING

The compiler is notified by a special format of the ASSIGN clause that an ASCII (American National Standard Code for Information Interchange) file is to be created or read. The system-name in the ASSIGN clause must have the following format to indicate that an ASCII file is to be processed:

UT(-device)-C-(buffer offset)-name

where:

UT
: must be specified for a utility.

device
: which if specified must be a magnetic device, since ASCII support is only for magnetic tape files. If the device is omitted here, the magnetic tape device may instead be specified at execution time through JCL.

C
: an organization code which specifies that an ASCII-encoded sequential file is to be processed, or that an ASCII-collated sort is to be performed.

buffer offset
: a two-character field that indicates the length of the block prefix for that file. This entry is required if a non-zero block prefix exists; it must, however, be omitted when an ASCII-collated sort is requested.

name
: a field of 1 to 8 characters that specifies the system-recognized name of the file. It is this external name that appears in the name field of the DD card for the file.

PROCESSING ASCII FILES

Record formats allowed for ASCII files are the following: mode F (fixed length), mode U (undefined), and mode D (variable length). D-mode records are of variable length with a four-byte record descriptor field for each record. The COBOL programmer processing variable-length records specifies V-mode records. Then the format information generated from the DCB parameter is internally converted to D mode. Format-D records cannot be explicitly specified by the user in a COBOL program.

Block Prefix

An ASCII file may have a variable-length field, called a block prefix, preceding the first logical record in a physical record. If this prefix exists on an ASCII file, its length must be indicated at compile time in the buffer offset field of the ASSIGN clause. The compiler places this length in the DCB parameter at compile time.

Whether the optional block prefix contains the block length or simply user information depends on the type of file specified (input or output) and the internal record mode (i.e., F, U, or D). These distinctions are made in the discussion that follows.

Files Opened as Input: Input files with either blocked or unblocked records have an optional block prefix of 0 to 99 bytes that does not contain the block length but may contain user information. For D-mode records, however, a block prefix of length four may contain the block length. Regardless of the record format, file processing is identical to that for files coded in EBCDIC.

Files Opened as Output: The block prefix for output files applies only to D-mode records and, when specified, must have a length of 4. The prefix must contain the length of the block, which length includes the buffer offset.

For any ASCII output file the ASSIGN clause may include a buffer offset of four. Alternatively, the programmer may omit this specification from the ASSIGN clause, instead making use of the phrase BLOCK CONTAINS 0 RECORDS. The offset can then be specified at execution time in the JCL. However, if BLOCK CONTAINS 0 RECORDS is used, the following options must be included in the JCL:

BUFOFF=(n)
: must be included in the DCB parameter of the EXEC card, where n is the length of the block prefix from 0 to 99 characters on input, and either 0 or 4 on output.

BLKSIZE=(n)
: must be included on the DD card, where n is the size of the block, including the length of the block prefix.

Notes:

- If a block prefix exists on an ASCII file and the BLOCK CONTAINS clause with the CHARACTERS option is used, the length of the block prefix must be included in the BLOCK CONTAINS clause.

- If either the RECORDS option is specified or the BLOCK CONTAINS clause is omitted, the compiler compensates for the block prefix (if specified).

Additional JCL considerations for ASCII data sets follow.

$$\text{LABEL=} \begin{Bmatrix} \text{AL} \\ \text{AUL} \\ \text{NL} \end{Bmatrix}$$

where AL specifies American National Standard labels, AUL specifies American National Standard and user labels, and NL indicates no labels.

The subparameters below are specified in the DCB parameter of the DD statement:

OPTCD=Q, where Q specifies an ASCII-encoded data set.

RECFM=D, where D represents a variable-length record, is an optional parameter. Whether or not this parameter is specified at execution time, the programmer must specify an ASCII file in the ASSIGN clause as well as a mode-V record. The compiler converts from mode V to mode D, or to the internal representation for a variable-length record.

BUFOFF=(L), where L indicates a four-byte block prefix that contains the block length including the block prefix.

Handling Numeric Data Items from ASCII Files

It is highly recommended that the programmer take advantage of the separately signed numeric data type. The SIGN clause (see "SIGN Clause" in the chapter "Programmer Considerations") can be used to specify the position and the mode of representation of the operational sign of numeric data items.

DIRECT FILE PROCESSING

The direct file processing technique is characterized by the use of the relative track addressing scheme. When this addressing scheme is used, the tracks of mass storage devices are consecutively numbered from 0 to n (where 0 equals the first track of the file, and n equals the last track). The positioning of logical records in a file is determined by the ACTUAL KEY supplied by the user in the Environment Division. The first part of the key, called the track identifier, specifies either the track on which space for the record is first sought or the track at which the search for a record is to begin. The second part, called the record identifier, serves as a unique identifier for the record. Files with direct data organization must be assigned to mass storage devices.

Format
ACTUAL KEY IS data-name

Data-name may be any fixed item from 5 through 259 bytes in length and must be defined in the File Section, Working-Storage Section, or Linkage Section. The following considerations apply when defining the ACTUAL KEY:

- Track Identifier
  The first four bytes of data-name are the track identifier. The identifier is used to specify the relative track address for the record and must be defined as a 5-integer binary data item whose maximum value does not exceed 65,535.
- Record Identifier
  The remainder of data-name, which is 1 through 255 bytes in length, is the record identifier. It represents the symbolic portion of the key field used to identify a particular record on a track.

The following example illustrates the use of the ACTUAL KEY clause:

```
ENVIRONMENT DIVISION.
         .
         .
         .
   ACTUAL KEY IS THE-ACTUAL-KEY.
         .
         .
         .
DATA DIVISION.
         .
         .
         .
WORKING-STORAGE SECTION.
01   THE-ACTUAL-KEY.
     05   TRACK-IDENT   PIC S9(5) COMP SYNC.
     05   RECORD-IDENT PIC X(25).
```

Note: The same record identifier may appear more than once in the same file when using COBOL. However, using the same record identifier is not recommended for the following reasons:

1. If they appear on the same track, only the first occurrence can be retrieved (using BDAM).

2. If an extended search is used in either creating or updating a file, the position of records containing duplicate record identifiers may be unpredictable.

With direct file processing, records must be unblocked and may be V-, U-, F-, or S-mode records. Figure 14 illustrates those parts of a directly organized file that are of importance to a COBOL programmer.

```
    INDEX
    POINT    GAP
      |       |                R0                R1                     R2
      |       |
      |       |
      V       V  TRACK        CAPACITY
              G  ADDRESS  G   RECORD    G   COUNT KEY DATA  G   COUNT KEY DATA  G
                                                                                   . . .
```

Figure 14. Directly Organized Data as it Appears on a Mass Storage Device

Each track contains the following:

Index Point
: There is one index point to indicate the physical beginning of each track.

G (Gaps)
: Gaps separate the different areas on the track. Certain equipment functions take place as the gap is rotating past the read/write head. The length of the gap varies with the device, the location of the gap, and the length of the preceding area. For instance, the gap that follows the index point is a different length than the gap that follows the track address. The length of the gap that follows a record depends on the length of that record.

Track Address
: This field defines the physical location of the track. It indicates the cylinder in which the track is located and the read/write head that services the track.

R0 (Capacity Record)
: This field indicates the amount of unused space available for additional records on the track.

R1, R2, ..., Rn
: These are physical records that contain the following:

count area -- control information

key area -- the-record identifier (1-255 bytes) as specified by the programmer in the ACTUAL KEY clause.

data area -- the-data moved into the FD before a WRITE statement was executed.

The following example illustrates the relationship between the ACTUAL KEY and the positioning of records on a mass storage device during the creation of a direct file.

```
ENVIRONMENT DIVISION.
        .
        .
        .
    ACTUAL KEY IS THE-ACTUAL-KEY.
        .
        .
        .
DATA DIVISION.
FILE SECTION.
FD  DIRECT-FILE
    LABEL RECORDS ARE STANDARD.
01  REC-1              PIC X(200).
        .
        .
        .
WORKING-STORAGE SECTION.
01  THE-ACTUAL-KEY.
    05  TRACK-IDENT   PIC S9(5) COMP SYNC.
    05  RECORD-IDENT  PIC X(3).
```

```
┌──────────────────────────────────────────────────────────────────────────────────────┐
│                      Count Key Data       Count Key Data       Count Key Data        │
│                     ┌─────┬───┬─────┐    ┌─────┬───┬─────┐    ┌─────┬───┬─────┐      │
│  TRACK 0        G   │     │AAA│REC-1│ G  │     │CCC│REC-1│ G  │     │BBB│REC-1│      │
│         . . .       └─────┴───┴─────┘    └─────┴───┴─────┘    └─────┴───┴─────┘ . . .│
│                                                                                      │
│                     ┌─────┬───┬─────┐    ┌─────┬───┬─────┐    ┌─────┬───┬─────┐      │
│  TRACK 1        G   │     │DDD│REC-1│ G  │     │FFF│REC-1│ G  │     │EEE│REC-1│      │
│         . . .       └─────┴───┴─────┘    └─────┴───┴─────┘    └─────┴───┴─────┘ . . .│
│                                                                                      │
└──────────────────────────────────────────────────────────────────────────────────────┘
```
Figure 15. Sample Format of the First Two Tracks of a Direct File

Consider REC-1 being written six times; the contents of THE-ACTUAL-KEY varying with each WRITE instruction:

```
                 THE-ACTUAL-KEY

                  TRACK|RECORD
                  IDENT|IDENT
                  ┌────┬────┐
       WRITE  1   │ 0  │ AAA│
                  ├────┼────┤
       WRITE  2   │ 0  │ CCC│
                  ├────┼────┤
       WRITE  3   │ 0  │ BBB│
                  ├────┼────┤
       WRITE  4   │ 1  │ DDD│
                  ├────┼────┤
       WRITE  5   │ 1  │ FFF│
                  ├────┼────┤
       WRITE  6   │ 1  │ EEE│
                  └────┴────┘
```

Relative track 0 and relative track 1 of the mass storage device will appear as shown in Figure 15.

When the WRITE statement is executed, the system seeks the track that corresponds to the number contained in TRACK-IDENT. It then searches for the next available position into which a record may be placed. The system writes a count area, writes the contents of RECORD-IDENT in the key area, and writes the information contained in REC-1 in the data area.

Note: The record identifier is not included in the level-01 record description (REC-1). It will, however, be moved into the output buffer before being written on the mass storage device. Buffer areas, therefore, will be large enough to accommodate both the contents of REC-1 and the record identifier.

Dummy and Capacity Records

Once a direct file has been created, records can be added randomly on tracks formatted sequentially. Unless a track is already filled with data records, it is formatted by the compiler via the writing of dummy records (mode F) or of one capacity record (mode U, V, or S).

In order to format tracks, a COBOL subroutine executes instructions to write dummy records for F-mode files or write capacity records for V-, U-, or S-mode files. Dummy records are identified by the presence of the figurative constant HIGH-VALUE in the first byte of the record identifier portion of the ACTUAL KEY. This indicates to the system that a record can be added to the file in the space assigned to the dummy record. (The user should not attempt to retrieve a dummy record by moving this configuration to the record identifier because it is considered an invalid key.) A capacity record is a single record at the physical beginning of each track that indicates the amount of space available for additional records. As V-, U-, or S-mode records are added to a track, the capacity record is written accordingly. Capacity records are never made available to the user.

When a file is created, it should contain enough dummy records, or appropriately written capacity records, to allow for future expansion. Once the file is closed, more space cannot be allocated and the extent of the file cannot be increased.

Note: Tracks that have been assigned to a file but are not formatted, are considered "allocated." The user should not attempt to write on tracks that have been allocated but not formatted.

Sequential Creation of Direct Data Set

The file processing technique used to create a direct file sequentially is BSAM (Basic Sequential Access Method).

- The associated COBOL statements are summarized in Table 13.
- The associated JCL parameters are summarized in Table 14.

The ACTUAL KEY is required. It specifies the relative track number on which the record is to be written. Since access is sequential, all records will be written serially in the sequence in which they are moved into the output buffer. It is, therefore, necessary that all records to be written on the first track (track identifier = 0) be processed before records to be written on the 2nd, 3rd, ..., *n*th track (track identifier = 1, 2, ..., *n*-1) are processed.

When records are written sequentially, the user need not update the contents of the track identifier portion of the ACTUAL KEY. A COBOL subroutine will update it as follows:

- Records will be written on the first available track until space is no longer available. At such time, the COBOL subroutine will increment the track identifier by 1, and continue writing on the next track.
- The value of track identifier used by the system is made available to the user in the track identifier portion of the ACTUAL KEY after the record is written.
- After a CLOSE or CLOSE UNIT statement has been executed, the COBOL subroutine places the relative track number of the last track written on (for a data, dummy, or capacity record) in the track identifier of the ACTUAL KEY.
- If the user updates the contents of track identifier and attempts to write on track 2 when tracks 0 through 4 are already full, the system will automatically adjust the track identifier to 5 (the next track with available space).

If the user wishes to skip tracks, the number of tracks, equal to the number of tracks to be advanced, must be added to the track identifier. The COBOL subroutine will then add dummy records (F-mode) or write capacity records (V-, U-, or S-mode) to complete the intervening track(s) (see "Dummy and Capacity Records"). If the value of track identifier for the initial WRITE is not 0, the subroutine will complete the preceding tracks with dummy or capacity records.

SPACE ALLOCATION FOR SINGLE VOLUME FILES: When a file is created sequentially, the number of primary tracks specified on the DD card must be available on the primary volume. If this quantity is not available, the job will not begin execution. Once execution begins however, the final allocation of space will not be made until the file is closed.

The following discussion illustrates the space allocated to a direct file created using BSAM. Figure 16 is an example of a user program that:

- Writes 350-1/2 tracks and then closes the file.
- Specifies SPACE=(TRK,(200,100)) on the associated DD card.

TRACK-LIMIT Clause Specified:

1. If the TRACK-LIMIT clause specifies TRACK-LIMIT = 500 and the file is closed after writing only 350-1/2 tracks:

   Note: A COBOL subroutine will format all remaining tracks up to and including the 500th track. This represents 150 extra tracks on which records may be added.

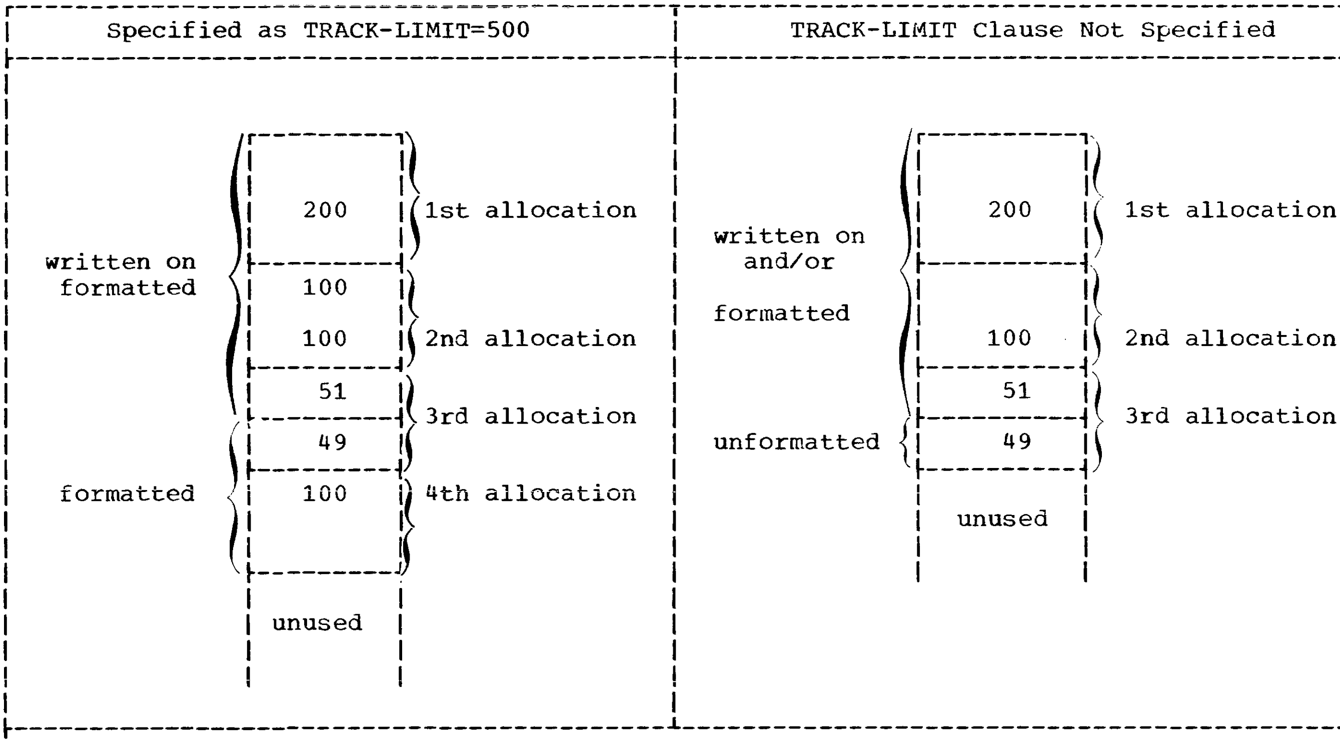
2. If the TRACK-LIMIT clause specifies TRACK-LIMIT = 300 and the program continues writing all 350-1/2 tracks:

   Note: The TRACK-LIMIT clause is ignored and the system allocates and formats as if no TRACK-LIMIT clause had been specified.

TRACK-LIMIT Clause Not Specified: If the TRACK-LIMIT clause is not specified, the system will allocate the primary extent (i.e., 200 tracks) and up to 16 secondary extents (i.e., 100 tracks each), as required. In Figure 16, the system allocates the first 200 tracks, all of which are completed. The second allocation, of 100 tracks, is also completed. The next 100-track allocation is, however, only partially used. The file is closed after writing on 350-1/2 tracks. At this time:

- A COBOL subroutine will format the rest of the 351st track. (Note that 351 tracks are actually relative tracks 0 through 350)
- The balance of 49 tracks will remain allocated but will not be formatted.

Note: In some of the foregoing cases, the number of tracks allocated to the file exceeds the number of tracks formatted by the COBOL subroutine. If the excess space was requested in track or block units, it should be released by specifying the RLSE option of the SPACE parameter.



Notes:
1. SPACE=(TRK,(200,100)) on a single volume.
2. The user program writes 350-1/2 tracks before closing the file.

Figure 16. Sample Space Allocation for Sequentially Created Direct Files

```
                     A
                  +------+                     B
                  |      |                  +------+
                  | 300  |                  | 100  |
                  |      |                  +------+                    C
                  |      |                  |      |                 +------+
 Specified as     |      |                  | 100  |                 |  50  |} formatted
 TRACK-LIMIT=950  +------+   formatted      +------+   formatted     +------+
                  |      |                  |      |                 |  50  |} allocated
                  | 100  |                  | 100  |                 +------+
                  +------+                  +------+                 |      |
                  |      |                  |      |                 |unused|
                  | 100  |                  | 100  |                 |      |
                  +------+                  +------+                 |      |
-------------------------------------------------------------------------------------
                     A
                  +------+
                  |      |
                  | 300  |   formatted         B                        C
                  |      |                  +------+                 +------+
                  |      |                  |      |                 |      |
                  |      |                  | 100  |   formatted     | 100  |  formatted
 TRACK-LIMIT      +------+                  +------+                 +------+
 clause not       |      |                  |      |                 |      |
 specified        |unused|                  |unused|                 |unused|
                  |      |                  |      |                 |      |
                  |      |                  |      |                 |      |
-------------------------------------------------------------------------------------
    SPACE=(TRK,(300,100))  on Volumes A, B, and C
```

Figure 17. Sample Space Allocation for Randomly Created Direct Files

Random Creation of a Direct Data Set

The file processing technique used to create a direct file randomly is BDAM (Basic Direct Access Method).

- The associated COBOL statements are summarized in Table 13.
- The associated JCL parameters are summarized in Table 14.

Figure 18 (sample program) illustrates the random creation of a direct data set.

The ACTUAL KEY is required. When a direct file is created randomly, records need not be written in any particular sequence. The system seeks the track specified in the track identifier portion of the ACTUAL KEY and writes the record in the next available position on that track.

When a file is created using BDAM, the number of tracks specified in the primary extent must be available on the primary volume. If there are secondary volumes, one secondary extent must be available on each of the secondary volumes. If these extents are not available, the job will not begin execution. Once execution begins, the final allocation of space is determined by the TRACK-LIMIT clause and the SPACE and volume-count parameters of the DD card when the file is opened as an output file. Figure 17 illustrates the allocation and formatting of space when the TRACK-LIMIT clause is specified as well as when it is not specified (see "Dummy and Capacity Records" for a definition of allocate and format).

1. When a TRACK-LIMIT clause is specifed (Figure 17), the system will do the following:

   a. Allocate tracks, by blocks, until the quantity specified by the TRACK-LIMIT clause has been equaled or just exceeded.

   b. Format only the space specified in the TRACK-LIMIT clause, even if the space formatted is less than the space allocated.

2. When a TRACK-LIMIT clause is not specified (Figure 17), the first volume will be allocated and formatted according to the primary allocation quantity, and any succeeding volumes will be allocated and formatted from the secondary quantity, one quantity per volume.

Records cannot be written on those tracks that were allocated but unformatted. Any attempt to do so will have unpredictable results. Unformatted tracks can be released by specifying the RLSE option in the SPACE parameter on the corresponding DD statement. Only space requested in track or block units can be released. If the CYL subparameter was specified, the unformatted tracks cannot be released.

Unlike direct files created with BSAM, the BDAM processing technique allocates and formats tracks when the file is opened. This is significant because the system will not allocate secondary extents if the user attempts to write on more tracks than the quantity initially formatted.

Note: The extended search option may be used during random creation. See "Random Reading, Updating, and Adding to Direct Data Sets" for a detailed description.

Sequential Reading of Direct Data Sets

The file processing technique used to read a direct file sequentially is BSAM (Basic Sequential Access Method).

- The associated COBOL statements are summarized in Table 13.
- The associated JCL parameters are summarized in Table 14.

When a direct file is being read sequentially, records are retrieved in logical sequence. This logical sequence corresponds exactly to the physical sequence of the records on the mass storage device. Dummy records, if present, are also made available.

For reading a file sequentially, the ACTUAL KEY clause need not be specified; however:

- If the key is not specified, the user will have no way of distinguishing between real and dummy records (F-mode only). Dummy records can be recognized by testing for the presence of the figurative constant "HIGH VALUE" in the first position of the record identifier.
- If the ACTUAL KEY clause is specified, the record's key will be placed in the record identifier portion of the ACTUAL KEY during the execution of a READ statement. The track identifier, however, remains unchanged.

Random Reading, Updating, and Adding to Direct Data Sets

The file processing technique used to read, update, and add to a direct file randomly is BDAM (Basic Direct Access Method).

- The associated COBOL statements are summarized in Table 13.
- The associated JCL parameters are summarized in Table 14.

When records are being retrieved from a direct file randomly, the ACTUAL KEY is required to determine the track and to locate a particular record on that track. When a match is found, the data portion of the record is read. For an add operation, after locating the track, the system searches for the next available position on the track, and writes the new record. For an update operation, after locating the track, the system searches for the record specified in the record identifier portion of the ACTUAL KEY.

In all of the foregoing cases, the specified track is the only one searched. If the desired record cannot be found, or room for an additional record cannot be found, the search terminates with an INVALID KEY condition. If the user wishes to extend the search to a specific number of tracks or to the entire file, the DCB OPTCD and LIMCT subparameters should be specified on the corresponding DD card. (Figure 18 illustrates the use of extended search.)

Multivolume Data Sets

Multivolume data sets, like single-volume data sets, may be created either randomly or sequentially.

Sequential Creation: When a file is created sequentially, the number of tracks specified in the primary extent must be available on the primary volume and the number of tracks specified in the secondary extent must be available on each of the secondary volumes. If extents are not available, execution of the job will not begin. Once execution begins, the primary, and as many secondary allocations as possible, are given to the first volume (up to 16 extents per volume). Subsequent volumes are allocated from the secondary specification.

If the CLOSE UNIT statement is executed, the current extent is formatted, volume switching procedures are executed, and the contents of ACTUAL KEY are updated to reflect the relative track number of the last track on the old volume. This is illustrated in the following example.

Consider the creation of a multivolume file whose space is allocated by:

```
SPACE=(TRK,(300,100))
```

1. When execution begins, the system allocates 300 tracks on the first volume. When the 300 tracks are used up, the system allocates 100 tracks more. Up to 16 allocations of 100 tracks each are possible.

2. If, after writing on 450 tracks, a CLOSE UNIT statement is executed, a COBOL subroutine will format the remaining 50 tracks of the current allocation before making the next unit available.

3. After the CLOSE UNIT statement is executed, a COBOL subroutine places the relative track number of the last track written on (for a data, dummy, or capacity record) in the track identifier of the ACTUAL KEY.

Note: A CLOSE UNIT statement always formats the tracks remaining on that unit from the current allocation. The formatting of tracks on the last unit, when a CLOSE file-name statement is executed, depends on the presence or absence of a TRACK-LIMIT clause, just as it did for single-volume files (see "Space Allocated for Single- Volume Files"). The RLSE option of the SPACE parameter applies only to the unformatted tracks at the end of the last unit.

Automatic Volume Switching: The user may choose to permit volume switching to occur automatically. This can be accomplished by writing on all allocated tracks until no more are available, or may be made available. This procedure, however, does not guarantee a specific distribution of records over the volumes, the placement of a particular record on a particular volume, or whether the data set is, in fact, multivolume.

Note: If the user permits system controlled volume switching, but specifies the file be created on more than one volume [e.g., VOL=SER=(V1,V2,V3)]; the system may write the entire file on the primary volume if there is enough room. The next time an attempt is made to open that file, since the system expects it to reside on three volumes, an ABEND will occur. This can be avoided by specifying:

```
VOL=(,,,3,SER=(V1,V2,V3))
```

This specifies the file be contained on one or more volumes.

To create a file with records distributed as evenly as posible over several volumes, the programmer must calculate the amount of space his file will require (see "Determination of File Space") and divide by the number of volumes. The result of this calculation (rounded) should be specified as both the primary and secondary allocation of the SPACE parameter of the associated DD statement. The programmer should execute CLOSE UNIT before the end of the initially allocated space on the first volume (that is, execute the CLOSE UNIT before writing the record that is to be first on the second volume).

For example, to distribute 2232 80-byte records as evenly as posible on three 2311 volumes, 34 tracks per volume are required and the SPACE parameter should specify (34,34). After writing the 744th record the programmer should execute CLOSE UNIT and continue writing.

If the required space is overestimated and the records do not fill the last track(s), the compiler will write dummy records to complete them. These records are included in the record count and should be taken into account when trying to address records on subsequent volumes.

If the space required is underestimated, automatic volume switching may occur before the CLOSE UNIT is executed since space on the first volume is filled. If this has happened, the CLOSE UNIT starts a third volume.

If no secondary allocation has been specified and the program issues a CLOSE UNIT statement, the job will terminate abnormally, since the allocation of subsequent volumes is taken from the secondary allocation field of the SPACE parameter.

In the creation of an output file, performance is improved by specifying the CONTIG subparameter of the SPACE parameter in the DD statement. However, space allocation is more efficient if CONTIG is not specified.

Random Creation: When a file is created randomly, space allocation and formatting is done as described in "Random Creation of a Direct Data Set" (Figure 17). It is important to note that a CLOSE UNIT statement is not permitted when creating a file randomly.

The following description pertains to Figure 17:

1. When the TRACK-LIMIT clause is specified, the total extent of the file is 950 tracks. The only valid track identifiers are 0 through 949:

   - Tracks 000 through 499 are contained on volume A.
   - Tracks 500 through 899 are contained on volume B.
   - Tracks 900 through 949 are contained on volume C.

2. When the TRACK-LIMIT clause is not specified, the total extent of the file is 500 tracks. The only valid track identifiers are 0 through 499:

   - Tracks 000 through 299 are contained on volume A.
   - Tracks 300 through 399 are contained on volume B.
   - Tracks 400 through 499 are contained on volume C.

File Organization Field of the System-Name

The single character "D" or "W", specifying the file organization, must be coded as part of the system-name. The user should be aware of the following differences:

- Sequentially accessed files must specify organization "D".

- Randomly accessed files may specify "D" or "W". When opened input or output "D" and "W" function identically.

1. Opened output ("D" and "W"):

   WRITE adds a new record. If a record containing the same key already exists, the system will add the record anyway. The result will be records with duplicate keys.

2. Opened I-O ("W"):

   a. REWRITE automatically searches for a record with a matching record identifier, and updates it.

   b. WRITE adds a new record to the file whether or not a duplicate key already exists.

3. Opened I-O ("D"):

   a. WRITE updates the file only if the preceding input/output statement was a READ of the same record.

   b. WRITE adds a new record to the file, whether or not a duplicate key already exists, if the preceding input/output statement was anything other than a READ of the same record.

Note: When a file is opened I-O (BDAM "D") the contents of ACTUAL KEY are moved to a save area during the execution of a READ statement. During the execution of a WRITE statement, the contents of ACTUAL KEY are compared to the contents of the save area to determine whether the system should add or update a record. A check is also made to assure that the preceding input/output statement was a READ. If it was a WRITE of any record, a new record is added to the file. Opening a file I-O (BDAM "W") omits the save and compare steps entirely. The system adds a record when a WRITE statement is executed and updates a record when a REWRITE statement is executed. It is, therefore, more efficient to use BDAM "W" than it is to use BDAM "D" if it is known in advance whether the record should be added or updated.

Determination of File Space: To determine the amount of space a data set requires, the following variables should be considered:

Device Type
Track Capacity
Tracks per Volume
Cylinders per Volume
Data length (block size)
Key Length
Device Overhead

Device overhead refers to the space required on each track for hardware data, i.e., address markers, count areas, inter-record gaps, Record 0, etc. Device overhead varies with each device and also depends on whether the blocks are written with keys. The formulas in Table 9 may be used to compute the actual space required for each block, including device overhead.

Table 10 lists device storage capacity, and Table 11 lists capacity in records per track for several mass storage devices.

Programmers who require more detailed information on mass storage devices may refer to the publications that follow:

IBM OS Component-Description -- 2841 Storage Control; 2302 Disk Storage, Models 3 and 4; 2311 Disk Storage Drive, Model 7; 2321 Data Call Drive; 2303 Drum; Order No. A26-5988.

Component Summary -- 2835 Storage Control, 2305 Fixed Head Storage, Order No. GA26-1589.

Component Summary -- 3830 Storage Control, 3330 Disk Storage, Order No. GA26-1592.

Note: Specification of the "S" option in the DCB subparameter RECFM can markedly increase 3330 performance (see the description of RECFM earlier in this chapter).

Randomizing Techniques

One method of determining the value of the track identifier portion of the ACTUAL KEY is called indirect addressing. Indirect addressing generally is used when the range of keys for a file includes a high percentage of unused values. For example, employee numbers may range from 000001 to 009999, but only 3000 of the possible 9999 numbers are currently assigned. Indirect addressing can also be used with nonnumeric keys. A nonnumeric field (e.g., alphanumeric), when moved to a computational field, will be packed and then converted to binary notation. Since packing eliminates the zone fields, the final binary item will be numeric.

Indirect addressing means that the key is converted to a value for the track identifier by use of some algorithm intended to limit the range of addresses. Such an algorithm is called a randomizing technique. Randomizing techniques need not produce a unique address for every record; in fact, such techniques usually produce synonyms. Synonyms are records whose keys randomize to the same address.

Two objectives must be considered in selecting a randomizing technique:

1. Every possible key in the file must randomize to an address within the designated range.

2. The addresses should be distributed evenly across the range so that there are as few synonyms as possible.

Note that one way to minimize synonyms is to allocate more space for the file than is actually required to hold all the records. For example, the percentage of locations actually used might comprise only 80 to 85 percent of the allotted space.

Division/Remainder Method: One of the simplest ways to address a directly organized file indirectly is to use the division/remainder method.

1. Determine the amount of locations required to contain the data file. Include a packing factor for additional space to eliminate synonyms. The packing factor should be approximately 20 percent of the total space allotted to contain the data file.

2. Select the nearest prime number that is less than the total of step 1. A prime number is a number divisible only by itself and the integer 1. Table 12 is a partial list of prime numbers.

3. Clear any zones from the key that is to be used to calculate the track identifier of actual key. This can be accomplished by moving the key to a field described as COMPUTATIONAL.

4. Divide the key by the prime number selected.

5. Ignore the quotient; utilize the remainder as the relative location within the data file.

For example, assume that a company is planning to create an inventory file on a 2311 disk storage device. There are 8,000 different inventory parts, each identified by an 8-character part number. Using a 20 percent packing factor, 10,000 record positions are allocated to store the data file.

Method A: The closest prime number to 10,000, but under 10,000, is 9973. Using one inventory part number as an example, in this case #25DF3514, and clearing the zones, we have 25463514. Dividing by 9973 a quotient of 2553 results in a remainder of 2445. Thus, 2445 is the relative location of the record within the data file corresponding to part number 25DF3514. The record address can be determined from the relative location as follows:

1. Determine the number of records that can be stored on a track (e.g., 12 per track on a 2311, assuming each inventory record is 200-bytes long).

   Note: Because each data record has nondata components, such as a count area and inter-record gaps, track capacity for data storage will vary with record length. As the number of separate records on a track increases, inter-record gaps occupy additional byte positions so that data capacity is reduced. Track capacity formulas provide the means to determine total byte requirements for records of various sizes on a track (see Tables 9, 10, and 11).

2. Divide the relative number (2445) by the number of records to be stored on each track.

3. The result, quotient = 203, now becomes the track identifier of the actual key.

Method B: Utilizing the same example, another approach will also provide the relative track address. Method B is illustrated in Figure 17:

1. The number of records that may be contained on one track is 12. Therefore, if 10,000 record locations are to be provided, 834 tracks must be reserved.

2. The prime number nearest, but less than 834, is 829.

3. Divide the zone-stripped key by the prime value. (In the example, 25463514 divided by 829 provides a quotient of 30715 and a remainder of 779. The remainder is the track identifier.)

Table 9. Mass Storage Device Overhead Formulas

Device Type	Bytes Required by Each Data Block			
	Blocks With Keys		Blocks Without Keys	
	Bi	Bn	Bi	Bn
2311	81+1.049(KL+DL)	20+KL+DL	61+1.049(DL)	DL
2314(2319)	146+1.043(KL+DL)	45+KL+DL	101+1.043(DL)	DL
2302	81+1.049(KL+DL)	20+KL+DL	61+1.049(DL)	DL
2303	146+KL+DL	38+KL+DL	108+DL	DL
2301	186+KL+DL	53+KL+DL	133+DL	DL
2321	100+1.049(KL+DL)	16+KL+DL	84+1.049(DL)	DL
2305-1	634+KL+DL	634+KL+DL	432+DL	432+DL
2305-2	289+KL+DL	289+KL+DL	198+DL	198+DL
3330	191+KL+DL	191+KL+DL	135+DL	135+DL

Bi is any block but the last on the track.
Bn is the last block on the track.
DL is data length.
KL is key length.

Table 10. Mass Storage Device Capacities

Device Type	Volume Type	Track Capacity	Tracks per Cylinder	Number of Cylinders	Total Capacity
2311	Disk	3625	10	200	7,250,000
2314(2319)	Disk	7294	20	200	29,176,000
2302	Disk	4984	46	246	56,398,944
2303	Drum	4892	10	80	3,913,600
2301	Drum	20483	8	25**	4,096,600
2321	Cell	2000	20***	980***	39,200,000
2305-1	Drum	14136	8	48	5,428,224
2305-2	Drum	14660	8	96	11,258,880
3330	Disk	13030	19	404	101,751,270

*Capacity indicated in bytes.
**There are 25 logical cylinders in a 2301 Drum.
***A volume is equal to one bin in a 2321 Data Cell.

Table 11. Mass Storage Device Track Capacity

Maximum Bytes per Record Formatted without Keys									Records per Track
2311	2314 (2319)	2302	2303	2301	2321	2305-1	2305-2	3330	
3625	7294	4984	4892	20483	2000	14136	14660	13030	1
1740	3520	2403	2392	10175	935	6852	7231	6447	2
1131	2298	1570	1558	6739	592	4424	4754	4253	3
830	1693	1158	1142	5021	422	3210	3516	3156	4
651	1332	912	892	3990	320	2480	2773	2498	5
532	1092	749	725	3303	253	1996	2278	2059	6
447	921	634	606	2812	205	1648	1924	1745	7
384	793	546	517	2444	169	1388	1659	1510	8
334	694	479	447	2157	142	1186	1452	1327	9
295	615	425	392	1928	119	1024	1287	1181	10
263	550	381	346	1741	101	892	1152	1061	11
236	496	344	308	1585	86	782	1040	962	12
213	450	313	276	1452	73	688	944	877	13
193	411	286	249	1339	62	608	863	805	14
177	377	164	225	1241	53	538	792	742	15
162	347	244	204	1155	44	478	730	687	16
149	321	225	186	1079	37	424	676	639	17
138	298	209	169	1012	30	376	627	596	18
127	276	196	155	952	24	334	584	557	19
118	258	183	142	897	20	296	544	523	20
109	241	171	130	848	15	260	509	491	21
102	226	161	119	804	10	230	477	463	22
95	211	151	109	763	6	200	448	437	23
88	199	143	100	726		174	421	413	24
82	187	135	92	691		150	396	391	25
77	176	127	84	659		128	373	371	26
72	166	121	77	630		106	352	352	27
67	157	114	70	603		88	332	335	28
63	148	108	64	577		70	314	318	29
59	139	102	58	554		52	297	303	30

Records per Track	Maximum Bytes per Record Formatted with Keys								
	2311	2314 (2319)	2302	2303	2301	2321	2305-1	2305-2	3330
1	3605	7249	4964	4854	20430	1984	13934	14569	12974
2	1720	3476	2383	2354	10122	920	6650	7140	6391
3	1111	2254	1505	1520	6686	576	4222	4663	4197
4	0811	1649	1139	1104	4968	406	3008	3425	3100
5	632	1288	893	854	3937	305	2278	2682	2442
6	512	1049	730	687	3250	238	1794	2187	2003
7	428	877	614	568	2759	190	1446	1833	1689
8	364	750	527	479	2391	154	1186	1568	1454
9	315	650	460	409	2104	126	984	1361	1271
10	275	571	406	354	1875	103	822	1196	1125
11	244	506	362	308	1688	85	690	1061	1005
12	217	452	325	270	1532	70	580	949	906
13	194	407	294	238	1399	58	486	853	821
14	174	368	267	211	1286	47	406	772	749
15	158	333	245	187	1188	38	336	701	686
16	143	304	224	166	1102	29	276	639	631
17	130	277	206	148	1026	21	222	585	583
18	119	254	190	131	959	15	174	536	540
19	108	233	176	117	899	9	132	493	501
20	99	215	163	104	844		94	453	467
21	90	198	152	92	795		58	418	435
22	82	183	142	81	751			386	407
23	76	168	132	71	710			357	381
24	69	156	123	62	673			330	357
25	63	144	116	54	638			305	335
26	58	133	108	46	606			282	315
27	53	123	102	39	577			261	296
28	48	114	95	32	550			241	279
29	44	105	89	26	524			223	262
30	40	96	83	20	501			206	247

Table 12. Partial List of Prime Numbers (Part 1 of 2)

Number	Nearest Prime Number Less than Number
500	499
600	599
700	691
800	797
900	887
1000	997
1100	1097
1200	1193
1300	1297
1400	1399
1500	1499
1600	1597
1700	1699
1800	1789
1900	1889
2000	1999
2100	2099
2200	2179
2300	2297
2400	2399
2500	2477
2600	2593
2700	2699
2800	2797
2900	2897
3000	2999
3100	3089
3200	3191
3300	3299
3400	3391
3500	3499
3600	3593
3700	3697
3800	3797
3900	3889
4000	3989
4100	4099
4200	4177
4300	4297
4400	4397
4500	4493
4600	4597
4700	4691
4800	4799
4900	4889
5000	4999
5100	5099
5200	5197
5300	5297
5400	5399
5500	5483
5600	5591
5700	5693
5800	5791
5900	5897

Table 12. Partial List of Prime Numbers (Part 2 of 2)

Number	Nearest Prime Number Less than Number
6000	5987
6100	6091
6200	6199
6300	6299
6400	6397
6500	6491
6600	6599
6700	6691
6800	6793
6900	6899
7000	6997
7100	7079
7200	7193
7300	7297
7400	7393
7500	7499
7600	7591
7700	7699
7800	7793
7900	7883
8000	7993
8100	8093
8200	8191
8300	8297
8400	8389
8500	8467
8600	8599
8700	8699
8800	8793
8900	8899
9000	8899
9100	9091
9200	9199
9300	9293
9400	9397
9500	9497
9600	9587
9700	9697
9800	9791
9900	9887
10,000	9973
10,100	10,099
10,200	10,193
10,300	10,289
10,400	10,399
10,500	10,499
10,600	10,597

Table 12. Partial List of Prime Numbers

Figure 18 is a sample COBOL program that creates a direct file using method B (see "Randomizing Technique") and provides for the possibility of synonym overflow. Synonym overflow will occur if a record randomizes to a track that is already full. The following discussion highlights some basic features. Circled numbers in the program example refer to corresponding numbers in the text that follows.

1. Since this randomizing technique (1) employs the prime number 829 as its divisor, the largest possible remainder is 828. By the interaction between the TRACK-LIMIT clause (2) and the SPACE parameter (3), the program formats 830 tracks (i.e., relative tracks 000-829). This establishes track 829 as the only track that can contain synonym overflow from track 828.

2. The DCB subparameter (4) OPTCD=E is specified. If a synonym overflow condition arises, an extended search will be employed, and the additional record will be written in the first available position on the following track(s).

3. The DCB subparameter (5) LIMCT=5 is specified. This limits the extended search to five tracks. If no room is found within this limit, an invalid key condition results. A value should always be specified for the LIMCT subparameter when OPTCD=E is indicated. Otherwise, the default value of LIMCT, which is zero, will result in an error that will be treated as an exceptional input/output condition.

Note: The randomizing technique chosen should minimize the number of synonym overflows for two reasons:

1. The more extended search is employed during file creation, the more it will be required during record retrieval. Extended searches increase access time proportionately.

2. When an extended search is employed, the adjusted value of the track identifier is not made available to the user after the execution of a WRITE statement. The user, therefore, has no way of knowing the track on which an overflow record is actually written.

```
00101  IDENTIFICATION DIVISION.
00102  PROGRAM-ID.  METHOD B.
00103  ENVIRONMENT DIVISION.
00104  CONFIGURATION SECTION.
00105  SOURCE-COMPUTER.  IBM-360.
00106  OBJECT-COMPUTER.  IBM-360.
00107  INPUT-OUTPUT SECTION.
00108  FILE-CONTROL.
00109      SELECT D-FILE ASSIGN DA-2314-D-MASTER
00110      ACCESS IS RANDOM ACTUAL KEY IS ACT-KEY
00112      TRACK-LIMIT IS 830.
**00103    SELECT C-FILE ASSIGN UT-S-CARDS.
00114  DATA DIVISION.
00115  FILE SECTION.
00116  FD  D-FILE
00117      LABEL RECORDS ARE STANDARD.
00118  01  D-REC.
00119      02  PART-NUM PIC X(8).
00120      02  NUM-ON-HAND PIC 9(4).
00121      02  PRICE  PIC 9(5)V99.
00122      02  FILLER PIC X(181).
00201  FD  C-FILE
00202      LABEL RECORDS ARE OMITTED.
00203  01  C-REC.
00204      02  PART-NUM PIC X(8).
00205      02  NUM-ON-HAND  PIC 9(4).
00206      02  PRICE PIC 9(5)V99.
           02  FILLER PIC X(61).
00207  WORKING-STORAGE SECTION.
00209  77  SAVE  PIC S9(8) COMP SYNC.
00210  77  QUOTIENT  PIC S9(5) COMP SYNC.
00211  01  ACT-KEY.
00212      02  TRACK-ID  PIC S9(5) COMP SYNC.
00213      02  REC-ID    PIC X(8).
00214  PROCEDURE DIVISION.
               OPEN INPUT C-FILE OUTPUT D-FILE.
00303  READS.
00304      READ C-FILE AT END GO TO EOJ.
00305      MOVE CORRESPONDING C-REC TO D-REC.
00306      MOVE PART-NUM OF C-REC TO REC-ID SAVE.
00307      DIVIDE SAVE BY 829 GIVING QUOTIENT REMAINDER TRACK-ID.
00308  WRITES.
00309      EXHIBIT NAMED TRACK-ID C-REC.
00310      WRITE D-REC INVALID KEY GO TO INVALID-KEY.
00311      GO TO READS.
00312  INVALID-KEY.
00313      DISPLAY 'INVALID KEY  ' TRACK-ID REC-ID.
00314  EOJ.
00315      CLOSE C-FILE D-FILE.
00316      STOP RUN.
```

Figure 18. Sample Program for a Randomly Created Direct File (Part 1 of 2)

```
STEP STEP2     TERMINATED. TIME 00.00 HR.HDRTH/HR * 00.00.23.30 HR.MIN.SEC.HDRTH/SEC*DATE 70.139
//STEP3   EXEC  PGM=*.STEP2.SYSLMOD
//SYSOUT   DD  SYSOUT=G
//SYSUDUMP DD   SYSOUT=A
//MASTER   DD  SPACE=(TRK,(500,100),RLSE),      3                        X
//              DCB=(OPTCD=E,LIMCT=5),UNIT=2314
//CARDS   DD *        4                5
//

TRACK-ID = 00801 C-REC = 82900801CD1
TRACK-ID = 00801 C-REC = 82900801CD2
TRACK-ID = 00801 C-REC = 82900801CD3
TRACK-ID = 00801 C-REC = 82900801CD4
TRACK-ID = 00031 C-REC = 82900031
TRACK-ID = 00801 C-REC = 82900801CD5
TRACK-ID = 00801 C-REC = 82900801CD6
TRACK-ID = 00801 C-REC = 82900801CD7
TRACK-ID = 00801 C-REC = 82900801CD8
TRACK-ID = 00801 C-REC = 82900801CD9
TRACK-ID = 00801 C-REC = 82900801CD10
TRACK-ID = 00801 C-REC = 82900801CD11
TRACK-ID = 00801 C-REC = 82900801CD12
TRACK-ID = 00801 C-REC = 82900801CD13
TRACK-ID = 00801 C-REC = 82900801CD14
TRACK-ID = 00801 C-REC = 82900801CD15
TRACK-ID = 00801 C-REC = 82900801CD16
TRACK-ID = 00000 C-REC = 829000003
TRACK-ID = 00801 C-REC = 82900801CD17
TRACK-ID = 00801 C-REC = 82900801CD18
TRACK-ID = 00801 C-REC = 82900801CD19
TRACK-ID = 00801 C-REC = 82900801CD20
TRACK-ID = 00809 C-REC = 82900809
TRACK-ID = 00801 C-REC = 82900801CD21
TRACK-ID = 00801 C-REC = 82900801CD22
TRACK-ID = 00801 C-REC = 82900801CD223
TRACK-ID = 00801 C-REC = 82900801CD24
TRACK-ID = 00801 C-REC = 82900801CD25
TRACK-ID = 00801 C-REC = 82900801CD26
```

Figure 18. Sample Program for a Randomly Created Direct File (Part 2 of 2)

Table 13. Direct File Processing on Mass Storage Devices

File Organization	Data Management Techniques	Access Method	KEY Clauses	OPEN Statement	Access Verbs	CLOSE Statement
	BSAM	SEQUENTIAL	ACTUAL	INPUT	READ [INTO] AT END	[UNIT] [WITH LOCK]
				OUTPUT	WRITE [FROM] INVALID KEY	
D	BDAM	RANDOM	ACTUAL	INPUT	SEEK READ [INTO] INVALID KEY	[WITH LOCK]
				OUTPUT	SEEK WRITE [FROM] INVALID KEY	
				I-O	SEEK READ [INTO] INVALID KEY WRITE [FROM] INVALID KEY	
W	BDAM	RANDOM	ACTUAL	I-O	SEEK READ [INTO] INVALID KEY WRITE [FROM] INVALID KEY REWRITE [FROM] INVALID KEY	[WITH LOCK]

Table 14. JCL Applicable to Directly Organized Files

DD Statement Parameters Applicable to BSAM Input Files

DSNAME	Device	UNIT VOLUME	LABEL	SPACE	SUBALLOC	SPLIT	DISP	DCB
as	Mass Storage required	not required if cataloged	[SL or SUL]		na		{OLD/SHR} {,PASS / ,KEEP / ,CATLG / ,DELETE / ,UNCATLG}	na

DD Statement Parameters Applicable to BSAM Output Files

DSNAME	Device	UNIT VOLUME	LABEL	SPACE	SUBALLOC	SPLIT	DISP	DCB
as	Mass Storage required	as	[SL or SUL]	as RLSE	as	na	NEW {,KEEP / ,CATLG / ,PASS / ,DELETE} Note: MOD not meaningful	[DSORG=DA] OPTCD=[W,T]

DD Statement Parameters Applicable to BDAM Input and I-O Files

DSNAME	Device	UNIT VOLUME	LABEL	SPACE	SUBALLOC	SPLIT	DISP	DCB
as	Mass Storage required	not required if cataloged	[SL or SUL]		na		{OLD/SHR} {,PASS / ,KEEP / ,CATLG / ,UNCATLG / ,DELETE}	as specified at file creation

DD Statement Parameters Applicable to BDAM Output Files

DSNAME	Device	UNIT VOLUME	LABEL	SPACE	SUBALLOC	SPLIT	DISP	DCB
as	Mass Storage required	as	[SL or SUB]	as RLSE	as	na	NEW {,KEEP / ,CATLG / ,PASS / ,DELETE} Note: MOD not meaningful	[DSORG=DA] OPTCD=[W,E] LIMCT=n

as = Applicable subparameters
na = Not applicable

RELATIVE FILE PROCESSING

Relative file processing is characterized by the use of the relative record addressing scheme. When this addressing scheme is used, the position of the logical records in a file is determined relative to the first record of the file starting with the initial value of zero. A NOMINAL KEY is used to identify randomly accessed records. Files with relative data organization must be assigned to mass storage devices.

Format
<u>NOMINAL</u> KEY IS data-name

Data-name must be defined as an 8-integer binary item whose value must not exceed 16,777,215. NOMINAL KEY must be defined in the Working-Storage Section.

The following example illustrates use of the NOMINAL KEY clause:

```
ENVIRONMENT DIVISION.
        .
        .
        .
    NOMINAL KEY IS THE-NOMINAL-KEY.
        .
        .
        .
DATA DIVISION.
        .
        .
        .
WORKING-STORAGE SECTION.
77  THE-NOMINAL-KEY PIC S9(8) COMP SYNC.
```

The relative file processing technique supports only unblocked fixed-length records.

Figure 19 illustrates those parts of a relatively organized file that are of importance to a COBOL programmer. The track format is similar to the format described for directly organized files (see section "Direct File Processing"). The following is a list of significant differences:

1. The records (R1, R2, ..., Rn) are formatted without a key area.

2. The COUNT area contains a record ID:

   a. 2 bytes containing the cylinder number.

   b. 2 bytes containing the read/write head.

   c. 1 byte containing a record number from 1 through 255.

Records on mass storage devices will always appear sequentially ranging from 0 to <u>n</u>, where <u>n</u> equals the highest key contained in the file.

The following example illustrates the relationship between the NOMINAL KEY and the positioning of records on a mass storage device.

```
ENVIRONMENT DIVISION.
        .
        .
        .
    NOMINAL KEY IS THE-NOMINAL-KEY.
        .
        .
        .
DATA DIVISION.
FILE SECTION.
FD  RELATIVE-FILE
    LABEL RECORDS ARE STANDARD.
01  REC-1              PIC  X(80).
        .
        .
        .
WORKING-STORAGE SECTION.
        .
        .
        .
77  THE-NOMINAL-KEY  PIC S9(8) COMP SYNC.
```

Consider REC-1 being written 50 times. With each execution of the WRITE statement, the content of THE-NOMINAL-KEY is incremented by 1, from 0 through 49. Since a 2311 mass storage device has room for only twenty-five 80-character records on each track (see "Determination of File Space" in "Direct File Processing") REC-1 will be written as follows:

- Relative records 0 through 24 will be on the first track.

- Relative records 25 through 49 will be on the second track.

```
        INDEX
        POINT   GAP
          |      |                 R0            R1             R2
          |      |
          |      |   +--------+  +---------+  +-----+----+  +-----+----+
          V      |   |TRACK   |  |CAPACITY |  |     |    |  |     |    |
                 G   |ADDRESS| G |RECORD   | G|COUNT|DATA| G|COUNT|DATA| G
                     +--------+  +---------+  +-----+----+  +-----+----+    . . .
```

Figure 19. Relatively Organized Data as it Appears on a Mass Storage Device

```
                 Count    Data       Count    Data       Count    Data           Count     Data
              +--------+------+   +--------+------+   +--------+------+      +---------+------+
            G |01,00,1 |REC-1 | G |01,00,2 |REC-1 | G |01,00,3 |REC-1 |G    G|01,00,25 |REC-1 |
1st TRACK     |        | (0)  |   |        | (1)  |   |        | (2)  |      |         |(24)  |
          ... +--------+------+   +--------+------+   +--------+------+ ...  +---------+------+

              +--------+------+   +--------+------+   +--------+------+      +---------+------+
            G |01,01,1 |REC-1 | G |01,01,2 |REC-1 | G |01,01,3 |REC-1 |G    G|01,01,25 |REC-1 |
2nd TRACK     |        |(25)  |   |        |(26)  |   |        |(27)  |      |         |(49)  |
          ... +--------+------+   +--------+------+   +--------+------+ ...  +---------+------+
```

Figure 20. Sample Format of Two Tracks of a Relative File

If the two tracks assigned to RELATIVE FILE are "cylinder 01 track 00" and "cylinder 01 track 01," they would appear as shown in Figure 19.

It is important to note that information about the length of each record, the capacity of each track and the relative record number, as indicated by the NOMINAL KEY is used by the system to determine the exact location of each record. As indicated in Figure 20, the system converts each relative record number into a unique cylinder number, head number, and record number, which are written in the count area of each physical record.

Note: Since count areas do not appear in I-O buffers and there are no key areas, buffer size need be only large enough to accommodate data in REC-1.

Sequential Creation

Relative files must be created sequentially using the file processing technique BSAM (Basic Sequential Access Method).

- The associated COBOL statements are summarized in Table 15.
- The associated JCL statements are summarized in Table 16.

Figure 21 illustrates the creation of a relative data set.

Records in relative files, are arranged sequentially in the order in which they were written. The first record written is relative record 0, the second record is relative record 1, the nth record written is relative record n-1. A file containing 1000 records will thus contain relative records 0 through 999. The clause that allows the user to specify the relative record needed is the NOMINAL KEY clause.

When a relative file is being created, the NOMINAL KEY clause may be specified.

- If the NOMINAL KEY is specified and the value in the NOMINAL KEY (when a WRITE statement is executed) is greater than the next sequential relative number, the necessary number of dummy records is written by the compiler so that the actual record is written in the specified relative position. If the NOMINAL KEY for a WRITE statement is less than the next sequential relative record number, the key is ignored and the record is written in the next available position.

- If the NOMINAL KEY is not specified, the system begins writing at relative record 0 and increments the relative record number by 1 for each additional WRITE statement. When the key is not specified, the user is responsible for insertion of dummy records. The only time the compiler will add dummy records is during the execution of a CLOSE or CLOSE UNIT statement.

  Note: Dummy records are identified by the presence of the figurative constant HIGH-VALUE in the first position of the record.

The relative block number of the last record written is placed in the NOMINAL KEY after a WRITE, CLOSE, or CLOSE UNIT statement, if the key is specified.

Once a file is created, more space cannot be allocated and the extent of the file cannot be increased. The only way to add records to an already existing file is to replace dummy records. Therefore, to allow for future additions, the user should create the file with as many excess dummy records as desired.

The allocation of space to a relative file (both single-volume and multivolume) is similar to the allocation of space described for a sequentially created direct file. Highlights and essential differences are discussed below:

- The relative file processing technique does not include a TRACK-LIMIT clause. Space allocation and formatting will, therefore, be determined by an interaction between the SPACE parameter of the DD card and the number of records written.

- The total number of tracks formatted will be determined when the file is closed. Dummy records will be added to complete the current track, if necessary.

- Tracks that are allocated but unformatted, and have been requested in track or block units, can be released by specifying the RLSE subparameter on the DD statement.

- When a unit of a multivolume file is closed, all tracks that have been allocated on the current unit are formatted (initialized with dummy records) before the next unit is made available. The RLSE subparameter of the DD statement applies only to the allocated tracks at the end of a data set.

Note: In order to determine the amount of space a data set requires, see Tables 9-11.

Sequential Reading

The file processing technique used to read a relative file sequentially is BSAM (Basic Sequential Access Method).

- The associated COBOL statements are summarized in Table 15.

- The associated JCL parameters are summarized in Table 16.

When a relative file is being read sequentially, the records are made available in the sequence in which the records were written. Dummy records are also made available. The NOMINAL KEY, if specified, will be ignored.

Random Access

The file processing technique used to read or update a relative file randomly is BDAM (Basic Direct Access Method).

- The associated COBOL statements are summarized in Table 15.

- The associated JCL statements are summarized in Table 16.

Since a relative file cannot be created randomly, the following restrictions exist:

1. The file cannot be opened as an output file.

2. The WRITE verb is not permitted.

A relative file with BDAM can be opened as input or I-O. Records are made available according to the contents of NOMINAL KEY. If the user wishes to update a file, it must be opened as I-O. Records can then be read into a single buffer, updated in that buffer, and rewritten from that buffer. If the user wishes to add records to a file, the file must have been created with excess dummy records. If dummy records are present, the file can be opened as I-O and dummy records can be replaced by the additions. If dummy records are not present, additions cannot be made.

Note: Records cannot be deleted, but can be replaced by dummy records.

Figure 21 illustrates several basic characteristics of the relative file processing technique. It creates a relative file (R-FILE) using a card file (C-FILE) as input. C-FILE consists of 11 cards in the following sequence:

Card Number		Card Contents
1	010	NAME01
2	020	NAME02
3	030	NAME03
4	040	NAME04
5	050	NAME05
6	060	NAME06
7	000	THIS CARD IS OUT OF SEQUENCE
8	070	NAME07
9	080	NAME08
10	090	NAME09
11	100	NAME10

The program, during creation, exhibits the contents of NOMINAL KEY after the execution of each WRITE statement. After creation, the relative file is closed, reopened as an input file, and written out on the printer. The following discussion highlights some basic features. Circled numbers in the program example refer to corresponding numbers in the text.

1. The nominal keys, 1 , that have been exhibited contain the relative record numbers of real records on the file. Relative records 10, 20, 30, 40, 50, 60, 61, 70, 80, 90, and 100 are real; all others are dummy records formatted by a COBOL subroutine. Note the nominal key N-KEY = 61. The initial value taken from C-FILE, card 7, was 000. This value, however, was not in logical sequence since relative records 000 through 060 had already been written. Therefore, a COBOL subroutine ignored the value 000 and adjusted it to the next appropriate relative record number (i.e., 61).

2. The contents of N-KEY for the first WRITE, 2 , was 10. This means that a COBOL subroutine formatted relative records 0 through 9, placing the constand HIGH-VALUE in the first position of each record.

   Note: The constant HIGH-VALUE is exhibited as a blank since FF is not a printable character.

3. The contents of N-KEY for the second WRITE, 3 , was 20. Therefore, the COBOL subroutine formatted relative records 11 through 19.

4. The contents of N-KEY for the seventh WRITE, 4 , was initially 000. As explained in step 1 , N-KEY was adjusted to 61 and the record was written in the next available position.

5. Since this file was created on a 2311 mass-storage device, the track capacity for R-FILE is 25 record per track. Relative record 100 is, therefore, the first record written on track 4 (remember: the first 5 tracks of a file are actually relative tracks 0 through 4). Since the file is closed after writing relative record 100, the COBOL subroutine formats the rest of track 4. In this case, it means the addition of 24 dummy records, 5

```
00101  IDENTIFICATION DIVISION.
00102  PROGRAM-ID. CREATER.
00103  REMARKS. ILLUSTRATE CREATION OF A RELATIVE FILE.
00104  ENVIRONMENT DIVISION.
00105  CONFIGURATION SECTION.
00106  SOURCE-COMPUTER. IBM-360.
00107  OBJECT-COMPUTER. IBM-360.
00108  INPUT-OUTPUT SECTION.
00109  FILE-CONTROL.
00110      SELECT R-FILE ASSIGN DA-2311-R-MASTER
00111          ACCESS IS SEQUENTIAL
00112          NOMINAL KEY IS N-KEY.
001125     SELECT C-FILE ASSIGN UR-S-CARDS.
001126     SELECT R-FILE2 ASSIGN DA-2311-R-MASTER.
001127     SELECT PRTFILE ASSIGN UR-S-PRTOUT.
00113  DATA DIVISION.
00114  FILE SECTION.
00115  FD  R-FILE
00116      LABEL RECORDS ARE STANDARD
00117      RECORDING MODE IS F
00118      DATA RECORD IS DISK.
001184 01  DISK PIC X(80).
001185 FD  R-FILE2 LABEL RECORDS ARE STANDARD.
001186 01  DISK2 PIC X(80).
00201  FD  C-FILE
00202      LABEL RECORDS ARE OMITTED
00203      DATA RECORD IS CARD.
00204  01  CARD.
002041     02  C-KEY PIC 9(3).
002042     02  FILLER PIC X(77).
002043 FD  PRTFILE  LABEL RECORDS ARE OMITTED.
002044 01  PRT.
002045     02  FILLER PIC X.
002046     02  FIELD1  PIC X(132).
00205  WORKING-STORAGE SECTION.
00206  77  N-KEY PIC S9(8) COMP SYNC.
00207  PROCEDURE DIVISION.
00208      OPEN INPUT C-FILE
00209           OUTPUT R-FILE.
00210  R1. READ C-FILE AT END GO TO EOJ1.
00211      MOVE C-KEY TO N-KEY.
00212      WRITE DISK FROM CARD.
00213      EXHIBIT NAMED N-KEY.  GO TO R1.
00214  EOJ1.
00215      CLOSE C-FILE R-FILE.
00216      OPEN INPUT R-FILE2  OUTPUT PRTFILE.
00217  R2. READ R-FILE2 AT END GO TO EOJ2.
00218      MOVE DISK2 TO FIELD1.
00219      WRITE PRT AFTER 1 LINES  GO TO R2.
00220  EOJ2.
00230      CLOSE R-FILE2 PRTFILE.  STOP RUN.
```

Figure 21. Sample Program for Relative File Processing (Part 1 of 4)

```
IEF285I   PROCPAST                                      PASSED
IEF285I   VCL SER NCS= LSASIA.
IEF285I   SYS69184.TC3C423.RVCCC.RFILE.UT1              CELETED
IEF285I   VCL SER NOS= MVTRES.
IEF285I   SYS69184.TC3C423.RV000.RFILE.AJCB             PASSED
IEF285I   VCL SER NOS= 222222.
IEF285I   SYS1.CCBLIB                                   KEPT
IEF285I   VCL SER NOS= LSASIA.
IEF285I   SYS69184.TC3C423.SVCCC.RFILE.RCOCC032         SYSCLT
IEF285I   VCL SER NOS= 231400.
IEF285I   SYS69184.TC3C423.RVCOC.RFILE.FMCH             CELETEC
IEF285I   VCL SER NOS= 222222.
STEP STEP2     TERMINATEC. TIME CO.CC MR.MCRTM/MR * CC.CO.18.08 HR.MIN.SEC.HORTH/SEC*CATE 69.184
//STEP3  EXEC   PGM=*.STEP2.SYSLMCD
//SYSCUT DC     SYSCUT=A
//SYSLDUMP  CC  SYSCLT=A
//MASTER  DC  UNIT=2311,VCLUME=SER=CAC28,SPACE=(TRK,(5,5),,CCNTIG),     X
//              CSNAME=RFILE,CISP=(NEW,KEEP)
//PRTCUT  CC  SYSCLT=A
//CARDS  DD  *
//
IEF236I ALLCC. FCR RFILE     STEP3
IEF237I JCBLIB    CN 193
IEF237I PGM=*.DD CN 19C
IEF237I SYSCLT    CN 23C
IEF237I SYSLDUMP CN 235
IEF237I MASTER    CN 192
IEF237I PRTCLT    CN 23C
IEF237I CARDS     CN 235

N-KEY = CCCCCC1C
N-KEY = CCOCOC2C
N-KEY = C0000030
N-KEY = CCCCCC4C
N-KEY = CCCCCC5C
N-KEY = C00000C6C    1
N-KEY = CCCCCC61
N-KEY = COCOOC7C
N-KEY = CCCCOC8C
N-KEY = CCCCCC9C
N-KEY = C000010C


 10  NAMEC1
 10  NAMEC1
 10  NAMEO1
 10  NAMEO1
 10  NAMEO1
 10  NAMEO1    2
 10  NAMEO1
 10  NAMEO1
 10  NAMEO1
 10  NAMEC1
010  NAMEO1
 20   NAMEO2
 20   NAMEO2
 20   NAMEC2
 20   NAMEO2
 20   NAMEO2    3
 20   NAMEO2
 20   NAMEC2
 20   NAMEC2
 20   NAMEC2
```

Figure 21. Sample Program for Relative File Processing (Part 2 of 4)

```
020   NAME02
 30  NAMEC3
 30  NAMEC3
 30  NAMEC3
 30  NAMEC3
 30  NAMEC3
 30  NAMEC3
 30  NAMEC3
 30  NAMEC3
 30  NAMEC3
03C  NAMEC3
 40   NAME04
 40   NAME04
 40   NAME04
 4C   NAME04
 40   NAME04
 40   NAME04
 40   NAME04
 40   NAME04
 4C   NAME04
040   NAME04
 50  NAME05
 50  NAME05
 50  NAME05
 50  NAME05
 50  NAME05
 50  NAME05
 50  NAME05
 50  NAMEC5
 50  NAME05
050  NAMEC5
 60   NAME06
 60   NAME06
 60   NAME06
 60   NAMEC6
 60   NAMEC6
 60   NAME06
 60   NAME06
 60   NAME06
 6C   NAME06
060   NAMEC6
000  THIS  CARC  IS  CLT  CF  SEQUENCE    4
 70   NAME07
 70   NAME07
 70   NAME07
 70   NAME07
 70   NAME07
 70   NAME07
 70   NAME07
 70   NAME07
070   NAME07
 8C  NAMEC8
 80  NAME08
 80  NAME08
 80  NAME08
 80  NAME08
 80  NAMEC8
 80  NAMEC8
 80  NAME08
 80  NAMEC8
080  NAMEC8
 90   NAME09
 90   NAMEC9
 90   NAMEC9
 9C   NAMEC9
 90   NAME09
```

Figure 21. Sample Program for Relative File Processing (Part 3 of 4)

```
90    NAME09
90    NAME09
90    NAME09
90    NAME09
090   NAME09
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
100  NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10      5
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
00   NAME10
```

Figure 21. Sample Program for Relative File Processing (Part 4 of 4)

Table 15. Relative File Processing on Mass Storage Devices

Data Management Techniques	Access Method	KEY Clauses	OPEN Statement	Access Verbs	CLOSE Statement
BSAM	SEQUENTIAL	[NOMINAL]	INPUT	READ [INTO] AT END	[UNIT] [WITH LOCK]
		NOMINAL	OUTPUT	WRITE [FROM] INVALID KEY	
BDAM	RANDOM	NOMINAL	INPUT	READ [INTO] INVALID KEY	[WITH LOCK]
			INPUT OUTPUT	READ [INTO] INVALID KEY REWRITE [FROM] INVALID KEY	

Table 16. JCL Applicable to Relatively Organized Files

DD Statement Parameters Applicable to BSAM Input Files							
DSNAME	Device	UNIT	VOLUME	LABEL	SPACE,SUBALLOC,SPLIT	DISP	DCB
as	Mass Storage required	not required if cataloged		[SL or SUL]	na	{OLD SHR} {,PASS ,KEEP ,CATLG ,DELETE ,UNCATLG}	na

DD Statement Parameters Applicable to BSAM Output Files									
DSNAME	Device	UNIT	VOLUME	LABEL	SPACE	SUBALLOC	SPLIT	DISP	DCB
as	Mass Storage required	as		[SL or SUL]	as RLSE	as	na	NEW {,KEEP ,CATLG ,PASS ,DELETE} Note: MOD not meaningful	OPTCD={W,T} [DSORG=DA]

DD Statement PARAMETERS Applicable to BDAM Input and I-O Files							
DSNAME	Device	UNIT	VOLUME	LABEL	SPACE,SUBALLOC,SPLIT	DISP	DCB
as	Mass Storage required	not required if cataloged		[SL or SUL]	na	{OLD SHR} {,PASS ,KEEP ,CATLG ,UNCATLG ,DELETE}	as has been specified

as = Applicable subparameters
na = Not applicable

INDEXED FILE PROCESSING

The indexed file processing technique arranges records on the tracks of a mass-storage device in a sequence determined by keys. The key is a control field that is a physical part of the record (defined in the FD) and is specified by the RECORD KEY clause in the Environment Division. The RECORD KEY clause identifies for the compiler the location and length of that item within the data record that will contain the key. It must always be specified.

Format
<u>RECORD</u> KEY IS data-name

<u>Data-name</u> may be any fixed-length item from 1 through 255 bytes in length.

When two or more record descriptions are associated with a file, a similar field must appear in each description, and must be in the same relative position from the beginning of the record, although the same data-name need not be used for both files.

Data-name must be defined to exclude the first byte of the record in the following cases:

1. Files with unblocked records.

2. Files from which records are to be deleted.

3. Files whose keys might start with a delete-code character (HIGH-VALUE).

With these exceptions, the item specified by data-name may appear anywhere within the record.

The position of each logical record in a file is determined by indexes created with the file and maintained by the system. The indexes are based on the RECORD KEYS and provide the following capabilities:

- Write and later read or update logical records in a sequential, ascending order (using QISAM) based on the collating sequence of the keys. This is done in a manner similar to that for sequential organization.

- Read or update individual logical records in a random manner (using BISAM). This method is somewhat slower per record than reading according to a collating sequence, since a search for pointers in indexes is required for the retrieval of each record.

- Insert new logical records at any point within the file (using BISAM). Using the indexes, the system locates the proper position for the new record and makes all necessary adjustments so that the sequence of the records, according to the keys, is maintained.

<u>Indexes</u>

There are two basic types of indexes: track indexes and cylinder indexes. There is one track index for each cylinder in the prime area (see "Indexed File Areas" for a description of prime area). The track index is written on the first track of the cylinder that it indexes. Each entry in the track index contains the identification of a specific track in the cylinder and the highest key on that track (Figure 22).

Figure 22 is the representation of a track index with the following areas:

<u>Home Address</u> -- This field defines the physical location of the track in which the index appears. It indicates the cylinder in which the track is located and the read/write head that services the track.

```
               R0        Normal         Overflow        Normal         Overflow

  |0100|     |COCR|   |00010|0001|   |00010|0001|   |00025|0002|   |00025|0002|

   Home                Key   Data     Key   Data     Key   Data     Key   Data
  Address
```

Figure 22. Track Index

COCR (Cylinder Overflow Control Record) -- When a cylinder overflow area is specified (see "Indexed File Areas" for a description of overflow areas), R0 of each track index is used to keep track of overflow records and space available in the cylinder overflow area.

Normal Entry -- There is one normal and one overflow entry for each usable track in the cylinder. The Normal Entry contains two areas:

- Key -- the key of the highest record on the track specified in the Data area

- Data -- the home address of one of the prime tracks in the cylinder

Figure 22 shows that the highest key on track 1 is 10 and the highest key on track 2 is 25.

Overflow Entry -- The overflow entry is originally the same as the normal entry. It is changed when an attempt is made to add a record to a prime track on which space is no longer available. In this case, the overflow entry keeps track of the logical sequence of records although physically the record may be added to an overflow area.

There is one cylinder index for each file in which prime area data occupies more than one cylinder. The cylinder index contains one entry for each cylinder in the prime area; each entry pointing to the track index for a particular cylinder (Figure 23).

```
|00500|0000|  |00945|0001|  |01550|0002|...
 Key   Data    Key   Data    Key   Data
          cylinder address
```

Figure 23. Cylinder Index

The cylinder index is formatted in the same fashion as the track index. Figure 23 shows that the highest key on cylinder 0 is 500, the highest key on cylinder 01 is 945, the highest key in cylinder 02 is 1550, etc.

Note: If an indexed file is being read randomly, the system locates the given record by its key after a search of the cylinder index and the track index within the indicated cylinder. If the file is being read sequentially, starting, with the first record, no index search is performed.

Records, in indexed files, may be either blocked or unblocked; but must be F-mode records. Figures 24 and 25 illustrate blocked and unblocked records as they appear on prime tracks of mass storage devices.

BLOCKED RECORDS

Count: contains control information

Key: contains the key of highest record in the block

Data(1, 2, ..., 6): each contains the information defined in the FD; including its own record key.

```
| COUNT | KEY | DATA1 | DATA2 | DATA3 |   | COUNT | KEY |DATA4 | DATA5 | DATA6 |

              1st Block                                2nd Block
```

Figure 24. Blocked Records on an Indexed File

```
┌──────────────────────────────────────────┐
│                                          │
│  ┌─────┬───┬─────┐    ┌─────┬───┬─────┐  │
│  │COUNT│KEY│DATA1│    │COUNT│KEY│DATA2│  │
│  └─────┴───┴─────┘    └─────┴───┴─────┘  │
│                                          │
│       1st Block           2nd Block      │
└──────────────────────────────────────────┘
```
Figure 25. Unblocked Records on an Indexed File

UNBLOCKED RECORDS

Count: contains control information

Key: contains the key of the record that is in the block.

Data (1), (2), etc.: each contains the information defined in the FD; including its own record key.

Indexed File Areas

The programmer specifies the structure of an indexed file and space to be allocated for it in the DD statement for the file when the file is created. In some instances, more than one DD statement is required. (These DD statements are described in "Using the DD Statements -- Single Volume Files.") The space being allocated must be divided into one, two, or three areas, depending on the needs of the programmer. These areas are: prime area, index area, and overflow area. The overflow area is optional.

Prime Area: The prime area is the area in which data records are written when the file is created or reorganized. These records are in a sequence determined by the record keys. The track indexes also use a portion of the reserved prime area. To reserve prime area space so that new logical records may be inserted without forcing records into an overflow area (described below), dummy records (records containing the figurative constant HIGH-VALUE in the first character position) may be written when the file is being created. The prime area may span multiple volumes and may consist of several noncontiguous areas.

Index Area: The index area contains the cylinder indexes and, if requested, master indexes (described later) for the file. This area exists for any file that has a prime area on more than one cylinder. Space for this area will be allocated separately from the prime area if specifically requested. The index area must be contained within one volume, but that volume need not be the same device type as the prime area volume. If not specifically requested, the index area will automatically be constructed in the independent overflow area, or, if there is no independent overflow area, it is constructed in the prime area.

Overflow Area: The overflow area is the area in which space is allocated for records forced from their original (prime) tracks by the insertion of new records. The fact that some records are stored in these areas, physically out of sequence, does not change the ability of QISAM to read the file in a logical sequence. An overflow area need not be specified if records are either not going to be added to the file, or sufficient space was originally reserved by writing dummy records in the prime area.

There are three ways in which space for an overflow area may be allocated:

1. Cylinder Overflow (Figure 26). Tracks on each cylinder can be reserved to hold the overflow of that cylinder (cylinder overflow option).

2. Independent Overflow (Figure 27). Space may be requested for an independent overflow area, using the dsname (OVFLOW) DD statement, either on the same volume or on a separate volume of the same device type as that of the prime area.

3. If the prime area is not filled when the file is created, the space remaining on the last cylinder on which data has been written will be designated as an independent overflow area (even though it is not requested directly). If a separate independent overflow area is requested, the remainder of the prime area is available for resuming a load operation.

Additional information about indexed file structure is contained in the publication IBM OS Data Management Services.

```
┌──────────────────────────────────────────┐
│  |    |    |    |    |    |    |    |     │
│  |CYL0|CYL1|CYL2|CYL3|CYL4|CYL5|CYL6|     │
│  ├────┼────┼────┼────┼────┼────┼────┤     │
│  |    |    |Track Indexes  |    |    |     │
│  ├────┼────┼────┼────┼────┼────┼────┤     │
│  |    |    |    |    |    |    |    |     │
│  |    |    |    |    |    |    |    |     │
│  |         Prime Area               |     │
│  |    |    |    |    |    |    |    |     │
│  |    |    |    |    |    |    |    |     │
│  |    |    |    |    |    |    |    |     │
│  |    |    |    |    |    |    |    |     │
│  ├────┼────┼────┼────┼────┼────┼────┤     │
│  |    |    |    |    |    |    |    |     │
│  |     Cylinder Overflow Area       |     │
│  |    |    |    |    |    |    |    |     │
│  └────┴────┴────┴────┴────┴────┴────┘     │
│                                           │
└──────────────────────────────────────────┘
```

Figure 26. Cylinder Overflow Area

An advantage of having a cylinder overflow area is that additional seek operations are not required to locate overflow records. A disadvantage is that there will be unused space if additions are unevenly distributed throughout the file.

```
┌──────────────────────────────────────────┐
│  |    |    |    |    |   |     |     |   │
│  |    |    |    |    |   |     |     |   │
│  |CYL0|CYL1|CYL2|CYL3|   |CYLX |CYLY |   │
│  ├────┼────┼────┼────┤   ├─────┼─────┤   │
│  |  Track  Indexes   |   |     |     |   │
│  ├────┼────┼────┼────┤   |     |     |   │
│  |    |    |    |    |   |     |     |   │
│  |    Prime Area     |   |Independent|   │
│  |    |    |    |    |   | Overflow  |   │
│  |    |    |    |    |   |    Area   |   │
│  |    |    |    |    |   |     |     |   │
│  └────┴────┴────┴────┘   └─────┴─────┘   │
│                                          │
└──────────────────────────────────────────┘
```

Figure 27. Independent Overflow Area

An advantage of having an independent overflow area is that less space need be reserved for overflows. A disadvantage is that accessing overflow records requires additional seek operations.

A suggested approach is to have cylinder overflow areas large enough to contain the average number of overflows caused by additions and an independent overflow area to be used as the cylinder overflow areas are filled.

Creating Indexed Files

Indexed files must be created sequentially using QISAM (Queued Indexed Sequential Access Method). Records must be arranged and written in ascending order according to the contents of RECORD KEY. If a WRITE statement is executed and the current contents of RECORD KEY is less than or equal to the previous contents of RECORD KEY, an INVALID KEY condition will result.

The structure of an indexed file, and the space to be allocated to it, is specified in a DD statement(s). The space, which can be allocated in several different ways, must be sufficient for all areas of the file.

DD STATEMENT REQUIREMENTS FOR INDEXED FILES: The special parameter requirements for DD statements that define new indexed files are discussed below. The discussion is oriented to indexed files on one volume. Many of the parameters used for creating multivolume files are not discussed here. For more detailed information about parameters for both single-volume and multivolume files, see either of the publications IBM OS Job Control Language Reference or IBM OS Job Control Language Programmer's Guide.

ddname (name field)
: The name field of the first or only DD statement defining the indexed sequential file can contain the symbolic identification ddname or procstep.ddname. Succeeding DD statements for the file must not be named.

DSNAME (DSN)
: This parameter must be specified and is coded as follows:

$$\left\{\begin{matrix}\text{DSNAME}\\ \text{DSN}\end{matrix}\right\} = \left\{\begin{matrix}\text{dsname}\\ \text{\&\&name}\end{matrix}\right\}\ \text{[(element)]}$$

The first subparameter, dsname, or &&name must be the same in all the DD statements defining one data set. The element subparameter, INDEX, PRIME, or OVFLOW, indicates the type of area defined by the DD statement. If more than one DD statement is used to define a file, the order in which the statements should be placed in the input stream is as follows:

```
DD  DSNAME=dsname(INDEX)
DD  DSNAME=dsname(PRIME)
DD  DSNAME=dsname(OVFLOW)
```

Deviation from this sequence results in abnormal termination of the job. If the element subparameter is omitted PRIME is assumed. Note that an indexed file cannot be specified by statements containing only index and overflow elements.

SPACE

This parameter specifies the space to be allocated for each of the separate areas on the device and must be included. Only cylinder (CYL) or absolute track (ABSTR) requests are permitted, and with ABSTR the designated tracks must encompass an integral number of cylinders. All the DD statements defining one indexed file must specify the same subparameter, either CYL or ABSTR. When all the DD statements specify CYL, all must also specify or omit CONTIG, depending on whether the space allocated is to be contiguous or noncontiguous. The directory or index quantity subparameter of the SPACE parameter is used to request the number of cylinders to be allocated for an index area embedded within the prime area (see "Space Parameter" in "Job Control Procedures"). An embedded index resides in the middle of a track and saves searching time by first determining which half of the track contains the requested record.

SPLIT

This parameter should never be specified for an indexed file, either for sharing a cylinder with indexed files or for sharing it with an indexed file and another type of file.

DISP

This parameter is written as it would be for any new file that cannot be cataloged. The CATLG subparameter must not be specified unless only one DD statement is used to allocate the file space (see "Cataloging Files" for additional information about cataloging indexed files).

DCB

This parameter must be specified for each DD statement and is coded as follows:

```
DCB=(DSORG=IS
  [,BUFNO=integer]
  [,OPTCD={Y|I|R|W|L|M|U,NTM=integer}]
  [,BLKSIZE=integer])
```

The DSORG=IS subparameter is required and indicates that the organization of the file is sequential. The DCB subparameters of all the DD statements defining one file must not conflict. For example, if the OPTCD=Y subparameter appears in the first DD statement, the subsequent DD statements should also contain OPTCD=Y. To avoid any errors, code all the DCB subparameters on the first DD statement. Code DCB=*.ddname on the remaining statements; ddname is the name of the DD statement that contains the DCB subparameters. The subparameters are discussed below.

BUFNO=number of buffers

This subparameter is used to specify the number of buffers to be assigned to the file if no RESERVE or SAME AREA clause is specified for the file in the source program. The maximum number is 255; however, the maximum number allowed for an installation may differ and is established at system generation time.

OPTCD=options

This subparameter is used to tell the system that certain additional facilities are to be provided for this file. Any combination of the following options can be specified for the OPTCD subparameter. If more than one option is specified, the options are written as a character string (i.e., without intervening commas or blanks). Note that if certain of these options are used, an additional subparameter must also be specified as indicated. In addition to the information supplied, the following default services are provided: (1) the COBOL compiler will supply OPTCD=L; and (2) in the case of an IBM 2321 mass storage device, the operating system will supply OPTCD=W.

- OPTCD=L: This option requests that the control program delete marked records. Marked records will be deleted when space for new records is required.
- OPTCD=Y: This option requests that a cylinder overflow area be created. It specifies that a certain number of tracks on each cylinder are to be reserved to contain any overflow records from other tracks on that cylinder. Another DCB subparameter, CYLOFL=xx, must also be written. The xx specifies the number of tracks on the cylinder to be reserved for the overflow area. The maximum number is 99.
- OPTCD=I: This option requests that an independent overflow area be reserved. It is used in

conjunction with DSNAME=dsname (OVFLOW) parameter in the DD statement used to allocate the independent area.

- OPTCD=M: This option requests that a master index be created (see "Master Index" for a discussion of master indexes). Another DCB subparameter, NTM=xx, must also be written. It specifies the maximum number of tracks to be contained in the cylinder index before a higher level index is created. The maximum value that can be specified is 99.

- OPTCD=R: This option requests reorganization criteria feedback, as described in "Reorganizing Files."

- OPTCD=W: This option requests the system to perform a write-validity check.

- OPTCD=U: This option requests that track index entries be accumulated in core storage until there are enough entries to fill a track. When the track is full all the entries will be written out. If enough core storage cannot be obtained entries will be written two at a time.

The following is an example of how the OPTCD subparameter can be used:

```
DCB=(DSORG=IS,OPTCD=M,NTM=20)
```

The foregoing example requests that a master index be created when the cylinder index exceeds 20 tracks.

BLKSIZE=integer
specifies the blocksize. This clause is used only if BLOCK CONTAINS 0 RECORDS was specified at compile time.

Note: Figure 28 shows the parameters that may be used in a DD statement when processing indexed files opened as output. Additional information about indexed file structure is contained in the publication IBM OS Data Management Services.

Using the DD Statements -- Single-Volume Files: The following examples refer to files that can be contained on one volume. Additional information about DD statements, including details on multivolume file allocation, can be found in the publication IBM OS Job Control Language Reference.

All three areas for an indexed file can be contained on a single volume if they are small enough. If such is the case and the programmer elects to allow the system to subdivide storage into the prime and index areas when the file is created, he need only code the following DD statement:

```
//ddname  DD  DSNAME=dsname(PRIME),        X
//            SPACE=(CYL,(no. of           X
//            cylinders)),UNIT=unit,       X
//            DCB=(DSORG=IS,...)
```

The DD statement given will produce a prime area with the index area occupying the last cylinder(s) of the space in the prime area. If any track(s) remain on the last cylinder after the index area, they are used as an independent overflow area; if no track(s) remain, an overflow area does not exist.

If the programmer definitely wants an independent overflow area, he must provide a second DD statement as follows:

```
//ddname  DD  DSNAME=dsname(PRIME),        X
//            SPACE=(CYL,(no. of           X
//            cylinders)),UNIT=unit,       X
//            VOLUME=SER=222222,           X
//            DCB=(DSORG=IS,OPTCD=I,...)
//        DD  DSNAME=dsname(OVFLOW),       X
//            SPACE=(CYL,(no. of           X
//            cylinders)),UNIT=unit,       X
//            VOLUME=SER=222222,           X
//            DCB=*.ddname
```

These DD statements will produce a prime area and a separate overflow area with the index area at the end of the overflow area. All three areas reside on the same volume.

Note: When more than on DD statement is used, only the first can be named. The others must not have a data definition name (ddname) but all must have the same data set name (dsname).

ddname	ddname used only for first DD statement of each file
DSNAME (DSN)	{dsname / &&name} (INDEX) (PRIME) (OVFLOW) Note: If more than one DD statement is used, elements must be in this order.
Device	Mass storage required
UNIT	DEFER not permitted
SEP, AFF	Restricted, see "Job Control Procedures"
VOLUME	Volume sequence number subparameter not applicable
LABEL	SL
SPACE	CYL / ABSTR ,... [,,CONTIG]
SUBALLOC	Not applicable
SPLIT	Not applicable
DISP	NEW[1] [,KEEP / ,PASS / ,DELETE]
DCB[2]	Required: DSORG=IS Optional: BUFNO=xxx BLKSIZE=xxxx OPTCD={W|M|Y|I|R|L|U}

[1]MOD not meaningful. CATLG allowed only if all areas are allocated with a single DD statement
[2]The DCB parameter should be the same for each DD statement

Figure 28. DD Statement Parameters Applicable to Indexed Files Opened as Output

If the programmer desires more control in the placement of the index area, he can subdivide storage before the data set is created by providing another DD statement as follows:

```
//ddname  DD  DSNAME=dsname(INDEX),            X
//              SPACE=(CYL,(no. of             X
//              cylinders)),UNIT=unit,         X
//              VOLUME=SER=333333,             X
//              DCB=(DSORG=IS,...)
//        DD  DSNAME=dsname(PRIME),            X
//              SPACE=(CYL,(no. of             X
//              cylinders)).UNIT=unit,         X
//              VOLUME=SER=333333,             X
//              DISP=(disp),DCB=*.ddname
```

These DD statements will produce two separate areas: index and prime. Each area is on the same volume.

If, along with more control of his index, the programmer wishes an independent overflow area, a third DD statement (OVFLOW) can be specified, as detailed earlier. The sequence will be:

```
//ddname  DD  DSNAME=dsname(INDEX),...
//        DD  DSNAME=dsname(PRIME),...
//        DD  DSNAME=dsname(OVFLOW),...
```

These DD statements will produce three separate areas: index, prime, and overflow.

Note that the OPTCD subparameter of the DCB parameter in each of the DD statements must specify an independent overflow area (OPTCD=I). All three areas reside on the same volume if so specified in the VOLUME parameter.

Note: The sequence of the DSNAME parameter elements in all of the foregoing examples must be followed when placing the DD statements into the input stream, or an abnormal termination of the job will result.

The example in Figure 29 defines a new indexed file that consists of three separate areas. All three areas reside on the same volume. The volume is on an IBM 2311 Disk Storage Drive.

```
//FILE    DD  DSNAME=ISM(INDEX),UNIT=2311,SPACE=(CYL,(1)),   X
//            VOLUME=SER=111111,DCB=(DSORG=IS,OPTCD=I,...)
//        DD  DSNAME=ISM(PRIME),UNIT=2311,SPACE=(CYL,(5)),   X
//            VOLUME=SER=111111,DISP=(,KEEP),DCB=*.FILE
//        DD  DSNAME=ISM(OVFLOW),UNIT=2311,SPACE=(CYL,(1)),  X
//            VOLUME=SER=111111,DISP=(,KEEP),DCB=*.FILE
```

Figure 29. Example of DD Statements for New Indexed Files

Cataloging Files: An indexed file can be cataloged if:

- All the areas of the file are allocated with a single DD statement. Such a file is cataloged in the usual manner by specifying the DISP parameter in the DD statement:

  DISP=(NEW,CATLG)

- The areas are allocated with more than one DD statement, but all volumes are on the same type of device. Such a file is cataloged using the IEHPROGM utility program (see the publication IBM OS Utilities).

An indexed file that is being created cannot be cataloged if its areas are on different device types. An existing indexed file cannot be cataloged through the specification of the CATLG subparameter of the DISP parameter in the DD statement.

Note: The DD statement(s) defining a new or existing indexed file can appear in cataloged procedures.

Calculating Space Requirements: To determine the number of cylinders required for an indexed file, the programmer must consider the number of records that will fit on a cylinder, the number of records that will be processed, and the amount of space required for indexes and overflow areas. In making the computations, additional space is also required for device overhead.

Note: The allocation of space to the different areas of an indexed file is permanent. New allocations can be achieved only by recreating the file. It is, therefore, important to remember:

- Unused space on the last cylinder on which data was written, in the prime area, is converted to an independent overflow area. Space allocated in excess of this cannot be released and will be wasted.

- Excess space allocated to overflow or index areas cannot be released.

Detailed information on space allocation can be found in the publication IBM OS Data Management Services.

Master Index: QISAM provides a master index facility to avoid inefficient serial searches of large cylinder indexes. The master index provides an index to the cylinder index. The programmer can specify with the DCB parameter in his DD statement(s) (see "DD Statement Requirements for Indexed Files" in "Creating Indexed Files") that a master index be built if the size of a cylinder index exceeds a certain number of tracks. Each entry in the master index points to a track of the cylinder index. If the size of the master index exceeds the number of tracks specified in the NTM parameter of the DD statement, the master index is automatically indexed by a higher level master index. Three such higher level master indexes can be constructed.

COBOL Considerations: When creating indexed files, the QISAM file processing technique is used. The following COBOL programming considerations should be noted:

- RECORD KEY Clause. The RECORD KEY clause in the SELECT sentence of the Environment Division is required. It is used to specify the location of the key within the record itself. If the RECORD KEY clause has a PICTURE clause that specifies that the item is binary (COMPUTATIONAL), zero is the lowest number acceptable as the first record. A negative key is considered to be larger than a positive key; therefore, if a record is inserted into the file, a negative key would place the record after those records with positive keys.

- Dummy Records. To reserve space for records to be added at a later time, when creating indexed files, dummy records can be written with the delete code (the figurative constant HIGH-VALUE) in the first byte. Dummy records and their deletion are described in "Using the WRITE Statement."

- Required and optional COBOL statements are summarized in Table 17.

Reading or Updating Indexed Files Sequentially

QISAM can be used to read or update an existing indexed file. Adding a record to an already existing file, however, can be done only with BISAM (see "Accessing an Indexed File Randomly").

When QISAM is used to read an input file, the READ statement makes available one logical record at a time in an ascending sequence determined by the record keys. Dummy records are not made available. If there are records in the overflow area, this sequence will not correspond exactly to the physical sequence of the records in the file. The file must have been created using QISAM.

When QISAM is used to update an I-O file, the READ and REWRITE statements permit updating-in-place or deletion of a logical record. Logical records are read sequentially and may be either updated and rewritten, or rewritten unaltered, from the same area. Alteration of record length or insertion of new records is not permitted. A logical record is marked for deletion by moving the figurative constant HIGH-VALUE into the first character position of the record and then using the REWRITE statement. Records in the file that contain this deletion code are not made available on input.

The discussion that follows is primarily concerned with indexed files that can be contained on a single volume. Additional information about processing existing indexed files accessed sequentially, including multivolume files, can be found in the publication IBM OS Job Control Language Reference.

Parameter Requirements: In the DD statement(s) indicating an existing indexed file, the following differences and requirements should be noted:

DCB
: The DSORG=IS subparameter must be specified, whereas the BUFNO subparameter is optional. The OPTCD field must not be specified again. Any OPTCD subparameter facilities that were specified when the file was created are in effect as long as the data set exists. For example, if the programmer specified the write-validity check option (OPTCD=W) when he created the file, the option is still in effect at the time of any subsequent WRITE statement. The BLKSIZE subparameter must not be specified. LRECL does not have to be specified if it already exists in the data set label.

DSNAME (DSN)
: This parameter is written DSNAME=dsname. The element subparameters (INDEX, PRIME, OVFLOW), must not be written.

DISP
: The first subparameter must be OLD. The second subparameter cannot be CATLG or UNCATLG (see "Cataloging Files" above for more information on cataloging indexed files).

Note: For further information about Indexed parameters, see "DD Statement Requirements for Indexed Files" in "Creating Indexed Files."

Only one DD statement is needed to specify an existing file if all of the areas are on one volume. The following is an example of a DD statement that can be used when processing a single-volume QISAM file.

```
//ddname  DD  DSNAME=dsname,                X
//            DCB=(DSORG=IS,...),           X
//            UNIT=unit,DISP=OLD
```

Further details about DD statements for existing single-volume and multivolume indexed files can be found in the publication IBM OS Job Control Language Reference.

Note: Figure 30 shows the parameters that may be used in a DD statement when processing indexed files opened as input or I-O. Additional information about indexed file structure is contained in the publication IBM OS Data Management Services.

Reorganizing Files: As new records are added to an indexed file, chains of records may be created in the overflow area if one exists. The access time for retrieving records in an overflow area is greater than that required for retrieving records in the prime area. Input/output performance is, therefore, sharply reduced when many overflow records develop. For this reason, an indexed file can be reorganized as soon as the need becomes evident. The system maintains a set of statistics to assist the programmer when reorganization is desired. These statistics are maintained as fields of the file's data control block. They are made available when APPLY REORG-CRITERIA is specified. If these statistics are desired, the OPTCD subparameter of the DCB parameter must have included the OPTCD=R parameter in each of the DD statements when the file was created. Additional information about reorganizing files is contained in the publication IBM OS Data Management Services.

Sequential Retrieval Using the START Statement: For indexed INPUT and I-O files, retrieval starts with the first nondummy record in the file. If the programmer wishes to begin processing at a point other than the beginning of the file, he can do so through the use of the START verb. When the START statement is used, the retrieval starts sequentially from the record specified in the NOMINAL KEY.

ddname	ddname used only for first DD statement of each file
DSNAME	dsname Note: Element subparameter must not be used.
Device	Mass storage required
UNIT	Applicable subparameter Note: Not needed if file is cataloged.
SEP, AFF	Restricted; see "Job Control Procedures"
VOLUME	Applicable subparameters
LABEL	SL
SPACE	Not applicable
SUBALLOC	Not applicable
SPLIT	Not applicable
DISP	OLD[1] [,KEEP / ,PASS / ,DELETE]
DCB	Required: DSORG=IS Optional: BUFNO=xxx (not allowed for BISAM) LRECL=xxx
[1]CATLG UNCATLG not permitted.	

Figure 30. DD Statement Parameters Applicable Indexed Files Opened as INPUT or I-O

COBOL Considerations: When processing an already existing file with QISAM, the following COBOL programming considerations should be noted:

- RECORD KEY Clause. The RECORD KEY always in the SELECT sentence of the Environment Division is required, just as it is when creating the file. Note other record key considerations under "Accessing an Indexed File Randomly."

- Delete Option. In order to keep the number of records in the overflow area to a minimum, and to eliminate unnecessary records, an existing record may be marked for deletion. This is done by moving the figurative constant HIGH-VALUE into the first character position of the record. The record is not physically deleted unless it is forced off its prime track by the insertion of a new record (see "Using the WRITE Statement" in "Accessing an Indexed File Randomly"), or if the file is reorganized. Records marked for deletion may be replaced (using BISAM) by new records containing equivalent keys. Execution of the READ statement in QISAM does not make available a record marked for deletion, whether the record has been physically deleted or not. Dummy records and deletion are discussed further in "Accessing an Indexed File Randomly."

Accessing an Indexed File Randomly

The file processing technique used for random retrieval of a logical record, the random updating of a logical record, and/or the random insertion of a record is BISAM (Basic Indexed Sequential Access Method). When accessing an indexed file randomly, both NOMINAL KEY and RECORD KEY must be specified. The format of the NOMINAL KEY is described briefly below:

Format
NOMINAL KEY IS data-name

Data-name may be any fixed-length Working Storage item from 1 through 255 bytes in length. If it is part of a logical record, it must be at a fixed displacement from the beginning of that record description (see the publication IBM OS Full American National Standard COBOL for additional information).

Since a RECORD KEY is used to identify a record to the system, the record keys associated with the logical records of the file may be thought of as a table of arguments. When a record is read or written, the contents of NOMINAL KEY is used as a search argument that is compared to the record keys of the file.

The following example illustrates the use of the NOMINAL KEY clause.

```
ENVIRONMENT DIVISION.
         .
         .
         .
    NOMINAL KEY IS NOM-KEY
    RECORD KEY IS REC-KEY.
         .
         .
         .
DATA DIVISION.
FILE SECTION.
FD  INDEXED-FILE
    LABEL RECORDS ARE STANDARD.
01  REC-1.
    02  DELETE-CODE   PIC X.
    02  REC-KEY       PIC 9(5).
         .
         .
         .
WORKING-STORAGE SECTION.
77  NOM-KEY           PIC 9(5).
```

Because of their complementary use of the indexed file organization, much of the information discussed above for QISAM also applies to BISAM. Differences are noted below.

Using the WRITE Statement: The programmer can use the WRITE statement to add a new record into an indexed file. The record is added on the basis of the value specified in the NOMINAL KEY. The contents of the NOMINAL KEY are used to locate the two records in the file between which the new record is to be inserted. The records sought are those that have values less than and greater than the values in the nominal key field. Two methods can be used to add records.

In the first method, the key to be added is a new key value. The record is inserted in place so that the sequence of the keys is maintained. If an overflow area exists, the insertion may cause records to be forced off the prime track into the overflow area. Dummy records forced off the track in this way are physically deleted and are not written in the overflow area.

In the second method, the key of the record to be added has the same value as that of a known dummy record. If the dummy record has not been physically deleted, it is replaced by the new record. If it has been physically deleted, the record is inserted as though it had a new key value. If the key of the record to be added has the same value as a record other then a dummy record, an INVALID KEY condition will result.

Notes:

- Records with a key higher (or lower) than the current highest (or lowest) key of the file may be added.

- Whenever a WRITE statement is executed the contents of RECORD KEY and NOMINAL KEY must be identical. Except in the case of dummy records, this value must be unique in the file.

Using the REWRITE Statement: If a record is to be updated, the indexed file should be opened as I-O and the REWRITE statement should be used. All REWRITE statements must be preceded by a READ statement. However, a READ statement can be followed by either a WRITE, REWRITE, or another READ.

Note: Whenever a REWRITE statement is executed the value contained in NOMINAL KEY and RECORD KEY must be identical.

Using the READ Statement: Records are retrieved on the basis of the value specified in the NOMINAL KEY. If the key of a record marked for deletion is specified and the record has not been physically deleted, it will be produced. If the record has been physically deleted, the READ statement will cause an INVALID KEY condition and control will go to the INVALID KEY routine if specified.

Note: Although the RECORD KEY clause must be specified, no value need be moved to the record key field before the execution of the READ statement. The search for the desired record is based on the contents of NOMINAL KEY.

COBOL Considerations: When processing an indexed file randomly, the following COBOL programming considerations should be noted:

- RECORD KEY Clause and NOMINAL KEY Clause. The RECORD KEY and NOMINAL KEY clauses in the SELECT sentence of the Environment Division are required. The RECORD KEY clause is used to specify the location of the key within the record itself. The NOMINAL KEY is used as a search argument to locate the proper record, and must not be defined within the file being processed. Note that since a RECORD KEY is defined within a record, the contents of RECORD KEY are not available after a WRITE statement has been executed for that record.

Table 17. Indexed File Processing on Mass Storage Devices

Data Management Techniques	Access Method	KEY Clauses	OPEN Statement	Access Verbs	CLOSE Statement
QISAM	SEQUENTIAL	RECORD NOMINAL	INPUT	READ [INTO] AT END START INVALID KEY	[WITH LOCK]
			OUTPUT	WRITE [FROM] INVALID KEY	
			I-O	READ [INTO] AT END START INVALID KEY REWRITE [FROM] INVALID KEY	
BISAM	RANDOM	RECORD NOMINAL	INPUT	READ [INTO] INVALID KEY	[WITH LOCK]
			I-O	READ [INTO] INVALID KEY WRITE [FROM] INVALID KEY REWRITE [FROM] INVALID KEY	

- TRACK-AREA Clause. Specifying the clause results in a considerable improvement in efficiency when a record is added to the file. If a record is added and the TRACK-AREA clause was not specified for the file, the contents of the NOMINAL KEY field are unpredictable after the WRITE statement is executed. In this case, the key must be reinitialized before the next WRITE statement is executed.

- APPLY REORG-CRITERIA Clause. If the OPTCD=R parameter was specified on the DD card for an indexed file when it was created, the APPLY REORG-CRITERIA clause can be used to obtain the reorganization statistics when the file is closed. These statistics are moved from the data control block to the identifier specified in the clause when a CLOSE statement is executed for the file.

- APPLY CORE-INDEX Clause. This clause specifies that the highest level index will reside in core storage during input/output operations. Otherwise, the index will be searched on the volume, and processing time will be longer.

- Required and optional COBOL statements are summarized in Table 17.

USING THE DD STATEMENT

Each data set that is defined by a DD statement is either to be created, or has been previously created and is to be retrieved. In either case, the data set must have a disposition; for example, if the data set is being created, the disposition must indicate whether the data set is to be cataloged, kept, or deleted. Other DD parameters may simply indicate that the data set is in the input stream or that ultimately the data set is to be printed or punched.

The following sections summarize the DD statement parameters and show examples for various uses of the DD statement. These sections include information about cataloging data sets and creating or referring to generation data groups; examples of cataloged data sets and partitioned data sets are included. For additional information about partitioned data sets see "Libraries." Also see "Appendix I: Checklist for Job Control Procedures" for additional examples of the DD statement used in job control procedures.

CREATING A DATA SET

When creating a data set, the programmer ordinarily will be concerned with the following parameters:

1. The data set name (DSNAME) parameter, which assigns a name to the data set being created.

2. The unit (UNIT) parameter, which allows the programmer to state the type and quantity of input/output devices to be allocated for the data set.

3. The volume (VOLUME) parameter, which allows specification of the volume in which the data set is to reside. This parameter also gives instructions to the system about volume mounting.

4. The space (SPACE), split cylinder (SPLIT), and suballocation (SUBALLOC) parameters, for mass storage devices only, which permit the specification of the type and amount of space required to accommodate the data set.

5. The label (LABEL) parameter, which specifies the type and some of the contents of the label associated with the data set.

6. The disposition (DISP) parameter, which indicates what is to be done with the data set by the system when the job step is completed.

7. The DCB parameter, which allows the programmer to specify additional information to complete the DCB associated with the data set (see "User-Defined Files"). This allows additional information to be specified at execution time to complete the DCB constructed by the compiler for a data set defined in the source program.

Figure 31 shows the subparameters that are frequently used in creating data sets. Additional subparameters are discussed in "Job Control Procedures."

Creating Unit Record Data Sets

Data sets whose destination is a printer or card punch are created with the DD statement parameters UNIT and DCB.

```
{DSNAME}   {dsname               }
{      } = {dsname(element)      }
{DSN   }   {&&name               }
           {&&name(element)      }

UNIT=(name[,n])

{VOLUME}
{      } =([PRIVATE][,RETAIN][,volume-sequence-number][,volume-count]
{VOL   }

         [,SER=(volume-serial-number[,volume-serial-number]...)]
         [                                                     ]
         [          dsname                                     ]
         [          *.ddname                                   ]  )
         [,REF=     *.stepname.ddname                          ]
         [          *.stepname.procstep.ddname                 ]

SPACE=(  {TRK                  }
         {CYL                  }  ,(primary-quantity[,secondary-quantity]
         {average-record-length}

         [,directory-quantity]))

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

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

DISP=(    [NEW]   [,DELETE]   [,DELETE]
          [   ]   [,KEEP  ]   [,KEEP  ]    )
          [MOD]   [,PASS  ]   [,CATLG ]
                  [,CATLG ]

DCB=(subparameter-list)
```

Figure 31. DD Statement Parameters Frequently Used in Creating Data Sets

UNIT: Required. Code unit information using the 3-digit address (e.g., UNIT=00E), the type (e.g., UNIT=1403), or the system-generated group name (e.g., UNIT=PRINTER).

DCB: Required only if the data control block is not completed in the processing program. Valid DCB subparameters are listed in "Appendix C: Fields of the Data Control Block."

Creating Data Sets on Magnetic Tape

Tape data sets are created using combinations of the DD statement parameters UNIT, LABEL, DSNAME, DCB, VOLUME, and DISP.

UNIT: Required, except when volumes are requested using VOLUME=REF. A unit can be assigned by specifying its address, type, or group name, or by requesting unit affinity with an earlier data set. Multiple output units and defer volume mounting can also be requested with this parameter.

LABEL: Required when the tape has user labels or does not have standard labels, and when the data set does not reside first on the reel. It is also used to assign a retention period and password protection.

DSNAME: Required for data sets that are to be cataloged or used by a later job.

DCB: Required only when data control block information cannot be specified in COBOL. Usually, such attributes as the logical record length (LRECL) and buffering technique (BFTEK) will have been specified in the processing program. Other attributes, such as the OPTCD field and the tape recording technique (TRTCH), are more appropriately specified in the DD statement. Valid DCB subparameters are listed in "Appendix C: Fields of the Data Control Block."

VOLUME: Optional, this parameter is used to request specific volumes. If VOLUME=REF is specified, and the existing data sets on the specified volume(s) are to be saved, indicate the data set sequence number in the LABEL parameter.

DISP: Required for data sets that are to be cataloged, passed, or kept. The programmer can specify conditional disposition as the third term in the DISP parameter to indicate how the data set is to be treated if the job step abnormally terminates.

Creating Sequential (BSAM or QSAM) Data Sets on Mass Storage Devices

Sequential data sets are created using combinations of the DD statements parameters UNIT, DSNAME, VOLUME, LABEL, DISP, DCB, and one of the space allocation parameters SPACE, SPLIT, or SUBALLOC.

UNIT: Required, except when volumes are requested using VOLUME=REF or space is allocated using SPLIT or SUBALLOC. Assign a unit by specifying its address, type, or group name, or by requesting unit affinity.

DSNAME: Required for all but temporary data sets.

Label: Required to specify label type and to assign a retention period or password protection.

DCB: Required only when data control block information is not completely specified in the processing program. Usually, such attributes as the logical record length (LRECL) and buffering technique (BFTEK) will have been specified in the processing program. Other attributes, such as the OPTCD field are more appropriately specified in the DD statement. Valid DCB subparameters are listed in "Appendix C: Fields of the Data Control Block."

VOLUME: Optional. This parameter requests specific volumes (SER and REF), specific volumes when the data set resides on more than one volume (seq #), multiple nonspecific volumes (volcount), private volumes (PRIVATE), or private volumes that are to remain mounted until the end of the job (RETAIN).

DISP: Required for data sets that are to be cataloged, passed, or kept. The programmer can specify conditional disposition as the third term in the DISP parameter to indicate how the data set is to be treated if the job step abnormally terminates.

SPACE, SPLIT, SUBALLOC: One of these is required for all new mass storage data sets.

Creating Direct (BDAM) Data Sets

Direct (BDAM) data sets are created using the same subset of DD statement parameters as sequential data sets, with the exception of the SPLIT parameter. Valid DCB subparameters for BDAM data sets are listed in "Appendix C: Fields of the Data Control Block."

Creating Indexed (BISAM and QISAM) Data Sets

Indexed (ISAM) data sets are created using combinations of the DD statement parameters UNIT, DSNAME, VOLUME, LABEL, DISP, DCB, and SPACE. The ISAM data sets occupy three areas of storage: an *index area* that contains master and cylinder indexes, a *prime area* that contains the data records and track indexes, and an optional *overflow area* to hold additional records when the prime area is exhausted. Detailed information on creating and retrieving indexed data sets is presented in "Appendix H: Creating and Retrieving Indexed Sequential Data Sets."

Creating Data Sets in the Output Stream

New data sets can be written on a system output device in much the same way as messages. When using a sequential scheduler, a data set is directed to the output stream with the SYSOUT and DCB parameters.

SYSOUT: Required. The output class through which the data set is routed must be specified. Output classes are identified by a single alphanumeric character.

DCB: Required only if complete data control block information has not been specified in the processing program.

When using a priority scheduler, data sets are not routed directly to a system output device. They are stored by the processing program on an intermediate mass storage device and later written on a system output device. In addition to the SYSOUT and DCB parameters, DD statements defining a data set of this type can also contain UNIT and SPACE parameters. All other parameters must be absent.

SYSOUT: Required. The output class through which the data set is routed must be specified. Output classes are identified by a single alphanumeric character. (Do not use classes 0 through 9 except in cases where the other classes are not sufficient.)

DCB: Required only if complete data control block information has not been specified in the processing program. Data control block information is used when the data set is written on an intermediate mass storage volume and read by the output writer. However, the output writer's own DCB attributes are used when the data set is written on the system output device. Valid DCB parameters are listed in "Appendix C: Fields of the Data Control Block."

UNIT: Optional. An intermediate mass storage device is assigned if UNIT is specified. A default device is assigned if this parameter is omitted.

SPACE: Optional. Estimate the amount of mass storage space required. A default estimate is assumed if this parameter is omitted.

Note: When a Direct SYSOUT Writer is used, the priority scheduler functions as a sequential scheduler. The SYSOUT data sets of the particular output class from any of the elegible job classes are not stored on an intermediate storage device, but are written directly to the system output device. When Direct SYSOUT Writer is used, all the parameters on the DD card are ignored. For detailed information on Direct SYSOUT Writer, see the publication *IBM OS Operator's Reference*, Order No. GC28-6691.

Examples of DD Statements Used To Create Data Sets

The following examples show various ways of specifying DD statements for data sets that are to be created. In general, the number of parameters and subparameters that are specified depend on the disposition of the data set at the end of the job step. If a data set is used only in the job step in which it is created and is deleted at the end of the job step, a minimum number of parameters are required. However, if the data set is to be cataloged, more parameters should be specified.

Example 1: Creating a data set for the current job step only.

```
//SYSUT1    DD UNIT=SYSDA,SPACE=(TRK,(50,10))
```

This example shows the basic required DD statement for creating and storing a data set on a mass storage device. The UNIT parameter is required unless the unit information is available from another source. If the data set were to be stored on a unit record or a tape device, the SPACE parameter would not be needed. The operating system assigns a temporary data set name and assumes a disposition of (NEW, DELETE).

Example 2: Creating a data set that is used only for the current job.

```
//SYSLIN    DD DSNAME=&&TEMP,DISP=(MOD,PASS),UNIT=SYSSQ,                X
//             SPACE=(TRK,(50))
```

This example shows a DD statement that creates a data set for use in more than one step of a job. The system assigns a unique symbol for the name, and this same symbol is substituted for each recurrence of the &&TEMP name within the job. The data set is allocated space on any available mass storage or tape device. If a tape device is selected, the SPACE parameter is ignored. The disposition specifies that the data set is either new or is to be added to (MOD), and is to be passed to the next job step (PASS). This DD statement can be used for specifying the data set that is created as output from the compiler and that is to be used as input to the linkage editor. By specifying MOD, separately compiled object modules can be placed in sequence in the same data set.

Note: If MOD is specified for a data set that does not already exist, the job may be abnormally terminated when a volume reference name, a volume serial number, or the disposition CATLG is specified or when the dsname is indicated by a backwards reference.

Example 3: Creating a data set that is to be kept but not cataloged.

```
//TEMPFILE  DD DSN=FILEA,DISP=(,KEEP),SPACE=(TRK,(30,10)),              X
//             UNIT=DIRECT,VOL=(,RETAIN,SER=AA70)
```

The example shows a DD statement that creates a data set that is kept but not cataloged. The data set name is FILEA. The disposition (,KEEP) specifies that the data set is being created in this job step and is to be kept. It is kept until a disposition of DELETE is specified on another DD statement. The KEEP parameter implies that the volume is to be treated as private. Private implies that the volume is unloaded at the end of the job step but because RETAIN is specified, the volume is to remain mounted until the end of the job unless another reference to it is encountered. The DIRECT parameter is a hypothetical device class, containing only mass storage devices. The volume with serial number AA70, mounted on a device in this class, is assigned to the data set. Space for the data set is allocated as specified in the SPACE parameter. The data set has standard labels since it is on a mass storage volume.

If the volume serial number were not specified in the foregoing example, the system would allocate space in an available nonprivate volume. Because KEEP is specified, the volume becomes private. (Another data set cannot be stored on a private volume unless its volume serial number is specified or affinity with a data set on the volume is stated.) The volume serial number of the volume assigned, if applicable, is included in the disposition message for the data set. Disposition messages are messages from the job scheduler, generated at the end of the job step.

Example 4: Creating a data set and cataloging it.

```
//DDNAMEA   DD DSNAME=INVENT.PARTS,DISP=(NEW,CATLG),                    X
//             LABEL=(,,EXPDT=71031),UNIT=DACLASS,                      X
//             VOLUME=(,REF=*.STEP1.DD1),                               X
//             SPACE=(CYL,(5,1),,CONTIG)
```

This example shows a DD statement that creates a data set named INVENT.PARTS and catalogs it in the previously created system catalog. The data set is to occupy the same volume as the data set referred to in the DD statement named DD1 occurring in the job step named STEP1. The UNIT parameter is ignored since REF is specified. Five cylinders are allocated to the data set, and if this space is exhausted, more space is allocated, one cylinder at a time. The five cylinders are to be contiguous. The disposition (CATLG), implies that the volume is to be private. The INVENT.PARTS is to have standard labels. The expiration date is the 31st day of 1971.

Example 5: Adding a member to a previously created library.

```
//SYSLMOD   DD DSNAME=SYS1.LINKLIB(INVENT),DISP=OLD
```

This DD statement adds a member named INVENT to the link library (SYS1.LINKLIB). When a member is added to a previously created data set, OLD should be specified. The member INVENT takes on the disposition of the library.

Example 6: Creating a library and its first member.

```
//SYSLMOD    DD DSNAME=USERLIB(MYPROG),DISP=(,CATLG),                    X
//              SPACE=(TRK,(50,30,3)),UNIT=2311,VOLUME=SER=111111
```

This DD statement creates a library, USERLIB, and places a member, MYPROG, in it. The disposition (,CATLG) indicates that the data set is being created in this job step (NEW is the default condition for the DISP parameter and is indicated by the comma) and is to be cataloged. The data set is to have standard labels. Space is allocated for the data set in a volume on a mass storage device that is an IBM 2311 unit. Initially, 50 tracks are allocated to the data set, but when this space is exhausted, more tracks are added, 30 at a time. The SPACE parameter must be specified when the library is created, and it must include allocation of space for the directory. SPACE cannot be specified when new members are added. If additional space is required when new members are added, the secondary allocation, if specified, will be used. Three 256-byte records are to be used for the directory. The volume serial number of the volume on which the library is to reside, is 111111.

Example 7: Replacing a member of an existing library.

```
//SYSLMOD    DD DSNAME=MYLIB(CASE3),DISP=OLD
```

This DD statement replaces the member named CASE3 with a new member with the same name. If the named member does not exist in the library, the member is added as a new member. In the foregoing example, the library is cataloged.

Example 8: Creating and adding a member to a library used only for the current job.

```
//SYSLMOD    DD DSNAME=&&USERLIB(MYPROG),DISP=(,PASS),UNIT=SYSDA,        X
//              SPACE=(TRK,(50,,1))
```

This DD statement creates and adds a member to a temporary library. It is similar to the DD statement shown in Example 6, except that a temporary name is used and the data set is not cataloged nor kept but is simply passed to the next job step. Since the data set is to be used only for this one job, it is not necessary to specify VOLUME and LABEL information. This statement can be used for a linkage edit job step in which the module is to be passed to the next step.

Note: If DISP=(,DELETE) is specified for a library, the entire library will be deleted.

RETRIEVING PREVIOUSLY CREATED DATA SETS

The parameters that must be specified in a DD statement to retrieve a previously created data set depend on the information that is available to the system about the data set. For example,

1. If a data set on a magnetic-tape or mass storage volume was created and cataloged in a previous job or job step, all information for the data set, such as volume, space, etc., is stored in the catalog and data set label. This information need not be repeated. Only the dsname and disposition parameters need be specified.

2. If the data set was created and kept in a previous job but has not been cataloged, information concerning the data set, such as space, record format, etc., is stored in the data set label. However, the unit and volume information must be specified unless available elsewhere.

3. If the data set was created in the current job step, or in a previous job step in the current job, the information in the previous DD statement is available to the system and is accessible by referring to the previous DD statement. Only the dsname and disposition parameters need be specified.

Note: A programmer may wish to change the previous disposition of a data set. For example, if KEEP was specified when the data set was created, the DD statement that retrieves the data set may change the disposition by specifying CATLG.

Figure 32 shows the parameters that are used to retrieve previously created data sets.

Retrieving Cataloged Data Sets

Input data sets, assigned a disposition of CATLG or cataloged by the IEHPROGM utility program, are retrieved using the DD statement parameters DSNAME, DISP, LABEL, and DCB. The device type, volume serial number, and data set sequence number (if tape) are stored in the catalog.

```
              dsname
 DSNAME       dsname(element)
          =   *.ddname
 DSN          *.stepname.ddname
              &&name
              &&name(element)

 UNIT=(name[,n])

 DCB=(subparameter-list)

                    ,DELETE     ,DELETE
          OLD       ,KEEP       ,KEEP
 DISP=(   SHR       ,PASS       ,CATLG       )
          MOD       ,CATLG      ,UNCATLG
                    ,UNCATLG

 LABEL=(subparameter-list)

 VOLUME
          =(subparameter-list)
 VOL
```

Figure 32. Parameters Frequently Used in Retrieving Previously Created Data Sets

DSNAME: Required. The data set must be identified by its cataloged name. If the catalog contains more than one index level, the data set name must be fully qualified.

DISP: Required. The status (OLD or SHR) of the data set must be given and an indication made as to how it is to be treated after its use, unless it is to remain cataloged. The programmer can specify as the third term in the DISP parameter a conditional disposition to indicate how the data set is to be treated if the job step abnormally terminates.

LABEL: Required only if the data set does not have a standard label.

DCB: Required only if complete data control block information is not specified by the processing program and the data set label. To save recoding time, DCB attributes can be copied from an existing DCB parameter and modified if necessary. Valid DCB subparameters are listed in "Appendix C: Fields of the Data Control Block."

Note: In addition to the disposition UNCATLG, a cataloged data set can be passed to a later step (PASS) or deleted (DELETE).

Retrieving Noncataloged (KEEP) Data Sets

Input data sets that were assigned a disposition of KEEP are retrieved by their tabulated name and location, using the DD statement parameters DSNAME, UNIT, VOLUME, DISP, LABEL, and DCB.

DSNAME: Required. The data set must be identified by the name assigned to it when it was created.

UNIT: Required, unless VOLUME=REF is used. The unit must be identified by its address, type, or group name. If the data set requires more than one unit, give the number of units. Deferred volume mounting and unit separation can be requested with this parameter.

VOLUME: Required. The volume(s) must be identified with serial numbers or, if the data set was retrieved earlier in the same job, with VOLUME=REF. If the volume is to be PRIVATE, it must be so designated. If a private volume is to remain mounted until a later job step uses it, RETAIN should be designated.

DISP: Required. The status (OLD or SHR) of the data set must be given and an indication made as to how it is to be treated after its use. The programmer can specify conditional disposition as the third term in the DISP parameter to indicate how the data set is to be treated if the job step abnormally terminates.

LABEL: Required if the data set does not have a standard label. If the data set resides with others on tape, its sequence number must be given.

DCB: Required for all indexed data sets. Otherwise, required only if complete data control block information is not supplied by the processing program and the data set label. To save recoding time, copy DCB attributes from an existing DCB parameter, and modify them if necessary. Valid DCB subparameters are listed in Appendix C.

Retrieving Passed Data Sets

Input data sets used in a previous job step and passed are retrieved using the DD statement parameters DSNAME, DISP, and UNIT. The data set's unit type, volume location, and label information remain available to the system from the original DD statement.

DSNAME: Required. The original data set must be identified by either its name or the DD statement reference term *.stepname.ddname. If the original DD statement occurs in a cataloged procedure, the procedure stepname must be included in the reference term.

DISP: Required. The data set must be identified as OLD, and an indication made as to how it is to be treated after its use. The programmer can specify conditional disposition as the third term in the DISP parameter to indicate how the data set is to be treated if the job step abnormally terminates.

UNIT: Required only if more than one unit is allocated to the data set.

Extending Data Sets with Additional Output

A processing program can extend an existing data set by adding records to it instead of reading it as input. Such a data set is retrieved using the same subsets of DD statement parameters described under the preceding three topics, depending on whether it was cataloged, kept, or passed when created. In each case, however, the DISP parameter must indicate a status of MOD. When MOD is specified, the system positions the appropriate read/write head after the last record in the data set. If a disposition of CATLG for an extended data set that is already cataloged is indicated, the system updates the catalog to reflect any new volumes caused by the extension.

When extending a multivolume data set where number of volumes might exceed the number of units used, the programmer should either specify a volume count or deferred mounting as part of the volume information. This ensures data set extension to new volumes.

Retrieving Data through an Input Stream

Data sets in the form of decks of cards or groups of card images can be introduced to the system through an input stream by interspersing them with control statements. To define a data set in the input stream, mark the beginning of the data set with a DD statement and the end with a delimiter statement. The DD statement must contain one of the parameters * or DATA. Use DATA if the data set contains job control statements and an * if it does not. Two DCB subparameters can also be coded when defining a data set in the input stream. In systems with MFT or MVT, data in the input stream is temporarily transferred to a mass storage device. The DCB subparameters BLKSIZE and BUFNO allow blocking of this data as it is placed on the mass storage device.

Notes:

When using a sequential scheduler:

- The input stream must be on a card reader or magnetic tape.
- Each job step and procedure step can be associated with only one data set in the input stream.
- The DD statement must be the last in the job step or procedure step.
- The records must be unblocked, and 80-characters in length.
- The characters in the records must be coded in BCD or EBCDIC.

When using a priority scheduler:

- The input stream can be on any device supported by QSAM.
- Each job step and procedure step can be associated with several data sets in an input stream. All such data sets except the first in the job must be preceded by DD * or DD DATA statements.
- The characters in the records must be coded in BCD or EBCDIC.

Examples of DD Statements Used to Retrieve Data Sets

Example 1: Retrieving a cataloged data set.

```
//CALC      DD    DSNAME=PROCESS,DISP=(OLD,PASS,KEEP)
```

This DD statement retrieves a cataloged data set named PROCESS. No UNIT or VOLUME information is needed. Since PASS is specified, the volume in which the data set is written is retained at the end of the job step. PASS implies that a later job step will refer to the data set. The last step in the job referring to the data set should specify the final disposition. If no other DD statement refers to the data set, it is assumed that the status of the data set is as it existed before this job. In the event of an abnormal termination, the KEEP disposition explicitly states the disposition of the data set.

Example 2: Retrieving a data set that was kept but not cataloged.

```
//TEMPFILE DD    DSNAME=FILEA,UNIT=DIRECT,VOLUME=SER=AA70,DISP=OLD
```

This DD statement retrieves a kept data set named FILEA. (This data set is created by the DD statement shown in Example 3 for creating data sets.) The data set resides on a device in a hypothetical device class, DIRECT. The volume serial number is AA70.

Example 3: Referring to a data set in a previous job step.

```
//SAMPLE    JOB
//STEP1     EXEC PGM=IKFCBL00,PARM=DECK
            .
            .
            .
//SYSLIN    DD   DSNAME=ALPHA,DISP=(NEW,PASS),UNIT=SYSSQ
//STEP2     EXEC PGM=IEWL
//SYSLIN    DD   *.STEP1.SYSLIN,DISP=(OLD,DELETE)
```

The DD statement SYSLIN in STEP2 refers to the data set defined in the DD statement SYSLIN in STEP1.

Example 4: Retrieving a member of a library.

```
//BANKING   DD   DSNAME=PAYROLL(HOURLY),DISP=OLD
```

The DD statement retrieves a member, HOURLY, from a cataloged library, PAYROLL.

DD STATEMENTS THAT SPECIFY UNIT RECORD DEVICES

A DD statement may simply indicate that data follows in the input stream or that the data set is to be punched or printed. Figure 33 shows the parameters of special interest for these purposes.

```
{*   }
{DATA}

SYSOUT=A

UNIT=name

DCB=(subparameters)
```

Note: The DCB parameter can be specified, where permissible, for data sets on unit record devices. For example, it can be specified for compiler data sets (other than SYSUT1, SYSUT2, SYSUT3, and SYSUT4) and data sets specified by the DD statements required for the ACCEPT and DISPLAY statements, when any of these data sets are assigned to unit-record devices.

Figure 33. Parameters Used To Specify Unit Record Devices

Example 1: Specifying data in the card reader.

```
//SYSIN     DD *
```

The asterisk indicates that data follows in the input stream. This statement must be the last DD statement for the job step. The data must be followed by a delimiter statement.

Example 2: Specifying a printer data set.

```
//SYSPRINT  DD SYSOUT=A
```

SYSOUT is the system output parameter; A is the standard device class for printer data sets.

Example 3: Specifying a card punch.

```
//SYSPUNCH  DD SYSOUT=B
```

B is the standard device class for punch devices.

CATALOGING A DATA SET

A data set is cataloged whenever CATLG is specified in the DISP parameter of the DD statement that creates or uses it. This means that the name and volume identification for the data set are placed in a system index called the catalog. (See "Processing with QISAM" in the section "Execution Time Data Set Requirements" for information about cataloging indexed data sets.) The information stored in the catalog is always available to the system; consequently, only the data set name and disposition need be specified in subsequent DD statements that retrieve the data set. See Example 4 in "Creating Data Sets," and Example 1 in "Retrieving Data Sets."

If DELETE is specified for a cataloged data set, any reference to the data set in the catalog is deleted unless the DD statement containing DELETE retrieves the data set in some way other than by using the catalog. If UNCATLG is specified for a cataloged data set, only the reference in the catalog is deleted; the data set itself is not deleted.

Note: A "cataloged data set" should not be confused with a "cataloged procedure" (see "Using the Cataloged Procedures").

GENERATION DATA GROUPS

It is sometimes convenient to save data sets as elements or generations of a generation data group (DSNAME=dsname (element)). A *generation data group* is a collection of successive, historically related data sets. Identification of data sets that are elements of a generation data group is based upon the time the data set is added as an element. That is, a generation number is attached to the generation data group name to refer to a particular element. The name of each element is the same, but the generation number changes as elements are added or deleted. The most recent element is 0, the element added previous to 0 is -1, the element added previous to -1 is -2, etc. A generation data group must always be cataloged.

For example, a data group named PAYROLL might be used for a weekly payroll. The elements of the group are:

PAYROLL(0)
PAYROLL(-1)
PAYROLL(-2)

where PAYROLL(0) is the data set that contains the information for the most current weekly payroll, and is the most recent addition to the group.

When a new element is added, it is called element(+n), where n is an integer greater than 0. For example, when adding a new element to the weekly payroll, the DD statement defines the data set to be added as PAYROLL(+1); at the end of the job the system changes its name to PAYROLL(0). The element that was PAYROLL(0) at the beginning of the job becomes PAYROLL(-1) at the end of the job, and so on.

If more than one element is being added in the same job, the first is given the number (+1), the next (+2) and so on.

NAMING DATA SETS

Each data set must be given a name. The name can consist of alphanumeric characters and the special characters, hyphen and the +0 (12-0 multipunch). The first character of the name must be alphabetic. The name can be assigned by the system, it can be given a temporary name, or it can be given a user-assigned name. If no name is specified on the DD statement that creates the data set, the system assigns to the data set a unique name for the job step. If a data set is used only for the duration of one job, it can be given a temporary name (DSNAME=&&name). If a data set is to be kept but not cataloged, it can be given a simple name. If the data set is to be cataloged it should be given a fully qualified data set name. The fully qualified data set name is a series of one or more simple names joined together so that each represents a level of qualification. For example, the data set name DEPT999.SMITH.DATA3 is composed of three simple names that are separated by periods to indicate a hierarchy of names. Starting from the left, each simple name indicates an index or directory within which the next simple name is a unique entry. The rightmost name identifies the actual location of the data set.

Each simple name consists of one to eight characters, the first of which must be alphabetic. The special character period (.) separates simple names from each other. Including all simple names and periods, the length of a data set name must not exceed 44 characters. Thus, a maximum of 21 qualification levels is possible for a data set name.

Programmers should not use fully qualified data set names that begin with the letters SYS and that also have a P as the nineteenth character of the name. Under certain conditions, data sets with the above characteristics will be deleted.

ADDITIONAL FILE PROCESSING INFORMATION

The following topics are discussed in this section: the data control block, error processing for COBOL files, and volume and data set labels.

More information about input/output processing is contained in the publication IBM OS Data Management Services.

DATA CONTROL BLOCK

Each data set is described to the operating system by a data control block (DCB). A data control block consists of a group of contiguous fields that provide information about the data set to the system for scheduling and executing input/output operations. The fields describe the characteristics of the data set (e.g., data set organization) and its processing requirements (e.g., whether the data set is to be read or written). The COBOL compiler creates a skeleton DCB for each data set and inserts pertinent information specified in the Environment Division, FD entry, and input/output statements in the source program. The DCB for each file is part of the object module that is generated. Subsequently, other sources can be used to enter information into the data control block fields. The process of filling in the data control block is completed at execution time.

Additional information that completes the DCB at execution time may come from the DD statement for the data set and, in certain instances, from the data set label when the file is opened.

Overriding DCB Fields

Once a field in the DCB is filled in by the COBOL compiler, it cannot be overridden by a DD statement or a data set label. For example, if the buffering factor for a data set is specified in the COBOL source program by the RESERVE clause, it cannot be overridden by a DD statement. In the same way, information from the DD statement cannot be overridden by information included in the data set label.

Identifying DCB Information

The links between the DCB, DD statement, data set label, and input/output statements are the filename, the system name in the ASSIGN clause of the SELECT statement, the ddname of the system-name, and the dsname (Figure 34).

1. The filename specified in the SELECT statement and in the FD entry of the COBOL source program is the name associated with the DCB.

2. Part of the system-name specified in the ASSIGN clause of the source program is the ddname link to the DD statement. This name is placed in the DCB.

3. The dsname specified in the DD statement is the link to the physical data set.

The fields of the data control block are described in the tables in Appendix C. They identify those fields for which information must be supplied by the source program, by a DD statement, or by the data set label. For further information about the data control block, see the discussion of the DCB macro instruction for the appropriate file processing technique in the publication IBM OS Data Management Services.

ERROR PROCESSING FOR COBOL FILES

System Error Recovery

During the processing of a COBOL file, data transmission to or from an input/output device may not be successful the first time it is attempted. If it is not successful, standard error recovery routines, provided by the operating system, attempt to clear the failure and allow the program to continue uninterrupted.

If an input/output error cannot be corrected by the system, an abnormal termination (ABEND) of the program may occur unless the programmer has specified some means of error analysis. Error processing routines initiated by the programmer are discussed in the following paragraphs, and in "Appendix G: Input/Output Error Conditions."

For sequential files, the programmer can specify a DD statement option (EROPT) that specifies the type of action to be taken by the system if an error occurs. This option can be specified whether or not a declarative is written. If a declarative is specified, the DD statement option is executed when a normal exit is taken from the declarative. See "Accessing a Standard Sequential File" for further information.

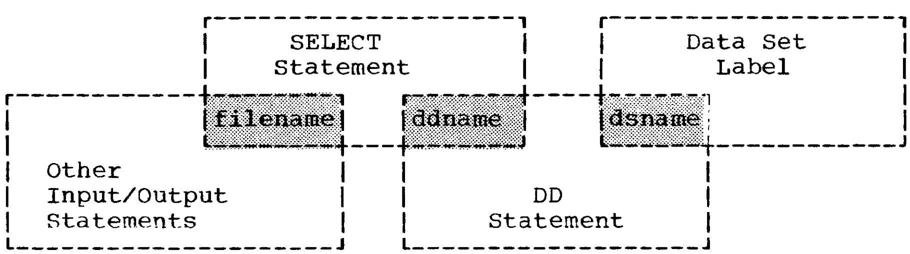


Figure 34. Links between the SELECT Statement, the DD Statement, the Data Set Label, and the Input/Output Statements

INVALID KEY Option

INVALID KEY errors may occur for files accessed randomly, or for output files accessed sequentially. A test to determine these errors may be made by using the INVALID KEY option of the READ, WRITE, REWRITE, or START verb.

Note: Secondary space allocation must be specified when the INVALID KEY option is used in a WRITE statement for QSAM and BSAM.

USE AFTER ERROR Option

The programmer may specify the USE AFTER ERROR option in the declarative section of the Procedure Division to determine the type of the input/output error. With the USE AFTER ERROR option, the programmer can pass control to an error-processing routine to investigate the nature of the error. If the GIVING option of the USE AFTER ERROR declarative is specified, data-name-1 will contain information about the error condition. Data-name-2, if specified, will contain the block in error if the last input/output operation was a read. If the file was opened as output, data-name-2 in the GIVING option cannot be referenced.

Data-name-2 of the GIVING option contains valid data only if data was actually transferred on the last input/output operation. For example, if the declarative is entered after execution of a START verb for a QISAM file on which no INVALID KEY option was present, an attempt to access data-name-2 results in an abnormal termination, because no transfer of data has taken place. Hence, the user should specify data-name-2 only within declaratives associated with READ statements. Otherwise, the user should define data-name-2 within the linkage section, so the user can examine data-name-1 and decide whether data-name-2 will be helpful.

Either or both the INVALID KEY clause and the USE AFTER ERROR declarative may be specified for a file. If both have been specified and an INVALID KEY error occurs, the imperative-statement specified in the INVALID KEY option will be executed. If both have been specified and any other type of input/output error occurs, the USE AFTER ERROR declarative will be entered. If an error occurs and neither has been specified, the program may terminate abnormally or may continue executing with incorrect data. Table 18 is a generalized summary of the means available for recovery from an invalid key condition or an input/output error. Table 19 lists the error processing facilities available for each type of file organization. The following discussion summarizes the action taken by each facility for each type. For further information on the USE AFTER ERROR option, see the publication IBM OS Full American National Standard COBOL.

STANDARD SEQUENTIAL

- Operating System: If the error cannot be corrected (read only), the program will ABEND in the absence of a DD statement option, USE AFTER STANDARD ERROR declarative, or INVALID KEY option. If both the DD statement option and USE section are specified, the control program will execute the USE declarative first and then the DD option if normal exit is taken from the declarative section. If no EROPT subparameter is indicated, or if ABS is specified and a USE AFTER STANDARD ERROR declarative exists, the declarative will receive control. After a normal exit, the job will abnormally terminate.

- DD Statement Option: The EROPT subparameter in the DCB parameter specifies one of three actions: accept the error block (ACC), skip the error block (SKP), or terminate the job (ABE).

- INVALID KEY: A transfer of control to the procedure indicated in the INVALID KEY phrase occurs if additional space cannot be allocated to write the record requested. This condition occurs when either no more space is available or 16 extents have already been allocated on the last volume assigned to the data set. The transfer of control occurs only if a secondary-quantity is specified in the DD statement SPACE, SPLIT, or SUBALLOC parameter. If no secondary-quantity is specified, the primary-quantity is assumed to be the exact amount of space required for the data set, and any attempt to write a record beyond the storage allocated causes the program to end abnormally. When an INVALID KEY error occurs, the file can be closed so that it may subsquently be reopened for retrieval as INPUT or I-O.

Table 18. Recovery from an Invalid Key Condition or from an Input/Output Error

Error \ Specified in COBOL Program	INVALID KEY option	USE AFTER STANDARD ERROR	Both	Neither on This Statement	Neither in Entire Program
Invalid key	Go to invalid key routine	Go to user's routine	Go to invalid key routine	Error ignored; next sequential instruction executed	ABEND
Input/Output Error	Return to system	Go to user's routine	Go to user's routine	Return to system	ABEND

Table 19. Input/Output Error Processing Facilities

File Organization \ Error Processing Facility Available	Operating System	DD Statement Option	COBOL Clauses: INVALID KEY	COBOL Clauses: USE AFTER STANDARD ERROR
Sequential	X	X	Note 1	X
Indexed (Sequential) WRITE	X		X	X
Indexed (Sequential) READ	X		Note 2	X
Indexed (Random)	Note 3		X	X
Direct or Relative (Sequential)	X		Note 1	X
Direct or Relative (Random)	X		X	X

Notes:
1. Holds only for WRITE.
2. Error cannot be caused by an invalid key.
3. No system error processing facility is available. If errors occur, they are ignored and processing continues, unless a programmer-specified error processing routine is indicated.

- USE AFTER STANDARD ERROR: The programmer may specify this option in order to display the cause of the error. Control goes to the declarative section; the programmer can then display a message indicating the error and execute his DD statement option on a normal exit from the declarative section.

INDEXED (RANDOM)

- INVALID KEY: If the error is caused by an invalid key, recovery is possible. If the error is not an invalid key and the USE AFTER ERROR option is not specified, the program is terminated.

- USE AFTER STANDARD ERROR: Control goes to the declarative section. The programmer can check the error type in the section by specifying data-name-1 in the GIVING option. If the error is caused by a key error or the "no space found" condition, recovery is possible. On a READ error, the block can be skipped by executing additional READ statements. If the error persists (more bad READ statements than the blocking factor), processing is limited to a CLOSE statement. Any other error cannot be corrected. The program may continue executing, but processing of the file is limited to CLOSE. If the programmer closes the file, he may do so in either the declarative section or in the main body of his program.

INDEXED (SEQUENTIAL)

A. WRITE (load mode)

- Operating System: If the error cannot be corrected, the program will ABEND unless an error processing option is specified.
- INVALID KEY: If the error is caused by an invalid key, recovery is possible. (The programmer may attempt to reconstruct the key and retry the operation, or may bypass the error record.)
- USE AFTER STANDARD ERROR: Control goes to the declarative section. The programmer can check the error type in the section by specifying data-name-1 in the GIVING option. If the error is the result of a key error, recovery is possible. If the error is not a key error, the error cannot be corrected. The program may continue executing, but processing of the file is limited to CLOSE. If the programmer closes the file, he may do so in either the declarative section or in the main body of his program.

B. READ, REWRITE (scan mode)

- Operating System: If the error cannot be corrected, the program will ABEND unless an error processing option is specified.
- INVALID KEY: The error cannot be caused by an invalid key. A source program coding error is implied and a compiler diagnostic message is generated.
- USE AFTER STANDARD ERROR: The programmer may specify this option in order to display the cause of the error. Control goes to the declarative section. The programmer can check the error type in the section by specifying data-name-1 in the GIVING option. Since the error cannot be caused by an invalid key, processing of the file is limited to CLOSE. If the programmer elects to close the file, he may do so in either the declarative section or in the main body of his program.

DIRECT or RELATIVE (RANDOM)

- Operating System: If the error cannot be corrected, the program will ABEND unless an error processing option is specified.
- INVALID KEY: If the error is caused by an invalid key, recovery is possible.
- USE AFTER STANDARD ERROR: Control goes to the declarative section. The programmer can check the error type in the section by specifying data-name-1 in the GIVING option. If the error is the result of a key error or the "no space found within the search limit" condition, recovery is possible. Any other error cannot be corrected. The program may continue executing, but processing of the file is limited to CLOSE. If the programmer closes the file, he may do so in either the declarative section or in the main body of his program.

DIRECT or RELATIVE (SEQUENTIAL)

- Operating System: If no error processing option is specified, a message is written to the console providing identification of the file and type of input/output error. Then control is returned to the system. For sequential data sets, if EROPT has SKP or ACC (as specified in the JCL for the data set), an ABEND will not occur and processing will continue.
- INVALID KEY: A transfer of control to the procedure indicated in the INVALID KEY phrase occurs if additional space cannot be allocated to write the record requested. This condition occurs when either no more space is available or 16 extents have already been allocated on the last volume assigned to the data set. The transfer of control occurs only if a secondary-quantity is specified in the DD statement SPACE, SPLIT, or SUBALLOC parameter. If no secondary-quantity is specified, the primary-quantity is assumed to be the exact amount of space required for the data set and any attempt to write a record beyond the storage allocated causes the program to end abnormally. When an INVALID KEY error occurs, the file can be closed

so that it may subsequently be reopened for retrieval as INPUT or I-O.

- USE AFTER STANDARD ERROR: The programmer may specify this option in order to display the cause of the error. Control goes to the declarative section. The programmer can check the error type in the section by specifying data-name-1 in the GIVING option. If the error is not the result of an invalid key, processing of the file is limited to CLOSE. If the programmer elects to close the file, he may do so in either the declarative section or in the main body of his program.

Notes: The user should consider the following when a relatively large number of INVALID KEY exits or declarative sequences (with GO TO exits) are to be executed:

1. The distinction between error processing via an error declarative and the INVALID KEY clause. When an input/output operation is requested, a storage area (called an input/output block, or IOB) is allocated until the request is satisfied (or, in the event of an error, until return from the user-provided error-handling routine). If the error declarative is used, a normal exit from the declarative returns control to the system and frees the IOB. When the INVALID KEY routine is used, however, the system does not regain control, and the IOB is not freed.

2. The error declarative dynamically allocates storage for a register save area upon entry. If a GO TO statement is used to exit from the declarative, neither this save area nor the IOB is freed.

   To make the maximum space available to other users, the programmer should rely on the declarative as much as possible, taking a normal exit from it. Otherwise, it is recommended that the programmer specify a larger region.

VOLUME LABELING

Various groups of labels may be used in secondary storage to identify magnetic-tape and mass storage volumes, as well as the data sets they contain. The labels are used to locate the data sets and are identified and verified by label processing routines of the operating system.

There are two different kinds of labels, standard and nonstandard. Magnetic tape volumes can have standard or nonstandard labels, or they can be unlabeled. The type(s) of label processing for tape volumes to be supported by an installation is selected during the system generation process. Mass storage volumes are supported with standard labels only.

Standard labels consist of volume labels and groups of data set labels. The volume label group precedes or follows data on the volume; it identifies and describes the volume. The data set label groups precede and follow each data set on the volume, and identify and describe the data set.

- The data set labels that precede the data set are called header labels.

- The data set labels that follow the data set are called trailer labels. They are almost identical to the header labels.

- The data set label groups can optionally include standard user labels except for ISAM files.

- The volume label groups can optionally include standard user labels for QSAM files.

Nonstandard labels can have any format and are processed by routines provided by the programmer. Unlabeled volumes contain only data sets and tapemarks. In the job control statements, a DD statement must be provided for each data set to be processed. The LABEL parameter of the DD statement is used to describe the data set's labels.

Specific information about the contents and physical location of labels is contained in the publications IBM OS Data Management Services, Order No. GC26-3746, and IBM OS Tape Labels, Order No. GC28-6680.

STANDARD LABEL FORMAT

Standard labels are 80-character records that are recorded in EBCDIC and odd parity on 9-track tape; or in BCD and even parity on 7-track tape. The first four characters are always used to identify the labels. These identifiers are:

VOL1	--	volume label
HDR1 and HDR2	--	data set header labels
EOV1 and EOV2	--	data set trailer labels (end-of-volume)
EOF1 and EOF2	--	data set trailer labels (end-of-data set)
UHL1 to UHL8	--	user header labels
UTL1 to UTL8	--	user trailer labels

The format of the mass storage volume label group is the same as the format of the tape volume label group, except one of the data set labels of the initial volume label consists of the data set control block (DSCB). The DSCB appears in the volume table of contents (VTOC) and contains the equivalent of the tape data set header and trailer information, in addition to space allocation and other control information.

STANDARD LABEL PROCESSING

Standard label processing as performed by the system consists of the following basic functions:

- Checking the labels on input data sets to ensure that the correct volume is mounted, and to identify, describe, and protect the data set being processed.
- Checking the existing labels on output data sets to ensure that the correct volume is mounted and to prevent overwriting of vital data.
- Creating and writing new labels on output data sets.

When a data set is opened for input, the volume label and the header labels are processed. For an input end-of-data condition, the trailer labels are processed when a CLOSE statement is executed. For an input end-of-volume condition, the trailer labels on the current volume are processed, and then the volume label and header labels on the next volume are processed.

When a data set is opened for output, the existing volume label and HDR1 label are checked, and new header labels are written. For an output end-of-volume condition, trailer labels are written on the current volume, the existing volume labels and header labels on the next volume are checked, and then new header labels are written on the next volume. When an output data set is closed, trailer labels are written.

STANDARD USER LABELS

Standard user labels contain user-specified information about the associated data set. User labels are optional within the standard label groups. The format used for user header labels (UHL1-8) and user trailer labels (UTL1-8) consists of a label 80 characters in length recorded in EBCDIC on 9-track tape units, or in BCD on 7-track tape units. The first three bytes consist of the characters that identify the label: UHL for a user header label (at the beginning of a data set) or UTL for a user trailer label (at the end-of-volume or end-of-data set). The next byte contains the relative position of this label within a set of labels of the same type and can be any number from 1 through 8. The remaining 76 bytes consist of user-specified information.

User labels are generally created, examined, or updated when the beginning or end of a data set or volume (reel) is reached. User labels are applicable for sequential, direct, and relative data sets. For sequentially processed data sets, end or beginning of volume exits are allowed (i.e., "intermediate" trailers and headers may be created or examined). For direct or relative data sets, user label routines will be given control only during OPEN or CLOSE condition for a file opened as INPUT, OUTPUT, or I-O. Trailer labels for files opened as INPUT or I-O are processed when a CLOSE statement is executed for the file that has reached an AT END condition. Thus, for standard sequential data sets, the user may create, examine, or update up to eight header labels and eight trailer labels on each volume of the data set, whereas for direct or relative data sets the user may create, examine, or update up to eight header labels during OPEN and up to eight trailer labels during CLOSE. Note that these labels reside on the initial volume of a multi-volume data set. This volume must be mounted at CLOSE if trailer labels are to be created, examined, or updated.

When standard user label processing is desired, the user must specify the label type of the standard and user labels (SUL) on the DD statement that describes the dataset. For mass storage volumes, specification of a LABEL subparameter of SUL results in a separate track being allocated for use as a user-label track when the data set is created. This additional track is allocated at initial allocation and for sequential data sets at end-of-volume (volume switch) time. The user-label track (one per volume of a sequential data set) will contain both user header and user trailer labels.

User Label Totaling
(BSAM and QSAM only)

When creating or processing a data set with user labels on a sequential file, the programmer may develop control totals to obtain exact information about each volume of the data set. This information can be stored in his user labels. For example, a control total accumulated as the data set is created, can be stored in a user label and later compared with a total accumulated while processing a volume. The user totaling facility enables the programmer to synchronize the control data that he has created while processing a data set with records physically written on a volume. In this way, he can tell exactly what records were written. This information can also be used for accurately labeling tape reels (i.e., assigning physical adhesive labels).

To request this option, specify OPTCD=T in the DCB parameter of the DD statement. The user's TOTALING area, where control data is accumulated, is provided by the user. In this area, the user can store information on each record he writes. When an input/output operation is scheduled, the control program sets up a user TOTALED save area that preserves an image of the information in the user's TOTALING area. When the output USE LABEL declarative is entered, the values accumulated in the user's TOTALING area corresponding to the last record actually written on the volume are stored in the TOTALED area. These values can be included in user labels.

When using this facility for an output data set (i.e., when creating the data set), the programmer must update his control data in the TOTALING area prior to issuing a WRITE instruction. When subsequently using this data set for input, the programmer can accumulate the same information as each record is read. These values can be compared with the ones previously stored in the user label when the records were created.

Variable length records with APPLY WRITE-ONLY or records with SAME RECORD AREA specified require special considerations when using the TOTALING option. Since the control program determines whether a variable-length record will fit in a buffer after a WRITE instruction has been issued, the values accumulated may include one more record than is actually written on the volume. In this case, the programmer must update his TOTALING area after issuing a WRITE instruction.

User label totaling is not available with S-mode records.

For further information on user label totaling, see the Program Product publication IBM OS Full American National Standard COBOL.

NONSTANDARD LABEL FORMAT

Nonstandard labels do not conform to the standard label formats. They are designed by programmers and are written and processed by programmers. Nonstandard labels can be any length less than 4096 bytes. There are no requirements as to the length, format, contents, and number of nonstandard labels, except that the first record on the volume cannot be a standard volume label. In other words, the first record cannot be 80 characters in length with the identifier VOL1 as its first four characters.

NONSTANDARD LABEL PROCESSING

To use nonstandard labels (NSL), the programmer must:

- Create nonstandard label processing routines for input header labels, input trailer labels, output header labels, and output trailer labels.
- Insert these routines into the operating system as part of the SVC library (SYS1.SVCLIB).
- Code NSL in the LABEL parameter of the DD statement at execution time.

The system verifies that the tape has a nonstandard label. Then if NSL is specified in the LABEL parameter, it loads the appropriate NSL routines into transient areas. These NSL routines are entered at OPEN, CLOSE, and END-OF-VOLUME conditions by the respective executors.

For a data set opened as output, the NSL routines entered include:

- At OPEN time, a header routine to check the old header and/or create the new header;
- At CLOSE time, a trailer-creation routine;
- At EOV time, a trailer-creation routine and a header routine.

For a data set opened as input essentially the same types of routines are required.

Note: The NSL routines must observe the following conventions:

1. Follow Type-IV SVC routine conventions.

2. Use GETMAIN and FREEMAIN for work areas.

3. Be reentrant load modules of 1024 bytes each.

4. Use EXCP for I/O operations and XCTL for passing control among load modules and then returning to the I/O-support routines.

5. Begin with the letters NSL if the system branches to them directly. (Other user-written modules having to do with nonstandard labels must begin with the letters IGC.)

6. Have as their entry points the first byte in each load module.

In addition, the NSL routines must write their own tapemarks, do all I/O operations necessary (via EXCP), determine when all labels have been processed, and take care of data set positioning. These routines may communicate at the LABEL source level with USE BEFORE LABEL PROCEDURE declaratives by means of linkage described under "User Label Procedure."

USER LABEL PROCEDURE

The USE...LABEL PROCEDURE statement provides the user with label handling procedures at the COBOL source level to handle nonstandard or user labels. The BEFORE option indicates processing of nonstandard labels. The AFTER option indicates processing of standard user labels. The labels must be listed as data-names in the LABEL RECORDS clause in the File Description entry for the file. When the file is opened as input, the label is read in and control is passed to the USE declarative if a USE...LABEL PROCEDURE is specified for the OPEN option or for the file. If the file is opened as output, a buffer area for the label is provided and control is passed to the USE declarative if a USE...LABEL PROCEDURE is specified for the OPEN option or for the file. For files opened as INPUT or I-O, control is passed to the USE declarative to process trailer labels when a CLOSE statement is executed for the file that has reached the AT END condition. A more detailed discussion of the USE...LABEL PROCEDURE statement is contained in the Program Product publication IBM OS Full American National Standard COBOL.

One of the concerns of the programmer is linkage between the nonstandard label SVC routine and the USE BEFORE LABEL PROCEDURE section. Other problems related to writing nonstandard label SVC routines are discussed in the publication IBM OS System Programmer's Guide.

When the nonstandard label SVC routine has determined that a particular DCB has nonstandard labels, the nonstandard label routine must inspect the DCB exit list for an active entry to ensure that there is a USE BEFORE...LABEL section for this DCB and for that type of label processing. The DCB field EXLST contains a pointer to this exit list. An active entry is defined as a 1-byte code other than X'00' or X'80' followed by a 3-byte address of the appropriate label section (Figure 35).

Code	Exit List
1	(USE section for header labels)
2	(USE section for trailer labels)
.	.
.	.
.	.

Notes:
1. Code 1 is set to X'01' indicating INPUT, or X'02' indicating OUTPUT.
2. Code 2 is set to 'X'OD' indicating INPUT, or X'04' indicating OUTPUT.

Figure 35. Exit List Codes

Once the nonstandard label SVC routine tests that the exit list confirms an appropriate active entry, it must pass the address of a parameter list in register 1.

The parameter list (Figure 36) must have the following format.

	1 byte	3 bytes
Byte 0	0	A(label buffer)
Byte 4	Flag byte	A(DCB)
Byte 8	Error flag	

Figure 36. Parameter List Formats

The A(label buffer) is the address of the label record on input and the address where the label will be created on output.

The A(DCB) is the address of the DCB. The DCB contains a pointer to the DEB. The nonstandard label SVC routine must test the

EOF bit in the OFLGS field of the DEB (data extend block) to determine whether to return control to the EOV or CLOSE module. Control is given to the CLOSE module only at EOF.

The error flag byte will have bit 0 set to 1 if an input/output error occurs when reading or writing a label.

Routine Type	Return Code	Applicable Note
Input header and/or trailer	0 4 16	1 2 3
Output header and/or trailer	4 8	1 2
Update header and/or trailer	8 12 16	1 2 3

Notes:

1. For output mode, the label is written or rewritten. For input mode, normal processing is resumed; any additional user labels are ignored.
2. Another label is read (for input mode) and control is returned to the USE BEFORE LABEL PROCEDURE section. For output mode, the labels should be written and control should be returned to the USE BEFORE LABEL PROCEDURE section. When control is returned to the nondeclarative portion, either normal processing will continue or the label section will be re-entered, depending on whether the return code is 4 or 8.
3. A return code of 16 indicates that the USE BEFORE LABEL PROCEDURE section has determined that an incorrect volume was mounted. When LABEL-RETURN is set to a nonzero value, the return code is set to 16.

Figure 37. Label Routine Return Codes

When the USE BEFORE LABEL PROCEDURE section returns control to the nonstandard label SVC routine, it will pass a return code that will indicate whether or not more labels are to be processed (Figure 37). This return code is set by assigning a value to the special register LABEL-RETURN.

The maximum size of the label record is stored on a halfword boundary at the EXITLIST address +38.

The user's nonstandard label routines are responsible for all tape positioning. For multifile volumes, the user may specify a file sequence number in the LABEL parameter on the DD card. The nonstandard label routines can inspect this information in the JFCB and position the files accordingly. For additional information, see the <u>IBM OS System Programmer's Guide</u>.

ASCII File Labels

ASCII files on magnetic tape may have American National Standard labels or American National Standard and user labels, or they may have no label. Any labels on an ASCII tape must be in ASCII code. Tapes containing a combination of ASCII and EBCDIC labels are not read. All the record formats supported (<u>i.e.</u>, fixed, undefined, and variable) are allowed on ASCII files, regardless of whether or not the files are labeled. Spanned records are not supported under ASCII.

When American National Standard labels are being processed, the label type must be specified in the DD statement that describes the data set. The parameter for American National Standard labels is LABEL=AL. The parameter for American National Standard and user labels is LABEL=AUL. Nonstandard labels are not permitted for ASCII files. The user may indicate no labels as LABELS=NL.

ASCII Standard Label Processing

Standard label processing for ASCII files is identical to standard label processing for files coded in EBCDIC. ASCII code is translated into EBCDIC code prior to processing.

ASCII User Label Processing

All American National Standard user labels (LABEL=AUL) are optional. ASCII files may have user header labels (UHLn) and user trailer labels (UTLn), which are processed very much like the standard user labels on EBCDIC files. However, there is no limit to the number of user labels possible at the beginning and the end of a file. No check is made on the number of labels written. It is left to the user to determine how many labels he wants written.

All user labels must be 80 bytes in length, but they may contain any user information desired.

Note: USE BEFORE STANDARD LABEL procedures are not allowed, because they are nonstandard.

User Label Exits

To create or verify user labels, the programmer must code for the file a USE AFTER STANDARD LABEL procedure.

Logical records may be in one of four formats: fixed-length (format F), variable-length (format V), unspecified (format U), or spanned (format S). F-mode files must contain records of equal lengths. Files containing records of unequal lengths must be V-mode, U-mode, or S-mode. Files containing logical records that are longer than physical records must be S-mode.

The record format is specified in the RECORDING MODE clause in the Data Division. If this clause is omitted, the compiler determines the record format from the record descriptions associated with the file. If the file is to be blocked, the BLOCK CONTAINS clause must be specified in the Data Division.

The prime consideration in the selection of a record format is the nature of the file itself. The programmer knows the type of input his program will receive and the type of output it will produce. The selection of a record format is based on this knowledge as well as an understanding of the type of input/output devices on which the file is written and of the access method used to read or write the file.

FIXED-LENGTH (FORMAT F) RECORDS

Format F records are fixed-length records. The programmer specifies format F records by including RECORDING MODE IS F in the file description entry in the Data Division. If this clause is omitted and both of the following are true:

- All records in the file are the same size
- BLOCK CONTAINS [integer-1 TO] integer-2... does not specify integer-2 less than the length of the maximum level-01 record

the compiler determines the recording mode to be F. All records in the file are the same size if there is only one record description associated with the file and it contains no OCCURS clause with the DEPENDING ON option; or if multiple record descriptions are all the same length.

The number of logical records within a block (blocking factor) is normally constant for every block in the file. When fixed-length records are blocked, the programmer specifies the BLOCK CONTAINS clause in the file description (FD) entry in the Data Division.

In unblocked format F, the logical record constitutes the block. The BLOCK CONTAINS clause is unnecessary for unblocked records.

Format F records are shown in Figure 38. The optional control character, represented by the letter C in Figure 37 is used for stacker selection and carriage control. When carriage control or stacker selection is desired, the WRITE statement with the ADVANCING or POSITIONING option is used to write records on the output file. In this case, one character position must be included as the first character of the record. This position will be automatically filled in with the carriage control or stacker select character. The carriage control character never appears when the file is written on the printer or punched on the card punch.

Note: Illustrations of unblocked Format F records do not take into account the key field required when direct organization is used.

```
┌─────────────────────────────────────────┐
│                                         │
│             Logical Record              │
│    ┌───┬─────────────────────────┐      │
│    │ C │         Data            │      │
│    └───┴─────────────────────────┘      │
│                                         │
│             Blocked Records             │
│  ┌───────────┬───────────┬───────────┐  │
│  │ Logical   │ Logical   │ Logical   │  │
│  │ Record    │ Record    │ Record    │  │
│  └───────────┴───────────┴───────────┘  │
│  <------------Fixed Length----------->  │
│                                         │
│             Unblocked Record            │
│  ┌─────────────────────────────────┐    │
│  │          Logical Record         │    │
│  └─────────────────────────────────┘    │
│  <------------Fixed Length----------->  │
│                                         │
└─────────────────────────────────────────┘
```
Figure 38. Fixed-length (Format F) Records

UNSPECIFIED (FORMAT U) RECORDS

Format U is provided to permit the processing of any blocks that do not conform to F, V, or S formats. Format U records are shown in Figure 39. The optional control character C, as discussed under "Fixed-Length (Format F) Records," may be used in each logical record.

The programmer specifies format U records by including RECORDING MODE IS U in the file description (FD) entry in the Data Division. U-mode records may be specified only for direct or standard sequential files.

If the RECORDING MODE clause is omitted, and BLOCK CONTAINS [integer-1 TO] integer-2... does not specify integer-2 less than the maximum level-01 record, the compiler determines the recording mode to be U if the file is direct and one of the following conditions exist:

- The FD entry contains two or more level-01 descriptions of different lengths.
- A record description contains an OCCURS clause with the DEPENDING ON option.
- A RECORD CONTAINS clause specifies a range of record lengths.

Each block on the external storage media is treated as a logical record. There are no record-length or block-length fields.

When a READ INTO statement is used for a U-mode file, the size of the longest record for that file is used in the MOVE statement. All other rules of the MOVE statement apply.

Note: Illustrations of Format U records do not take into account the key field required when direct organization is used.

```
┌─────────────────────────────────────────┐
│            Logical Record               │
│    ┌───┬─────────────────────┐          │
│    │ C │     Data            │          │
│    └───┴─────────────────────┘          │
│                                         │
│            Format U Record              │
│  ┌─────────────────────────────────┐    │
│  │         Logical Record          │    │
│  └─────────────────────────────────┘    │
│                                         │
└─────────────────────────────────────────┘
```
Figure 39. Unspecified (Format U) Records

VARIABLE LENGTH (FORMAT V) RECORDS

The programmer specifies format V records by including RECORDING MODE IS V in the file description entry in the Data Division. V-mode records may be specified only for direct or standard sequential files. If the RECORDING MODE clause is omitted and BLOCK CONTAINS [integer-1 TO] integer-2... does not specify integer-2 less than the maximum level-01 record, the compiler determines the recording mode to be format V if the file is standard sequential and one of the following conditions exist:

- The FD entry contains two or more level-01 descriptions of different lengths.
- A record description contains an OCCURS clause with the DEPENDING ON option.
- The RECORD CONTAINS clause specifies a range of record lengths.

V-mode records, unlike U-mode or F-mode records, are preceded by fields containing control information. These control fields are illustrated in Figures 40 and 41.

The first four bytes of each block contain control information (CC):

LL -- represents two bytes designating the length of the block (including the 'CC' field).

BB -- represents two bytes reserved for system use.

The first four bytes of each logical record contain control information (cc):

ll -- represents two bytes designating the logical record length (including the 'cc' field).

bb -- represents two bytes reserved for system use.

For unblocked V-mode records (Figure 40), the Data portion + CC + cc constitute the block.

For blocked V-mode records (Figure 41), the Data portion of each record + the cc of each record + CC constitute the block.

```
┌──────────────────────────────────────────┐
│                                          │
│                                          │
│     4         4           variable       │
│<--bytes-><--bytes--><------bytes------->│
│┌────┬────┬────┬────┬──────────────────┐│
││ LL │ BB │ 11 │ bb │       Data       ││
│└────┴────┴────┴────┴──────────────────┘│
│  \_______/  \_______/                    │
│    'CC'       'cc'                       │
│                                          │
└──────────────────────────────────────────┘
```

Figure 40. Unblocked V-Mode Records

```
FD  VARIABLE-FILE-1
    RECORDING MODE IS V
    BLOCK CONTAINS 35 TO 80 CHARACTERS
    RECORD CONTAINS 27 TO 72 CHARACTERS
    DATA RECORD IS VARIABLE-RECORD-1
    LABEL RECORDS ARE STANDARD.

01  VARIABLE-RECORD-1.
    LOGICAL RECORD
    05 FIELD-A      PIC X(20).
    05 FIELD-B      PIC 99.
    05 FIELD-C   OCCURS 1 TO 10 TIMES
       DEPENDING ON
       FIELD-B      PIC 9(5).
```

Variable-length record descriptions, for input and output files, must not define space for the control bytes. Control bytes are automatically provided when a record is written and are not communicated to the user when a file is read. Although they do not appear in the descriptions of logical records, control bytes do appear in the buffer areas of main storage. The compiler automatically allocates input and output buffers that are large enough to contain the required control bytes.

When variable-length records are written on unit record devices, control bytes are neither printed nor punched. They do appear, however, on other external storage devices. V-mode records moved from an input buffer to a working storage area will be moved without the control bytes.

Note: When a READ INTO statement is used for a V-mode file, the size of the longest record for that file is used in the MOVE statement. All other rules of the MOVE statement apply.

Example 1:

Consider the following standard sequential file consisting of unblocked V-mode records:

The LABEL RECORDS clause is always required. The DATA RECORD(S) clause is never required. If the RECORDING MODE clause is omitted, the compiler determines the mode as V since the record associated with VARIABLE-FILE-1 varies in length depending on the contents of FIELD-B. The RECORD CONTAINS clause is never required. The compiler determines record sizes from the record description entries. The BLOCK CONTAINS clause is also not required, since the compiler assumes unblocked records if the clause is omitted. Note: Record length calculations are affected by the following:

- When the BLOCK CONTAINS clause with the RECORDS option is used, the compiler adds four bytes to the logical record length and four more bytes to the block length.

- When the BLOCK CONTAINS clause with the CHARACTERS option is used, the user must include each cc + CC in the length calculation. In the definition of VARIABLE-FILE-1, the BLOCK CONTAINS clause specifies eight more bytes than does the RECORD CONTAINS clause. Four of these bytes are the logical record control bytes and the other four are the block control bytes.

```
┌──────────────────────────────────────────────────────────────────────────────┐
│                       1st                 2nd                  3rd           │
│                 Logical Record      Logical Record       Logical Record      │
│                \______________/   \______________/    \______________/       │
│   ┌────┬────┬────┬────┬─────────┬────┬────┬─────────┬────┬────┬─────────┐   │
│   │ LL │ BB │ 11 │ bb │ DATA-1  │ 11 │ bb │ DATA-2  │ 11 │ bb │ DATA-3  │   │
│   └────┴────┴────┴────┴─────────┴────┴────┴─────────┴────┴────┴─────────┘   │
│    \_______/ \_______/           \_______/           \_______/               │
│      'CC'                          'cc'                                      │
│  (block control              (record control                                 │
│     bytes)                        bytes)                                     │
└──────────────────────────────────────────────────────────────────────────────┘
```

Figure 41. Blocked V-Mode Records

In Example 1, assume that FIELD-B contains the value 02 for the first record of a file and FIELD-B contains the value 03 for the second record of the file. The first two records will appear on an external storage device and in buffer areas of main storage as shown in Figure 42.

If the file described in Example 1 had a blocking factor of 2, the first two records would appear on an external storage medium as shown in Figure 43.

Example 2:

If VARIABLE-FILE-2 is blocked, with space allocated for three records of maximum size per block, the following FD entry could be used when the file is created:

```
FD VARIABLE-FILE-2
    RECORDING MODE IS V
    BLOCK CONTAINS 3 RECORDS
    RECORD CONTAINS 20 TO 100 CHARACTERS
    DATA RECORDS ARE VARIABLE-RECORD-1,
      VARIABLE-RECORD-2
    LABEL RECORDS ARE STANDARD.

01  VARIABLE-RECORD-1.
    05 FIELD-A  PIC X(20).
    05 FIELD-B  PIC X(80).

01  VARIABLE-RECORD-2.
    05 FIELD-X  PIC X(20).
```

As mentioned previously, the RECORDING MODE, RECORD CONTAINS, and DATA RECORDS clauses are unnecessary. By specifying that each block contains three records, the programmer allows the compiler to provide space for three records of maximum size plus additional space for the required control bytes. Hence, 316 character positions are reserved by the compiler for each output buffer. If this size is other than that required, the BLOCK CONTAINS clause with the CHARACTERS option should be specified. If the block size is to be specified at execution time by use of the BLKSIZE subparameter on an associated DD card, BLOCK CONTAINS 0 CHARACTERS must be specified.

Note: Blocked variable-length records are permitted only when the file processing technique is standard sequential.

In Example 2, assume that the first six records written are five 100-character records followed by one 20-character record. The first two blocks of VARAIBLE-FILE-2 will appear on the external storage device as shown in Figure 44.

1st Block								2nd Block							
0040	BB	0036	bb	FIELD-A	02	FIELD-C	FIELD-C	0045	BB	0041	bb	FIELD-A	03	FIELD-C	FIELD-C	FIELD-C

Note: Lengths appear in decimal notation for illustrative purposes.

Figure 42. Fields in Unblocked V-Mode Records

	1st Record						2nd Record						
0081	BB	0036	bb	FIELD-A	02	FIELD-C	FIELD-C	0041	bb	FIELD-A	03	FIELD-C	FIELD-C	FIELD-C

Note: Lengths appear in decimal notation for illustrative purposes.

Figure 43. Fields in Blocked V-Mode Records

1st Block | 2nd Block

316	BB	104	bb	Data	104	bb	Data	104	bb	Data		236	BB	104	bb	Data	104	bb	Data	24	bb	Data

Note: Lengths appear in decimal notation for illustrative purposes.

Figure 44. First Two Blocks of VARIABLE-FILE-2

The buffer for the second block is truncated after the sixth WRITE statement is executed since there is not enough space left for a maximum size record. Hence, even if the seventh WRITE to VARIABLE-FILE-2 is a 20-character record, it will appear as the first record in the third block. This condition can be eliminated by using the APPLY WRITE-ONLY clause when creating files of variable-length blocked records.

Note: Illustrations of unblocked Format V records do not take into account the key field required when direct organization is used.

APPLY WRITE-ONLY Clause

The APPLY WRITE-ONLY clause is used to make optimum use of buffer space when creating a standard sequential file with blocked V-mode records.

Suppose VARIABLE-FILE-2 is being created with the following file description entry:

```
FD  VARIABLE-FILE-2
    RECORDING MODE IS V
    BLOCK CONTAINS 316 CHARACTERS
    DATA RECORDS ARE VARIABLE-RECORD-1,
      VARIABLE-RECORD-2
    LABEL RECORDS ARE STANDARD.

01  VARIABLE-RECORD-1.
    05 FIELD-A  PIC X(20).
    05 FIELD-B  PIC X(80).

01  VARIABLE-RECORD-2.
    05 FIELD-X  PIC X(20).
```

The first three WRITE statements to the file create one 20-character record followed by two 100-character records. Without the APPLY WRITE-ONLY clause, the buffer is truncated after the third WRITE statement is executed since the maximum size record no longer fits. The block is written as shown below:

236	bb	24	bb	Data	104	bb	Data	104	bb	Data

Using the APPLY WRITE-ONLY clause causes a buffer to be truncated only when the next record does not fit in the buffer. That is, if the next three WRITE statements to the file specify VARIABLE-RECORD-2, the block is created containing six logical records, as shown below:

308	bb	24	bb	Data	104	bb	Data	104	bb	Data

24	bb	Data	24	bb	Data	24	bb	Data

Note: When using the APPLY WRITE-ONLY clause, records must not be constructed in buffer areas. An intermediate work area must be used with a WRITE FROM statement.

SPANNED (FORMAT S) RECORDS

A spanned record is a logical record that may be contained in one or more physical blocks. Format S records may be specified for direct (BDAM, BSAM) files and for standard sequential (QSAM) files assigned to magnetic tape or to mass storage devices.

When creating files with S-mode records, if a record is larger than the remaining space in a block, a segment of the record is written to fill the block. The remainder of the record is stored in the next block or blocks, as required.

When retrieving a file with S-mode records, only complete records are made available to the user.

Spanned records are preceded by fields containing control information. Figure 44 illustrates the control fields.

BDF (Block Descriptor Field):

LL -- represents two bytes designating the length of the physical block (including the block descriptor field itself).

BB -- represents two bytes reserved for system use.

SDF (Segment Descriptor Field):

ll -- represents two bytes designating the length of the record segment (including the segment descriptor field itself).

bb -- represents two bytes reserved for system use.

Note: There is only one block descriptor field at the beginning of each physical block. There is, however, one segment descriptor field for each record segment within the block.

Each segment of a record in a block, even if it is the entire record, is preceded by a segment descriptor field. The segment descriptor field also indicates whether the segment is the first, the last, or an intermediate segment. Each block includes a block descriptor field. These fields are not described in the Data Division; provision is automatically made for them. These fields are not available to the user.

A spanned blocked file may be described as a file composed of physical blocks of fixed length established by the programmer. The logical records may be either fixed or variable in length and that size may be smaller, equal to, or larger than the physical block size. There are no required relationships between logical records and physical block sizes. Records of a spanned file may only be blocked when organization is sequential (QSAM).

A spanned unblocked file may be described as a file composed of physical blocks each containing one logical record or one segment of a logical record. The logical records may be either fixed or variable in length. When the physical block contains one logical record, the length of the block is determined by the logical record size. When a logical record has to be segmented, the system always writes the largest physical block possible. The system segments the logical record when the entire logical record cannot fit on the track.

Figure 46 is an illustration of blocked spanned records of SFILE. SFILE is described in the Data Division with the following file description entry:

```
FD  SFILE
    RECORD CONTAINS 250 CHARACTERS
    BLOCK CONTAINS 100 CHARACTERS
    .
    .
    .
```

Figure 46 also illustrates the concept of record segments. Note that the third block contains the last 50 bytes of REC-1 and the first 50 bytes of REC-2. Such portions of logical records are called record segments. It is therefore correct to say that the third block contains the last segment of REC-1 and the first segment of REC-2. The first block contains the first segment of REC-1 and the second block contains an intermediate segment of REC-1.

S-MODE CAPABILITIES

Formatting a file in the S-mode allows the user to make the most efficient use of external storage while organizing data files with logical record lengths most suited to his needs.

1. Physical record lengths can be designated in such a manner as to make the most efficient use of track capacities on mass storage devices.

2. The user is not required to adjust logical record lengths to maximum physical record lengths and their device-dependent variants when designing his data files.

3. The user has greater flexibility in transferring logical records across DASD types.

Spanned record processing will support the 2400 tape series, the 2311 and 2314 disk storage devices, and the 2321 data cell drive.

SEQUENTIAL S-MODE FILES (QSAM) FOR TAPE OR MASS STORAGE DEVICES

When the spanned format is used for QSAM files, the logical records may be either fixed or variable in length and are completely independent of physical record length. A logical record may span physical records. A physical record may contain one or more logical records and/or segments of logical records.

```
 <--4 bytes---> <--4 bytes--> <----------------Variable bytes----------------->
 | LL  |  BB  |  ll  |  bb  |          Data Record or Segment                 |
    BDF            SDF
```

Figure 45. Control Fields of an S-Mode Record

```
<--------100 bytes-------->   <--------100 bytes------->   <-50 bytes-> <-50 bytes->
|         REC-1          | G |          REC-1         | G |   REC-1   |   REC-2   |
       1st Block                     2nd Block                    3rd Block
```

Figure 46. One Logical Record Spanning Physical Blocks

Source Language Considerations

The user specifies S-mode by describing the file with the following clauses in the file description (FD) entry of his COBOL program:

- BLOCK CONTAINS integer-2 CHARACTERS
- RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS
- RECORDING MODE IS S

The size of the physical record must be specified using the BLOCK CONTAINS clause with the CHARACTERS option. Any block size may be specified. Block size is independent of logical record size.

The size of the logical record may be specified by the RECORD CONTAINS clause. If this clause is omitted, the compiler will determine the maximum record size from the record descriptions under the FD.

Format S may be specified by the RECORDING MODE IS S clause. If this clause is omitted, the compiler will set the recording mode to S if the BLOCK CONTAINS inteter-2 CHARACTERS clause was specified and either of the following conditions exist:

- Integer-2 is less than the largest fixed-length level-01 FD entry.
- Integer-2 is less than the maximum length of a variable level-01 FD entry (i.e., an entry containing one or more OCCURS clauses with the DEPENDING ON option).

Except for the APPLY WRITE-ONLY, APPLY RECORD-OVERFLOW, WRITE BEFORE ADVANCING, WRITE AFTER ADVANCING, or WRITE AFTER POSITIONING clauses, all the options for a variable file apply to a spanned file.

Processing Sequential S-Mode Files (QSAM)

Suppose a file has the following file description entry:

```
FD  SPAN-FILE
    BLOCK CONTAINS 100 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS DATAREC.

01  DATAREC.
    05  FIELD-A PIC X (100).
    05  FIELD-B PIC X (50).
```

Figure 47 illustrates the first four blocks of SPAN-FILE as they would appear on external storage devices (i.e., tape or mass storage) or in buffer areas of main storage.

Notes:

1. The RECORDING MODE clause is not specified. The compiler determines the recording mode to be S since the block size is less than the record size.

2. The length of each physical block is 100 bytes, as specified in the BLOCK CONTAINS clause. All required control fields, as well as data, must be contained within these 100 bytes.

3. No provision is made for the control fields within the level-01 entry DATAREC.

The preceding discussion dealt with S-mode records which were larger than the physical blocks that contained them. It is also possible to have S-mode records which are equal to or smaller than the physical blocks that contain them. In such cases, the RECORDING MODE clause must specify S (if so desired) since the compiler cannot determine this by comparing block size and record size.

One advangage of S-mode records over V-mode records is illustrated by a file with the following characteristics:

1. RECORD CONTAINS 50 TO 150 CHARACTERS

2. BLOCK CONTAINS 350 CHARACTERS

3. The first five records written are 150, 150, 150, 100, and 150 characters in length.

1st Block

4 bytes	4 bytes	92 bytes
LL	BB	ll	bb	DATAREC (1)

2nd Block

4 bytes	4 bytes	58 bytes	4 bytes	30 bytes
LL	BB	ll	bb	DATAREC (1)	ll	bb	DATAREC (2)

3rd Block

4 bytes	4 bytes	92 bytes
LL	BB	ll	bb	DATAREC (2)

4th Block

4 bytes	4 bytes	28 bytes	4 bytes	60 bytes
LL	BB	ll	bb	DATAREC (2)	ll	bb	DATAREC (3)

Figure 47. First Four Blocks of SPAN-FILE

RECORDING MODE IS V	RECORDING MODE IS S
150 | 150 G 150 | 100 G 150	150 | 150 | 50 G 100 | 100 | 150
R1 R2 R3 R4 R5	R1 R2 R3 R4 R5

Note: The enclosed diagrams are for illustrative purposes only. Neither takes into account the space required for control fields.

Figure 48. Advantage of S-Mode Records Over V-Mode Records

For V-mode records, buffers are truncated if the next logical record is too large to be completely contained in the block (Figure 48). This results in more physical blocks and more inter-record gaps on the external storage device.

Note: For V-mode records, buffer truncation occurs:

1. When the maximum level-01 record is too large.

2. If APPLY WRITE-ONLY or SAME RECORD AREA is specified and the actual logical record is too large to fit into the remainder of the buffer.

For S-mode records, all blocks are 350 bytes in length and records that are too large to fit entirely into a block will be segmented. This results in more efficient use of external storage devices since the number of inter-record gaps are minimized (Figure 48).

A second advangage of S-mode processing over that of V-mode is that the user is no longer limited to a record length that does not exceed the track of the mass storage device selected. Records may span tracks, cylinders, extents, and volumes.

QSAM spanned records differ from other QSAM record formats because of an allocation of an area of main storage known as the "Logical Record Area." If logical records span physical blocks, COBOL will use this Logical Record Area to assemble complete logical records. If logical records do not span blocks (i.e., they are contained within a single physical block) the Logical Record Area is not used. Regardless, only complete logical records are made available to the user. Both READ and WRITE statements should be thought of as manipulating complete logical records not record segments.

The allocation of a Logical Record Area ~~may be a disadvantage to the COBOL user.~~ Additional main storage, consisting of 36 bytes + the maximum record length, will always be required. The Logical Record Area is discussed in detail in "Finding Data Records in an Abnormal Termination Dump."

DIRECTLY ORGANIZED S-MODE FILES (BDAM AND BSAM)

When S-mode is used for directly organized files, only unblocked records are permitted. Logical records may be either fixed or variable in length. A logical record will span physical records if, and only if, it spans tracks. A physical record will contain only one logical record or a segment of a logical record. A track may contain a segment of a logical record, or segments of two logical records and/or whole logical records. Records may span tracks, cylinders, and extents, but not volumes.

```
      Sequential File 3                                   Direct File

  ┌─────┬──────────┬─────────────┐              ┌────┐   ┌──────────┐   ┌───────┐
  │ R1  │    R2    │      R3     │ ...1st track... │ R1 │ G │    R2    │ G │  R3   │
  └─────┴──────────┴─────────────┘              └────┘   └──────────┘   └───────┘

  ┌──────────────────────────────┐              ┌──────────────────────────────┐
  │              R3              │ ...2nd track... │              R3              │
  └──────────────────────────────┘              └──────────────────────────────┘

  ┌───────────┬──────────────────┐              ┌──────────────────┐   ┌─────────┐
  │     R3    │        R4        │ ...3rd track... │        R3        │ G │   R4    │
  └───────────┴──────────────────┘              └──────────────────┘   └─────────┘

                                   ...4th track... ┌───────┐
                                                │  R4   │
                                                └───────┘
```

Figure 49. Direct and Sequential Spanned Files on a Mass Storage Device

Source Language Considerations

The user specifies S-mode by describing the file with the following clauses in the file description (FD) entry of his COBOL program:

- BLOCK CONTAINS integer-2 CHARACTERS

- RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS

- RECORDING MODE IS S

The size of a logical record may be specified by the RECORD CONTAINS clause. If this clause is omitted, the compiler will determine the maximum record size from the record descriptions under the FD.

The spanned format may be specified by the RECORDING MODE IS S clause. If this clause is omitted, the compiler will set the recording mode to S if the BLOCK CONTAINS integer-2 CHARACTERS clause was specified and integer-2 is less than the greatest logical record size. This is the only use of the BLOCK CONTAINS clause. It is otherwise treated as comments.

The physical block size is determined by either:

1. The logical record length.

2. The track capacity of the device being used.

If, for example, the track capacity of a mass storage device is 3625 characters, any record smaller than 3625 characters may be written as a single physical block. If a logical record is greater than 3625 characters, the record is segmented. The first segment may be contained in a physical block of up to 3625 bytes, and the remaining segments must be contained in succeeding blocks. In other words, a logical record will span physical blocks if, and only if, it spans tracks.

Figure 49 illustrates four variable-length records (R1, R2, R3, and R4) as they would appear in direct and sequential files on a mass storage device. In both cases, control fields have been omitted for illustrative purposes. For both files, assume:

1. BLOCK CONTAINS 3625 CHARACTERS (track capacity = 3625)

2. RECORD CONTAINS 500 TO 5000 CHARACTERS

In the sequential file, each physical block is 3625 bytes in length and is completely filled with logical records. The file consists of three physical blocks, occupies three tracks, and contains no inter-record gaps.

In the direct file, the physical blocks vary in length. Each block contains only one logical record or one record segment. Logical record R3 spans physical blocks only because it spans tracks. The file consists of seven physical blocks, occupies more than three tracks, and contains three inter-record gaps.

Processing Directly Organized S-Mode Files (BDAM and BSAM)

When processing directly organized files, there are two advantages spanned format has over the other record formats:

1. Logical record lengths may exceed the length restriction of the track capacity of the mass storage device. If, for example, the track capacity of a mass storage device is 2000 bytes, this does not represent the maximum length of the logical record that can be specified (even when the device does not have a Track Overflow feature).

   Note: Even when the spanned format is used, the COBOL restriction on the length of logical records must be adhered to (i.e., a maximum length of 32,767 characters).

2. S-mode records give the user the same facility as the Track Overflow feature. If neither RECORDING MODE IS S nor APPLY RECORD-OVERFLOW is specified, only complete logical records can be written on any single track. This means that when a track has only 900 unoccupied bytes and a record of 1000 bytes is to be added, it will be written on the next available track. This is inefficient, since a 900 byte segment could be added to the current track by means of either APPLY RECORD-OVERFLOW or RECORDING MODE IS S.

   Note: If a choice exists between Track Overflow and S-mode records, neither has any particular advantage over the other with regard to the efficient use of storage space.

The disadvantage of BSAM and BDAM spanned records is similar to that mentioned for QSAM. A segment work area is always allocated which occupies additional main storage.

Like QSAM, the processing of BSAM and BDAM spanned records relies on an interaction between buffers, segment work areas, and Logical Record Areas. For QSAM, input-output buffers are used as the segment work area and complete logical records are assembled in a Logical Record Area before being made available to the user if the record is segmented. If the record is not segmented, the logical record is made available to the user within the buffer unless the SAME AREA clause is specified. For BSAM and BDAM, input-output buffers are used as a Logical Record Area and a separate segment work area must be allocated. Segment work areas and Logical Record Areas are described fully in "Finding Data Records in an Abnormal Termination Dump."

OCCURS CLAUSE WITH THE DEPENDING ON OPTION

If a record description contains an OCCURS CLAUSE WITH THE DEPENDING ON option, the record length is variable. This is true for records described in an FD as well as in the Working-Storage section. The previous sections discussed four different record formats. Three of them, V-mode, U-mode, and S-mode, may contain one or more OCCURS clauses with the DEPENDING ON option.

The following section discusses some factors that affect the manipulation of records containing OCCURS clauses with the DEPENDING ON option. The text indicates whether the factors apply to the File (FD) or Working-Storage sections, or both.

The compiler calculates the length of records containing an OCCURS clause with the DEPENDING ON option at two different times, as follows (the first applies to FD entries only, the second to both FD and Working-Storage entries):

1. When a file is read and the object of a DEPENDING ON option is within the record.

2. When the object of the DEPENDING ON option is changed as a result of a move to it or to a group that contains it. (The length is not calculated when a move is done to an item which redefines or renames it.)

Consider the following example:

```
WORKING-STORAGE SECTION.

77  CONTROL-1     PIC 99.
77  WORKAREA-1    PIC 9(6)V99.
        .
        .
        .
01  SALARY-HISTORY.
    05  SALARY OCCURS 0 TO 10 TIMES
        DEPENDING
        ON CONTROL-1  PIC 9(6)V99.
```

The Procedure Division statement MOVE 5 TO CONTROL-1 will cause a recalculation of the length of SALARY-HISTORY. MOVE SALARY (5) TO WORKAREA-1 will not cause the length to be recalculated.

The compiler permits the occurrence of more than one level-01 record, containing the OCCURS clause with the DEPENDING ON option, in the same FD entry (Figure 50). If the BLOCK CONTAINS clause is omitted, the buffer size is calculated from the longest level-01 record description entry. In Figure 50, the buffer size is determined by the description of RECORD-1 (RECORD-1 need not be the first record description under the FD).

During the execution of a READ statement, the length of each level-01 record description entry in the FD will be calculated (Figure 50). The length of the variable portions of each record will be the product of the numeric value contained in the object of the DEPENDING ON option and the length of the subject of the OCCURS clause. In Figure 50, the length of FIELD-1 is calculated by multiplying the contents of CONTROL-1 by the length of FIELD-1; the length of FIELD-2, by the product of the contents of CONTROL-2 and the length of FIELD-2; the length of FIELD-3 by the contents of CONTROL-3 and the length of FIELD-3.

```
FD  INPUT-FILE
    .
    .
    .
    DATA RECORDS ARE RECORD-1 RECORD-2 RECORD-3.

01  RECORD-1.
    02  CONTROL-1               PIC 99.
    02  FIELD-1 OCCURS 0 TO 10 TIMES DEPENDING ON CONTROL-1  PIC 9(5).

01  RECORD-2.
    02  CONTROL-2               PIC 99.
    02  FIELD-2 OCCURS 1 TO 5 TIMES DEPENDING ON CONTROL-2   PIC 9(4).

01  RECORD-3.
    02  FILLER                  PIC XX.
    02  CONTROL-3               PIC 99.
    02  FIELD-3 OCCURS 0 TO 10 TIMES DEPENDING ON CONTROL-3  PIC X(4).
```

Figure 50. Calculating Record Lengths When Using the OCCURS Clause with the DEPENDING ON Option

Since the execution of a READ statement makes available only one record type (i.e., RECORD-1 type, RECORD-2 type, or RECORD-3 type), two of the three record descriptions in Figure 50 will be inappropriate. In such cases, if the contents of the object of the DEPENDING ON option does not conform to its picture, the length of the corresponding record will not be calculated. For the contents of an item to conform to its picture:

- An item described as USAGE DISPLAY must contain decimal data.
- An item described as USAGE COMPUTATIONAL-3 must contain internal decimal data.
- An item described as USAGE COMPUTATIONAL must contain binary data.

The following example illustrates the length calculations made by the system when a READ statement is executed:

```
FD
    .
    .
    .
01  RECORD-1.
    05  A  PIC  99.
    05  B  PIC  99.
    05  C  PIC  99  OCCURS 5 TIMES
        DEPENDING ON A.

01  RECORD-2.
    05  D  PIC  XX.
    05  E  PIC  99.
    05  F  PIC  99.
    05  G  PIC  99  OCCURS 5 TIMES
        DEPENDING ON F.

WORKING-STORAGE SECTION.
    .
    .
    .
01  TABLE-3.
    05  H OCCURS 10 TIMES DEPENDING ON B.


01  TABLE-4.
    05  I OCCURS 10 TIMES DEPENDING ON E.
```

When a record is read, lengths are determined as follows:

1. The length of RECORD-1 is calculated using the contents of field A.

2. The length of RECORD-2 is calculated using the contents of field F.

3. The length of TABLE-3 is calculated using the contents of field B.

4. The length of TABLE-4 is calculated using the contents of field E.

The user should be aware of several additional factors that affect the successful manipulation of variable-length records. The following example illustrates a group item (i.e., REC-1) whose subordinate items contain an OCCURS clause with the DEPENDING ON option and the object of that DEPENDING ON option.

```
WORKING-STORAGE SECTION.
01  REC-1.
    05  FIELD-1             PIC S9.
    05  FIELD-2 OCCURS 5 TIMES DEPENDING ON
                FIELD-1  PIC X(5).
01  REC-2.
    05  REC-2-DATA          PIC X(50).
```

The results of executing a MOVE to the group item REC-1 will be affected by the following:

- The length of REC-1 may have been calculated at some time prior to the execution of this MOVE statement. The user should be sure that the current length of REC-1 is the desired one.

- The length of REC-1 may never have been calculated at all. In this case, the result of the move will be unpredictable.

- After the move, since the contents of FIELD-1 have been changed, an attempt will be made to recalculate the length of REC-1. This recalculation, however, will be made only if the new contents of FIELD-1 conform to its picture. In other words, if FIELD-1 does not contain an external decimal item, the length of REC-1 will not be recalculated.

Note: According to the COBOL description, FIELD-2 can occur a maximum of five times. If, however, FIELD-1 contains an external decimal item whose value exceeds five, the length of REC-1 will still be calculated. One possible consequence of this invalid calculation will be encountered if the user attempts to initialize REC-1 by moving zeros or spaces to it. This initialization would inadvertently delete part of the adjacent data stored in REC-2.

The following example applies to updating a record containing an OCCURS clause with the DEPENDING ON option and at least one other subsequent entry. In this case, the subsequent entry is another OCCURS clause with the DEPENDING ON option.

```
WORKING-STORAGE SECTION.
01  VARIABLE-REC.
    05  FIELD-A       PIC X(10).
    05  CONTROL-1     PIC S99.
    05  CONTROL-2     PIC S99.
    05  VARY-FIELD-1 OCCURS 10 TIMES
        DEPENDING ON CONTROL-1 PIC X(5).
    05  VARY-FIELD-2 OCCURS 10 TIMES
        DEPENDING ON CONTROL-2 PIC X(9).

01  STORE-VARY-FIELD-2.
    05  VARY-FLD-2 OCCURS 10 TIMES
        DEPENDING ON CONTROL-2 PIC X(9).
```

Assume that CONTROL-1 contains the value 5 and VARY-FIELD-1 contains 5 entries.

In order to add a sixth field to VARY-FIELD-1, the following steps are required:

```
MOVE VARY-FIELD-2 TO STORE-VARY-FIELD-2.
ADD 1 TO CONTROL-1.
MOVE 'additional field' TO
  VARY-FIELD-1 (CONTROL-1).
MOVE STORE-VARY-FIELD-2 TO VARY-FIELD-2.
```

For a discussion of the use of the OCCURS DEPENDING ON clause in a sort program, see "Sorting Variable-Length Records."

A programmer using the Full American National Standard COBOL Compiler, Version 4, under the IBM Operating System, has several methods available to him for testing and debugging his programs. Use of the symbolic debugging features is the easiest and most efficient method for testing and debugging and is described in detail in this chapter.

"Appendix A: A Sample Program" contains an example of a program run without the symbolic debugging features. The chapter "Program Checkout" contains information useful for finding the instruction that causes the abnormal termination and then correcting the problem. The chapters "Output" and "Using the Checkpoint/Restart Feature" include a discussion of compiler output and a description of taking checkpoints and restarting programs, respectively.

USE OF THE SYMBOLIC DEBUGGING FEATURES

As an aid to debugging, compiler options can be requested that provide additional diagnostic information for an abnormal termination other than "Canceled by Operator" or, under MVT, as a result of exceeding the system-state time slice. Three user options are available for object-time debugging -- the statement number option (STATE), the flow trace option (FLOW), and the symbolic debug option (SYMDMP).

The STATE option causes the number of the card for the last verb executed before termination to be printed out. The FLOW option causes a trace of the last user-specified number of procedures executed to be printed out (with a default of 99). Both STATE and FLOW cause the PROGRAM-ID, the condition code, and the last problem PSW to be printed out. The SYMDMP option enables the user to request a symbolic formatted dump of the data area of the object program for an abnormal termination, or to request dynamic dumps of data areas at strategic points during execution.

Use of these features requires no source language coding; rather the user specifies these options at compile time, through job control language. Operation of the SYMDMP option is dependent on execution-time control cards. Figure 51 illustrates the output generated for each of these features.

STATE Option

If the STATE option is in effect and an abnormal termination occurs, the printed output includes the compiler-generated card number or, if NUM is in effect, the card sequence number for the last verb executed. To use the STATE option, the programmer must:

- Request the option at compile time by specifying STATE in the PARM field or, if a cataloged procedure is used, in the PARM.COB field.

- Include a //SYSDBOUT DD card for the output data set at execution time.

For additional information, see "Options for the Compiler."

FLOW Option

If the FLOW option is in effect, a formatted list containing the PROGRAM-ID and either the compiler-generated card number or the line number (if NUM is in effect) of the last n executed procedures is printed on SYSDBOUT. The number of procedures traced can vary from 1 to 99 and is specified by the programmer. To use the flow trace facility, the programmer must:

- Request a trace at compile time by specifying FLOW in the PARM field or, if a cataloged procedure is used, in the PARM.COB field.

- Indicate the number of procedures to be traced at compile time or, by specifying FLOW[=nn] on the EXEC card, at execution time.

- Include a //SYSDBOUT DD card for the output data set to be used for the trace.

The number of procedures to be traced may be specified at compile time via either the PARM parameter or, if a cataloged procedure is used, the PARM.COB field. This number may be overridden at execution time via the PARM parameter or, if a cataloged procedure is used, the PARM.GO parameter. If the number of procedures traced is specified at neither compile time nor execution time, either the default value of 99 or the value specified at program product installation will be employed.

If batch compilation is used, FLOW can be specified at compile time and remain in effect for every program in the batch. To suppress a trace for a particular program within the batch, the programmer should specify NOFLOW at execution time as the last parameter in the PARM field for that program, or change the CBL card. For more information, see the sections "Options for the Compiler" and "Options for Execution."

Note: The FLOW option is completely independent of the READY/RESET TRACE feature of the debugging language.

SYMDMP Option

If the SYMDMP option is in effect, a symbolic formatted dump of the object program's data area is produced when the program abnormally terminates. This option also enables the programmer to request dynamic dumps of specified data-names at strategic points during program execution. If two or more COBOL programs are link-edited together and one of them terminates abnormally, a formatted dump is produced for all programs in the calling sequence compiled with the SYMDMP option, up to and including the main program.

Note: The terminating program need not have been compiled with the SYMDMP option.

The abnormal termination dump consists of the following parts:

1. An abnormal termination message, including the number of the statement and of the verb being executed at the time of an abnormal termination.

2. Selected areas in the Task Global Table.

3. A formatted dump of the Data Division including:

   (a) For an SD -- the card number, the sort-file-name, the type, and the sort record.

   (b) For an FD -- the card number, the file-name, the type, the ddname, the DECB and/or DCB status, the contents of the DECB and/or DCB in hexadecimal, and the fields of the record.

   (c) For an RD -- the card number, the report-name, the type, the report line, and the contents of PAGE-COUNTER and LINE-COUNTER if present.

   (d) For a CD -- the CD itself in its implicit format, as well as the area containing the message data currently being buffered.

   (e) For an index name -- the name, the type, and the contents in decimal.

The symbolic dump option is requested at compile time via the SYMDMP option, through the PARM parameter of the EXEC card. Operation of the symbolic dump option is dependent on object-time control cards placed in the SYSDBG data set. This data set must consist of unblocked 80-byte records. If the object-time control cards are not present, SYMDMP is cancelled at execution time. These cards are discussed below.

Object-Time Control Cards

The operation of the SYMDMP option is determined by two types of control cards:

Program-control card -- required if abnormal termination and/or dynamic dumps are requested.

Line-control card -- required only if dynamic dumps are requested.

Syntax Rules: The fields of both the program-control card and the line-control card must conform to the following rules:

1. Control cards are essentially free form, i.e., parameters coded on these cards can start in any column. However, parameters may not extend beyond column 71.

2. Each parameter except the last must be immediately followed by a comma or a blank.

3. No commas are needed to account for optional parameters that are not specified.

4. All upper-case letters in IBM documentation represent specifications that are to appear in the actual statement exactly as shown.

5. All lower-case letters represent generic terms that are to be replaced in the actual statement.

6. Brackets are used to indicate that a specification is optional and is not always required in the statement.

7. Brackets enclosing stacked items indicate that a choice of one item may, but need not, be made by the programmer.

8. Braces enclosing stacked items indicate that a choice of one item must be made by the programmer.

9. All punctuation marks and special characters shown in the statement formats other than hyphens, brackets, braces, and underscores, must be punched exactly as shown. This includes commas, parentheses, and the equal sign.

   Note: Blanks may be substituted for commas.

Continuation Cards: To continue either the program-control card or the line-control card, the programmer must code a nonblank character in column 72 of the continued card. Individual keywords and data-names cannot be split between cards.

Control Statement Placement: If a main program is compiled with the SYMDMP option, or if at least one subprogram called by the main program is a COBOL program compiled with the SYMDMP option, the control cards may either follow or precede the programmer's data, if any, in the input stream:

```
//GO          EXEC   PGM=
//GO.SYSDBG   DD     *

  {user's control cards}

/*
//GO.SYSIN    DD     *

  {user's data cards, if any}

/*
```

For an example of the control statements used to compile a program with the SYMDMP option, see Figure 51.

Program-Control Cards: A program-control card must be present at execution time for any program requesting a SYMDMP service. Program-control cards have the following format:

```
                            ┌   ┌ (HEX)   ┐ ┐
program-id,ddname[,ENTRY]   │ , ┤         ├ │
                            └   └ (NOHEX) ┘ ┘
```

where:

program-id
: is a 1-8 character program-name of a COBOL program compiled with the SYMDMP option. This parameter is required and must appear first on the program-control card.

ddname
: is the ddname assigned to the file that was produced at compile time on SYSUT5. This parameter is required and must follow the program-id.

ENTRY
NOENTRY
: ENTRY is used to provide a trace of a program-name when several programs are link-edited together. Each time the program whose PROGRAM-ID matches the "program-id" parameter is entered, its name is displayed.

HEX
NOHEX
: is optional and refers to the format of the Data Division area in the abnormal termination dump. If HEX is specified, level-01 items are provided in hexadecimal. Items subordinate to level-01 items are printed in EBCDIC, if possible. Level-77 items are provided both in EBCDIC and hexadecimal. If HEX is not specified, items subordinate to level-01 items and level-77 items are provided in EBCDIC. If these items are unprintable, hexadecimal notation is provided.

Line-Control Cards: Line-control cards have the following format:

```
line-num[,(verb-num)][,ON n][,m][,k]

 ┌ [,(HEX)  ]                               ┐
 │ [,(NOHEX)]  ,ALL                         │
 ┤                                          ├
 │ [,(HEX) ]                                │
 └ [,NOHEX)]   {,name1 [THRU name2]}...     ┘
```

line-num
: indicates the card number associated with the point in the Procedure Division at which the dynamic dump is to be taken. The card number is either the compiler-generated number or, if NUM is in effect, the user's number in card columns 1 through 6.

verb-num
: indicates the position of the verb in the card indicated by "line-num" before whose execution a dynamic dump is taken. When "verb-num" is not specified, the value 1 is assumed; when specified, "verb-num" must follow "line-num" and may not exceed 15.

ON n[,m][,k]

is equivalent to the COBOL statement ON n AND EVERY m UNTIL k... This option limits the requested dynamic dumps to specified times. For example, "ON n" would result in one dump, given the nth time "line-num" is reached during execution. "ON n,m" would result in a dump the first time at the nth execution of "line-num" and thereafter at every mth execution until end-of-job.

HEX
<u>NOHEX</u>

refers to the format of the Data Division areas provided in the dynamic dump. If HEX is specified, level-01 items are provided in hexadecimal. Items subordinate to level-01 items are printed in EBCDIC, if possible. Level-77 items are printed in both EBCDIC and hexadecimal. If HEX is not specified, items subordinate to level-01 items and level-77 items are provided in EBCDIC. If the items are unprintable, hexadecimal notation is provided. Note that if "name1" is specified and it represents a group item and HEX has not been specified, neither the group nor the elementary items in the group will be provided in hexadecimal.

name1 [THRU name2]

represents selected areas of the Data Division to be dumped. With the THRU option, a range of data-names appearing consecutively in the Data Division is dumped. "name1" and "name2" may be qualified but not subscripted. If the programmer wishes to see a subscripted item, specifying the name of the item without the subscript results in a dump of every occurrence of that item.

<u>ALL</u>

results in a dump of everything that would be dumped in the event of an abnormal termination. The purpose of ALL is to allow the programmer to receive a formatted dump at normal end-of-job. To do this, the generated statement number of the line on which a STOP RUN, EXIT PROGRAM, or GOBACK statement appears must be specified as the "line-num" parameter.

OVERALL CONSIDERATIONS

The end-of-file control card, slash asterisk (/*) must end the symbolic debug control card data set. If a run unit includes one or more programs that have been compiled with the SYMDMP option and no symbolic dump is required at execution time, the input data set is not required. In this case, SYMDMP responds with the following message:

IKF177I- SYMDMP CANCELLED. NO CONTROL CARDS.

SAMPLE PROGRAM -- TESTRUN

Figure 51 is an illustration of a program that utilizes the Symbolic Debugging feature. In the following description of the program and its output, letters identifying the text correspond to letters in the program listing.

(A) Because the SYMDMP option is requested in the PARM parameter of the EXEC card, the logical unit SYSUT5 must be assigned at compile time.

(B) The PARM parameter specifications on the EXEC card indicate that an alphabetically ordered cross-reference dictionary, a flow trace of 10 procedures, and the SYMDMP option are being requested.

(C) An alphabetically ordered cross-reference dictionary of data-names and procedure-names is produced by the compiler as a result of the SXREF specification in the PARM parameter of the EXEC card.

(D) The file assigned at compile time to SYSUT5 to store SYMDMP information is assigned to DD1 at execution time.

(E) The SYMDMP control cards placed in the input stream at execution time are printed along with any diagnostics.

(1) The first card is the program-control card where:

(a) TESTRUN is the PROGRAM-ID.

(b) DD1 is the ddname of the SYSUT5 file at execution time.

(2) The second card is a line-control card which requests a (HEX) formatted dynamic dump of KOUNT, NAME-FIELD, NO-OF-DEPENDENTS, and RECORD-NO prior to the first and every fourth execution of generated card number 70.

(3) The third card is also a line-control card which requests a (HEX) formatted dynamic dump of WORK-RECORD and B prior to the

execution of generated card number 81.

(F) The type code combinations used to identify data-names in abnormal termination and dynamic dumps are defined. Individual codes are illustrated in Table 20.

(G) The dynamic dumps requested by the first line-control card.

(H) The dynamic dumps requested by the second line-control card.

(I) Program interrupt information is provided by the system when a program terminates abnormally.

(J) The statement number information indicates the number of the verb and of the statement being executed at the time of the abnormal termination. The name of the program containing the statement is also provided.

(K) A flow trace of the last 10 procedures executed is provided because FLOW=10 was specified in the PARM parameter of the EXEC card.

(L) Selected areas of the Task Global Table are provided as part of the abnormal termination dump.

(M) For each file-name, the generated card number, the file type, the file status, the file organization, the DCB status, and the fields of the DCB and DECB, if applicable, are provided in hexadecimal.

(N) The fields of records associated with each FD are provided in the format requested on the program-control card.

(P) The contents of the fields of the Working-Storage Section are provided in the format requested on the program-control card.

(Q) The value associated with each of the possible subscripts is provided for each of the data items described with an OCCURS clause.

(R) Asterisks appearing within the EBCDIC representation of the value of a given field indicate that the type and the actual content of the field conflict.

Note: When the SYMDMP option is used, level numbers appear "normalized" in the symbolic dump produced. For example, a group of data items described as:

```
01   RECORDA.
     05 FIELD-A.
       10   FIELD-A1 PIC X.
       10   FIELD-A2 PIC X.
```

will appear as follows in SYMDMP output:

```
01 RECORDA...
02 FIELD-A...
03 FIELD-A1...
03 FIELD-A2...
```

Debugging TESTRUN

1. Reference to the statement number information J provided by the SYMDMP option shows that the abnormal termination occurred during the execution of the first verb on card 81.

2. Generated card number 81 contains the statement COMPUTE B = B + 1.

3. Through verification of the contents of B at the time of the abnormal termination R, it can be seen that the usage of B (numeric packed) conflicts with the value contained in the data area reserved for B (numeric display).

4. The abnormal termination occurred during an attempt to perform an addition on a display item.

More complex errors may require the use of dynamic dumps to isolate the problem area. Line-control cards are included in TESTRUN merely to illustrate how they are used and what output they produce.

Table 20. Individual Type Codes Used in SYMDMP Output

Code	Meaning
A	Alphabetic
B	Binary
D	Display
E	Edited
*	Subscripted Item
F	Floating Point
N	Numeric
P	Packed Decimal
S	Signed
OL	Overpunch Sign Leading
OT	Overpunch Sign Trailing
SL	Separate Sign Leading
ST	Separate Sign Trailing

```
   IEF298I  DEBUG    SYSOUT=U.
   //DEBUG  JOB   7074722674,'D. DAVIDSON',MSGLEVEL=1,MSGCLASS=G
   //JOBLIB DD DSN=DUMMY0S,UNIT=2314,VOL=SER=DC156,DISP=SHR
   // DD DSN=PRODVER4,DISP=SHR
B  // EXEC UCOB4CLG,PARM.COB='DMAP,PMAP,SXREF,FLOW=10,SYMDMP,QUOTE,NORES'
   XXCOB EXEC PGM=IKFCBL00,REGION=80K,PARM=(LOAD)                         00000010
   //COB.SYSPRINT  DD  SYSOUT=G,OUTLIM=1000                                    SMF
   X/SYSPRINT DD SYSOUT=U,OUTLIM=1000                                     00000SMF
   XXSYSUDUMP DD SYSOUT=U,OUTLIM=1000                                     00000SMF
   XXSYSUT1 DD  SPACE=(CYL,(10,2)),UNIT=2314                              00000040
   XXSYSUT2 DD  SPACE=(CYL,(10,2)),UNIT=(2314,SEP=SYSUT1)                 00000050
   XXSYSUT3 DD  SPACE=(CYL,(10,2)),UNIT=(2314,SEP=(SYSUT1,SYSUT2))        00000060
   XXSYSUT4 DD  SPACE=(CYL,(10,2)),UNIT=(2314,SEP=(SYSUT1,SYSUT2,SYSUT3)) 00000070
A  //COB.SYSUT5 DD    DSNAME=&&UT5,UNIT=SYSDA,SPACE=(TRK,(100,10)),
   //  DISP=(NEW,PASS)
   X/SYSUT5  DD  SPACE=(CYL,(10,2)),UNIT=2314,DSN=&SYMDBG,DISP=(NEW,PASS) 00000080
   XXSYSLIN DD DSN=&LOADSET,DISP=(MOD,PASS),UNIT=2314,SPACE=(CYL,(10,2))  00000090
   //COB.SYSIN DD *
```

Figure 51. Using the SYMDMP Option to Debug the Program TESTRUN (Part 1 of 11)

```
IEC130I SYSLIB   DD STATEMENT MISSING
IEF373I STEP /COB      / START 72144.0024
IEF374I STEP /COB      / STOP  72144.0029 CPU    0MIN 04.09SEC MAIN  78K LCS    0K
STEP COB       ENDED. COMP CODE 0004 CORE REQUSTED= 0080K. CORE USED= 0078K.                  MU=       2,02
XXLKED  EXEC PGM=IEWL,PARM=(XREF,LIST,LET),COND=(5,LT,COB),                 00000100
XX             REGION=96K                                                   00000110
XXSYSLIN DD DSN=&LOADSET,DISP=(OLD,DELETE)                                  00000120
XX     DD DDNAME=SYSIN                                                      00000130
XXSYSLMOD DD DSN=&GODATA(RUN),DISP=(NEW,PASS),                              00000140
XX        UNIT=2314,SPACE=(1024,(50,20,1))                                  00000150
//LKED.SYSLIB  DD   DSN=NEWSYMJB,UNIT=2314,VOL=SER=DC157,DISP=SHR
X/SYSLIB DD DSN=SYS1.DYNAMLIB,DISP=SHR                                      00000160
//             DD   DSNAME=SYS1.DYNAMLIB,DISP=SHR
X/   DD DSN=SYS1.TELCMLIB,DISP=SHR                                          00000170
XXSYSUT1 DD UNIT=(2314,SEP=(SYSLIN,SYSLMOD)),SPACE=(1024,(50,20))           00000180
//LKED.SYSPRINT  DD   SYSOUT=G,OUTLIM=1000                                       SMF
X/SYSPRINT DD SYSOUT=U,OUTLIM=1000                                          00000SMF
XXSYSUDUMP DD SYSOUT=U,OUTLIM=1000                                          00000SMF



IEF373I STEP /LKED     / START 72144.0029
IEF374I STEP /LKED     / STOP  72144.0030 CPU    0MIN 00.67SEC MAIN  96K LCS    0K
STEP LKED      ENDED. COMP CODE 0000 CORE REQUSTED= 0096K. CORE USED= 0096K.                  MU=        .60
XXGO   EXEC PGM=*.LKED.SYSLMOD,COND=((5,LT,COB),(5,LT,LKED))                 00000210
//GO.SYSUDUMP  DD   SYSOUT=G,OUTLIM=1000                                         SMF
X/SYSUDUMP DD SYSOUT=U,OUTLIM=1000                                          00000SMF
XXSYSDBOUT   DD   SYSOUT=U,OUTLIM=1000                                      00000SMF
(D)-->//GO.DD1 DD DSN=&&UT5,UNIT=SYSDA,DISP=(OLD,DELETE)
X/DD1   DD   DSN=&SYMDBG,DISP=(OLD,DELETE)                                  00000240
//GO.SAMPLE DD UNIT=2400,LABEL=(,NL),DISP=(NEW,DELETE),VOL=SER=TESTER
//GO.SYSOUT   DD   SYSOUT=G,OUTLIM=1000                                          SMF
//GO.SYSDBOUT  DD   SYSOUT=G,OUTLIM=1000                                         SMF
//GO.STEPLIB   DD   DSN=NEWSYMJB,UNIT=2314,VOL=SER=DC157,DISP=SHR
//             DD   DSNAME=SYS1.DYNAMLIB,DISP=SHR
//GO.SYSDBG     DD   *
//
```

Figure 51. Using the SYMDMP Option to Debug the Program TESTRUN (Part 2 of 11)

```
IEC130I SYSDTERM DD STATEMENT MISSING
A 0001 NYC 0
B 0002 NYC 1
C 0003 NYC 2
D 0004 NYC 3
E 0005 NYC 4
F 0006 NYC 0
G 0007 NYC 1
H 0008 NYC 2
I 0009 NYC 3
IEF460I WTP MESSAGE LIMIT EXCEEDED
COMPLETION CODE - SYSTEM=0C7  USER=0000
IEF242I ALLOC. FOR DEBUG    GO                    AT ABEND
IEF237I 136    ALLOCATED TO JOBLIB
IEF237I 355    ALLOCATED TO
IEF237I 240    ALLOCATED TO PGM=*.DD
IEF237I 242    ALLOCATED TO SYSUDUMP
IEF237I 242    ALLOCATED TO SYSDBOUT
IEF237I 241    ALLOCATED TO DD1
IEF237I 282    ALLOCATED TO SAMPLE
IEF237I 242    ALLOCATED TO SYSOUT
IEF237I 242    ALLOCATED TO SYSDBOUT
IEF237I 137    ALLOCATED TO STEPLIB
IEF237I 355    ALLOCATED TO
IEF237I 241    ALLOCATED TO SYSDBG
IEF285I   DUMMY0S                                   PASSED
IEF285I   VOL SER NOS= DC156 .
IEF285I   PRODVER4                                  PASSED
IEF285I   VOL SER NOS= DC160 .
IEF285I   SYS72144.T002347.RV000.DEBUG.GODATA       PASSED
IEF285I   VOL SER NOS= 231400.
IEF285I   SYS72144.T002347.SV000.DEBUG.R0000011     SYSOUT
IEF285I   VOL SER NOS= 231402.
IEF285I   SYS72144.T002347.SV000.DEBUG.R0000012     SYSOUT
IEF285I   VOL SER NOS= 231402.
IEF285I   SYS72144.T002347.RV000.DEBUG.UT5          DELETED
IEF285I   VOL SER NOS= 231401.
IEF285I   SYS72144.T002347.RV000.DEBUG.R0000013     DELETED
IEF285I   VOL SER NOS= TESTER.
IEF285I   SYS72144.T002347.SV000.DEBUG.R0000014     DELETED
IEF285I   VOL SER NOS= 231402.
IEF285I   SYS72144.T002347.SV000.DEBUG.R0000015     DELETED
IEF285I   VOL SER NOS= 231402.
IEF285I   NEWSYMJB                                  KEPT
IEF285I   VOL SER NOS= DC157 .
IEF285I   SYS1.DYNAMLIB                             KEPT
IEF285I   VOL SER NOS= DC160 .
IEF285I   SYS72144.T002347.RV000.DEBUG.S0000016     SYSIN
IEF285I   VOL SER NOS= 231401.
IEF285I   SYS72144.T002347.RV000.DEBUG.S0000016     DELETED
IEF285I   VOL SER NOS= 231401.
IEF373I STEP /GO      / START 72144.0030
IEF374I STEP /GO      / STOP  72144.0033 CPU    0MIN 03.20SEC MAIN   52K LCS    0K
STEP GO        ENDED. COMP CODE 00C7 CORE REQUSTED= 0052K. CORE USED= 0052K.                     MU=       1.16
IEF285I   DUMMY0S                                   KEPT
IEF285I   VOL SER NOS= DC156 .
IEF285I   PRODVER4                                  KEPT
IEF285I   VOL SER NOS= DC160 .
IEF285I   SYS72144.T002347.RV000.DEBUG.GODATA       DELETED
IEF285I   VOL SER NOS= 231400.
IEF375I  JOB /DEBUG   / START 72144.0024
IEF376I  JOB /DEBUG   / STOP  72144.0033 CPU    0MIN 07.96SEC
JOB DEBUG     ENDED. CODE= 00C7 JOB READ IN AT 00.40    ON 72144 JOB STRTED AT 00.41    ON 72144 JOB ENDED AT 00.56    ON  72144
```

Figure 51. Using the SYMDMP Option to Debug the Program TESTRUN (Part 3 of 11)

```
   1

IKF0011I-W SYSLIB NOT USABLE. COMPILATION CONTINUING.
$LBDMP    7     X
$LBDMP    H     X



   2

100010 IDENTIFICATION DIVISION.
100020 PROGRAM-ID.  TESTRUN.
100030   AUTHOR. PROGRAMMER NAME.
100040   INSTALLATION. NEW YORK PROGRAMMING CENTER.
100050   DATE-WRITTEN. JULY 12, 1968.
100060 DATE-COMPILED. JAN  6,1972
100070   REMARKS. THIS PROGRAM HAS BEEN WRITTEN AS A SAMPLE PROGRAM FOR
100080      COBOL USERS. IT CREATES AN OUTPUT FILE AND READS IT BACK AS
100090      INPUT.

100100 ENVIRONMENT DIVISION.
100110 CONFIGURATION SECTION.
100120   SOURCE-COMPUTER. IBM-360-H50.
100130   OBJECT-COMPUTER. IBM-360-H50.
100140 INPUT-OUTPUT SECTION.
100150 FILE-CONTROL.
100160      SELECT FILE-1 ASSIGN TO UT-2400-S-SAMPLE.
100170      SELECT FILE-2 ASSIGN TO UT-2400-S-SAMPLE.

100180 DATA DIVISION.
100190 FILE SECTION.
100200 FD  FILE-1
100210      LABEL RECORDS ARE OMITTED
100220      BLOCK CONTAINS 100 CHARACTERS
100225      RECORD CONTAINS 20 CHARACTERS
100230      RECORDING MODE IS F
100240      DATA RECORD IS RECORD-1.
100250   01 RECORD-1.
100260      02 FIELD-A PICTURE IS X(20).
100270 FD  FILE-2
100280      LABEL RECORDS ARE OMITTED
100290      BLOCK CONTAINS 5 RECORDS
100300      RECORD CONTAINS 20 CHARACTERS
100310      RECORDING MODE IS F
100320      DATA RECORD IS RECORD-2.
100330   01 RECORD-2.
100340      02 FIELD-A PICTURE IS X(20).

100350 WORKING-STORAGE SECTION.
100360      77 KOUNT PICTURE S99 COMP SYNC.
100370      77 NOMBER  PICTURE S99 COMP SYNC.
100375   01 FILLER.
100380      02 ALPHABET PICTURE X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
100395      02 ALPHA REDEFINES ALPHABET PICTURE X OCCURS 26 TIMES.
100405      02 DEPENDENTS PICTURE X(26)  VALUE "0123401234012340123401234
100410-     "0".
100420      02 DEPEND REDEFINES DEPENDENTS PICTURE X OCCURS 26 TIMES.
100440   01 WORK-RECORD.
100450      02 NAME-FIELD  PICTURE X.
100460      02 FILLER PICTURE X VALUE IS SPACE.
100470      02 RECORD-NO PICTURE 9999.
100480      02 FILLER PICTURE X VALUE IS SPACE.
100490      02 LOCATION PICTURE AAA VALUE IS "NYC".
100500      02 FILLER PICTURE X VALUE IS SPACE.
100510      02 NO-OF-DEPENDENTS PICTURE XX.
100520      02 FILLER PICTURE X(7) VALUE IS SPACES.
100521      01 RECORDA.
```

Figure 51. Using the SYMDMP Option to Debug the Program TESTRUN (Part 4 of 11)

```
  3

00058   100522        02  A PICTURE S9(4) VALUE 1234.
00059   100523        02  B REDEFINES A PICTURE S9(7) COMPUTATIONAL-3.
00060   100530 PROCEDURE DIVISION.
00061   100540 BEGIN. READY TRACE.
00062   100550     NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED
00063   100560     AND INITIALIZES COUNTERS.
00064   100570 STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO KOUNT NOMBER.
00065   100580     NOTE THAT THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE
00066   100590     CONTAINED IN THE FILE, WRITES THEM ON TAPE, AND DISPLAYS
00067   100600     THEM ON THE CONSOLE.
00068   100610 STEP-2. ADD 1 TO KOUNT. ADD 1 TO NOMBER. MOVE ALPHA (KOUNT) TO
00069   100620     NAME-FIELD.
00070   100630     MOVE DEPEND (KOUNT) TO NO-OF-DEPENDENTS.
00071   100640     MOVE NOMBER TO RECORD-NO.
00072   100650 STEP-3. DISPLAY WORK-RECORD UPON CONSOLE. WRITE RECORD-1 FROM
00073   100660     WORK-RECORD.
00074   100670 STEP-4. PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL TO 26.
00075   100680     NOTE THAT THE FOLLOWING CLOSES OUTPUT AND REOPENS IT AS
00076   100690     INPUT.
00077   100700 STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.
00078   100710     NOTE THAT THE FOLLOWING READS BACK THE FILE AND SINGLES OUT
00079   100720     EMPLOYEES WITH NO DEPENDENTS.
00080   100730 STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.
00081   100731        COMPUTE B = B + 1.
00082   100740 STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO
00083   100750     NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO
00084   100760     STEP-6.
00085   100770 STEP-8. CLOSE FILE-2.
00086   100780     STOP RUN.


 18

                                   (C)------->CROSS-REFERENCE DICTIONARY

 DATA NAMES                        DEFN      REFERENCE

 A                                 000058
 ALPHA                             000044    000068
 ALPHABET                          000043
 B                                 000059    000081
 DEPEND                            000047    000070
 DEPENDENTS                        000045
 FIELD-A                           000029
 FIELD-A                           000037
 FILE-1                            000017    000064   000072   000077
 FILE-2                            000018    000077   000080   000085
 KOUNT                             000040    000064   000068   000070   000074
 LOCATION                          000053
 NAME-FIELD                        000049    000068
 NO-OF-DEPENDENTS                  000055    000070   000082
 NOMBER                            000041    000064   000068   000071
 RECORD-NO                         000051    000071
 RECORD-1                          000028    000072
 RECORD-2                          000036    000080
 RECORDA                           000057
 WORK-RECORD                       000048    000072   000080   000083
```

Figure 51. Using the SYMDMP Option to Debug the Program TESTRUN (Part 5 of 11)

19

```
PROCEDURE NAMES                        DEFN      REFERENCE


BEGIN                                  000061
STEP-1                                 000064
STEP-2                                 000068    000074
STEP-3                                 000072    000074
STEP-4                                 000074
STEP-5                                 000077
STEP-6                                 000080    000083
STEP-7                                 000082
STEP-8                                 000085    000080
```

20

```
 CARD   ERROR MESSAGE

 58     IKF2190I-W     PICTURE CLAUSE IS SIGNED, VALUE CLAUSE UNSIGNED. ASSUMED POSITIVE.

PHASE      FILE1      FILE2      FILE3      FILE4      FILE5
1        00000000 00000000 0000034C 00000000 00000000
2        00000000 00000000 00000000 00000000 00000000
3        00000000 000002D6 00000000 00000000 00000000
4        00000000 00000000 00000000 0000040A 00000000
5        00000000 00000000 000002C1 00000000 00000000
6        00000000 00000000 00000000 000003EF 00000000
7        00000000 00000000 00000000 00000000 00000400
8        00000000 00000000 00000351 00000074 00000000
9        000005DD 00000000 0000000C 00000000 00000000
A        00000000 00000000 00000000 00000000 00000000
B        00000000 0000083C 00000035 00000000 00000000
C        00000A34 00000000 00000246 00000001 00000000
D        00000000 00000000 00000000 00000000 00000000
E        00000000 00000000 00000000 00000000 00000000
F        00000000 00000783 00000000 000000E2 00000000
G        000001EB 00000000 00000114 00000000 00000000
H        00000000 00000000 000000CC 00000000 00000600
```

Figure 51. Using the SYMDMP Option to Debug the Program TESTRUN (Part 6 of 11)

```
(1)→TESTRUN,DD1
(2)→70,ON 1,4,(HEX),KOUNT,NAME-FIELD,NO-OF-DEPENDENTS,RECORD-NO
(3)→81,(HEX),WORK-RECORD,B
          TESTRUN   UNIDENTIFIED ELEMENTS ON CONTROL CARDS
*ERROR* CARD/VERB
IKF160I   70        IDENTIFIER NOT FOUND
                                          001 ERRORS FOUND IN CONTROL CARDS
```

```
                              (F)→TYPE CODES USED IN SYMDMP OUTPUT
                         CODE      MEANING
                         A      = ALPHABETIC
                         AN     = ALPHANUMERIC
                         ANE    = ALPHANUMERIC EDITED
                         D      = DISPLAY (STERLING NONREPORT)
                         DE     = DISPLAY EDITED (STERLING REPORT)
                         F      = FLOATING POINT (COMP-1/COMP-2)
                         FD     = FLOATING POINT DISPLAY (EXTERNAL FLOATING POINT)
                         NB     = NUMERIC BINARY UNSIGNED (COMP)
                         NB-S   = NUMERIC BINARY SIGNED
                         ND     = NUMERIC DISPLAY UNSIGNED (EXTERNAL DECIMAL)
                         ND-OL  = NUMERIC DISPLAY OVERPUNCH SIGN LEADING
                         ND-OT  = NUMERIC DISPLAY OVERPUNCH SIGN TRAILING
                         ND-SL  = NUMERIC DISPLAY SEPARATE SIGN LEADING
                         ND-ST  = NUMERIC DISPLAY SEPARATE SIGN TRAILING
                         NE     = NUMERIC EDITED
                         NP     = NUMERIC PACKED DECIMAL UNSIGNED (COMP-3)
                         NP-S   = NUMERIC PACKED DECIMAL SIGNED
                         *      = SUBSCRIPTED
TESTRUN  AT CARD 00007C
 LOC      CARD   LV NAME                          TYPE    VALUE
(G)→0D0778  000040  77 KOUNT                      NB-S    +01
                                           (HEX)          0001
0D07B8  000049  02 NAME-FIELD                     AN      A
0D07BA  000051  02 RECORD-NO                      ND      ****
                                           (HEX)          4750C0FE

TESTRUN  AT CARD 00007C
 LOC      CARD   LV NAME                          TYPE    VALUE
0D0778  000040  77 KOUNT                          NB-S    +05
                                           (HEX)          0005
0D07B8  000049  02 NAME-FIELD                     AN      E
0D07BA  000051  02 RECORD-NO                      ND      0004

TESTRUN  AT CARD 000070
 LOC      CARD   LV NAME                          TYPE    VALUE
0D0778  000040  77 KOUNT                          NB-S    +09
                                           (HEX)          0009
0D07B8  000049  02 NAME-FIELD                     AN      I
```

Figure 51. Using the SYMDMP Option to Debug the Program TESTRUN (Part 7 of 11)

```
0D07BA   000051  02 RECORD-NO                              ND       0008


TESTRUN   AT CARD 000070
 LOC       CARD   LV NAME                                  TYPE     VALUE

0D0778   000040  77 KOUNT                                  NB-S     +13
                                                   (HEX)            000D

0D07B8   000049  02 NAME-FIELD                             AN       M

0D07BA   000051  02 RECORD-NO                              ND       0012


TESTRUN   AT CARD 000070
 LOC       CARD   LV NAME                                  TYPE     VALUE

0D0778   000040  77 KOUNT                                  NB-S     +17
                                                   (HEX)            0011

0D07B8   000049  02 NAME-FIELD                             AN       Q

0D07BA   000051  02 RECORD-NO                              ND       0016


TESTRUN   AT CARD 000070
 LOC       CARD   LV NAME                                  TYPE     VALUE

0D0778   000040  77 KOUNT                                  NB-S     +21
                                                   (HEX)            0015

0D07B8   000049  02 NAME-FIELD                             AN       U

0D07BA   000051  02 RECORD-NO                              ND       0020


TESTRUN   AT CARD 000070
 LOC       CARD   LV NAME                                  TYPE     VALUE

0D0778   000040  77 KOUNT                                  NB-S     +25
                                                   (HEX)            0019

0D07B8   000049  02 NAME-FIELD                             AN       Y

0D07BA   000051  02 RECORD-NO                              ND       0024


(H)->TESTRUN   AT CARD 000081
 LOC       CARD   LV NAME                                  TYPE     VALUE

         000048  01 WORK-RECORD
0D07B8                                             (HEX)          C140F0F0 F0F140D5 E8C340F0   40404040 40404040
0D07B8   000049  02 NAME-FIELD                             AN       A
0D07B9   000050  02 FILLER                                 AN
0D07BA   000051  02 RECORD-NO                              ND       0001
0D07BE   000052  02 FILLER                                 AN
0D07BF   000053  02 LOCATION                               A        NYC
0D07C2   000054  02 FILLER                                 AN
0D07C3   000055  02 NO-OF-DEPENDENTS                       AN       0
0D07C5   000056  02 FILLER                                 AN

0D07D0   000059  02 B                                      NP-S     *1*2*3*
                                                   (HEX)            F1F2F3C4
```

Figure 51. Using the SYMDMP Option to Debug the Program TESTRUN (Part 8 of 11)

```
(I)--PROGRAM        TESTRUN

     COMPLETION CODE = 0C7      LAST PSW BEFORE ABEND = FFD5000D000D0A06

(J)--LAST CARD NUMBER/VERB NUMBER EXECUTED -- CARD NUMBER 000081/VERB NUMBER 01.

                                                                FLOW TRACE
(K)--TESTRUN  000068 000072 000068 000072 000068 000072 000068 000072 000077 000080
```

```
                                   DATA DIVISION DUMP OF TESTRUN

(L)--TASK GLOBAL TABLE       LOC       VALUE

     SAVE AREA               0D0938    009A9200 000DC768 000DA2E8 700D0EB6   0000E23A 700D0ECE 000DA400 000D08B8
                             0D0958    00026CE4 00C000D0 700D0E0F 000D0778   000DA414 000DA400 000D0FFE 000D06F0
                             0D0978    700D0ECE 000D0B70
     SWITCH                  0D0980    7D00C04B
     TALLY                   0D0984    00000000
     SORT-SAVE               0D0988    C0C0C000
     ENTRY-SAVE              0D098C    000D0EDC
     SORT-CORE-SIZE          0D0990    00C00000
     RETURN-CODE             0D0994    05EF
     SORT-RETURN             0D0996    5891
     WORKING CELLS           0D0998    000D2456 000D2F1A FFFFFF2E 000DC7F8   0002635C 00000000 00108000 E940F0F0
                             0D09B8    F2F640D5 E8C340F0 02004020 4040404C   000D06F0 000D06F0 000D0B70 60089202
                             0D09D8    6C081B99 50910000 4140C0EA 47F0C1F8   D2001000 7000D200 D0584000 D700D058
                             0D09F8    7000DC00 18E00700 FA306058 C04F070C   5800D1D8 07FB4000 00000FA0 000DC658
                             0D0A18    20001000 30C00000 000A7788 00000000   00000001 000A7BC0 47F0F00F 000D0E96
                             0D0A38    000D080C 010090EC 400D0F84 000D108A   00000030 000D0E9E 000D080C 00026CF4
                             0D0A58    C00C00D0 700D0E0E 000D0778 000DA414   000D06F0 000D0FFE 000D06F0 000D03DC
                             0D0A78    000D0B70 000D0E9E 000D108A 00000030   8F0DC020 000D080C 00011A78 000D0F96
                             0D0A98    4270B001 9240B002 9200B003 43405000   444090F6 5060B004 06704470 917A50C0
                             0D0AB8    B00841B0 B00C41C7 C0014177 80019102
     SORT-FILE-SIZE          0D0AC8    00C00000
     SORT-MODE-SIZE          0D0ACC    00000000
     PGT-VN TBL              0D0AD0    86D29142
     TGT-VN TBL              0D0AD4    50E0D008
     VCON ADDR               0D0AD8    505CD00C
     VN TBL LENGTH           0D0ADC    4177
     LABEL RETURN            0D0ADE    0C
     CURRENT PRIORITY        0D0ADF    00
     DEBUG SAVE14            0D0AE0    700D0ECE
     COBOL INDICATOR         0D0AE4    ANSC
     A(INIT1)                0D0AE8    000D06F0
     DEBUG TABLE PTR         0D0AEC    00000478
     SUBCOM ADDR             0D0AF0    0C0D0630
     SORT DDNAME             0D0AF4
     UNUSED                  0D0AFC    5850D000 00C00000 1F744100 000C1815   58505004
     DEBUG SAVE11            0D0B10    000D0EDC
     UNUSED                  0D0B14    C0011815
     PRBADR CELL             0D0B18    000D0BD8
     GENCB TABLE             0D0B1C    DC081FD2
     UNUSED                  0D0B20    06
     TRANSIENT AREA LENGTH   0D0B21    7E9500
     UNUSED                  0D0B24    50049600 B0024140
     OVERFLOW CELLS          (NONE)
     BL CELLS                0D0B2C    000DA414 000DA400 000D0778
     DECBADR CELLS           (NONE)
     TEMP STORAGE            0D0B38    00000000 0000026C
     BLL CELLS               0D0B40    000C00C0 00C00000
```

Figure 51. Using the SYMDMP Option to Debug the Program TESTRUN (Part 9 of 11)

```
                                    DATA DIVISION DUMP OF TESTRUN


VLC CELLS              (NONE)
SEL CELLS              (NONE)
INDEX CELLS            (NONE)
OTHER (SEE MEMORY MAP) 0D0B48   000D0799 000D07B3 000D0D90 000D0D90   800D08B8 18141E11 4101100C 00000001
                       0D0B68   0A0C08CA 20C60A0A



                                    DATA DIVISION DUMP OF TESTRUN

      LOC      CARD    LV NAME                          TYPE    VALUE
(M)--------->000017   FD FILE-1                        QSAM    FILE: CLOSED   ORGANIZATION: PHYSICAL SEQUENTIAL

      0DC80C                                  DCB   C0000000 00000000 00000000 00000006   00810000 000DA391 00C04000 00000001
      0DC82C                                        46000001 900D07DC E2C1D4D7 D3C54040   02000048 00000001 060D2456 00000064
      0D084C                                        C0000000 00000000 00000000 00000001   00000014 00000001 00000000 00000000

               000028  01 RECORD-1
(N)-->0DA414   000029  02 FIELD-A                      AN      B 0002 NYC 1
(M)--------->000018   FD FILE-2                        QSAM    FILE: OPEN     ORGANIZATION: PHYSICAL SEQUENTIAL

      0D08E8                                  DCB   00000000 00000000 00000000 00000002   0081C300 020DA390 00004000 00000001
      0D08D8                                        460D0EC8 900D0888 007C4800 00026CE4   120FEE00 000FEC40 060D2456 00090064
      0DC8F8                                        28012828 000DBC30 000DA464 000DA400   00000014 00000001 00000000 000FF898
(N)
   \-> 0DA400  000036  01 RECORD-2
               000037  02 FIELD-A                      AN      A 0001 NYC 0
(P)-->0D0778   000040  77 KOUNT                        NB-S    +26
      0DC77A   000041  77 NUMBER                       NB-S    +26

               000042  01 FILLER
      0DC780   000043  02 ALPHABET                     AN      ABCDEFGHIJKLMNOPQRSTUVWXYZ
               000044  02 ALPHA                       *AN

                          (SUB1)
(Q)-->0D0780                 1                                 A
      0D0781                 2                                 B
      0DC782                 3                                 C
      0D0783                 4                                 D
      0DC784                 5                                 E
      0D0785                 6                                 F
      0DC786                 7                                 G
      0D0787                 8                                 H
      0DC788                 9                                 I
      0D0789                10                                 J
      0DC78A                11                                 K
      0D078B                12                                 L
      0DC78C                13                                 M
      0D078D                14                                 N
      0DC78E                15                                 O
      0D078F                16                                 P
      0DC790                17                                 Q
      0D0791                18                                 R
      0DC792                19                                 S
      0D0793                20                                 T
      0DC794                21                                 U
      0D0795                22                                 V
      0DC796                23                                 W
      0D0797                24                                 X
```

Figure 51. Using the SYMDMP Option to Debug the Program TESTRUN (Part 10 of 11)

DATA DIVISION DUMP OF TESTRUN

```
LOC      CARD    LV NAME                           TYPE    VALUE

0D0798                          25                          Y
0D0799                          26                          Z
0D079A   000045  02 DEPENDENTS                     AN      01234012340123401234012340
         000047  02 DEPEND                         *AN

               Q ──>(SUB1)
0D079A                           1                          0
0D079B                           2                          1
0D079C                           3                          2
0D079D                           4                          3
0D079E                           5                          4
0D079F                           6                          0
0D07A0                           7                          1
0D07A1                           8                          2
0D07A2                           9                          3
0D07A3                          10                          4
0D07A4                          11                          0
0D07A5                          12                          1
0D07A6                          13                          2
0D07A7                          14                          3
0D07A8                          15                          4
0D07A9                          16                          0
0D07AA                          17                          1
0D07AB                          18                          2
0D07AC                          19                          3
0D07AD                          20                          4
0D07AE                          21
0D07AF                          22                          .
0D07B0                          23                          2
0D07B1                          24                          3
0D07B2                          25                          4
0D07B3                          26                          0
                                          P
         000048  01 WORK-RECORD <──
0D07B8   000049  02 NAME-FIELD                     AN      A
0D07B9   000050  02 FILLER                         AN
0D07BA   000051  02 RECORD-NO                      NP      0001
0D07BE   000052  02 FILLER                         AN
0D07BF   000053  02 LOCATION                       A       NYC
0D07C2   000054  02 FILLER                         AN
0D07C3   000055  02 NO-OF-DEPENDENTS               AN      0
0D07C5   000056  02 FILLER                         AN

         000057  01 RECORDA
0D07D0   000058  02 A                              ND-OT   +1234
0D07D0   000059  02 B                              NP-S    *1*2*3*  <── R
                                           (HEX)           F1F2F3C4
```

DATA DIVISION DUMP OF TESTRUN

```
LOC      CARD    LV NAME                           TYPE    VALUE
```

END OF COBOL DIAGNOSTIC AIDS

Figure 51. Using the SYMDMP Option to Debug the Program TESTRUN (Part 11 of 11)

The compiler, linkage editor, COBOL load module, and other system components can produce output in the form of printed listings, punched card decks, diagnostic or informative messages, and data sets directed to tape or mass storage devices. This chapter describes the output listings that can be used to document and debug programs and the format of the output modules. The same COBOL program is used for each example. "Appendix A: Sample Program Output" shows the output formats in the context of a complete listing generated by a sample program.

COMPILER OUTPUT

The output of the compilation job step may include:

- A printed listing of the job control statements
- Device allocation and deallocation messages from the job scheduler
- A printed listing of the statements contained in the source module
- A glossary of compiler-generated information about data
- A printed listing of the object code
- Compiler diagnostic messages
- System messages
- Disposition messages from the job scheduler
- An object module
- A cross-reference listing
- A condensed listing containing source card numbers and the location of the generated instruction for each verb
- Compiler statistics

Diagnostic messages associated with the compilation of the source program are automatically generated as output. The other forms of output may be requested in the PARM parameter in the EXEC statement. The level of diagnostic messages printed depends upon the FLAGW or FLAGE options.

All output to be listed is written on the device specified by the SYSPRINT DD statement. Line spacing of the source listing and the number of lines per page can be controlled by the SPACEn and LINECNT options.

Figure 52 contains a portion of the compiler output listing shown in "Appendix A: Sample Program Output." Each type of output is numbered, and each format within each type is lettered. The text following Figure 50 is an explanation of the illustration.

1

```
//TEST    JOB   NY83922041,'A. HALL ',MSGLEVEL=(1,1),CLASS=C,MSGCLASS=U
//JOBLIB   DD    DSN=PRODVER4,DISP=SHR,VOLUME=SER=USAS,UNIT=2314
//STEP1   EXEC   PGM=IKFCBL00,PARM='DMAP,PMAP,XREF,QUOTE,OPT',REGION=76K
//SYSUT1 DD      DSNAME=&&UT1,UNIT=SYSDA,SPACE=(TRK,(100,10))
//SYSUT2 DD      DSNAME=&&UT2,UNIT=SYSDA,SPACE=(TRK,(100,10))
//SYSUT3 DD      DSNAME=&&UT3,UNIT=SYSDA,SPACE=(TRK,(100,10))
//SYSUT4 DD      DSNAME=&&UT4,UNIT=SYSDA,SPACE=(TRK,(100,10))
//SYSLIN DD      DSNAME=&&PNCH,UNIT=SYSDA,SPACE=(TRK,(100,10)),         X
//               DISP=(NEW,PASS)
//SYSPRINT DD    SYSOUT=U
//SYSIN  DD      *
```

2

```
IEF236I ALLOC. FOR TESTRUN  COB
IEF237I 230    ALLOCATED TO SYSPRINT
IEF237I 230    ALLOCATED TO SYSUDUMP
IEF237I 235    ALLOCATED TO SYSUT1
IEF237I 230    ALLOCATED TO SYSUT2
IEF237I 242    ALLOCATED TO SYSUT3
IEF237I 230    ALLOCATED TO SYSUT4
IEF237I 235    ALLOCATED TO SYSLIN
IEF237I 243    ALLOCATED TO STEPLIB
IEF237I 235    ALLOCATED TO SYSIN
```

Figure 52. Examples of Compiler Output (Part 1 of 3)

3

```
00001   100010 IDENTIFICATION DIVISION.
00002   100020 PROGRAM-ID.  TESTRUN.
00003   100030   AUTHOR. PROGRAMMER NAME.
00004   100040   INSTALLATION. NEW YORK PROGRAMMING CENTER.
00005   100050   DATE-WRITTEN. JULY 12, 1968.
00006   100060 DATE-COMPILED. FEB 19,1972
          .
          .
          .
00018   100180 DATA DIVISION.
00019   100190 FILE SECTION.
00020   100200 FD  FILE-1
00021   100210     LABEL RECORDS ARE OMITTED
00022   100220     BLOCK CONTAINS 100 CHARACTERS
00023   100225     RECORD CONTAINS 20 CHARACTERS
00024   100230     RECORDING MODE IS F
00025   100240     DATA RECORD IS RECORD-1.
00026   100250   01 RECORD-1.
00027   100260     02 FIELD-A PICTURE IS X(20).
00028   100270 FD  FILE-2
          .
          .
          .
00057   100530 PROCEDURE DIVISION.
00058   100540 BEGIN. READY TRACE.
00059   100550     NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED
00060   100560     AND INITIALIZES COUNTERS.
00061   100570 STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO KOUNT NOMBER.
          .
          .
          .
00078   100730 STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.
00079   100740 STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO
00080   100750     NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO
00081   100760     STEP-6.
00082   100770 STEP-8. CLOSE FILE-2.
00083   100780     STOP RUN.
```

4

```
   A             B    C                        D        E      F             G            H            I
INTRNL NAME     LVL SOURCE NAME              BASE     DISPL  INTRNL NAME   DEFINITION   USAGE        R  O  Q  M
DNM=1-148       FD  FILE-1                   DCB=01          DNM=1-148                  QSAM                  F
DNM=1-167       01  RECORD-1                  BL=1    000    DNM=1-167     DS 0CL20     GROUP
DNM=1-188       02  FIELD-A                   BL=1    000    DNM=1-188     DS 20C       DISP
DNM=1-205       FD  FILE-2                   DCB=02          DNM=1-205                  QSAM                  F
DNM=1-224       01  RECORD-2                  BL=2    000    DNM=1-224     DS 0CL20     GROUP
DNM=1-245       02  FIELD-A                   BL=2    000    DNM=1-245     DS 20C       DISP
DNM=1-265       77  KOUNT                     BL=3    000    DNM=1-265     DS 1H        COMP
   .             .    .                        .       .       .             .            .
   .             .    .                        .       .       .             .            .
   .             .    .                        .       .       .             .            .
DNM=2-071       02  B                         BL=3    058    DNM=2-071     DS 4P        COMP-3       R
```

5

```
A  MEMORY MAP

    TGT                    00248

SAVE AREA                  00248
SWITCH                     00290
TALLY                      00294
SORT SAVE                  00298
ENTRY-SAVE                 0029C
    .                        .
    .                        .
    .                        .
XSASW CELLS                00478
XSA CELLS                  00478


    PGT                    00498

OVERFLOW CELLS             00498
VIRTUAL CELLS              00498
PROCEDURE NAME CELLS       004B8
GENERATED NAME CELLS       004CC
DCB ADDRESS CELLS          004E4
VNI CELLS                  004EC
LITERALS                   004F0
DISPLAY LITERALS           00504
```

```
B  LITERAL POOL (HEX)

   004F0 (LIT+0)       00000001  1C00001A  004805EF  48000000

        DISPLAY LITERALS (BCD)

   00504 (LTL+20)    'WORK-RECORD'
```

6

```
REGISTER ASSIGNMENT

   REG 6    BL =3
   REG 7    BL =1
   REG 8    BL =2
```

7 WORKING-STORAGE STARTS AT LOCATION 00088 FOR A LENGTH OF 00060

8

```
A      B       C      D                    E      F     G                   H
58     VERB 1
              000510                       START  EQU   *
              000510  07 00                       BCR   0,0
              000512  58 F0 C 00C                 L     15,00C(0,12)        V(ILBODBG4)
              000516  05 EF                       BALR  14,15
              000518  58 F0 C 010                 L     15,010(0,12)        V(ILBOFLW1)
              00051C  05 1F                       BALR  1,15
              00051E  0000003A                    DC    X'0000003A'
              000522  58 F0 C 014                 L     15,014(0,12)        V(ILBODSP0)
              000526  05 1F                       BALR  1,15
              000528  000140                      DC    X'000140'
              00052B  05F5F840404040              DC    X'05F5F840404040'
58     VERB 2
              000532  96 40 D 048                 OI    048(13),X'40'       SWT+0
61     VERB 3
              000536  58 F0 C 00C                 L     15,00C(0,12)        V(ILBODBG4)
              00053A  05 EF                       BALR  14,15
              00053C  58 F0 C 010                 L     15,010(0,12)        V(ILBOFLW1)
              000540  05 1F                       BALR  1,15
              000542  0000003D                    DC    X'0000003D'
              000546  58 F0 C 014                 L     15,014(0,12)        V(ILBODSP0)
              00054A  05 1F                       BALR  1,15
              00054C  000140                      DC    X'000140'
              00054F  05F6F140404040              DC    X'05F6F140404040'
61     VERB 4
              000556  58 F0 C 00C                 L     15,00C(0,12)        V(ILBODBG4)
              00055A  05 EF                       BALR  14,15
```

Figure 52. Examples of Compiler Output (Part 2 of 3)

```
  ( *STATISTICS*     SOURCE RECORDS =     84      DATA DIVISION STATEMENTS =     25      PROCEDURE DIVISION STATEMENTS =     22
  ) *OPTIONS IN EFFECT*     SIZE =    81920  BUF =      2768  LINECNT = 57  SPACE1, FLAGW,    SEQ,    SOURCE
(9) *OPTIONS IN EFFECT*      DMAP,    PMAP, NOCLIST, NOSUPMAP, NOXREF,    SXREF,    LOAD, NODECK, QUOTE, NOTRUNC,    FLOW=    35
  ( *OPTIONS IN EFFECT*     NOTERM, NONUM, NOBATCH, NONAME, COMPILE=01,    STATE, NORESIDENT, NODYNAM, NOLIB, NOSYNTAX

                                                   CROSS-REFERENCE DICTIONARY

      DATA NAMES                       DEFN      REFERENCE
      .
      .
      .
      FIELD-A                          000027
      FIELD-A                          000035
      FILE-1                           000016    000061   000070   000075
      FILE-2                           000017    000075   000078   000082
      KOUNT                            000037    000061   000065   000068   000072
      LOCATION                         000050
      NAME-FIELD                       000046    000065
      NO-OF-DEPENDENTS                 000052    000068   000079
(10)  .
      .
      .

      PROCEDURE NAMES                  DEFN      REFERENCE

      BEGIN                            000058
      STEP-1                           000061
      STEP-2                           000065    000072
      STEP-3                           000070    000072
        :                                :
      STEP-8                           000082    000078

                          CARD   ERROR MESSAGE
(11)            IKF1100I-W      2 SEQUENCE ERRORS IN SOURCE PROGRAM.
                IKF2190I-W      PICTURE CLAUSE IS SIGNED, VALUE CLAUSE UNSIGNED. ASSUMED POSITIVE.

      IEF285I   SYS71023.T011209.RV000.TESTRUN.GODATA           PASSED
      IEF285I   VOL SER NOS= DDB116.
      IEF285I   SYS71023.T011209.SV000.TESTRUN.R0000011         DELETED
      IEF285I   VOL SER NOS= 231400.
(12)  IEF285I   SYS71023.T011209.SV000.TESTRUN.R0000012         SYSOUT
      IEF285I   VOL SER NOS= 231400.
      IEF285I   SYS71023.T011209.RV000.TESTRUN.R0000013         DELETED
      IEF285I   VOL SER NOS= L00300.
```

Figure 52. Examples of Compiler Output (Part 3 of 3)

1. Listing of job control statements associated with this job step. These statements are listed because MSGLEVEL=(1,1) is specified in the JOB statement. JCL statements with XX instead of // represent statements in a cataloged procedure.

2. Allocation messages from the job scheduler. These messages provide information about the device allocation for the data sets in the job step. They appear after the job control statements in the compile, linkage edit, and execution job steps. For example:

   IEF237I 235 ALLOCATED TO SYSUT1

   indicates that the data set for SYSUT1 has been assigned to the device 235.

3. Source module listing. The statements in the source module are listed exactly as submitted except that a compiler-generated card number is listed in the first column of each line. This number is referred to in diagnostic messages, on the XREF or SXREF listing, and in the object code listing. If NUM is specified, the programmer-encoded source numbers in columns 1 through 6 are used in each of these cases. (See the description of the NUM option under "Options for the Compiler.") The source module is not listed when the NOSOURCE option is specified.

The following notations may appear on the listing:

C   Denotes that the statement was inserted with a COPY statement. Statements copied will not be listed if SUPPESS is indicated.

** Denotes that the card is out of sequence.

I   Denotes that the card was inserted with an INSERT card.

If DATE-COMPILED is specified in the Identification Division, any sentences in that paragraph are replaced in the listing by the date of compilation in the following format:

DATE-COMPILED. month day, year

4. Glossary: The glossary is listed when the DMAP option is specified. The glossary contains information about names in the COBOL source program.

A and F. The internal name generated by the compiler. This name is used in the compiler object code ~~listing to represent the name used~~ in the source program. It is repeated for readability.

B. A normalized level number. This level number is determined by the compiler as follows: (1) the first level number of any hierarchy is always 01, and increments for other levels are always by one; (2) only level numbers 03 through 49 are affected -- level numbers 66, 77, as well as 88 and FD, SD, RD, and CD indicators are not changed.

C. The data name that is used in the source module.

Note that the following Report Writer internally generated data-names can appear under the SOURCE NAME column:

CTL.LVL	Used to coordinate control break activities.
GRP.IND	Used by coding generated for GROUP INDICATE clause.
TER.COD	Used by coding generated for TERMINATE statement.
FRS.GEN	Used by coding generated for GENERATE statement.
-nnnn	Generated report record associated with the file on which the report is to be printed.
RPT.RCD	Build area for print record
CTL.CHR	First or second position of RPT.RCD. Used for carriage control character.
RPT.LIN	Beginning of actual information that will be displayed. Second or third position of RPT.RCD.
CODE-CELL	Used to hold code specified in CODE clause.
E.nnnn	Name generated from COLUMN clause in a level-02 statement.
S.nnnn	Used for elementary level with SUM clause, but not with data-name.
N.nnnn	Used to save the total number of lines used by a report group when relative line numbering is specified.

D and E. For data names, these columns contain information about the address in the form of a base and displacement. For file names, the column contains information about the associated DCB and DECB, if any.

G. ~~This column defines storage for~~ each data item. It is represented in assembler-like terminology. Table 21 refers to information in this column.

H. Usage of the data name. For FD entries, the file processing technique is identified (e.g. QSAM, BDAM, etc.). For group items, GROUP is identified. For elementary items, the information in its USAGE clause is identified, or the USAGE that was specified on its group.

I. A letter under column:

R-Indicates that the data-name redefines another data-name.
O-Indicates that an OCCURS clause has been specified for that data-name.
Q-Indicates that the data-name is the object or contains the object of the DEPENDING ON option of the OCCURS clause.
M-Indicates that the format of the records of the file is:

F = fixed
V = variable
U = undefined
S = spanned

I-Indicates an input CD in a teleprocessing application
O-Indicates an output CD in a teleprocessing application

5. Global Tables and Literal Pool: The global table is listed when the PMAP, CLIST, or DMAP option is specified unless SUPMAP is also specified and an E-level diagnostic message is generated. A global table contains easily addressable information needed by the object program for execution. For example, in the Procedure Division source coding (3), the address of the first instruction under STEP-1, namely:

```
OPEN OUTPUT FILE-1.
```

would be found in the PROCEDURE NAME CELLS entry of the Program Global Table (PGT).

A. Task Global Table (TGT). This table consists of switches, addresses, and work areas whose information changes during execution of the program.

B. Literal Pool. The literal pool lists the collection of the literals in the program, with duplications eliminated. These literals include those specified by the programmer (e.g., MOVE "ABC" TO DATA-NAME) and those generated by the compiler (e.g., to align decimal points in arithmetic computation). The literals are divided into two groups: those that are referred to by instructions (marked "LITERAL POOL") and those that are referred to by the calling sequences to object time subroutines (marked "DISPLAY LITERALS").

C. Program Global Table (PGT). This table contains the remaining addresses and the literals used by the object program.

For further discussion, see "Appendix J: Fields of the Global Table."

6. Register Assignment: This contains the register assigned to each base locator (BL) in the object program.

7. Working-Storage: When PMAP, CLIST, or DMAP is specified, both the location and the length (in hexadecimal) of the Working-Storage Section, if any, are provided.

8. Object Code Listing: The object code listing is produced when the PMAP option is specified unless SUPMAP is also specified and an E-level error is encountered. The actual object code listing contains:

A. The compiler-generated card number or source card number, if NUM is specified. The number refers to the COBOL statement in the source module that contains the verb listed under column B.

B. The relative verb number for each card number.

The statement within which the COBOL verb appears determines the information under columns C, D, F, and G.

If VERB is specified in connection with PMAP or CLIST, procedure-names and verb-names are listed with the associated code.

C. The relative location, in hexadecimal notation, of the object code instruction in the module.

D. The actual object code instruction in hexadecimal notation.

E. The procedure-name number. A number is assigned only to those procedure-names to which reference is made in other Procedure Division statements. This may be a PN (procedure-name) or GN (generated-name) number.

F. The object code instruction in a form that closely resembles assembler language (with displacements in hexadecimal notation).

G. Compiler-generated information about the operands of the generated instruction. This includes names and relative locations of literals. Tables 21 and 22 refer to information in this column.

Note: The programmer can produce a condensed listing by specifying CLIST as an option in place of PMAP. The CLIST option produces only the source card number, the relative verb number, and the location of the first generated instruction, as follows:

```
55  VERB1  0004AC   58  VERB1  0004C0
58  VERB2  0004F2   62  VERB1  00050E
62  VERB2  00051A   62  VERB3  000526
```

9. Statistics: The compiler statistics list the options in effect for this run and the number of Data Division and Procedure Division statements specified. Each level number is counted as one statement in the Data Division. Each verb is counted as one statement in the Procedure Division.

10. Cross-Reference Dictionary: The XREF dictionary, produced when either the XREF or the SXREF option is specified, consists of two parts:

A. The XREF dictionary for data-names followed by the generated number or source card number of the card on which the statement begins, if NUM is in effect. For condition names, the data-name of the conditional variable appears in the XREF dictionary.

B. The XREF dictionary for procedure-names followed by the generated number or source card number of the card on which the statement begins.

For XREF, all the names begin in the order in which they are defined in the source program. For SXREF, the names appear sorted in alphanumeric order. The number of references appearing for a given name is based on the number of times the name is referenced in the compiler-generated code.

11. Diagnostic messages: The diagnostic messages associated with the compilation are always listed. The format of the diagnostic message is:

A. Compiler-generated line number or source card number. This is the number of a line in the source module related to the error.

Note: In this listing of TESTRUN, there were no error messages. However, a typical message is provided with the compiler output to serve as an example of message format.

B. Message identification. The message identification for COBOL compiler diagnostic messages always begins with the symbols IKF.

C. Severity level. There are four severity levels as follows:

W Warning -- This severity level indicates that an error was made in the source program. However, it is not serious enough to hinder the execution of the program. These warning messages are listed only if FLAGW is specified.

C Conditional -- This severity level indicates that an error was made but that the compiler makes an assumption, which in some cases corrects the error. The statement containing the error is retained. Execution can be attempted for its debugging value.

E Error -- This severity level indicates that a serious error has been detected. Usually the compiler makes no corrective assumption. The statement or operand containing the error is dropped. Execution of the program should not be attempted.

D Disaster -- This severity level indicates that a serious error was made. Compilation is not completed. Results are unpredictable.

There is a correlation between severity level and the return codes referred to by the COND parameter. For example, a compilation in which a W-level error is detected generates a return code of 4; a C-level error, a code of 8; an E-level error, of 12; and a D-level error, of 16.

D. Message text. The text identifies the condition that caused the error and indicates the action taken by the compiler.

Since Report Writer generates a number of internal data items and procedural statements, some error messages may reflect internal names. In cases where the error manifests itself mainly in these generated routines, the error messages may indicate the card number of the RD entry for the report under consideration. In addition, there are errors that may indicate the card number of the card upon which the statement containing the error ends rather than the card upon which the error occurred. Messages for errors in the files refer to the card number of the associated SELECT clause. Internal name formats for Report Writer are discussed in the "Glossary."

On a given page of the listing, all messages beginning with the symbols 'IKF6' follow all other messages, as in the example below.

```
CARD  ERROR MESSAGE

93    IKF1015I-E     EXTERNAL NAME IN
                       SYSTEM-NAME *****
                       INVALID.
                       SYSTEM-NAME IGNORED.

      ERROR MESSAGE

      IKF6006I-E     SUPMAP SPECIFIED AND
                       E-LEVEL DIAGNOSTIC
                       HAS OCCURRED....
```

A complete list of compiler diagnostic messages is contained in the Program Product publication IBM OS Full American National Standard COBOL, Version 4 Messages.

12. Disposition messages from the job scheduler: These messages contain information about the disposition of the data sets, including volume serial numbers of volumes in which the data sets reside.

OBJECT MODULE

The object module contains the external symbol dictionary, the text of the program, and the relocation dictionary. It is followed by an END statement that marks the end of the module. For more detailed information about the external symbol dictionary, text, and relocation dictionary, see the publication IBM OS Linkage Editor and Loader.

An object module deck is punched if the DECK option is specified unless SUPMAP is specified and an E-level diagnostic message is generated, and if a SYSPUNCH DD statement is included. An object module is written in an output volume if the LOAD option is specified unless SUPMAP is specified and an E-level diagnostic message is generated, and if a SYSLIN DD statement is included.

Table 21. Glossary Definition and Usage

Type	Definition[1]	Usage
Group Fixed Length	DS 0CLN	GROUP
Alphabetic	DS NC	DISP
Alphanumeric	DS NC	DISP
Alphanumeric Edited	DS NC	AN-EDIT
Group Variable Length	DS VLI=N	GROUP
Numeric edited	DS NC	NM-EDIT
Sterling Report	DS NC	RPT-ST
External Decimal	DS NC	DISP-NM
External Floating Point	DS NC	DISP-FP
Internal Floating Point	DS 1F[2] or 4C	COMP-1
	DS 1D[2] or 8C	COMP-2
Binary	DS 1H[2],1F[2],2F[2],2C,4C,8C	COMP
Internal Decimal	DS NP	COMP-3
Sterling Non-Report	DS NC	DISP-ST
Index-Name	BLANK	INDEX-NAME
File (FD)	BLANK	FILE PROCESSING TECHNIQUE
Condition (88)	BLANK	BLANK
Report Definition (RD)	BLANK	BLANK
Sort Definition (SD)	BLANK	BLANK

[1]In this column, N = size in bytes, except in group variable length where it is a variable-length cell number.
[2]If the SYNCHRONIZED clause appears, these fields are used.

Table 22. Symbols Used in the Listing and Glossary to Define Compiler-Generated Information

Symbol	Definition
DNM	Source Data Name
SAV	Save Area Cell
SAV2	Input/Output Error Save Cell
SAV3	OPEN Parameter
SWT	Switch Cell
TLY	Tally Cell
WC	Working Cell
TS	Temporary Storage Cell
TS2	Temporary Storage (Non-Arithmetic)
TS3	Temporary Storage (Synchronization)
TS4	Temporary Storage (Table-Handling)
VLC	Variable Length Cell
SBL	Secondary Base Locator
BL	Base Locator
BLL	Base Locator for Linkage Section
ON	On Counter
PFM	Perform Counter
PSV	Perform Save
VN	Variable Procedure Name
DEC	DECB Address
SBS	Subscript Address
XSW	Exhibit Switch
XSA	Exhibit Save Area
PRM	Parameter
PN	Source Procedure Name
GN	Generated Procedure Name
DCB	DCB Address
VNI	Variable Name Initialization
LTL	Literal
INX	Index Cell
V(BCDNAME)	Virtual
RSV	Report Save Area
SSV	Sort Save Area
CKP	Checkpoint Counter
PBL	Procedure Block (Optimizer)

LINKAGE EDITOR OUTPUT

The output of the linkage editor job step may include:

- A printed listing of the job control statements
- A map of the load module after it has been processed by the linkage editor
- A cross-reference list
- Informative messages
- Diagnostic messages
- Disposition messages
- A listing of the linkage-editor control statements
- A load module that must be assigned to a library

Any diagnostic messages or informative messages associated with the linkage editor are automatically generated as output. The other forms of output may be requested by the PARM parameter in the EXEC statement. All output to be listed is written in the data set specified by the SYSPRINT DD statement.

Figure 53 is an example of linkage editor output listing. It shows the job control statements, informative messages, and module map. The different types of output are numbered and each type to be explained is lettered. The text following Figure 53 is an explanation of the illustration.

```
          .
          .
          .
  XXLKED       EXEC  PGM=IEWL,PARM=(XREF,LIST,LET),COND=(5,LT,COB),         X00000100
  XX                 REGION=96K                                              00000110
  XXSYSLIN     DD DSNAME=&LOADSET,DISP=(OLD,DELETE)                          00000120
  XX           DD DDNAME=SYSIN                                               00000130
  XXSYSLMOD    DD DSNAME=&GODATA(RUN),DISP=(NEW,PASS),                      X00000140
1 XX                 UNIT=SYSDA,SPACE=(1024,(50,20,1))                       00000150
  XXSYSLIB  DD DSN=SYS1.COBULIB,DISP=SHR                                     00000160
  XXSYSUT1     DD UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),                        X00000170
  XX                 SPACE=(1024,(50,20))                                    00000180
  XXSYSPRINT DD SYSOUT=A                                                     00000190
  XXSYSUDUMP  DD  SYSOUT=A                                                   00000200

  IEF236I ALLOC. FOR TESTRUN  LKED
  IEF237I 235    ALLOCATED TO SYSLIN
  IEF237I 242    ALLOCATED TO SYSLMOD
2 IEF237I 243    ALLOCATED TO SYSLIB
  IEF237I 230    ALLOCATED TO SYSUT1
  IEF237I 230    ALLOCATED TO SYSPRINT
  IEF237I 230    ALLOCATED TO SYSUDUMP

3  F88-LEVEL LINKAGE EDITOR OPTIONS SPECIFIED XREF,LIST,LET
              VARIABLE OPTIONS USED -  SIZE=(90112,16384)-          DEFAULT OPTION(S) USED

                                                     CROSS REFERENCE TABLE
          A                                  B
   CONTROL SECTION                           ENTRY

     NAME      ORIGIN  LENGTH                  NAME     LOCATION     NAME    LOCATION     NAME    LOCATION     NAME    LOCATION

5  TESTRUN        00      800
   ILBODSP0*     800      6A6
   ILBOERR0*     EA8      1F6
                                           ILBOERR1      EA8     ILBOERR2      EEA     ILBOERR4      F18     ILBOERR3      F3E
                                           ILBOERR5      FA6
   ILBOSTP0*    1090       35
                                           ILBOSTP1     10A6
      A               B                 C
6  LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION          LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION

      438           ILBOSTP0        ILBOSTP0                   43C           ILBODSP0         ILBODSP0
      440           ILBOSTP1        ILBOSTP0                    EC           ILBOERR1         ILBOERR0
      198           ILBOERR1        ILBOERR0

5  ENTRY ADDRESS       00  C
   TOTAL LENGTH      10C8  D

4  ****RUN        DOES NOT EXIST BUT HAS BEEN ADDED TO DATA SET

   IEF285I    SYS71023.T011209.RV000.TESTRUN.LOADSET        DELETED
7  IEF285I    VOL SER NOS= 231401.
   IEF285I    SYS71023.T011209.RV000.TESTRUN.GODATA         PASSED
   IEF285I    VOL SER NOS= DDB116.
```

Figure 53. Linkage Editor Output Showing Module Map and Cross-Reference List

1. The job control statements. These statements are listed because MSGLEVEL=(1,1) is specified on the JOB statement for this job, shown in Figure 52.

2. Allocation messages from the job scheduler. These messages provide information about the device allocation for the data sets in the job step. For example, the message

   IEF2371 230 ALLOCATED TO SYSUT1

   indicates that the data set for SYSUT1 has been assigned to the device 230.

3. Linkage editor informative message. This message lists the PARM options that were specified.

4. Linkage editor informative message. This is a disposition message describing the disposition of the load module.

   A. Name of the load module specified in the DSNAME parameter of the SYSLMOD DD statement

   B. Text of message

5. Module map. The module map is listed when either the XREF or the MAP option is specified in linkage editor processing. The module map shows all control sections in the output module and all entry names in each control section. The control sections are arranged in ascending order according to their assigned origins. All entry names are listed below the control section in which they are defined. Each COBOL program is a control section, and any COBOL library subroutine is a separate control section (except as noted under segmentation).

   A. Control section. Under this heading the name, origin, and length of each control section is listed.
      Name. The name of the control section. This name is the PROGRAM-ID name in the main COBOL program or a called program. Each control section that is obtained from a library by an automatic library call is indicated by an asterisk.
      Origin. The relative origin in hexadecimal notation.
      Length. The number of bytes in each control section in hexadecimal notation.

   B. Entry. The entry names within each control section and their relative location. A called program may have more than one entry point. For a called COBOL program, the entry points are the same as the names specified by the ENTRY statements in the source program.

   C. Entry address. The relative address of the instruction with which processing of the module begins. It will always be INIT1 if the COBOL program is the main program of the load module.

   D. Total length. The total number of bytes, in hexadecimal notation, of the load module. It is the sum of the lengths of all control sections.

6. Cross reference list. The cross reference list, as well as a module map, is listed if the XREF option is specified. The MAP and XREF options should not be specified together. The cross reference list provides the following information:

   A. Location. The relative location in the program where another program is called.

   B. Symbol reference. The name of the entry point of the called program.

   C. In control section. The control section that contains the entry point.

   For example, 440 is the location where a COBOL subroutine is called. ILBOSRV1 is the entry point of the called program. ILBOSRV0 is the control section that contains the entry point ILBOSTP1.

   If XREF is specified, the cross reference list appears before the Entry Address.

7. Disposition messages from the job scheduler. These messages contain information about the disposition of the data sets.

Comments on the Module Map and Cross Reference List

The severity of linkage editor diagnostic messages may affect the production of the module map and the cross reference list.

Since various processing options will affect the structure of the load module, the text of the module map and cross reference list will sometimes provide additional information. For example, the load module may have an overlay structure. In this case, a module map will be listed for each segment in the overlay structure. The cross reference list is the same as that previously discussed, except that segment numbers also are listed to indicate the segment in which each symbol appears.

Listing the Linkage Editor Control Statements: If the LIST option is specified, linkage editor control statements, such as OVERLAY and LIBRARY, are listed.

Linkage Editor Messages

The linkage editor may generate informative or diagnostic messages. A complete list of these messages is included in the publication IBM OS Linkage Editor and Loader.

LOADER OUTPUT

Loader output consists of a collection of diagnostic and error messages, and, if MAP is specified, a storage map of the loaded program. The output data set, SYSLOUT is sequential and blocked as specified by the user in the DCB. For better performance, the user can also specify the number of buffers to be allocated.

Diagnostic messages include a loader heading and a list of options requested by the user. The error messages, identifying the source of error, will be written when the error is detected. After processing is complete, an explanation of the error will be written. A complete list of loader diagnostic messages is found in the publication IBM OS Linkage Editor and Loader.

The map includes the name and absolute address for each control section and entry point defined in the program. It is written on SYSLOUT concurrently with input processing so it appears in order of input ESD items. The total size and storage extent also are included. Figure 54 is an example of a module map.

COBOL LOAD MODULE EXECUTION OUTPUT

The output generated by program execution (in addition to data written in program output files) can include:

- Data displayed on the console, or on the printer
- Cards
- Messages to the operator
- System informative messages
- System diagnostic message
- A system dump
- Debugging information

Note: If a program ends abnormally and one of the options FLOW, STATE, or SYMDMP is in effect and the SYSDBOUT DD card has been included, debugging information appears in the program listing (see the chapter entitled "Symbolic Debugging Features").

A dump as well as system diagnostic messages are generated automatically if a program contains errors that cause abnormal termination.

Note: If a COBOL program abnormally terminates, then a formatted dump is provided for all COBOL programs compiled with the SYMDMP option which could include the abnormally terminating program and its callers, up to and including the main program. For a discussion of the SYMDMP option as well as of other COBOL symbolic debugging options, see the chapter entitled "Symbolic Debugging Features."

Figure 55 shows an example of output from the execution job step. The following text is an explanation of the illustration.

1. The job control statements. These statements are listed because MSGLEVEL=(1,1) is specified in the JOB statement for this job.

2. The job allocation messages from the job scheduler. These messages indicate the device that is allocated for each data set defined for the job step.

3. Disposition messages from the job scheduler. These messages are contained in the publication IBM OS Messages and Codes.

4. Program output on printer. The results of execution of the TRACE and EXHIBIT NAMED statements appear on program listing.

5. Console output. Data is printed on console as a result of execution of DISPLAY UPON CONSOLE.

```
                              OS/360 LOADER

OPTIONS USED - PRINT,MAP,NOLET,CALL,NORES,SIZE=424176

   NAME   TYPE  ADDR       NAME   TYPE  ADDR       NAME   TYPE  ADDR       NAME   TYPE  ADDR       NAME   TYPE  ADDR

  SAMPL2B   SD   161E0    SAMPL2BA  SD   16EC8    IHEMAIN   SD   17CF8    IHENTRY   SD   17D00    IHESPRT   SD   17D10
  SYSIN     SD   17D48    IHEVQC  * SD   17D80    IHEVQCA * LR   17D80    IHEVQB  * SD   17FD8    IHEVQBA * LR   17FD8
  IHEDIA  * SD   183C0    IHEDIAA * LR   183C0    IHEDIAB * LR   183C2    IHEVPE  * SD   18608    IHEVPEA * LR   18608
  IHEVPA  * SD   18870    IHEVPAA * LR   18870    IHEVFC  * SD   189D0    IHEVFCA * LR   189D0    IHEVPC  * SD   189F8
  IHEVPCA * LR   189F8    IHEVFE  * SD   18BE8    IHEVFEA * LR   18BE8    IHEVSC  * SD   18C08    IHEVSCA * LR   18C08
  IHEDNC  * SD   18CB8    IHEDNCA * LR   18CB8    IHEDOA  * SD   18F30    IHEDOAA * LR   18F30    IHEDOAB * LR   18F32
  IHEDMA  * SD   19010    IHEDMAA * LR   19010    IHEVFD  * SD   19108    IHEVFDA * LR   19108    IHEVFA  * SD   19160
  IHEVFAA * LR   19160    IHEVPB  * SD   19248    IHEVPBA * LR   19248    IHEXIS  * SD   193F0    IHEXISO * LR   193F0
  IHEIOB  * SD   19488    IHEIOBA * LR   19488    IHEIOBB * LR   19490    IHEIOBC * LR   19498    IHEIOBD * LR   194A0
  IHESARC * LR   1A9C8    IHESADD * LR   1A9DE    IHESAFF * LR   1AA18    IHEPRT  * SD   1AB70    IHEPRTA * LR   1AB70
  IHEBEGA * LR   1AE28    IHEERR  * SD   1AE68    IHEERRD * LR   1AE68    IHEERRC * LR   1AE72    IHEERRB * LR   1AE7C
  IHEERRA * LR   1AE86    IHEERRE * LR   1B4E2    IHEIOF  * SD   1B580    IHEIOFB * LR   1B580    IHEIOFA * LR   1B582
  IHEITAZ * LR   1B81E    IHEITAX * LR   1B82A    IHEITAA * LR   1B83E    IHEDCN  * SD   1B860    IHEDCNA * LR   1B860
  IHEDCNB * LR   1B862    IHEIOD  * SD   1BA50    IHEIODG * LR   1BA50    IHEIODP * LR   1BA52    IHEIODT * LR   1BB4[illegible]
  IHEVTB  * SD   1BCF0    IHEVTBA * LR   1BCF0    IHEVQA  * SD   1BD78    IHEVQAA * LR   1BD78



  IHEQINV   PR      00    IHEQERR   PR       4    SAMPL2BB  PR       8    SAMPL2BC  PR       C    IHEQSPR    PR     10
  SYSIN     PR      14    IHEQLSA   PR      18    IHEQLW0   PR      1C    IHEQLW1   PR      20    IHEQLW2    PR     24
  IHEQLW3   PR      28    IHEQLW4   PR      2C    IHEQLWE   PR      30    IHEQLCA   PR      34    IHEQVDA    PR     38
  IHEQFVD   PR      3C    IHEQCFL   PR      40    IHEQFOP   PR      48    IHEQADC   PR      4C    IHEQXLV    PR     50
  IHEQEVT   PR      58    IHEQSLA   PR      60    IHEQSAR   PR      64    IHEQLWF   PR      68    IHEQRTC    PR     6C
  IHEQSFC   PR      70

IEW1001   IHEUPBA
IEW1001   IHEUPAA
IEW1001   IHETERA
IEW1001   IHEM91C
IEW1001   IHEM91B
IEW1001   IHEM91A
IEW1001   IHEDDOD
IEW1001   IHEVPFA
IEW1001   IHEVPDA
IEW1001   IHEDBNA
IEW1001   IHEVSFA
IEW1001   IHEVSBA
IEW1001   IHEVCAA
IEW1001   IHEVSAA
IEW1001   IHEDNBA
IEW1001   IHEUPBB
IEW1001   IHEUPAB
IEW1001   IHEVSEB

   TOTAL LENGTH     5068
   ENTRY ADDRESS    17D00

IEW1001   WARNING - UNRESOLVED EXTERNAL REFERENCE (NOCALL SPECIFIED)
```

Figure 54. Module Map Format Example

```
(1)  XXGO       EXEC   PGM=*.LKED.SYSLMOD,COND=((5,LT,COB),(5,LT,LKED))    00000210
     XXSYSUDUMP DD SYSOUT=A                                                00000220
     //GO.SYSOUT DD SYSOUT=G
     //GO.SAMPLE DD UNIT=2400,LABEL=(,NL)

(2)  IEF236I ALLOC. FOR TESTRUN  GO
     IEF237I 242    ALLOCATED TO PGM=*.DD
     IEF237I 230    ALLOCATED TO SYSUDUMP
     IEF237I 230    ALLOCATED TO SYSOUT
     IEF237I 182    ALLOCATED TO SAMPLE

(3)  IEF285I    SYS71023.T011209.RV000.TESTRUN.GODATA        PASSED
     IEF285I    VOL SER NOS= DD8116.
     IEF285I    SYS71023.T011209.SV000.TESTRUN.R0000011      DELETED
     IEF285I    VOL SER NOS= 231400.
     IEF285I    SYS71023.T011209.SV000.TESTRUN.R0000012      SYSOUT
     IEF285I    VOL SER NOS= 231400.
     IEF285I    SYS71023.T011209.RV000.TESTRUN.R0000013      DELETED
     IEF285I    VOL SER NOS= L00300.

(4)  58
     62
     66
     68
     62
     66
     62
     66
     .
     .
     .
     (Repeat 21 times)
     .
     .
     .
     62
     66
     62
     66
     71
     74
     75
     WORK-RECORD = A 0001 NYC Z
     74
     75
     WORK-RECORD = B 0002 NYC 1
     74
     75
     WORK-RECORD = C 0003 NYC 2
     74
     75
     WORK-RECORD = D 0004 NYC 3
     74
     75
           .
           .
           .
     WORK-RECORD = V 0022 NYC 1
     74
     75
     WORK-RECORD = W 0023 NYC 2
     74
     75
     WORK-RECORD = X 0024 NYC 3
     74
     75
     WORK-RECORD = Y 0025 NYC 4
     74
     75
     WORK-RECORD = Z 0026 NYC Z
     74
     78

(5)  A 0001 NYC 0
     B 0002 NYC 1
     C 0003 NYC 2
     D 0004 NYC 3
           .
           .
           .
     V 0022 NYC 1
     W 0023 NYC 2
     X 0024 NYC 3
```

Figure 55. Execution Job Step Output

REQUESTS FOR OUTPUT

1. The programmer can request data to be displayed by using the DISPLAY statement and including the following in the job control procedure:

```
//SYSOUT     DD   SYSOUT=A
```

2. Message to the operator can also be displayed on the console when requested in the source program (DISPLAY UPON CONSOLE).

3. The programmer can request debugging information in case of an abnormal termination by specifying FLOW and/or STATE and including the following in the job control procedure:

```
//SYSDBOUT   DD   SYSOUT=A
```

4. The programmer can request a full dump, in case his program is terminated abnormally, by including the following in the job control procedure:

```
//SYSABEND   DD   SYSOUT=A
```

<u>Note</u>: Under MVT, the SPACE parameter should also be included in the DD statement. For example:

```
//SYSABEND   DD   SYSOUT=A,                X
//                SPACE=                   X
//                (125,(200,1000),RLSE)
```

Dumps and debugging facilities are explained in "Program Checkout."

OPERATOR MESSAGES

The COBOL load module may issue operator messages. A complete list of these messages and required operator responses can be found in the publication <u>IBM OS Full American National Standard COBOL, Version 4 Messages.</u> MCS considerations are discussed there also.

<u>SYSTEM OUTPUT</u>

Informative and diagnostic messages may appear in the listing during execution of any job step. Further information about system diagnostics is found in the publication <u>IBM OS Messages and Codes</u>. COBOL messages and associated documentation for this compiler appear in the Program Product publication <u>IBM OS Full American National Standard COBOL, Version 4 Messages</u>.

Each of these messages contains an identification code in the first three columns of the message to indicate the portion of the operating system that generated the message. Table 23 lists these codes, along with an identification of each.

Table 23. System Message Identification Codes

Code	Identification
IEA	An on-line console message from the supervisor.
IEC	An on-line console message from data management.
IEF	A message from the job scheduler.
IKF	A message from the COBOL compiler.
IER	A message from the Sort program.
IET	A message from the assembler.
IEW	A message from the linkage editor.
IHB	A message from the supervisor and data management.

A programmer using the COBOL compiler under the IBM Operating System has several methods available to him for testing and debugging his programs or revising them for increased efficiency of operation.

The syntax-checking options can be specified to save programmer and machine time while checking the source statements for syntax errors.

The COBOL debugging language can be used by itself or in conjunction with other COBOL statements. A dump can also be used for program checkout. For a discussion of the COBOL symbolic debugging options, see the chapter entitled "Symbolic Debugging Features."

SYNTAX-CHECKING COMPILATION

The compiler checks the source text for syntax errors and then generates the appropriate error messages. With the syntax-checking feature, the programmer can request a compilation either conditionally, with object code produced only if no messages or just W- or C-level messages are generated, or unconditionally, with no object code produced regardless of message level.

Selected test cases run with the syntax-checking feature have resulted in a compilation-time saving of as much as 70%. For a discussion of the syntax-checking options, SYNTAX and CSYNTAX, see the section "Options for the Compiler" under "Job Control Procedures."

DEBUGGING LANGUAGE

The COBOL debugging language is designed to aid the COBOL programmer in producing an error-free program in the shortest possible time. The sections that follow discuss the use of the debugging language and other methods of program checkout.

The three debugging language statements are TRACE, EXHIBIT, and ON. Any one of these statements can be used as often as necessary. They can be interspersed throughout a COBOL source program, or they can be in a packet in the input stream to the compiler.

Program debugging statements may not be desired after testing is completed. A debugging packet can be removed after testing. This allows elimination of the extra object program coding generated for the debugging statements.

The output produced by the TRACE and EXHIBIT statements is listed on the system logical output device (SYSOUT). If these statements are used, the SYSOUT DD statement must be specified in the execution time job step.

The following discussions describe ways to use the debugging language.

FOLLOWING THE FLOW OF CONTROL

The READY TRACE statement causes the compiler generated card numbers for each section and paragraph name to be listed on the system output unit when control passes to that point. The output appears as a list of card numbers.

To reduce execution time, a trace can be stopped with a RESET TRACE statement. The READY TRACE/RESET TRACE combination is helpful in examining a particular area of the program. The READY TRACE statement can be coded so that the trace begins before control passes to that area. The RESET TRACE statement can be coded so that the trace stops when the program has passed the area. The two trace statements can be used together where the flow of control is difficult to determine, e.g., with a series of PERFORM statements or with nested conditionals.

Another way to control the amount of tracing, so that it is done conditionally, is to use the ON statement with the TRACE statement. When the COBOL compiler encounters an ON statement, it sets up a mechanism such as a counter that is incremented during execution whenever control passes through the ON statement. For example, if an error occurs when a specific record is processed, the ON statement can be used to isolate the problem record. The statement should be placed where control passes only once for each record that is read. When the contents of the counter equal the number of the record (as specified in the ON statement), a trace can be taken on that record. The following example shows a way in which the processing of the 200th record could be selected for a TRACE statement.

```
Col.
1      8
----------------------------------------
       RD-REC.
       .
       .
       .
DEBUG  RD-REC.
       PARA-NM-1.    ON 200 READY TRACE.
                     ON 201 RESET TRACE.
```

If the TRACE statement were used without the ON statement, the processing of every record would be traced.

A common program error could be either (1) failing to break a loop, or (2) unintentionally creating a loop. If many iterations of the loop are required before it can be determined that there is a program error, the ON statement can be used to initiate a trace only after the expected number of iterations has been completed.

Note: If an error occurs in an ON statement, the diagnostic message may refer to the previous statement number.

DISPLAYING DATA VALUES DURING EXECUTION

A programmer can display the value of a data item during program execution by using the EXHIBIT statement. The three forms of this statement display (1) the names and values of the identifiers or nonnumeric literals listed in the EXHIBIT statement (EXHIBIT NAMED) whenever the statement is encountered during execution, (2) the values of the items listed in this statement only if the value has changed since the last execution (EXHIBIT CHANGED), and (3) the names and values of the items listed in the statement only if the values have changed since the previous execution (EXHIBIT CHANGED NAMED).

Note: The combined total length of all items displayed with EXHIBIT CHANGED and EXHIBIT CHANGED NAMED cannot exceed 32,767 bytes. The length of any one operand must be less than or equal to 256 bytes. The length of a "NAME" must be less than or equal to 120 characters.

Data can be used to check the accuracy of the program. For example, the programmer can display specified fields from records, work the calculations himself, and compare his calculations with the output from his program. The coding for a payroll problem could be:

```
Col.
1      8
----------------------------------------
              .
              .
              .
       GROSS-PAY-CALC.
           COMPUTE GROSS-PAY =
           RATE-PER-HOUR * (HRSWKD
           + 1.5 * OVERTIMEHRS).
       NET-PAY-CALC.
              .
              .
              .
DEBUG  NET-PAY-CALC
       SAMPLE-1. ON 10 AND
           EVERY 10 EXHIBIT NAMED
           RATE-PER-HOUR, HRSWKD,
           OVERTIMEHRS, GROSS-PAY.
```

This coding will cause the values of the four fields to be listed for every tenth data record before net pay calculations are made. The output could appear as:

```
RATE-PER-HOUR = 4.00 HRSWKD = 40.0
     OVERTIMEHRS = 0.0 GROSS-PAY = 160.00

RATE-PER-HOUR = 4.10 HRSWKD = 40.0
     OVERTIMEHRS = 1.5 GROSS-PAY = 173.23

RATE-PER-HOUR = 3.35 HRSWKD = 40.0
     OVERTIMEHRS = 0.0 GROSS-PAY = 134.00
```

Note: Decimal points are included in this example for clarity, but actual printouts depend on the data description in the program.

The preceding is an example of checking at regular intervals (every tenth record). A check of any unusual conditions can be made by using various combinations of COBOL statements in the debug packet. For example:

```
IF OVERTIMEHRS GREATER THAN 2.0
   EXHIBIT NAMED PAYRCDHRS
```

In connection with the previous example, this statement could cause the entire pay record to be displayed whenever an unusual condition (overtime exceeding two hours) is encountered.

The EXHIBIT CHANGED statement also can be used to monitor conditions that do not occur at regular intervals. The values of the items are listed only if the value has changed since the last execution of the statement. For example, suppose the program calculates postage rates to various cities. The flow of the program might be as shown in Figure 56.

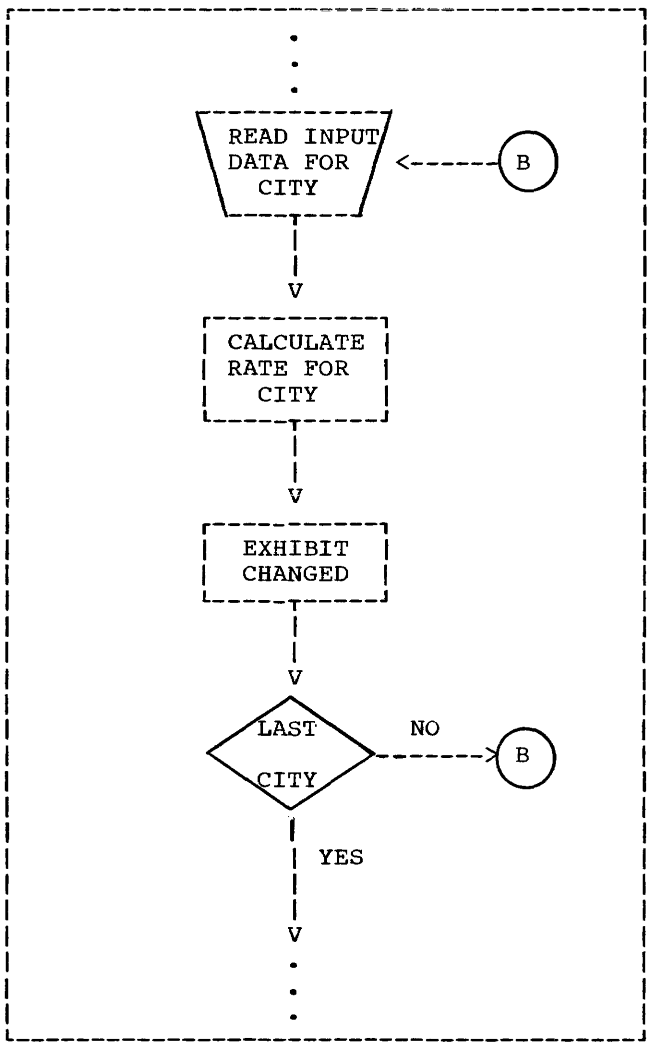


Figure 56. Example of Program Flow

The EXHIBIT CHANGED statement in the program could be:

```
EXHIBIT CHANGED STATE CITY RATE
```

The output from the EXHIBIT CHANGED statement could appear as:

```
01  01  10
    02  15
    03
    04  10
02  01
    02  20
    03  15
    04
03  01  10
     .
     .
     .
```

The first column contains the code for a state, the second column contains the code for a city, and the third column contains the code for the postage rate. The value of an item is listed only if it is changed since the previous execution. For example, since the postage rate to city 02 and 03 in state 01 are the same, the rate is not printed for city 03.

The EXHIBIT CHANGED NAMED statement lists the name of the data item and the value of that item if the value has changed. For example, the program might calculate the cost of various methods of shipping to different cities. After the calculations are made, the following statement could be in the program:

```
EXHIBIT CHANGED NAMED STATE CITY RAIL
   BUS TRUCK AIR
```

The output from this statement could appear as:

```
STATE = 01 CITY = 01 RAIL = 10
   BUS = 14 TRUCK = 12 AIR = 20

CITY = 02

CITY = 03 BUS = 06 AIR = 15

CITY = 04 RAIL = 30 BUS = 25
   TRUCK = 28 AIR = 34

STATE = 02 CITY = 01 TRUCK = 25

CITY = 02 TRUCK = 20 AIR = 30
      .
      .
      .
```

Note that the name of the item and its value are listed only if the value has changed since the previous execution.

TESTING A PROGRAM SELECTIVELY

A debug packet allows the programmer to select a portion of the program for testing. The packet can include test data and can specify operations the programmer wants performed. When the testing is completed, the packet can be removed. The flow of control can be selectively altered by the inclusion of debug packets, as shown in Figure 57.

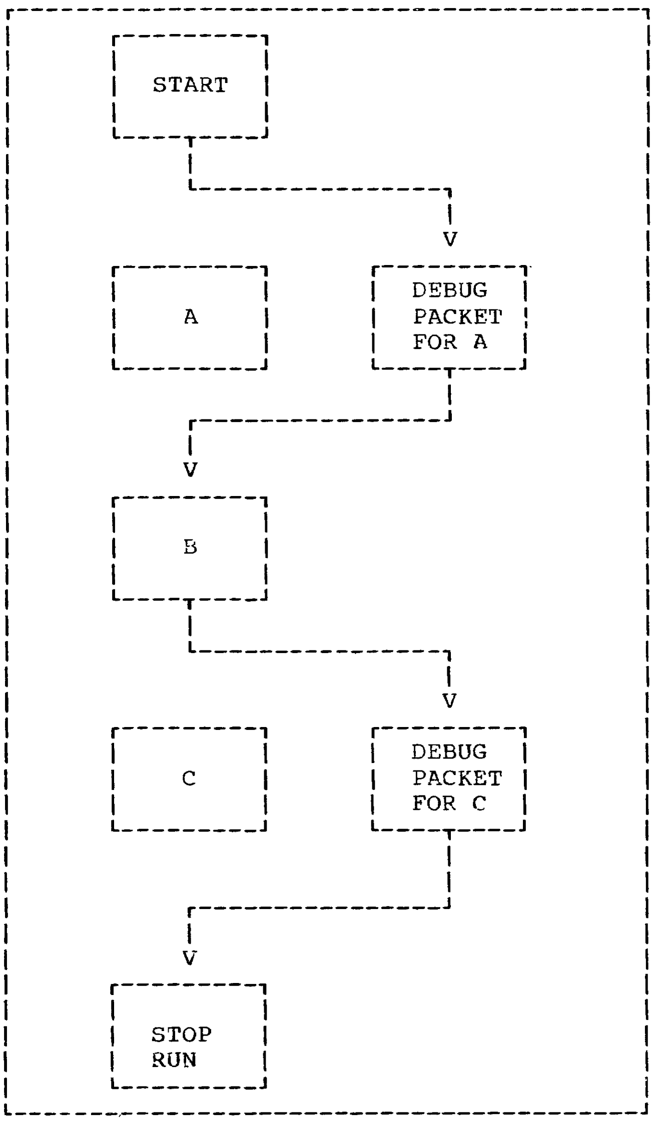


Figure 57. Selective Testing of B

In this program, A creates data, B processes it, and C prints it. The debug packet for A simulates test data. It is first in the program to be executed. In the packet, the last statement is GO TO B, which permits A to be bypassed. After B is executed with the test data, control passes to the debug packet for C, which contains a GO TO statement that transfers control to the end of the program, bypassing C.

TESTING CHANGES AND ADDITIONS TO PROGRAMS

If a program runs correctly but changes or additions can make it more efficient, a debug packet can be used to test changes without modifying the original source program.

If the changes to be incorporated are in the middle of a paragraph, the entire paragraph, with the changes included, must be written in the debug packet. The last statement in the packet should be a GO TO statement that transfers control to the next procedure to be executed.

There are usually several ways to perform an operation. Alternative methods can be tested by putting them in debug packets.

The source program library facility can be used for program checkout by placing a source program in a library (see "Libraries"). Changes or additions to the program can be tested by using the BASIS card and any number of INSERT and DELETE cards. Such changes or additions remain in effect only for the duration of the run.

A debug packet can also be used in conjunction with the BASIS card to debug a program or to test deletions or additions to it. The debug packet is inserted in the input stream immediately following the BASIS card and any INSERT or DELETE cards.

DUMPS

If a serious error occurs during execution of a program, the job is abnormally terminated; any remaining steps are bypassed, and a dump is generated. The programmer can use the dump for program checkout. (However, any pending transfers to an external device may not be completed. For example, if a READY TRACE statement is in effect when the job is abnormally terminated, the last card number may not appear on the external device.) In cases where the abnormal termination does not go to completion, a dump is not produced. This situation may cause duplicate name definition when the next job is run, and is discussed at the end of this section.

If a SYSUDUMP DD statement has been included in the execution-time job step, the system will provide the programmer with a printout, in hexadecimal and EBCDIC format, of main storage. Those areas occupied by the problem program and its data at the time the error occurred, will be included. This printout is called an abnormal termination dump and is identified by the heading

*** ABDUMP REQUESTED ***

If a SYSABEND DD statement is specified, the contents of the nucleus is also printed.

If neither a SYSUDUMP nor a SYSABEND DD statement is included in the execution-time job step, or its specification has been destroyed, an indicative dump is produced. This dump does not contain a printout of main storage and is not given under MVT.

All dumps include a completion code designating the condition that caused the termination. The completion code consists of a system code and a user code. Only one of the codes is nonzero. A nonzero system code indicates that the control program detected the error.

The COBOL programmer can now request dynamic dumps via a compile-time option. The SYMDMP option, requested in the PARM parameter of the EXEC statement, produces a symbolic formatted dump of the data area of the object program when the program abnormally terminates. At execution time, the user can also request a dynamic dump at any point in the Procedure Division.

Notes:

- If a COBOL program abnormally terminates, then a formatted dump is produced for all COBOL programs compiled with the SYMDMP option which could include the abnormally terminating program and its callers, up to and including the main program.
- The explanation of the system-generated completion codes and a complete description of the dumps are contained in the publication IBM OS Programmer's Guide to Debugging. For a discussion of the COBOL symbolic debugging options, see the chapter entitled "Symbolic Debugging Features."

ERRORS THAT CAN CAUSE A DUMP

Following is a discussion of some error conditions that can cause a program to be abnormally terminated and a dump to be listed.

Input/Output Errors

Errors can occur while a COBOL file is being processed. For example, during data transmission, an input/output error may occur that cannot be corrected. If the file being processed is organized sequentially and no error-processing declarative or INVALID KEY option has been specified for the file, the job is terminated. If it is a QSAM file, the job will be terminated when there is no declarative or INVALID KEY option and the EROPT=ABE option in the DD statement has been specified.

Referring to an input area before OPEN and READ statements are issued can cause unpredictable results, because base locator (BL) cells and registers are not properly initialized.

Another error that can cause termination is an attempt to read a file whose records are of a different size than those described in the source program. The section "Additional File Processing Information" contains more information about input/output errors.

Errors Caused by Invalid Data

Abnormal termination of a job occurs when a data item with an invalid format is processed in the Procedure Division.

Some of the program errors are:

1. A data item in the Working-Storage Section is not initialized before it is used, causing invalid data to be picked up.

2. For an item whose usage is COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2, either the alignment is incorrect, or the description of the item does not specify the proper alignment. Some examples are:

   a. A redefining entry contains one or more of the above items and the redefined entry is not properly aligned. Alignment will not be performed for items that cause the starting address of the redefining item to be changed.

   b. A record in the Linkage Section of a called program is described by an 01 entry and contains one or more of the above items, and the corresponding argument in the calling program is not properly aligned.

c. A file, containing one or more of the above items, is blocked, but the required inter-record slack bytes were not inserted when the file was created. If the file is later read as an input file, the alignment may not be correct.

3. An input file or received message contains invalid data or data incorrectly defined by its data description. For example, the contents of the sign position of an internal or external decimal data item in the file may be invalid. The compiler does not generate a test to check the sign position for a valid configuration before the item is used as an operand.

4. If a group item is moved to a group item and the subordinate data descriptions are incompatible, the new data in the receiving field may not match the corresponding data descriptions. (Conversion or editing is not performed in a move involving a group item.)

5. The SIZE ERROR option is not specified for the COMPUTE statement and the result of the calculation is larger than the specified resultant COMPUTATIONAL data name. Using the result in a subsequent calculation might cause an error.

6. The SIZE ERROR option is not specified for a DIVIDE statement, and an attempt is made to divide by zero.

7. The USAGE specified for a redefining data item is different from the USAGE specified for the redefined item. An error results when the item is referred to by the wrong name for the current content.

8. A record containing a data item described by an OCCURS clause with the DEPENDING ON data-name option, may cause data items in the record to be affected by a change in the value of data-name during the course of program execution. This may result in incorrectly described data. Additional information about how to correct this situation is included in "Programming Techniques."

9. The data description in the Linkage Section of a called program does not correctly describe the data defined in the calling program.

10. Blanks read into data fields defined as numeric generate an invalid sign.

11. Some common errors that occur when clearing group items in storage are:

   a. Moving ALL ZEROS to a group level item to clear several counters causes an invalid sign to be generated in all of the elementary fields except the lowest order field.

   b. Moving SPACES to a group level item will put invalid data in any numeric field in that group.

   c. Moving 0 to a group level item moves one zero and pads the rest of the fields with blanks.

12. Failure to initialize counters produces incorrect results. No initial values are generated by the compiler unless specifically instructed to do so with a VALUE clause. If such fields are defined as decimal, internal or external, invalid signs may result in addition to unpredictable initial values. If defined as binary, they will cause unpredictable results and, further, if used in subscripting, may exceed the range of the associated OCCURS clause and cause data to be fetched or stored erroneously. An addressing exception may occur if the uninitialized subscript generates a bad address.

13. Not testing to insure that a subscript or index does not exceed the range of the associated OCCURS clause may lead to fetching and storing data from and to some incorrect locations.

14. Failure to initialize an index produces incorrect results. No initial values are generated by the compiler unless a SET statement is executed. When indexing is then specified, the range of the OCCURS clause may be exceeded and cause data to be fetched or stored erroneously. An addressing exception may occur if the initialized index generates an address outside the range of the machine, or a protection exception if data is stored outside the partition of this program.

15. A subscript or index set at zero will address data outside the range of the table.

16. If either HIGH-VALUE or LOW-VALUE is moved to internal or external decimal fields and those fields are used for comparisons, computations, or subscripting, a data exception will occur. HIGH-VALUE and LOW-VALUE are the hexadecimal values X'FF' and X'00', respectively.

Other Errors

1. No DD statement is included for a file described in the source program and an attempt is made to access the file. When an OPEN statement for the file is executed, the system console message is written. The programmer can elect to direct the operator to continue processing his program, but any READ or WRITE associated with the unlocated file will result in an abnormal termination. A similar situation exists when a file is closed WITH LOCK and an attempt is made to reopen it (see the Program Product publication IBM OS Full American National Standard COBOL, Version 4 Messages for the format of the generated error message).

2. A file is not opened and execution of a READ or WRITE statement for the file is attempted, or a MOVE to a record area in the file is attempted.

3. A GO TO statement, with no procedure name following it, is not properly initialized with an ALTER statement before the first execution of the GO TO statement.

4. Reference is made to an item in a file after end of data. This includes the use of the TERMINATE statement of the Report Writer feature, if the CONTROL FOOTING, PAGE FOOTING, or REPORT FOOTING contain items that are in the file (e.g., SOURCE data-name, where data-name refers to an item in the file).

5. Block size for an F-format file is not an integral multiple of the record length.

6. In a blocked and/or double buffered file, a count cannot be kept directly in a record.

7. A READ is issued for a data set referenced on a DD DUMMY statement. The AT END condition is sensed immediately and any reference to a record in the data set produces unpredictable results.

8. Under MVT, a STOP RUN statement is executed before all files are closed.

9. A SORT did not execute successfully. The programmer may check SORT-RETURN.

10. An input/output statement is issued for a file after the AT END branch is taken, without closing and reopening the file.

11. A SEND or RECEIVE statement is issued when a message control program is not running.

12. A SEND or RECEIVE statement is issued for a QNAME (i.e., the "QNAME=" parameter of the DD card) that is unknown to the message control program.

In addition to errors that can result in an abnormal termination, errors in the source program can occur that cause parts of the program to be overlaid and the corresponding object code instructions to become invalid. If an attempt is then made to execute one of these instructions, an abnormal termination may result because the operation code of the instruction is invalid, the instruction results in a branch to an area containing invalid instructions, or the instruction results in a branch to an area outside the program, such as an address protected area.

Some COBOL source program errors that can cause this overlaying are:

1. Using a subscript whose value exceeds the maximum specified in the associated OCCURS clause.

2. Using a data-name as a counter whose value exceeds the maximum value valid for that counter.

COMPLETION CODES

The following cases represent some of the errors that can occur in a COBOL program and the interrupt or completion code associated with them. These errors do not necessarily cause an abnormal termination at the time they are recognized and do not always hold true.

1. 013--Check register 2 of registers at the entry to ABEND. This address points to the DCB in conflict.

2. 043--Error occurred during the attempted opening of a TCAM application program data set, as described below.

   a. A value of 01 in register 0 indicates the attempted opening of a TCAM application program data set without an active message control program (MCP) in the system.

   b. A value of 02 indicates that the QNAME= parameter of a DD statement associated with an input or output DCB for a COBOL program is not the name of a process entry defined in the terminal table.

   c. A value of 03 indicates that the process entry named by the QNAME= parameter of a DD statement associated with a COBOL program is currently being used by another COBOL program.

   d. A value of 04 indicates that insufficient main storage was available in the MCP to build internal control blocks associated with the COBOL program interface. Specify a larger region or partition size in the JOB statement for the MCP.

   e. A value of 05 indicates that insufficient main storage was available in the COBOL work area to build internal control blocks. Specify a larger region or partition size in the JOB statement for the COBOL program.

3. 046--Error occurred during the termination of the TCAM MCP because the COBOL program data set was still open. Specify the STOP RUN statement when COBOL processing is complete. Ensure that all COBOL programs have terminated processing before deactivating the MCP.

4. 0C1--Operation Exception:

   a. When the interrupt is at 000048 or at 004800, look for a missing DD card or an unopened file.

   b. When the interrupt is at 000050, look at register 1 of the registers at entry to ABEND. Add hexadecimal 28 to the address found in register 1. This should point to the DD name of a missing DD statement.

   c. When the interrupt is at 00004A, look for a missing card, i.e.,

      //SYSOUT DD SYSOUT=A

      any missing JCL card, or the wrong name of a JCL card. Add hexadecimal 28 to the address found in register 1 at entry to ABEND. This should point to the DD name of the DD statement in error.

   d. When interrupt is at 00004F, look for inconsistent JCL or check the system-name in the COBOL program.

5. 0C4--Protection Exception:

   a. Check for the block size and record size being equal for variable record input or output.

   b. Check for missing SELECT statement.

   c. If interrupt is at 004814, check for an attempt to READ an unopened input file or a missing DD card.

   d. Check for an uninitialized index or subscript.

6. 0C5 and 0C6--Addressing and Specification Exception:

   a. Subscript or index value may have exceeded maximum and instruction or table area was overlaid.

   b. Check for an improper exit from a procedure being operated on by a PERFORM statement.

   c. Check for duplicate close of an input or output file if DS formatting discontinued.

   d. A sort is being attempted with an incorrect catalog procedure.

   e. Attempting to reference an input/output area before a READ or OPEN statement, respectively.

   f. Alignment for COMPUTATIONAL data is incorrect when record is blocked, and inter-record slack bytes were not inserted.

   g. Check for initialized subscript or index value.

7. 0C7--Data Exception:

   a. Data field was not initialized.

   b. Input record numeric field contains blanks.

   c. Subscript or index value exceeded maximum and invalid data was referenced.

   d. Data was moved from the DISPLAY field to the COMPUTATIONAL or COMPUTATIONAL-3 field at group level. Therefore, no conversion was provided.

   e. The figurative constants ZERO or LOW-VALUE moved to a group level numeric field.

   f. Omission of USAGE clause or erroneous USAGE clause.

   g. Incorrect Linkage Section data definition, passing parameters in wrong order, omission or inclusion of a parameter, failure to carry over a USAGE clause when necessary, or defining the length of a parameter incorrectly.

8. 001--I/O Error:

   a. Register 1 of the SVRB points to the DCB which caused the input/output problem. Look for input record and blocking errors. That is, the input does not agree with the record and blocking descriptions in the DCB, the COBOL file description, or the DD statement LRECL parameter.

   b. Attempted to READ after EOF has been sensed.

9. 002--Register 2 of registers at the entry to ABEND contains the address of the DCB for the file causing the input/output problem. Check the DCB list for the specific file.

10. 213--Error during execution of OPEN statement for data set on mass storage device, as follows:

    a. DISP parameter of DD statement specified OLD for output data set.

    b. Input/output error cannot be corrected when reading or writing the DSCB. Recreate the data set or resubmit the job, check register 14 of the registers at entry to ABEND. This address points to the file that has no DSCB.

11. 214--Error during CLOSE for data set on tape; there is an input/output error that cannot be corrected either in tape positioning or volume disposition. Resubmit the job and inform the field engineer if error persists.

12. 237--Error at EOV:

    a. Incorrect volume serial number specified in SER subparameter of VOLUME parameter of DD statement.

    b. Incorrect volume mounted.

    c. Incorrect labels.

13. 400--If this completion code is generated during a compile step, the member to be compiled has not been extracted from the source library for compilation.

14. 413--Error during execution of an OPEN statement for a data set on tape:

    a. Volume serial number was not specified for input data set.

    b. Volume could not be mounted on the allocated device.

    c. There is an input/output error in reading the volume label that cannot be corrected.

15. 806--The error occurred during execution of a LINK, XCTL, ATTACH, or LOAD macro instruction. An error was detected by the control program routine for the BLDL macro instruction. The contents of register 15 indicates the nature of the error:

    04 The requested program was not found in the indicated source private, job, or link library.

    08 An uncorrectable input/output error occurred when the control program attempted to search the directory of the library indicated as containing the requested program.

16. 80A--Insufficient contiguous core storage for linkage to some phase of the compiler. The programmer should look to see if secondary data-set allocation has caused an extra DEB to be built at lower core addresses within the region. If so, this problem can be corrected by assigning sufficient primary extents for the data set in question. See "Data Set Requirements" for further information.

17. 813--Error during execution of an OPEN statement in verification of labels:

    a. Volume serial number specified in VOLUME parameter of DD statement is incorrect.

    b. Data set name specified in DSNAME parameter is incorrect.

    c. Wrong volume is mounted.

18. When compilation is terminated with diagnostic message IKF0010I-D, IKF0020I-D, or IKF0030I-D, an abnormal termination dump is generated to provide additional debugging information.

Finding Location of Program Interruption in COBOL Source Program Using the Condensed Listing

To determine the location of the interruption, the programmer should proceed as follows:

1. From first page of dump:

    a. Get completion code and program interruption storage location.

    b. Determine the starting address of the program (PRB address+20).

2. From linkage editor listing:

    a. Determine storage address for each module. Add starting address of the program to origin of each module.

    b. Determine module in which interrupt storage location falls.

    c. Determine relative address. Subtract module storage address from interrupt location.

3. From Procedure Division map:

    a. Find the highest previous relative address in the condensed listing. That statement is in error.

    b. Get line number and verb of COBOL source statement.

4. From source listing find the line number and verb of source statement causing program interruption.

USING THE ABNORMAL TERMINATION DUMP

The programmer can also determine the cause of an abnormal termination with the following material:

1. The COBOL program object code listing.

2. A knowledge of the layout of the COBOL object module.

3. The full abnormal termination dump in conjunction with the linkage editor map or cross reference list.

A description of the linkage editor output and of the COBOL object code listing is found in "Output." Figure 53 shows the layout of the COBOL program object module.

Note: The information in this section about the use of the abnormal termination dump applies only when running under MFT. For information about the abnormal termination dumps under MVT, see the publication IBM OS Programmer's Guide to Debugging. Note that under the MVT option no indicative dumps are given.

The abnormal termination dump provides the address at which the load module has been loaded (load address) and the address of the instruction that caused the interrupt. The programmer computes the load module area by adding the load address to the load module length, as shown in the linkage editor output. It is now possible to determine whether the instruction falls within the load module. If it does not, the interrupt could have resulted from an improper branch to a point outside the load module or an error occurring in another part of the system.

If the instruction does fall within the load module, the programmer now determines in which part: the main program, a COBOL library subroutine, or a called program. The ranges of the various parts are determined by adding their relative origins, as shown in the linkage editor output, to the load address.

If the instruction occurred in an object module generated for a COBOL program, (i.e., the main program), the programmer can determine whether or not the instruction was one of the generated object code instructions. He can determine the address of the first instruction in the Procedure Division (as found in the object code listing) by adding its relative location to the location of the object module (load address plus relative origin). If it was one of the object code instructions, a similar technique can be used to locate the exact instruction. If it was not one of these instructions, the error has occurred in another part of the object module. Control possibly went there because of an improper branch.

If the instruction that initiated the dump occurred in a COBOL library subroutine, or if the original program called another program and the instruction occurred in the called program, the instruction can be located by a similar technique. The linkage editor cross reference list indicates the locations where the call to the program or subroutine in question was made.

The following general rules can be used to determine the cause of the dump and the error.

1. Determine the COBOL statement that generated the code leading to the program check.

    a. The top of the system dump will tell the address of the PC (Program Check) instruction and the type of PC. Locate the instruction in the core dump.

    b. Determine the relocation factor of the program from the linkage editor map. Subtract the relocation factor from the address of the invalid instruction.

    c. The address that results may be located in the procedure division map generated by the MAP option. (The coding shown at this location of the map should correspond to the instruction located in step one.)

    d. Preceding the address and code found in step three, find the sequence number of the corresponding COBOL statement in the listing and the number of the element in the sentence that generated the code.

2. Be sure the COBOL statement is coded properly.

3. If the statement is coded properly, go back to the core dump and determine the type of PC.

    a. If it is a data exception, the programmer will probably find that the instruction is a decimal instruction, and that one of the fields either will not have a valid sign or will contain digits other than 0 to 9. To determine this, it will be necessary to find the fields in core storage. Inspect bits 4 through 7 of the low-order byte for a valid sign (A through F). If one is not present, this is the cause of the PC.

       If one or both of the fields being operated on are defined as external decimal, the programmer will find one or more pack instructions immediately ahead of the PC instruction. From these determine the address of the external decimal field that generated the invalid sign. Several common causes of data exceptions are given in "Errors Caused by Invalid Data."

    b. If it is a protection exception, one possible cause is that a base register used in the instruction has not been initialized. Base registers in COBOL are initialized at different times. For input files, the register is not initialized until the first <u>successful</u> read; it is not initialized when the file is opened. For output files, the registers are initialized during the processing of the OPEN statement. When faced with a protection exception, the programmer should go to the COBOL source program to ascertain that no data has been moved prior to the time when base registers are initialized.

c. If an addressing or specification exception occurs, the programmer may find upon inspection (but not always) that registers have been unexpectedly modified and the problem becomes one of finding out how. Two possible approaches are:

(1) Check the addresses in registers 14 and 15 against the address of the PC instruction. If the address of the PC instruction is equal to or slightly larger than the address in register 15, the address probably is in a subroutine, and the address in register 14 should be the return address. A BAL or BALR instruction probably will precede the return address. The programmer should look for this particularly when the problem is not with a COBOL statement. If the PC instruction has an address equal to or a bit larger than the address in register 14, then the programmer probably has just returned from a subroutine, and register 15 should still be pointing to the entry address of the subroutine. The programmer should check the coding to see if this could reasonably be so, and check the entry points listed on the linkage editor map. If this approach bears further action, a listing of the subroutine would be needed or the instructions from the dump must be interpreted.

(2) If the foregoing step does not locate the error, the programmer should check back through the dump to see what exists between the PC instruction and the last unconditional branch in order to determine the possible course of events.

The sample COBOL program ABEND and its output, shown in Figure 58 for a nonsegmented program and in Figure 59 for a segmented program, illustrates in detail the way in which an object code listing, a cross-reference table, and an abnormal termination dump can be used together to debug a program. The circled numerals in the figures are cited in the associated text. Note that all values are expressed in hexadecimal format unless otherwise indicated.

In both examples of the ABEND program, the completion code in the dump, (1), indicates the condition causing the abnormal termination. If the system part of the code is nonzero, the explanation can be found in the publication <u>IBM OS Programmer's Guide to Debugging</u>. In the program ABEND, the completion code is 0C7; invalid data is the reason for termination.

<u>Debugging a Nonsegmented Program</u>: Suggested below are general procedures for locating and correcting the source statement responsible for abnormal termination.

1. The PROGRAM INTERRUPTION (DATA) AT LOCATION hhhhhh entry, (2), gives the hexadecimal address of the instruction following the instruction that initiated the interrupt and caused the dump. This address can be used to determine the relative location of the instruction in the load module (see item 4 below). In the example, the address is B52D0.

2. To determine the main storage area occupied by the load module, add the total length of the module, in hexadecimal format, to its load address. The load address can be obtained from the USE/EP entry, (3), of the first ACTIVE RBS (Request Blocks) specification. The last six digits of this entry are the address of the entry point (INIT1) in the COBOL program. In this case, the address is B5020 in hexadecimal format.

   The total length of the load module is indicated in the TOTAL LENGTH entry, (4), in the linkage editor output (340, in the example). The highest location in the load module is:

   B5020 + 540 = B5560

   Thus, the range is from B5020 to B5560. Since the address B52D0 falls within this range, the instruction initiating the dump must be within the load module.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ABEND.
REMARKS.
        THIS IS A PROGRAM TO ILLUSTRATE THE ABNORMAL
        TERMINATION OF A NONSEGMENTED PROGRAM.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-360-H50.
OBJECT-COMPUTER.  IBM-360-H50.

DATA DIVISION.

WORKING-STORAGE SECTION.
01  RECORDA.
    02  A  PICTURE S9(4) VALUE 1234.
    02  B REDEFINES A PICTURE S9(7) COMPUTATIONAL-3.  8
PROCEDURE DIVISION.
    COMPUTE B = B + 1.  8
    STOP RUN.
```

INTRNL NAME	LVL	SOURCE NAME	BASE	DISPL	INTRNL NAME	DEFINITION	USAGE	R	O	Q	M
DNM=1-032	01	RECORDA	BL=1	000	DNM=1-032	DS OCL4	GROUP				
DNM=1-052	02	A	BL=1	000	DNM=1-052	DS 4C	DISP-NM				
DNM=1-063	02	B	BL=1	000	DNM=1-063	DS 4P	COMP-3	R			

MEMORY MAP

TGT	00090
SAVE AREA	00090
SWITCH	000D8
TALLY	000DC
SORT SAVE	000E0
ENTRY-SAVE	000E4
SORT CORE SIZE	000E8
RET CODE	000EC
SORT RET	000EE
WORKING CELLS	000F0
SORT FILE SIZE	00220
SORT MODE SIZE	00224
PGT-VN TBL	00228
TGT-VN TBL	0022C
VCONPTR	00230
LENGTH OF VN TBL	00234
LABEL RET	00236
CURRENT PRIORITY	00237
UNUSED	00238
INIT1 ADCON	00240
DEBUG TABLE PTR	00244
UNUSED	00248
OVERFLOW CELLS	0024C
BL CELLS	0024C
DECBADR CELLS	00250
TEMP STORAGE	00250
TEMP STORAGE-2	00250
TEMP STORAGE-3	00250
TEMP STORAGE-4	00250
BLL CELLS	00250
VLC CELLS	00258
SBL CELLS	00258
INDEX CELLS	00258
SUBADR CELLS	00258
ONCTL CELLS	00258
PFMCTL CELLS	00258
PFMSAV CELLS	00258
VN CELLS	00258
SAVE AREA =2	00258
SAVE AREA =3	00258
XSASW CELLS	00258
XSA CELLS	00258
PARAM CELLS	00258
RPTSAV AREA	00258
CHECKPT CTR	00258
VCON TBL	00258

LITERAL POOL (HEX)

00270 (LIT+0)   1C

PGT	00260
OVERFLOW CELLS	00260
VIRTUAL CELLS	00260
PROCEDURE NAME CELLS	00268
GENERATED NAME CELLS	00268
DCB ADDRESS CELLS	0026C
VNI CELLS	0026C
LITERALS	00270
DISPLAY LITERALS	00271

Figure 58. Nonsegmented COBOL Program with Abnormal Termination Dump (Part 1 of 3)

```
 REGISTER ASSIGNMENT

  REG 6   BL  =1

WORKING-STORAGE STARTS AT LOCATION 00088 FOR A LENGTH OF 00008.

 18      COMPUTE     000272 8                           START     EQU    *
                     000272  FA 30 6 000 C 010                  7 AP     000(4,6),010(1,12)     DNM=1-63       LIT+0
 17      STOP        000278                             GN=01     EQU    *
                     00027B  58 F0 C 004                          L      15,004(0,12)           V(ILBOSTP1)
                     00027C  07 FF                                BCR    15,15
                     00027E  50 D0 5 008                INIT2     ST     13,008(0,5)
                     000282  50 50 D 004                          ST     5,004(0,13)
                     000286  50 E0 D 054                          ST     14,054(0,13)
                     00028A  94 EF D 048                          NI     048(13),X'EF'          SWT+0
                     00028E  58 F0 C 000                          L      15,000(C,12)           VIR=1
                     000292  05 EF                                BALR   14,15
                     000294  50 10 D 188                          ST     1,188(0,13)
                     000298  12 00                                LTR    0,0
                     00029A  07 89                                BCR    8,9
                     00029C  96 10 D 048                          OI     048(13),X'10'          SWT+0
                     0002A0  05 F0                      INIT3     BALR   15,0
                     0002A2  91 20 D 048                          TM     048(13),X'20'          SWT+0
                     0002A6  47 E0 F 016                          BC     14,016(0,15)
                     0002AA  98 2D B 050                          LM     2,13,050(11)
                     0002AE  58 00 B 048                          L      0,048(0,11)
                     0002B2  58 E0 D 054                          L      14,054(0,13)
                     0002B6  07 FE                                BCR    15,14
                     0002B8  96 20 D 048                          OI     048(13),X'20'          SWT+0
                     0002BC  41 60 0 004                          LA     6,004(0,0)
                     0002C0  41 10 C 008                          LA     1,008(0,12)            GN=01
                     0002C4  41 70 C 010                          LA     7,010(0,12)            LIT+0
                     0002C8  06 70                                BCTR   7,0
                     0002CA  05 50                                BALR   5,0
                     0002CC  58 40 1 000                          L      4,000(0,1)
                     0002D0  1E 4B                                ALR    4,11
                     0002D2  50 40 1 000                          ST     4,000(0,1)
                     0002D6  87 16 5 000                          BXLE   1,6,000(5)
                     0002DA  41 80 D 1BC                          LA     8,1BC(0,13)            OVF=1
                     0002DE  41 70 D 1BF                          LA     7,1BF(0,13)            TS=01-1
                     0002E2  05 10                                BALR   1,0
                     0002E4  58 00 8 000                          L      0,000(0,8)
                     0002E8  1E 0B                                ALR    0,11
                     0002EA  50 00 8 000                          ST     0,000(0,8)
                     0002EE  87 86 1 000                          BXLE   8,6,000(1)
                     0002F2  58 60 D 1BC                          L      6,1BC(0,13)            BL  =1
                     0002F6  58 E0 D 054                          L      14,054(0,13)
                     0002FA  07 FE                                BCR    15,14
                     000000  90 EC D 00C                INIT1     STM    14,12,00C(13)
                     000004  18 5D                                LR     5,13
                     000006  05 F0                                BALR   15,0
                     000008  45 80 F 010                          BAL    8,010(0,15)
                     00000C  C1C2C5D5C4404040                     DC     X'C1C2C5D5C4404040'
                     000014  C1D5E2C3                             DC     X'C1D5E2C3'
                     000018  07 00                                BCR    0,0
                     00001A  98 9F F 024                          LM     9,15,024(15)
                     00001E  07 FF                                BCR    15,15
                     000020  96 02 1 034                          OI     034(1),X'02'
                     000024  07 FE                                BCR    15,14
                     000026  41 F0 0 001                          LA     15,001(0,0)
                     00002A  07 FE                                BCR    15,14
                     00002C  000002A0                             ADCON  L4(INIT3)
                     000030  00000000                             ADCON  L4(INIT1)
                     000034  00000000                             ADCON  L4(INIT1)
                     000038  00000260                             ADCON  L4(PGT)
                     00003C  00000090                             ADCON  L4(TGT)
                     000040  00000272                             ADCON  L4(START)
                     000044  0000027E                             ADCON  L4(INIT2)
                     000048                                       DS     15F


*STATISTICS*      SOURCE RECORDS =    19     DATA DIVISION STATEMENTS =      3     PROCEDURE DIVISION STATEMENTS =     2
*OPTIONS IN EFFECT*     SIZE =   81920  BUF =     2768  LINECNT = 57  SPACE1, FLAGW,    SEQ,    SOURCE
*OPTIONS IN EFFECT*       DMAP,    PMAP, NOCLIST, NOSUPMAP, NOXREF, NOSXREF,   LOAD, NODECK, APOST, NOTRUNC, NOFLOW
*OPTIONS IN EFFECT*     NOTERM, NONUM, NOBATCH, NONAME, COMPILE=01, NOSTATE,  LIB, VERB, ZWB, SYST


  F88-LEVEL LINKAGE EDITOR OPTIONS SPECIFIED XREF,LIST,LET
          VARIABLE OPTIONS USED - SIZE=(92160,3192)                  DEFAULT OPTION(S) USED

                                                CROSS REFERENCE TABLE

  CONTROL SECTION                          ENTRY

    NAME     ORIGIN  LENGTH                  NAME   LOCATION     NAME   LOCATION     NAME   LOCATION     NAME   LOCATION

  ABEND         00      2FC 5
  ILBOSTP0*    300       30
                                           ILBOSTP1     31C

  LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION              LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION

     260          ILBOSTP0        ILBOSTP0                        264          ILBOSTP1        ILBOSTP0

 ENTRY ADDRESS        00
 TOTAL LENGTH        340 4

****RUN        DOES NOT EXIST BUT HAS BEEN ADDED TO DATA SET
```

Figure 58. Nonsegmented COBOL Program with Abnormal Termination Dump (Part 2 of 3)

```
 * ABDUMP REQUESTED *

JOB ABEND           STEP GO                  TIME 013907   DATE 71097

COMPLETION CODE     SYSTEM = 0C7 (1)

PROGRAM INTERRUPTION (DATA) AT LOCATION 014292

INTERRUPT AT 014298 (2)

PSW AT ENTRY TO ABEND  FF15000D C0014298

TCB  0053D8     RB    0006C080   PIE      00000000   DEB  0006CD04   TIOT 0006CF10   CMP   800C7000   TPN  00000000
                MSS   00005460   PK/FLG 10910408   FLG  000002F8   LLS  00000000   JLB   00000000   JSE  00000000
                FSA   1806CFB0   TCB      00000000   TME  00005488   PIB  F0009FC0   NSTAF 00000000   TCT  0000087C
                USER  00000000


ACTIVE RBS
                                                      (3)
PRB    014000   NM RUN          SZ/STAB 006C00C0    USE/EP 00014020   PSW FF15000D C0014298    Q 000000   WT/LNK 000053D8

SVRB   06CDE0   NM SVC-401C     SZ/STAB 0012D172    USE/EP 000041A8   PSW FF040033 40004370    Q E803E8   WT/LNK 00014000
                RG 0-7   000140A8     50014304      0006D000     0006D000     00014020     500142EC     000140A8     0001426F
                RG 8-15  00014270     000142C0      00014020     00014020     00014280     00014080     00014292     500142C2

SVRB   06CD80   NM SVC-105A     SZ/STAB 000CD172    USE/EP 000041A8   PSW FF040230 8000043C    Q C803C8   WT/LNK 0006CDE0
                RG 0-7   00014360     000143B8      000012C0     400041AA     00000000     00000000     00014360     8000435E
                RG 8-15  0000545E     0001442E      00060000     000053D8     000053D8     000143C8     600043FA     00014292

P/P STORAGE BOUNDARIES 00014000 TO 00060000

FREE AREAS      SIZE

  014668       000582A8
  06CD50       00000030
  06CEF8       00000028

SAVE AREA TRACE

RUN       WAS ENTERED

SA    06CFB0   WD1 00000000    HSA 00000000    LSA 00014080    RET 000064CC    EPA 50014020    R0  00000068
               R1  0006CFF8    R2  0006D000    R3  0006D000    R4  0006CF70    R5  00000060    R6  000053D8
               R7  0006CC30    R8  0006CF78    R9  00000000    R10 0006CFB0    R11 0006CFF8    R12 600143EA

SA    014080   WD1 00000000    HSA 0006CFB0    LSA 08000026    RET 00000001    EPA 00000001    R0  00000000
               R1  00000000    R2  00000000    R3  80000000    R4  00000000    R5  00000000    R6  00000000
               R7  00000000    R8  00000000    R9  00000001    R10 00004000    R11 00000001    R12 42000001



REGS AT ENTRY TO ABEND

FL.PT.REGS 0-6        00.000000 00000000      00.000000 00000000      00.000000 00000000      00.000000 00000000

REGS 0-7            000140A8   50014304   0006D000   0006D000            00014020   500142EC   000140A8   0001426F
REGS 8-15           00014270   000142C0   00014020   00014020            00014280   00014080   00014292   500142C2

P/P STORAGE

014000    D9E4D540 40404040 006C00C0 00014020     FF15000D C0014298 00000000 000053D8    *RUN      ...... .................Q*
014020    90ECD00C 185D05F0 4580F010 C1C2C5D5     C4404040 C1D5E2C3 0700989F F02407FF    *.......0..0.ABEND   ANSC....0...*
014040    96021034 07FE41F0 000107FE 000142C0     00014020 00014020 00014280 00014080    *.......0.......... ... ....... .*
014060    00014292 0001429E 00000000 00000000     00000000 00000000 00000000 00000000    *................................*
014080    00000000 00000000 00000000 00000000     00000000 00000000 00000000 00000000    *................................*
0140A0    00000000 00000000 F1F2F3C4 00000000     00000000 0006CFB0 08000026 00000001    *........123D....................*
0140C0    00000001 00000000 00000000 00000000     80000000 00000000 00000000 00000000    *................................*
```

Figure 58. Nonsegmented COBOL Program with Abnormal Termination Dump (Part 3 of 3)

3. To determine the relative location within the load module of the instruction indicated in the INTERRUPTION entry, subtract the load address from the address of the instruction. In the example, this becomes:

   B52D0 - B5020 = 2B0

4. To determine whether or not the instruction occurred in the object module generated for the program, compare its relative location (2B0) with the total length, (4), of the object module. If the relative location were greater than the size of the object module, then the error would not be part of this program. A relative location between the size of the program, (5), and the total length would indicate that the abnormal termination had occurred in one of the COBOL library subroutines. Such an error could be located by comparing the relative location with the relative origin of the subroutines. In this example, 2B0 is less than the program size (356), so the instruction occurred in the main program.

5. To determine whether or not the abnormal termination occurred in one of the object code instructions generated as a result of a statement in the Procedure Division of the source program, compare its relative location with the relative location of the first generated instruction in the Procedure Division, (6). In this example, the relative location of the instruction is greater than that of the first generated instruction (2B0 > 2AA) and so it can be found by locating the corresponding relative location. The immediately preceding object code instruction then is the instruction that initiated the dump, (7). In this example, it is an instruction generated as a result of a COMPUTE statement. Checking back to the source program listing, the corresponding statement, (8), is located and 'B' is seen to be the data-name that caused the trouble. Data item B is defined in the Data Division, (9), as a COMPUTATIONAL-3 or internal decimal item, but the value at B is there as a result of a VALUE clause for A, the item that B redefines. This value is in external decimal format since there is no USAGE clause specified. The configuration of A is invalid for B and results in an interrupt.

<u>Determining the Location of an ABEND When Running Dynamically</u>: When running dynamically, the programmer should do the following to determine whether the abend occurred in the main program.

1. The compiler produces a Load List that contains the COBOL subroutine library names and the addresses used in the program. These are anything beginning with ILBO. (A) Figure 59 is a Load List, the letters corresponding to the explanation in the text. The programmer is particularly interested in any ILBO subroutine that does not end in a zero, such as ILBORNT, ILBOREC, ILBODSP, etc.

2. In this case, the abend has occurred at 441CC. To determine whether this is within the main program, go to the Load List, and look for the subroutine with its address closest to that of the abend. ILBOREC (B) has an address of 043E18.

3. Look below to the second part of the Load List. This contains the length of the subroutines that begin at the address specified above. In this case at 043E18, under the LN column, the length of the subroutine is 9E8 (C). Adding the length of the subroutine 9E8 to the starting address 043E18, results in a number falling within the confines of the main program.

4. After this is determined, the programmer continues his debugging in the specified manner.

```
    NE 00023890    RSP-CDE 02023928       NE 00023898    RSP-CDE 01028308       NE 000238A0    RSP-CDE 01028238
    NE 000238A8    RSP-CDE 01028208       NE 00023B20    RSP-CDE 010281D8       NE 000241B0    RSP-CDE 01023B38
    NE 00024388    RSP-CDE 01023D20       NE 000244A8    RSP-CDE 01023DF8       NE 000245B0    RSP-CDE 01028308
    NE 000246E8    RSP-CDE 01028268       NE 00024960    RSP-CDE 010281A8       NE 00024B80    RSP-CDE 01028338
    NE 00025368    RSP-CDE 01023E10       NE 000253A8    RSP-CDE 01023E28       NE 000253C0    RSP-CDE 010247C8
    NE 000261D8    RSP-CDE 010248A8       NE 00000000    RSP-CDE 01024968

CDE

  0253D8      ATR1 0B   NCDE 000000   ROC-RB 00025200   NM RUN        USE 01   EPA 041698   ATR2 20   XL/MJ 0253C8
  023928      ATR1 30   NCDE 023A78   ROC-RB 00000000   NM IGC0A05A   USE 02   EPA 04C960   ATR2 28   XL/MJ 023908

                                                                                                    PAGE 0002
  028308      ATR1 B0   NCDE 028338   ROC-RB 00000000   NM IGG019CD   USE 02   EPA 07E130   ATR2 20   XL/MJ 0282F8
  028238      ATR1 B0   NCDE 028268   ROC-RB 00000000   NM IGG019CJ   USE 02   EPA 07DD10   ATR2 20   XL/MJ 028228
  028208      ATR1 B0   NCDE 028238   ROC-RB 00000000   NM IGG019BA   USE 02   EPA 07DB80   ATR2 20   XL/MJ 0281F8
  0281D8      ATR1 B0   NCDE 028208   ROC-RB 00000000   NM IGG019BB   USE 02   EPA 07DA58   ATR2 20   XL/MJ 0281C8
  023B38      ATR1 31   NCDE 023D20   ROC-RB 00000000   NM IGG019RL   USE 01   EPA 04DB48   ATR2 20   XL/MJ 023B28
  023D20      ATR1 31   NCDE 023DF8   ROC-RB 00000000   NM IGG019RG   USE 01   EPA 04DC90   ATR2 20   XL/MJ 023D10
  023DF8      ATR1 33   NCDE 023F10   ROC-RB 00000000   NM ILBORNT    USE 01   EPA 041088   ATR2 20   XL/MJ 023DE8
  028308      ATR1 B0   NCDE 028338   ROC-RB 00000000   NM IGG019CD   USE 02   EPA 07E130   ATR2 20   XL/MJ 0282F8
  028268      ATR1 B0   NCDE 028298   ROC-RB 00000000   NM IGG019CI   USE 02   EPA 07E020   ATR2 20   XL/MJ 028258
  0281A8      ATR1 B0   NCDE 0281D8   ROC-RB 00000000   NM IGG019AI   USE 02   EPA 07D9D8   ATR2 20   XL/MJ 028198
  028338      ATR1 B0   NCDE 028368   ROC-RB 00000000   NM IGG019AP   USE 02   EPA 07E3A0   ATR2 20   XL/MJ 028328
  023E10      ATR1 03   NCDE 0241B8   ROC-RB 00000000   NM ILBOSRV0   USE 01   EPA 04145A   ATR2 20   XL/MJ 0246A8
  023E28      ATR1 37   NCDE 024870   ROC-RB 000241D0   NM ILBOREC0   USE 00   EPA 043E1A   ATR2 10   XL/MJ 0241B8
  0241B8      ATR1 33   NCDE 023E28   ROC-RB 00000000   NM ILBOREC    USE 01   EPA 043E18   ATR2 20   XL/MJ 024798  <-- A
  0247C8      ATR1 37   NCDE 0248C0   ROC-RB 000241D0   NM ILBODSP0   USE 00   EPA 043062   ATR2 10   XL/MJ 024870
  024870      ATR1 33   NCDE 0247C8   ROC-RB 00000000   NM ILBODSP    USE 01   EPA 043060   ATR2 20   XL/MJ 0247B8
  0248A8      ATR1 37   NCDE 024B88   ROC-RB 000241D0   NM ILBONTR0   USE 00   EPA 042872   ATR2 10   XL/MJ 0248C0
  0248C0      ATR1 33   NCDE 0248A8   ROC-RB 00000000   NM ILBONTR    USE 01   EPA 042872   ATR2 20   XL/MJ 024898
  024968      ATR1 16   NCDE 0253D8   ROC-RB 000241D0   NM ILBOCOM0   USE 00   EPA 0415D8   ATR2 10   XL/MJ 024B88
  024B88      ATR1 12   NCDE 024968   ROC-RB 00000000   NM ILBOCOM    USE 01   EPA 0415D8   ATR2 20   XL/MJ 024DB0

                                                                       B
XL                                                   LN          ADR        LN          ADR          LN          ADR

  0253C8   SZ 00000010   NO 00000001           80001168    00041698
  023908   SZ 00000010   NO 00000001           800006A0    0004C960
  0282F8   SZ 00000010   NO 00000001           80000270    0007E130
  028228   SZ 00000010   NO 00000001           80000110    0007DD10
  0281F8   SZ 00000010   NO 00000001           80000190    0007DB80
  0281C8   SZ 00000010   NO 00000001           80000128    0007DA58
  023B28   SZ 00000010   NO 00000001           80000148    0004DB48
  023D10   SZ 00000010   NO 00000001           80000B70    0004DC90
  023DE8   SZ 00000010   NO 00000001           800003D0    00041088
  0282F8   SZ 00000010   NO 00000001           80000270    0007E130
  028258   SZ 00000010   NO 00000001           80000110    0007E020
  028198   SZ 00000010   NO 00000001           80000080    0007D9D8
  028328   SZ 00000010   NO 00000001           80000100    0007E3A0
  0246A8   SZ 00000010   NO 00000001           80000180    00041458
  024798   SZ 00000010   NO 00000001           800009F8    00043E18  <-- C
  0247B8   SZ 00000010   NO 00000001           800007A0    00043060
  024898   SZ 00000010   NO 00000001           80000790    00042870
  024DB0   SZ 00000010   NO 00000001           800000C0    000415D8

DEB

023940                              00000B94 00000B94   C0000B94 0007DD10 00000B94 00000000   *................................*
023960   00000306 00002BE0 11000000 04023E68   10023DC8 88000000 8F000000 01000000   *..................H.............*
023980   1B000000 EF04F2A0 04023948 1000296C   00000010 00000014 00090032 00010001   *......2.........................*
0239A0   00000000 00000000 0000007D C2C2C2C1   C3D1C3C4 00000000 00000000 00000000   *............BBBACJCD............*
0239C0   00000000 00000000 00000000 03000065                                         *..................8301C.........*

DEB

023DA0                                                          00000C00 00000C00   *...0...............0...........*
```

Figure 59. Load List

Debugging a Segmented Program: Below are the recommended steps for identifying the segment executing at the time of the error causing abnormal termination.

1. The PROGRAM INTERRUPTION (DATA) AT LOCATION hhhhhh entry, 2, gives the hexadecimal address of the instruction following the instruction that initiated the interrupt and caused the dump. This address can be used to determine the relative location of the instruction in the load module (see item 3 below). In the example (Figure 60), the address is 14BCE.

2. To determine the main storage load address of the load module, subtract the length of the segment table ($SEGTAB) from the entry point address. The load address can be obtained from the USE/EP entry, 3, of the first active RBS (Request Blocks) specification. The last six digits of this entry are the address of the entry point (INIT1) in the COBOL program. In this case, the address is 14050.

   The length of the segment table, 4, in the linkage editor output is 20 in this example. The load address of the module is:

   14050 - 20 = 14030.

3. To determine the relative location of the instruction indicated in the INTERRUPTION entry, subtract the load address from the address of the instruction. In the example, this becomes:

   14BCE - 14030 = B9E.

4. To determine whether or not the abnormal termination occurred in the object module generated for the program, compare its relative location (B66) with the starting address of each of the modules following $SEGTAB. The last of these modules ($ENTAB) begins at B70 and ends at B87. Because the location B9E is beyond the range of locations in the main program and the COBOL subroutines, the instruction initiating the dump would appear to be in another program. However, another check must be made. If the location B9E is less than the TOTAL LENGTH, 5, but greater than the end of $ENTAB, it is in the transient area, 6.

5. Subtract the starting address of the transient area from the relative location of the abnormally terminating instruction, as below:

   B9E - B88 = 16

   Therefore, 16 is the relative address of the instruction immediately following the one responsible for the abnormal termination. It remains to identify the segment of the program in which this instruction occurs.

6. To find location 16 in the segment being executed at the time of the abnormal termination, compute the sum of the location of CURSEGM, 7, and the load address (computed in item 3).

   14030 + B29 = 14B59

   The hexadecimal contents of the byte indicated, 8, identify this segment. In this example, the hexadecimal value is 32 (which is equivalent to the decimal value 50), so the priority number of the current segment is 50.

7. In the section of priority 50 (SEC50), the instruction immediately following the one that caused the abnormal termination is a Load, 9, so that the instruction causing the data interrupt was the Add Decimal instruction at location 10. In this example, as in the nonsegmented program ABEND (Figure 58), this instruction was generated as the result of a COMPUTE statement.

Finding Data Records in an Abnormal Termination Dump

The glossary, listed when the DMAP option is specified, contains information about all data-names described in the COBOL source program. The location assigned to a given data-name may be found by using the BL number and displacement specified for that entry in the glossary, and then locating the appropriate BL cell in the TGT. The hexadecimal sum of the glossary displacement and the contents of the cell should give the relative address of the area desired. This can be converted to an absolute address as described in the text associated with Figure 59 (for a nonsegmented program) and Figure 60 (for a segmented program).

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ABEND.
REMARKS.
       THIS IS A PROGRAM TO ILLUSTRATE THE ABNORMAL
       TERMINATION OF A SEGMENTED PROGRAM.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-360-H50.
OBJECT-COMPUTER.  IBM-360-H50.

DATA DIVISION.

WORKING-STORAGE SECTION
01  RECORDA
    02  A  PICTURE S9(4) VALUE 1234.
    02  B REDEFINES A PICTURE S9(7) COMPUTATIONAL-3.
PROCEDURE DIVISION.
SEC10 SECTION 10.
    DISPLAY 'START TEST'.
BEGIN.
    READY TRACE.
SEC50 SECTION 50.
    COMPUTE B = B + 1.
    STOP RUN.
```

```
INTRNL NAME   LVL SOURCE NAME                              BASE   DISPL  INTRNL NAME   DEFINITION   USAGE      R  O  Q  M
DNM=1-080     01  RECORDA                                  BL=1    000   DNM=1-080     DS 0CL4      GROUP
DNM=1-100     02  A                                        BL=1    000   DNM=1-100     DS 4C        DISP-NM
DNM=1-111     02  B                                        BL=1    000   DNM=1-111     DS 4P        COMP-3     R
```

```
          MEMORY MAP

     TGT                       00090

  SAVE AREA                    00090
  SWITCH                       000D8
  TALLY                        000DC
  SORT SAVE                    000E0
  ENTRY-SAVE                   000E4
  SORT CORE SIZE               000E8
  RET CODE                     000EC
  SORT RET                     000EE
  WORKING CELLS                000F0
  SORT FILE SIZE               00220
  SORT MODE SIZE               00224
  PGT-VN TBL                   00228
  TGT-VN TBL                   0022C
  VCONPTR                      00230
  LENGTH OF VN TBL             00234
  LABEL RET                    00236
  CURRENT PRIORITY             00237
  UNUSED                       00238
  INIT1 ADCON                  00240
  DEBUG TABLE PTR              00244
  UNUSED                       00248
  OVERFLOW CELLS               0024C
  BL CELLS                     0024C
  DECBADR CELLS                00250
  TEMP STORAGE                 00250
  TEMP STORAGE-2               00250
  TEMP STORAGE-3               00250
  TEMP STORAGE-4               00250
  BLL CELLS                    00250
  VLC CELLS                    00258
  SBL CELLS                    00258
  INDEX CELLS                  00258
  SUBADR CELLS                 00258
  ONCTL CELLS                  00258
  PFMCTL CELLS                 00258
  PFMSAV CELLS                 00258
  VN CELLS                     00258
  SAVE AREA =2                 00258
  SAVE AREA =3                 00258
  XSASW CELLS                  00258
  XSA CELLS                    00258
  PARAM CELLS                  00258
  RPTSAV AREA                  00258
  CHECKPT CTR                  00258
  VCON TBL                     00258


     PGT                       00268

  OVERFLOW CELLS               00268
  VIRTUAL CELLS                00268
  PROCEDURE NAME CELLS         00278
  GENERATED NAME CELLS         0027C
  DCB ADDRESS CELLS            00280
  VNI CELLS                    00280
  LITERALS                     00280
  DISPLAY LITERALS             00281
```

Figure 60. Segmented COBOL Program with Abnormal Termination Dump (Part 1 of 4)

```
REGISTER ASSIGNMENT

  REG 6    BL =1

WORKING-STORAGE STARTS AT LOCATION 00088 FOR A LENGTH OF 00008.

                          SEGMENT OF PTY 50

 22               000000                      PN=01   EQU   *
                  000000  58 F0 C 004                 L     15,004(0,12)        V(ILBODSP0)
                  000004  05 1F                       BALR  1,15
                  000006  000140                      DC    X'000140'
                  000009  05F2F240404040              DC    X'05F2F240404040'
 23               000010  FA 30 6 000 C 018           AP    000(4,6),018(1,12)  DNM=1-111     LIT+0
 24               000016                      GN=01   EQU   *
                  000016  58 F0 C 00C               1 L     15,00C(0,12)        V(ILBOSTP1)
                  00001A  07 FF                       BCR   15,15

                          ROOT SEGMENT

LITERAL POOL (HEX)

00280 (LIT+0)      1C


    DISPLAY LITERALS (BCD)

00281 (LIL+1)     'START TEST'

 18               00028C                      START   EQU   *
                  00028C  58 F0 C 004                 L     15,004(0,12)        V(ILBODSP0)
                  000290  05 1F                       BALR  1,15
                  000292  000140                      DC    X'000140'
                  000295  05F1F840404040              DC    X'05F1F840404040'
 19               00029C  58 F0 C 004                 L     15,004(0,12)        V(ILBODSP0)
                  0002A0  05 1F                       BALR  1,15
                  0002A2  0001                        DC    X'0001'
                  0002A4  10                          DC    X'10'
                  0002A5  00000A                      DC    X'00000A'
                  0002A8  0C000019                    DC    X'0C000019'         LIT+1
                  0002AC  0000                        DC    X'0000'
                  0002AE  FFFF                        DC    X'FFFF'
 20               0002B0  58 F0 C 004                 L     15,004(0,12)        V(ILBODSP0)
                  0002B4  05 1F                       BALR  1,15
                  0002B6  000140                      DC    X'000140'
                  0002B9  05F2F040404040              DC    X'05F2F040404040'
 21               0002C0  96 40 D 048                 OI    048(13),X'40'       SWT+0
                  0002C4  58 00 C 010                 L     0,010(0,12)         PN=01
                  0002C8  13 00                       LCR   0,0
                  0002CA  58 F0 C 008                 L     15,008(0,12)        V(ILBOSGM0)
                  0002CE  05 EF                       BALR  14,15
                  0002D0  50 D0 5 008         INIT2   ST    13,008(0,5)
                  0002D4  50 50 D 004                 ST    5,004(0,13)
                  0002D8  50 E0 D 054                 ST    14,054(0,13)
                  0002DC  94 EF D 048                 NI    048(13),X'EF'       SWT+0
                  0002E0  58 F0 C 000                 L     15,000(0,12)        VIR=1
                  0002E4  05 EF                       BALR  14,15
                  0002E6  50 10 D 1B8                 ST    1,1B8(0,13)
                  0002EA  12 00                       LTR   0,0
                  0002EC  07 89                       BCR   8,9
                  0002EE  96 10 D 048                 OI    048(13),X'10'       SWT+0
                  0002F2  05 F0               INIT3   BALR  15,0
                  0002F4  91 20 D 048                 TM    048(13),X'20'       SWT+0
                  0002F8  47 E0 F 016                 BC    14,016(0,15)
                  0002FC  98 2D B 050                 LM    2,13,050(11)
                  000300  58 00 B 048                 L     0,048(0,11)
                  000304  58 E0 D 054                 L     14,054(0,13)
                  000308  07 FE                       BCR   15,14
                     .       .                         .       .
                     .       .                         .       .
                     .       .                         .       .
                  00033A  87 16 5 000                 BXLE  1,6,000(5)
                  00033E  41 80 D 1BC                 LA    8,1BC(0,13)         OVF=1
                  000342  41 70 D 1BF                 LA    7,1BF(0,13)         TS=01-1
                  000346  05 10                       BALR  1,0
                  000348  58 00 8 000                 L     0,000(0,8)
                  00034C  1E 0B                       ALR   0,11
                  00034E  50 00 8 000                 ST    0,000(0,8)
                  000352  87 86 1 000                 BXLE  8,6,000(1)
                  000356  58 60 D 1BC                 L     6,1BC(0,13)         BL =1
                  00035A  58 E0 D 054                 L     14,054(0,13)
                  00035E  07 FE                       BCR   15,14
                  000000  90 EC D 00C         INIT1   STM   14,12,00C(13)
                  000004  18 5D                       LR    5,13
                  000006  05 F0                       BALR  15,0
                  000008  45 80 F 010                 BAL   8,010(0,15)
                  00000C  C1C2C5D5C4404040            DC    X'C1C2C5D5C4404040'
                  000014  C1D5E2C3                    DC    X'C1D5E2C3'
                  000018  07 00                       BCR   0,0
                  00001A  98 9F F 024                 LM    9,15,024(15)
                  00001E  07 FF                       BCR   15,15
                  000020  96 02 1 034                 OI    034(1),X'02'
                  000024  07 FE                       BCR   15,14
                  000026  41 F0 0 001                 LA    15,001(0,0)
                  00002A  07 FE                       BCR   15,14
                  00002C  000002F2                    ADCON L4(INIT3)
                  000030  00000000                    ADCON L4(SEGMT)
                  000034  00000000                    ADCON L4(INIT1)
                  000038  00000268                    ADCON L4(PGT)
                  00003C  00000090                    ADCON L4(TGT)
                  000040  0000028C                    ADCON L4(START)
                  000044  000002D0                    ADCON L4(INIT2)
                  000048                              DS    15F
```

Figure 60. Segmented COBOL Program with Abnormal Termination Dump (Part 2 of 4)

```
*STATISTICS*     SOURCE RECORDS =      24      DATA DIVISION STATEMENTS =        3      PROCEDURE DIVISION STATEMENTS =        4
*OPTIONS IN EFFECT*      SIZE =    81920  BUF =      2768  LINECNT = 57  SPACE1,  FLAGW,    SEQ,     SOURCE
*OPTIONS IN EFFECT*         DMAP,     PMAP,  NOCLIST,  NOSUPMAP,  NOXREF,  NOSXREF,    LOAD,  NODECK,  APOST,  NOTRUNC,  NOFLOW
*OPTIONS IN EFFECT*      NOTERM,  NONUM,  NOBATCH,  NONAME,  COMPILE=01,  NOSTATE,  LIB,  VERB,  ZWB,  SYST


 F88-LEVEL LINKAGE EDITOR OPTIONS SPECIFIED LIST,LET,XREF,OVLY
              VARIABLE OPTIONS USED - SIZE=(92160,8192)                 DEFAULT OPTION(S) USED
IEW0000        INSERT ABEND
IEW0000        OVERLAY A
IEW0000        INSERT ABEND50
IEW0000        ENTRY ABEND

                                            CROSS REFERENCE TABLE

  CONTROL SECTION                          ENTRY

    NAME    ORIGIN  LENGTH  SEG. NO.      NAME    LOCATION       NAME    LOCATION       NAME    LOCATION       NAME    LOCATION

 $SEGTAB      00       20      1 (4)
  ABEND       20      360      1
  ILBODSP0*  380      6CA      1
  ILBOSGM0*  A50       DB      1
                                        CURSEGM      B29 (7)
  ILBOSTP0*  B30       3D      1
                                        ILBOSTP1     B4C
 $ENTAB      B70       18      1


  LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION  SEG. NO.    LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION  SEG. NO.

      284          ABEND50         ABEND50          2            288          ILBOSTP0        ILBOSTP0          1
      28C          ILBODSP0        ILBODSP0         1            290          ILBOSGM0        ILBOSGM0          1
      294          ILBOSTP1        ILBOSTP0         1             50          ABEND50         ABEND50           2

  CONTROL SECTION                          ENTRY

    NAME    ORIGIN  LENGTH  SEG. NO.      NAME    LOCATION       NAME    LOCATION       NAME    LOCATION       NAME    LOCATION

  ABEND50    B88       1C      2  (6)


  LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION  SEG. NO     LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION  SEG. NO.

 ENTRY ADDRESS      20
 TOTAL LENGTH      BA8 (5)

****RUN        DOES NOT EXIST BUT HAS BEEN ADDED TO DATA SET



 * ABDUMP REQUESTED *

JOB ABEND               STEP GO                 TIME 173122    DATE 71086

COMPLETION CODE         SYSTEM = 0C7 (1)

PROGRAM INTERRUPTION (DATA) AT LOCATION 014BC8

INTERRUPT AT 014BCE (2)

PSW AT ENTRY TO ABEND  FF15000D C0014BCE

TCB  005298    RB   0006C888    PIE     00000000    DEB  0006C80C    TIOT 0006CF10    CMP    800C7000    TRN  00000000
               MSS  00005320    PK/FLG 10910408    FLG  000002F9    LLS  0006CBA0    JLB    00000000    JSE  00000000
               FSA  1406CFB0    TCB     00000000    TME  00005348    PIB  E0009F98    NSTAE 00000000    TCT  00000E0E
               USER 00000000
```

Figure 60. Segmented COBOL Program with Abnormal Termination Dump (Part 3 of 4)

```
ACTIVE RBS

PRB    014000  NM RUN        SZ/STAB 017B00C0   USE/EP 00014050    PSW FF15000D C0014BCE   Q 000000   WT/LNK 00005298

SVRB   06CB10  NM SVC-401C   SZ/STAB 0012D172   USE/EP 000041A8    PSW FF040033 40004370   Q E803E8   WT/LNK 00014000
               RG 0-7   CDFEB448     00014BC8   0006D000    0006D000      32014BCE    5001436E    000140D8    0001429F
               RG 8-15  000142A0     00014342   00014BB8    00014050      000142B8    000140E0    00014BB8    000143E0

SVRB   06C888  NM SVC-105A   SZ/STAB 000CD172   USE/EP 000041A8    PSW FF040235 8000D53C   Q C803C8   WT/LNK 0006CB10
               RG 0-7   00014BD8     00014C30   000012C0    400041AA      00000000    00000000    00014BD8    8000435E
               RG 8-15  0000531E     00014CA6   0006D000    00005298      00005298    00014C40    600043FA    00014BC8

LOAD LIST

LPRB   06CBA8  NM IEWSZOVR   SZ/STAB 00412010   USE/EP 0106CBC8    PSW FF040232 800073F0   Q 000000   WT/LNK 0006CB10


P/P STORAGE BOUNDARIES 00014000 TO 0006D000

FREE AREAS     SIZE

  014EE0     00057590
  06C858     00000030
  06CDB0     00000008
  06CEE8     00000028

SAVE AREA TRACE

RUN      WAS ENTERED

SA    06CFB0   WD1 00000000    HSA 00000000    LSA 000140E0    RET 000064CC    EPA 50014050    R0  00000068



               R1  0006CFF8    R2  0006D000    R3  0006D000    R4  0006CF70    R5  00000060    R6  00005298
               R7  0006CC30    R8  0006CF78    R9  00000000    R10 0006CFB0    R11 0006CFF8    R12 600143EA

RUN      WAS ENTERED VIA CALL

SA    0140E0   WD1 00000000    HSA 0006CFB0    LSA 00000000    RET 00014BB8    EPA 00014A80    R0  5001464C
               R1  9200CCC0    R2  CDFEB448    R3  8F06CE10    R4  0006D000    R5  0006D000    R6  00014BC2
               R7  00000005    R8  0006CA21    R9  00000079    R10 900144B2    R11 6000CD10    R12 0000CF58



REGS AT ENTRY TO ABEND

FL.PT REGS 0-6        00.000000 00000000      00 000000 00000000      00 000000 00000000      00.000000 00000000

REGS 0-7          CDFEB448    00014BC8    0006D000    0006D000                 32014BCE    5001436E    000140D8    0001429F
REGS 8-15         000142A0    00014342    00014BB8    00014050                 000142B8    000140E0    00014BB8    000143E0

P/P STORAGE

014000    D9E4D540 40404040 017B00C0 00014050    FF15000D C0014BCE 00000000 00005298    *RUN      .......................*
014020    00014030 C014CC00 00000500 00000B00    0006CF78 00014020 02020000 00000000    *.. ..................  ........*
014040    00000000 00000000 00000002 01000002    90ECD00C 185D05F0 4580F010 C1C2C5D5    *.................... 0..0 ABEN*
014060    C4404040 C1D5E2C3 0700989F F02407FF    96021034 07FE41F0 000107FE 00014342    *D   ANSC....0......... 0........*
014080    00014BB8 00014050 000142B8 000140E0    000142DC 00014320 00000000 00000000    *...... ...... ................*
0140A0    00000000 00000000 00000000 00000000    00000000 00000000 00000000 00000000    *..............................*
0140C0    00000000 00000000 08000056 00000001    00000001 00000000 F1F2F3C4 00000000    *...................... 123D....*
0140E0    00000000 0006CFB0 00000000 00014BB8    00014A80 5001464C 9200CCC0 CDFEB448    *...............................*
014100    8F06CE10 0006D000 0006D000 00014BC2    00000005 0006CA21 00000079 900144B2    *..............E...............*
014120    6000CD10 0000CF58 7000004B 00000000    00000000 000142DC 00000000 00000000    *...............................*
                 .
                 .
                 .
014AE0    4020F0D8 422D01A7 41000032 19204740    F0B49140 F0DA4710 F084D500 F0D9E000    * .0Q............0...0...0.N.0R..*
014B00    4780F0B4 481D01A4 88100002 585D019C    586D0198 12114780 F0B49180 60004710    *..0.......................0.......*
014B20    F0A8D203 50006000 41550004 41660004    4610F09A 503D000C 98ECD00C 94BFF0DA    *0.K...............0...........0.*
014B40    07FE5420 F0D447F0 F0244898 CDFEB448  8 0000007F 7FFFFFFF 00320088 80004780    *....0M.00.......................*
014B60    47F0F006 B1001B00 9101F03C 071E4100    00019601 F03C50D0 F03807FE 9400F020    *.00.......0.........0...0.....0.*
014B80    58E0F01C 48FD005C 58D0E004 58ED000C    980CD014 07FE5820 000140E0 01B4900F    *..0.......................  .....*
014BA0    47FF000C 00014BB8 02000000 0A2D58FF    000407FF 01014030 58F0C004 051F0001    *.........................0.......*
014BC0    4005F2F2 40404040 FA306000 C01858F0    C00C07FF 00001068 00000100 2D000000    * .22      .......0...............*
014BE0    0007000C 0400724A 00280C9A 00000001    00004000 00000001 04000001 54000000    *................................*
014C00    002C0020 0006C80C 9200D588 0000D518    0B000001 00000372 30040048 4106CDB8    *......H...N...N.................*
014C20    0100DC40 0000DC40 0000007D 00000001    000020BE 00014BD8 00005298 00000000    *.........................Q.......*
014C40    00000000 00057BA8 40404040 40404040    40404040 60014CF6 0000D518 0006C7E8    *........              ...6..N...GY*
014C60    00014C88 00005298 00004539 00014C20    800043AA 00000000 A000448C 00014DB1    *................................*
014C80    00014EE0 00000017 7F000000 40204040    00014BD8 0006C470 0006CDC0 000E0005    *....................Q..D.........*
014CA0    98E0D15C 12EE4780 D096D27C F000D16C    41FF007D 50FD0160 411F007D 191047C0    *..J.......K.0.J.................*
014CC0    D0B61BFE 40FD0168 D203E000 D16841FE    000450FD 0160411D 00489220 100558F1    *....  ...K...J..................1*
014CE0    000858F0 F03005EF 411D0048 58E01008    58F0E034 05EF4110 0001480D 005C9560    *...00.............0..............*
014D00    D1704740 D0CE4720 D0CC1A01 1A011A01    9240D170 D277D171 D1704110 00381901    *J................ J.K.J.J........*
014D20    47A0D0EA 400D005C 07F5481D 005E4111    0001401D 005E92F1 D170D203 D1DDD12C    *..........5.............1J.K.J.J.*
014D40    4E1D0150 F333D1E2 D15496F0 D1E5D201    D05CD130 47F0D060 98E0D15C 12EE0785    *....3.JSJ..0JVK...J..0....J.....*
014D60    411E0004 191F0785 47F0D082 D7C1C7C5    FFFF07FE 00000000 00000000 00014C30    *.........0..PAGE................*
014D80    00014D80 00000EE0 00000000 F0F0F0F0    C6F0C6F0 6C00005C 80004262 0006C470    *................4030..........0.*
014DA0    0006C6E5 0006C7E8 036F4040 007D4040    40F0F1F4 C4C1F040 4040F0F0 F0F6C3F6    *..FV..GY..   ..                  *
014DC0    40404040 40404040 F0F4F040 C6F0C6F4    C6F0F4F0 40C3F6C6 F0C3F6C6 F4404040    *                                 *
   LINES     014DE0-014E00     SAME AS ABOVE
```

Figure 60. Segmented COBOL Program with Abnormal Termination Dump (Part 4 of 4)

Since the sample problem program shown in Figure 61 was interrupted because of a data exception, the programmer should locate the contents of field B at the time of the interrupt. The numerals encircled in the two techniques given below refer to information similarly labeled in the sample program.

Using the General Registers: The general registers usually contain information that can be helpful to the programmer who is trying to locate specific data.

1. Locate data-name B, 1 , in the glossary. It appears under the column headed SOURCE-NAME. Source-name B has been assigned to base locator 3 (i.e., BL=3) with a displacement of 058. The sum of the value of base locator 3 and the hexadecimal displacement value 58 is the address of data-name B.

2. The Register Assignment table, 2 , lists the registers assigned to each base locator. Register 6 has been assigned to BL=3.

3. The contents of the 16 general registers at the time of the interrupt are displayed at the beginning of the dump, 3 . Register 6 contains the address 00014200.

4. The location of data-name B, 4 , can now be determined by adding the contents of register 6 and the hexadecimal displacement value 58. The result, 14258, is the address of the leftmost byte of the 4-byte field B. Field B contains F1F2F3C4. This is external decimal representation and does not correspond to the USAGE COMPUTATIONAL-3 defined in the source listing.

Using the TGT Memory Map: If the general registers appear not to contain meaningful information, it may be that errors in the problem program have destroyed their contents. In such a case, the alternate method of locating data-names given below should be helpful.

1. The location assigned to a given data-name may also be found by using the BL CELLS relocation value given in the TGT Memory Map, 5 . To find the location of the BL cells, add 003FC (from the TGT table) to the entry point address[1], 14020, of the object module, 6. In this example, the BL cells begin at location 1441C:

   003FC + 14020 = 1441C

2. The first four bytes are the first BL cell, the second four bytes are the second BL cell, etc. Note that the third BL cell, 7, contains the value 14200. This is the same value as that contained in register 6.

Note: Use of the FLOW and STATE options eliminates the need for the calculations described above. All that is needed for program debugging is the output from FLOW and STATE printed at the end of the listing, 8, and described below.

A. Specification of either FLOW or STATE causes the PROGRAM-ID, the completion code, and the PSW for the last problem program executed before the abnormal termination to be printed out.

B. If STATE is in effect, the printed output includes the compiler-generated card number for the last verb executed.

C. If FLOW is in effect, the words FLOW TRACE are printed out, together with the PROGRAM-ID and the card numbers of the procedure-names executed for all COBOL programs with the FLOW option in effect.

For further discussion of the FLOW and STATE compiler options, including their relationship to the NUM option and to the SYMDMP option, see the chapter entitled "Symbolic Debugging Features."

----

[1]For nonsegmented programs, the entry point address and the load point address are the same. (For a discussion on computing the load point address for a segmented program, see the section "Debugging a Segmented Program.")

```
100010 IDENTIFICATION DIVISION.
100020 PROGRAM-ID.  TESTRUN.
100030   AUTHOR. PROGRAMMER NAME.
100040   INSTALLATION. NEW YORK PROGRAMMING CENTER.
100050   DATE-WRITTEN. JULY 12, 1968.
100060 DATE-COMPILED. MAY  6,1971
100070   REMARKS. THIS PROGRAM HAS BEEN WRITTEN AS A SAMPLE PROGRAM FOR
100080      COBOL USERS. IT CREATES AN OUTPUT FILE AND READS IT BACK AS
100090      INPUT.
100100 ENVIRONMENT DIVISION.
100110 CONFIGURATION SECTION.
100120   SOURCE-COMPUTER. IBM-360-H50.
100130   OBJECT-COMPUTER. IBM-360-H50.
100140 INPUT-OUTPUT SECTION.
100150 FILE-CONTROL.
100160      SELECT FILE-1 ASSIGN TO UT-2400-S-SAMPLE.
100170      SELECT FILE-2 ASSIGN TO UT-2400-S-SAMPLE.
100180 DATA DIVISION.
100190 FILE SECTION.
100200 FD  FILE-1
100210      LABEL RECORDS ARE OMITTED
100220      BLOCK CONTAINS 100 CHARACTERS
100225      RECORD CONTAINS 20 CHARACTERS
100230      RECORDING MODE IS F
100240      DATA RECORD IS RECORD-1.
100250   01 RECORD-1.
100260      02 FIELD-A PICTURE IS X(20).
100270 FD  FILE-2
100280      LABEL RECORDS ARE OMITTED
100290      BLOCK CONTAINS 5 RECORDS
100300      RECORD CONTAINS 20 CHARACTERS
100310      RECORDING MODE IS F
100320      DATA RECORD IS RECORD-2.
100330   01 RECORD-2.
100340      02 FIELD-A PICTURE IS X(20).
100350 WORKING-STORAGE SECTION.
100360      77 KOUNT PICTURE S99 COMP SYNC.
100370      77 NOMBER  PICTURE S99 COMP SYNC.
100375   01 FILLER.
100380      02 ALPHABET PICTURE X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
100395      02 ALPHA REDEFINES ALPHABET PICTURE X OCCURS 26 TIMES.
100405      02 DEPENDENTS PICTURE X(26)  VALUE "0123401234012340123401234
100410-     "0".
100420      02 DEPEND REDEFINES DEPENDENTS PICTURE X OCCURS 26 TIMES.
100440   01 WORK-RECORD.
100450      02 NAME-FIELD  PICTURE X.
100460      02 FILLER PICTURE X VALUE SPACE.
100470      02 RECORD-NO PICTURE 9999.
100480      02 FILLER PICTURE X VALUE SPACE.
100490      02 LOCATION PICTURE AAA VALUE "NYC".
100500      02 FILLER PICTURE X VALUE SPACE.
100510      02 NO-OF-DEPENDENTS PICTURE XX.
100520      02 FILLER PICTURE X(7) VALUE SPACES.
100522 01   RECORDA.
100524      02  A PICTURE S9(4) VALUE 1234.
100526      02  B REDEFINES A PICTURE S9(7) COMPUTATIONAL-3.
100530 PROCEDURE DIVISION.
100540 BEGIN.
100550      NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED
100560      AND INITIALIZES COUNTERS.
100570 STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO KOUNT NOMBER.
100580      NOTE THAT THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE
100590      CONTAINED IN THE FILE, WRITES THEM ON TAPE, AND DISPLAYS
100600      THEM ON THE CONSOLE.
100610 STEP-2. ADD 1 TO KOUNT, ADD 1 TO NOMBER, MOVE ALPHA (KOUNT) TO
100620      NAME-FIELD.
100625           COMPUTE B = B + 1.
100630      MOVE DEPEND (KOUNT) TO NO-OF-DEPENDENTS.
100640      MOVE NOMBER TO RECORD-NO.
100650 STEP-3. DISPLAY WORK-RECORD UPON CONSOLE. WRITE RECORD-1 FROM
100660      WORK-RECORD.
100670 STEP-4. PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL TO 26.
100680      NOTE THAT THE FOLLOWING CLOSES OUTPUT AND REOPENS IT AS
100690      INPUT.
100700 STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.
100710      NOTE THAT THE FOLLOWING READS BACK THE FILE AND SINGLES OUT
100720      EMPLOYEES WITH NO DEPENDENTS.
100730 STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.
100740 STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO
100750      NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO
100760      STEP-6.
100770 STEP-8. CLOSE FILE-2.
100780      STOP RUN.
```

Figure 61. Sample Program (Part 1 of 5)

INTRNL NAME	LVL	SOURCE NAME	BASE	DISPL	INTRNL NAME	DEFINITION	USAGE	R	O	Q	M
CNM=1-148	FC	FILE-1	DCB=01		DNM=1-148		CSAM				F
CNM=1-167	C1	RECCRC-1	BL=1	0C0	DNM=1-167	DS CCL20	GROUP				
CNM=1-188	C2	FIELC-A	BL=1	000	DNM=1-188	CS 20C	DISP				
CNM=1-205	FC	FILE-2	DCB=02		DNM=1-2C5		CSAM				F
DNM=1-224	C1	RECCRC-2	BL=2	0C0	DNM=1-224	CS CCL20	GROUP				
CNM=1-245	C2	FIELC-A	BL=2	CC0	CNM=1-245	CS 20C	DISP				
CNM=1-265	77	KCUNT	BL=3	CC0	DNM=1-265	DS 1H	CCMP				
DNM=1-28C	77	NCMBER	BL=3	002	CNM=1-280	CS 1H	COMP				
CNM=1-296	C1	FILLER	BL=3	CC8	DNM=1-296	CS 0CL52	GROUP				
CNM=1-315	C2	ALPHABET	BL=3	CC8	DNM=1-315	CS 26C	CISP				
DNM=1-333	C2	ALPHA	BL=3	0C8	CNM=1-333	CS 1C	DISP	R	O		
CNM=1-351	C2	CEPENCENTS	BL=3	C22	DNM=1-351	CS 26C	DISP				
DNM=1-371	C2	CEPENC	BL=3	022	DNM=1-371	CS 1C	DISP	R	C		
CNM=1-387	C1	WCRK-RECORC	BL=3	040	CNM=1-387	CS 0CL20	GROUP				
CNM=1-411	C2	NAME-FIELC	BL=3	C40	DNM=1-411	CS 1C	DISP				
DNM=1-431	C2	FILLER	BL=3	C41	CNM=1-431	CS 1C	DISP				
CNM=1-45C	C2	RECORC-NC	BL=3	C42	CNM=1-450	CS 4C	CISP-NM				
CNM=1-469	C2	FILLER	BL=3	C46	DNM=1-469	DS 1C	DISP				
CNM=1-488	C2	LCCATICN	BL=3	C47	CNM=1-488	CS 3C	CISP				
CNM=2-C0C	C2	FILLER	BL=3	C4A	DNM=2-00C	CS 1C	CISP				
CNM=2-C19	C2	NC-CF-DEPENCENTS	BL=3	04B	DNM=2-C19	DS 2C	CISP				
CNM=2-C45	C2	FILLER	BL=3	C4D	CNM=2-C45	CS 7C	DISP				
CNM=2-C64	C1	RECCRCA	BL=3	C58	CNM=2-C64	CS CCL4	GROUP				
CNM=2-C84	C2	A	BL=3	C58	CNM=2-C84	CS 4C	DISP-NM				
CNM=2-C95	C2	B 1	BL=3	C58	CNM=2-095	CS 4P	COMP-3	R			

MEMCRY MAP

TGT	CC24C
SAVE AREA	CC24C
SWITCH	CC288
TALLY	CC28C
SCRT SAVE	CC29C
ENTRY-SAVE	CC294
SCRT CCRE SIZE	CC298
RET CCCE	CC29C
SCRT RET	CC29E
WCRKING CELLS	CC2A0
SCRT FILE SIZE	CC3DC
SCRT MCCE SIZE	0C3C4
PGT-VN TBL	CC3C8
TGT-VN TBL	CC3DC
VCCNPTR	CC3EC
LENGTH OF VN TBL	CC3E4
LABEL RET	CC3E6
CURRENT PRICRITY	C03E7
UNUSED	CC3E8
INIT1 ACCCN	CC3FC
CEBUG TABLE PTR	CC3F4
UNUSEC	CC3F8
CVERFLCW CELLS	0C3FC
5 BL CELLS	CC3FC
CECBACR CELLS	CC4C8
TEMP STCRAGE	CC4C8
TEMP STORAGE-2	CC410
TEMP STCRAGE-3	CC41C
TEMP STCRAGE-4	CC41C
BLL CELLS	CC41C
VLC CELLS	CC418
SBL CELLS	CC418
INCEX CELLS	CC418
SUPACR CELLS	CC418
CNCTL CELLS	CC42C
PFMCTL CELLS	CC42C
PFMSAV CELLS	CC42C
VN CELLS	CC424
SAVE AREA =2	CC428
SAVE AREA =3	CC428
XSASW CELLS	CC43C
XSA CELLS	CC43C
PARAM CELLS	CC43C
RPTSAV AREA	CC434
CHECKPT CTR	CC434
VCCN TBL	CC438
CEBUG TABLE	CC438

Figure 61. Sample Program (Part 2 of 5)

```
LITERAL POOL (HEX)

CC4AE (LIT+0)     CCCOCOC1  1COOCO1A  CO4805EF  480000CO  C00000CO

     CISPLAY LITERALS (BCC)

CC4BC (LTL+2C)    'WCRK-RECCRC'


                  PGT                       CC450

               CVERFLCW CELLS               CC45C
               VIRTLAL CELLS                CC45C
               PRCCEDLRE NAME CELLS         CC46C
               GENERATEC NAME CELLS         CC48C
               CCB ADCRESS CELLS            0C498
               VNI CELLS                    CC4AC
               LITERALS                     CC4A8
               CISPLAY LITERALS             004BC


 REGISTER ASSIGNMENT

  REG 6    BL =3   2
  REG 7    BL =1
  REG 8    BL =2

WCRKING-STCRAGE STARTS AT LCCATICN 001EC FOR A LENGTH OF C006C.

58      *BEGIN
                     0C04C8                         START     ECU    *
                     CCC4C8   C7 CC                           BCR    0,0
                     0C04CA   58 FC C 00C                     L      15,00C(0,12)          V(ILBODBG4)
                     CCC4CE   C5 EF                           BALR   14,15
                     CCC4CC   5E FC C 01C                     L      15,01C(0,12)          V(ILBOFLW1)
                     CC04C4   C5 1F                           BALR   1,15
                     CCC4D6   CC00CC3A                        DC     X'0000003A'
61      *STEP-1
                     CCC4CA   58 FC C 00C                     L      15,CCC(C,12)          V(ILBODBG4)
                     CCC4DE   C5 EF                           BALR   14,15
                     0CC4EC   58 FC C 01C                     L      15,C1C(C,12)          V(ILBCFLW1)
                     CCC4E4   C5 1F                           BALR   1,15
                     CCC4E6   CCCCCC3D                        DC     X'0000003D'
61       CPEN        CC04EA   5P FC C 00C                     L      15,CCC(C,12)          V(ILBODBG4)
                     CCC4EE   C5 EF                           BALR   14,15
                     CCC4FC   5E 1C C 048                     L      1,C48(C,12)           CCB=1
                     CCC4F4   C2 C1 1 032 C 06C               MVC    032(2,1),C6C(12)                     LIT+8
                     CCC4FA   D2 C1 1 06C C C62               MVC    060(2,1),C62(12)                     LIT+1C
                     CCC5CC   5C 1C D 1E8                     ST     1,1E8(0,13)           SAV3
                     CCC5C4   92 8F C 1E8                     MVI    1E8(13),X'8F'         SAV3
                     CCC5C8   41 1C C 1E8                     LA     1,1E8(0,13)           SAV3
                     CCC5CC   CA 13                           SVC    19
                     CCC5CE   58 1C C 048                     L      1,048(0,12)           CCB=1
                     CCC512   18 21                           LR     2,1
                     CC0514   58 FC 1 030                     L      15,C3C(C,1)
                     CCC51E   C5 EF                           BALR   14,15
                     CCC51A   5C 10 C 1BC                     ST     1,1BC(C,13)           BL =1
                     CCC51E   5E 7C D 1BC                     L      7,1BC(0,13)           BL =1
61       MCVE        0C0522   C2 C1 6 000 C C5E               MVC    CC0(2,6),C5E(12)      DNM=1-265      LIT+C
                     CCC528   C2 01 6 002 C C58               MVC    002(2,6),C5E(12)      DNM=1-280      LIT+C
65      *STEP-2
                     CCC52E                         PN=C1     ECL    *
                     CCC52E   58 FC C 00C                     L      15,00C(0,12)          V(ILBODBG4)
                     CCC532   C5 EF                           BALR   14,15
                     CC0534   5E FC C 01C                     L      15,01C(C,12)          V(ILBCFLW1)
                     CCC538   C5 1F                           BALR   1,15
                     CCC53A   CCCCCC41                        DC     X'00000C41'
65       ACC         C0C53E   48 3C C 05A                     LH     3,C5A(C,12)           LIT+2
                     CCC542   4A 3C 6 000                     AH     3,0CC(0,6)            CNM=1-265
                     CCC546   4C 3C 6 0CC                     STH    3,CC0(C,6)            CNM=1-265
65       ADC         CCC54A   48 3C C 05A                     LH     3,C5A(C,12)           LIT+2
                     CCC54E   4A 3C 6 CC2                     AH     3,002(0,6)            CNM=1-280
                     C0C552   4C 3C 6 002                     STH    3,0C2(C,6)            CNM=1-280
65       MOVE        CCC556   41 4C 6 0C8                     LA     4,008(0,6)            CNM=1-333
                     CCC55A   48 2C 6 CCC                     LH     2,0C0(C,6)            CNM=1-265
                     CCC55E   4C 2C C C5A                     MH     2,05A(0,12)           LIT+2
                     CCC562   1A 42                           AR     4,2
                     CCC564   5B 4C C 058                     S      4,C58(C,12)           LIT+0
                     CCC56B   5C 4C C 1D8                     ST     4,1C8(C,13)           SBS=1
                     CCC56C   58 EC C 1D8                     L      14,1C8(C,13)          SBS=1
                     CCC57C   C2 CC 6 040 E CCC               MVC    04C(1,6),0CC(14)      CNM=1-411      CNM=1-333
67       CCMPLTE     CC0576   FA 3C 6 058 C C5C               AP     C58(4,6),05C(1,12)    DNM=2-95       LIT+4
68       MCVE        CCC57C   41 4C 6 C22                     LA     4,C22(0,6)            CNM=1-371
                     CCC58C   48 2C 6 0CC                     LH     2,CCC(C,6)            CNM=1-265
                     CCC584   4C 2C C C5A                     MH     2,C5A(0,12)           LIT+2
                     CCC58B   1A 42                           AR     4,2
                     CCC58A   5B 40 C 058                     S      4,C58(C,12)           LIT+C
                     CCC58E   5C 4C C 1DC                     ST     4,1CC(0,13)           SBS=2
                     CCC592   58 EC C 1DC                     L      14,1CC(C,13)          SBS=2
                     CCC596   C2 0C 6 048 E 0C0               MVC    048(1,6),CCC(14)      DNM=2-19       CNM=1-371
                     CCC59C   92 4C 6 04C                     MVI    04C(6),X'40'          CNM=2-19+1
                        .        .                            .          .                   .
                        .        .                            .          .                   .
                        .        .                            .          .                   .
```

Figure 61. Sample Program (Part 3 of 5)

```
*STATISTICS*     SCURCE RECORCS =    83     CATA DIVISION STATEMENTS =    25     PROCEDURE DIVISION STATEMENTS =    21
*OPTICNS IN EFFECT*     SIZE =   8192C  BUF =    2768  LINECNT = 57  SPACE1, FLAGW,   SEQ,   SOURCE
*CPTICNS IN EFFECT*      CMAP,   PMAP, NCCLIST, NCSUPMAP, NOXREF, NCSXREF,   LCAC, NCCECK, CUCTE, NOTRUNC,   FLCW=   35
*CPTICNS IN EFFECT*     NCTERM, NCNUM, NOBATCH, NONAME, CCMPILE=01,   STATE,   LIB; VERB, ZWB, SYST




 CARC   ERRCR MESSAGE

 55     IKF219CI-W     PICTURE CLAUSE IS SIGNEC, VALUE CLAUSE UNSIGNEC. ASSUMED PCSITIVE.




 * ABCUMP RECUESTEC *

JCB TESTRUN          STEP CC              TIME C64342   DATE 71126

CCMPLETICN CCDE      SYSTEM = OC7

PRCGRAM INTERRUPTICN (CATA) AT LCCATION C14596

INTERRUPT AT CC4C34

PSW AT ENTRY TC ABENC  CCC4CCCD EC01459C

TCB  CC5298      RB   CCC6CA40    PIE     CCCCCCCC    DEB  CCC6C9C4    TICT CCC6CEE8    CMP    800C7000    TRN  000C0000
                 MSS  CCCC532C    PK/FLG 10910408     FLG  000004F9    LLS  C00CC00C    JLB    CCCCC0CC    JSE  CCC0000C
                 FSA  14C6CFBC    TCB     CCCCCCCO    TME  0CCC5348    PIB  ECCC9F98    NSTAE ACC6CCF8     TCT  0C0C4D8C
                 USER CCCCCCCC


ACTIVE RBS

PRB    C14CCC  NM RUN        SZ/STAB C51BCCCC    USE/EP COC1402C    PSW FF15CCCC 60CC4C34    C 0C0C00    WT/LNK 0C0C5298

SVRB   C6CD68  NM SVC-8C1C   SZ/STAB 0012C172    USE/EP 0CCC41A8    PSW FFF5000D ECC1459C    Q 18C318    WT/LNK 0CC14000
               RG C-7   CCCCCC3C    CCC1455E    CCC0C001    0CCC0CC1    C0C14208    50C1485E    CCC142CC    CCC6CB98
               RG 8-15  CCC14C2C    0CC14E26    CCC14C2C    CCC14C2C    CCC1447C    CCC1426C    0CC14208    0CC16C06

SVRB   C6CAAC  NM SVC-4C1C   SZ/STAB 0012C172    USE/EP 00CC41A8    PSW FF040033 4CCC437C    Q E8C3E8    WT/LNK 0CC6CD68
               RG C-7   FACCCC6E    8CCC7CCC    8CCC5E64    0000647C    C0C05298    CC06CC68    0006CDFE    CC01400C
               RG 8-15  0CC14C0C    000C44E2    C0C6CDC8    C0CC5EC4    CCCC5298    CCC6CAE8    400C41AA    CCCC0000

SVRB   C6CA4C  NM SVC-1C5A   SZ/STAB C0CCD172    USE/EP CCCC41A8    PSW FF040235 8C00C53C    Q C8C3C8    WT/LNK CCC6CAAC
               RG C-7   CCC168D8    00C1653C    CCCC12CC    4CCC41AA    CCC00C0C    CC000000    0CC168C8    80CC435E
               RG 8-15  CCCC531E    00C169A6    0CC6D0CC    CCCC5298    CC005298    CCC1654C    6CCC43FA    2C014596

P/P STCRAGE BCUNCARIES CCC14CCC TC CCC6CCCC
```

Figure 61. Sample Program (Part 4 of 5)

```
REGS AT ENTRY TO ABEND

FL.PT.REGS 0-6      CC.CCCOCC CCCCC000     00.00000C 00CC0000     C0.CCC00C CCCOCCCC  (3)  CC.0C0000 00000000

REGS 0-7      00000030  0001455E  00000001  00000001       00014208  50014858  00014200  0006CB98
REGS 8-15     00014020  00014826  00014020  00014020       00014470  00014260  00014208  00016006

P/P STORAGE

C14CCC   C9E4D54C 4C4C404C C51BCCCC 00C1402C    FF15C00D 60004C34 CC0C00C0 00005298   *RUN      ...... ................*
C14C20   90ECD00C 185C05FC 4580F010 E3C5E2E3    C9E4D540 C1D5E2C3 C7CC989F FC24C7FF   *.......0..0.TESTRUN ANSC....0...*
C14C4C   96C21C34 C7FE41FC CCC1C7FE 0C014826    00C14020 00014C2C C001447C CC01426C   *.......C.......... ... .........*
C1406C   C0C144E8 CCC14EC4 CCCCCCCC 0CCCCCCC    CCCCOCC0 00000CC0 CCCCCCCC CCC00000   *...Y............................*
C14C8C   0CCCC00C CCCCOCCC CCCOCCCC 00000000    0CCCCC0C C000CCCC CCCCCCCC CCCCCCC0   *................................*
C14CAC   CCCCCCCC CCCCCCCC CCCCCCCC 00C0C0CC    C0C00CC0 C0000000 08014C46 CC000001   *................................*
C140CC   CCCCCCC1 CCCCCCCC CCCCCCCC 0CCCCCCC    8CCCCCCC CC00CCCC CCCCCCCC CCCCCC00   *................................*
C14CEC   CCCCCCCC CC8183CC 02C6CB90 00004000    0CCCCC01 46CC00C1 90C14CA4 CC68CC48   *............ .......... .....*
C141CC   CCC6CE2C 52CCCCCC CCCCCF58 06C15B78    CCC90C64 28012828 42C6CB3C 0006CBFC   *................................*
C1412C   CCC6CBAC CCCC0014 00C00001 0CCCCCCC    C0CCDE68 C5EFCCCC CCCCCCCC CCC00000   *................................*
C1414C   CCCCCCCC CCCCCCCC 0CCCCCCC 0CCCC000    CCCC000C CCC0CCCC CCCCOCCC CCCCCCCC   *................................*
C1416C   CCCCCCCC CBC14C46 CCCCCCC1 CCCCCCC1    CCCCCCCC CC000000 CCOCCCCC 8C000000   *...... .........................*
C1418C   0CCCCCCC CCCCCCCC CCCQCCOC 0CCCCOCC    0CCCCOCC CCCOOCC1 CCCC4CCC CCCCCCC1   *......................... .....*
C141AC   42CCCCC1 9CC1415C E2C1D4D7 D3C5404C    C2CC000C C00000C0 00015878 CCCCCC64   *........SAMPLE  ...............*
C141CC   0CCCCCCC CCCCCCCC CCCCCOCC CCCCCCCC    CCCCCC14 CCCOCCCC CCCCCCCC CCOCCC00   *................................*
C141EC   CCCCCCCC CCCCOCCC CCCC00CC C0000000    0CCCOCC0 COCCOCCC CCCCCCCC CCCOCCCC   *................................*
C142CC   CCC1CCC1 CCCCCCCC C1C2C3C4 C5C6C7C8    C9D1D2D3 D4D5D6D7 D8D9E2E3 E4E5E6E7   *........ABCDEFGHIJKLMNOPQRSTUVWX*
C1422C   E8E9F0F1 F2F3F4FC F1F2F3F4 FCF1F2F3    F4FCF1F2 F3F4F0F1 F2F3F4FC CCC00CC0   *YZ01234C1234C1234C1234012340....*
C1424C   C14CCCCC CCCC4CD5 E8C34CC0 004C4040    404C4C4C C0C0000Q F1F2F3C4 CCCCCCCC (4) *A .... NYC ..         ....123D....*
C1426C   CCCCCCCC CCC6CFBC CCC6CD5C 5CC14554    CCC16CC6 CC000030 5C01455A CCC140C4   *............................. M*
C1428C   CCC6DCCC CCC14C2C 50C14858 0CC142CC    CCC6CB98 00014020 CCC14E26 CCC14C20   *...... ............... ....... .*
C142AC   CCC14C2C CCC1447C 36CCC04E CCCCCCCC    CCCC0000 CCC144E8 CCOCCCCC CCCCCCCC   *.. ....................Y........*
C142CC   CCCCCCCC CCCCCCCC CCCCCOCC 0CCCC00C    0C000000 CC000000 CCCCOCCC CCCCCCCC   *................................*
  LINES     C142EC-C143EC     SAME AS ABOVE
(7) C144CC   CCCCCCCC CCCCCCCC CCCCCCCC 000CCC0G    CCC14C20 CCCOC438 CCC6CFF8 CCC6CB98   *................. .........8....*
C1442C   CCC1402C CCC142CC CCCCCOCC C0000000    0C0000CC CC000000 0C0142C8 CCCCCCC0   *.. .............................*
C1444C   CCCCCCCC CCC1462C 8FC14CD4 CCCCCCCC    CCCCCCCC CC0000C1 23CCC7E4 CCCOCCCC   *.......... M...............U....*

(A) PROGRAM TESTRUN (8)
    COMPLETION CODE = CC7     LAST PSW BEFORE ABEND = FFF5000D E001459C

(B) LAST CARD NUMBER/VERB NUMBER EXECUTED -- CARD NUMBER 000067/VERB NUMBER 01.

                                        FLOW TRACE
(C) TESTRUN   CCCC58 CCCCE1 CCCCE5

    END OF COBOL DEBUGGING AIDS
```

Figure 61. Sample Program (Part 5 of 5)

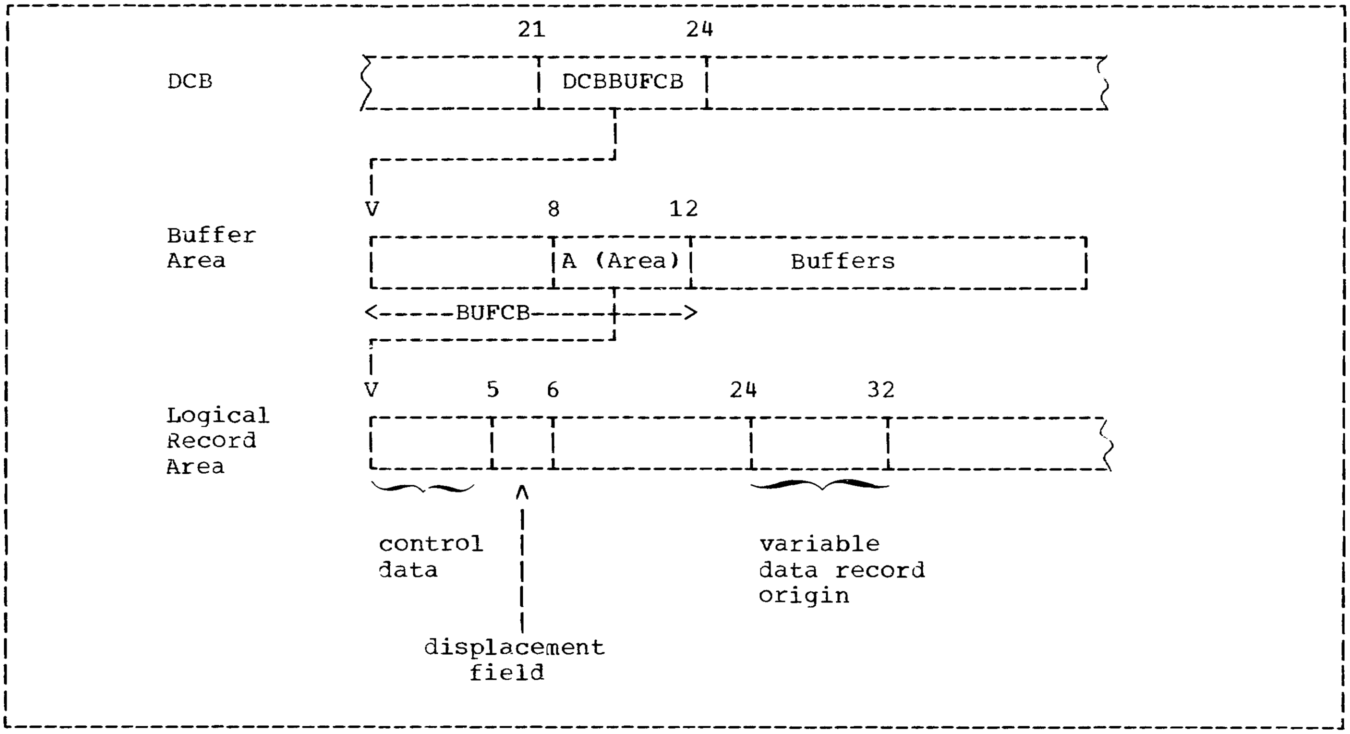


Figure 62. Locating the QSAM Logical Record Area

Locating Data Areas for Spanned Records

QSAM: QSAM (sequential) spanned records allocate a Logical Record Area in which complete logical records may be assembled (see "Record Formats"). Figure 62 illustrates the relationship between the DCB, the Buffer Areas, and the Logical Record Area.

1. The DCB contains the DCBBUFCB field at a displacement of 21 bytes from the origin of the DCB. The contents of DCBBUFCB points to the origin of the Buffer Control Block (BUFCB) in the Buffer Area.

2. The BUFCB field contains an Area-Address (A(Area)) at a displacement of 8 bytes from the origin of the Buffer Area. The Area-Address points to the origin of the Logical Record Area.

3. The Logical Record Area contains a displacement field at a displacement of 5 bytes from its origin. This field contains a value from 0 to 8 indicating the number of bytes the record has been displaced. The contents of this 1-7 byte field must be added to the value 24 (the first byte in the variable data record origin area) in order to locate the beginning of the logical data record within the Logical Record Area. Note that the first 4 bytes of the Logical Record Area are control data indicating the length of the Logical Record Area (including the 4 bytes of control data).

Note: The Logical Record Area is not allocated for QSAM records formatted in V, U, or F mode.

```
            variable        4               variable
        <----bytes----><-bytes-><----------bytes---------->
        ┌─────────────┬────┬──┬──────────────────────────┐
        |   REC-ID    | LL |ll|        Maximum 01        }
        └─────────────┴────┴──┴──────────────────────────┘

                       Logical Record Area


           18               variable     4            variable
 <--------bytes-------> <--bytes--><-bytes-><--------bytes-------->
 ┌─────────────────────┬──────────┬────┬────┬─────────────────────┐
 |        BUFCB        | REC-ID   | LL | ll |    Data Segment     }
 └─────────────────────┴──────────┴────┴────┴─────────────────────┘
 \_________ _________/  \_____________________ ____________________/
           V                                  V
         Buffer                     Segment Work Area
      Control Block
```

Figure 63. Logical Record Area and Segment Work Area for BDAM and BSAM Spanned Records

<u>BSAM and BDAM</u>: BSAM and BDAM (direct) spanned records allocate a Segment Work Area. This work area is used for temporary storage of record segments before a complete logical record is assembled in the Logical Record Area. Figure 63 illustrates the Logical Record Area and the Segment Work Area.

<u>Note</u>: The segment work area is not allocated for BSAM and BDAM records formatted in V, U, or F mode.

The following discussion illustrates the relationship between the DCB, the Logical Record Area, and the Segment Work Area as shown in Figure 63.

<u>BDAM</u>

1. The DCB address plus 100 bytes points to the beginning of the BUFCB (Buffer Control Block).

2. The contents of the BL assigned to the level-01 entry in an FD points to the Logical Record Area labeled "Maximum 01" in Figure 63 (see Figure 59 for an example of the BL pointer.)

<u>BSAM output</u>

1. The DCB address plus 76 bytes points to the beginning of the BUFCB (Buffer Control Block).

2. The DECB address plus 12 bytes points to the beginning of the Logical Record Area.

<u>BSAM input</u>

1. The DECB address plus 12 bytes points to the beginning of the Segment Work Area.

2. The DCB address plus 100 bytes points to the beginning of the Logical Record Area.

<u>Locating TCAM Data Areas</u>

In a teleprocessing application, control blocks, called <u>queue blocks</u>, are created for a given partition/region. For the RECEIVE statement, the number of queue blocks created agrees with the number of queues accessed. For the SEND statement, however, only one queue block is created for each partition/region. The encircled numerals in Figures 64 and 65 refer to the numbered paragraphs below.

1. The TGT address plus 440 bytes points to the SUBCOM field (see Figure 125 in Appendix J: "Fields of the Global Table"). The fullword at X'50' bytes into SUBCOM points to the first RECEIVE queue block. The fullword at

X'54' off SUBCOM points to the SEND queue block. In both cases, the first field (IHADCB) contains the data control block (DCB).

2. At X'58' bytes into either a RECEIVE or a SEND queue block, the first byte of the 4-byte BUFRADR field indicates whether the address that follows represents a TCAM buffer or a BSAM buffer. If the two high-order bits are on, the address contained in the next three bytes is for a TCAM buffer.

   Note: For TCAM there is only one buffer; for BSAM there is one buffer for each queue.

Relative Location	Field
0	IHADCB 1
58	BUFRADR 2
5C	BUFSIZE 3
5E	SUMGIVEN 4
60	DECBQB 5
74	MOREBLKS 6
78	AMTLEFT 7
7A	DATIMSOR 8
90	QNAMEQB 9
98	EORCHAR 10
99	HELDOVER 13
9A	ISITPART 14

Figure 64. Fields of the RECEIVE Queue Block

Relative Location	Field
0	IHADCB 1
58	BUFRADR 2
5C	BUFSIZE 3
5E	MOREROOM 7
60	ENDICATR 8
61	EORCHAR 10

Figure 65. Fields of the SEND Queue Block

3. In either a RECEIVE or a SEND queue block, the next field (BUFSIZE) specifies the size of the buffer, whose format is pictured in Figure 66. (For a list of codes used in the TCAM control byte, see Table 24.)

4. The SUMGIVEN field of the RECEIVE queue block indicates the number of bytes of data given to the user for this request.

5. The DECBQB field of the RECEIVE queue block contains the data event control block (DECB).

6. In the RECEIVE queue block, the MOREBLKS field provides the address of the next queue block. If the first byte of this field is zero, there are no additional queue blocks.

7. For BSAM only, the AMTLEFT field of the RECEIVE queue block indicates the amount of data being held in the buffer from the last request.

8. For BSAM only, the MOREROOM field of the SEND queue block indicates the number of unused bytes left in the buffer.

9. The 22-byte DATIMSOR field of the RECEIVE queue block contains the date and time of the last message received from this queue, as well as the source of the message.

10. The ENDICATR field of the SEND queue block contains the end indicator (in zoned decimal) specified in the COBOL source statement.

11. The QNAMEQB field contains the ddname for the queue block specified in the COBOL teleprocessing program. This is the name the COBOL programmer used in the SYMBOLIC QUEUE clause of the CD entry.

12. The EORCHAR field in both the RECEIVE queue block and the SEND queue block contains the record delimiter specified in the MCP.

13. The HELDOVER field in the RECEIVE queue contains a character that, in some instances, is the next data character.

14. The ISITPART field in the RECEIVE queue block is a switch byte.

```
1        5   6               14
--------T---T----------------T---/ /-----
--------|---|----------------|--/ /------
    V     TCAM  Source ID          Data
prefix   control
          Byte
```

Note: The prefix, the TCAM control byte, and the source ID must be user specified for a SAM file. However, if the user invokes the SEND statement to create a SAM file for subsequent input, then the COBOL compiler adds bytes 1 through 13 (see Figure XY in the chapter entitled "Using the Teleprocessing Feature").

Figure 66. Structure of a TCAM Record

INCOMPLETE ABNORMAL TERMINATION

If a job is abnormally terminated and the abnormal termination process goes to completion, the following procedures are carried out:

- A dump (ABDUMP) is produced by the system.
- The data sets in the job steps are disposed of as specified in the DISP parameter (i.e., kept, deleted, etc.). This is indicated in the job scheduler disposition messages produced for the job step.
- Temporary data sets, including those passed from previous job steps, are deleted.

When the abnormal termination process does not go to completion (i.e., no end of dump message is present), none of these procedures will be carried out. Those data sets in the job step that were in existence previous to the point at which the error condition occurred will remain in effect. For data sets on direct access volumes, the names will remain tabulated in the Volume Table of Contents (VTOC) of the volume (see "Additional File Processing Information" for details on the VTOC). The result of an incomplete abnormal termination is that space needed by a subsequent job will be unavailable, or, if the same job is then rerun, duplicate name definition will result for those data sets that are newly created in the job step. This is true for temporary data sets for which the system has assigned the name, as well as data sets for which the programmer has assigned the name.

Table 24. Codes Used in the TCAM Control Byte

Code	Meaning
X'F1'	The first block of a multiblock message
X'F5'	The first block of a multiblock ~~message, with end of segment~~ indicated
X'40'	An intermediate data block
X'F4'	An intermediate data block, with end of segment indicated
X'F2'	The last block of a multiblock message
X'F6'	The last block of a multiblock message, with end of segment indicated
X'F3'	A single block message
X'F7'	A single block message, with end of segment indicated

SCRATCHING DATA SETS

To avoid duplicate name definition and to ensure that space will be available for newly created data sets, the programmer can scratch his direct-access volume data sets by using the utility program IEHPROGM. To scratch such a data set means to remove its data set label (which includes its name) from the VTOC and to make the space assigned to it available for reallocation. Scratching does not uncatalog any cataloged data sets. This is done by the UNCATLG option of the IEHPROGM.

Note: The information in this section about scratching data sets applies only when running under MFT. Under the MVT option, direct-access volume data sets are scratched automatically. For use of the system utilities under MVT, see the publication IBM OS Utilities.

If a DSNAME parameter has been specified in the DD statement for the data set, the IEHPROGM utility program requires the name of the data set. For data sets named by the programmer, the specified name is the dsname. For data sets for which the DSNAME=&&name convention has been used, an internal name

name.jobname

is assigned by the system, where jobname is the name of the job and name is from the &&name. If no DSNAME parameter is specified, an internal name is assigned by the system. For data sets with no DSNAME parameter there exists an option by which the programmer can specify that all such data sets on the volume be scratched, without having to specify their names.

If the programmer wishes to obtain a listing of the names of all the data sets on a volume, including system-assigned internal names, he can use the utility program IEHLIST. This program provides a listing of the VTOC of the volume.

Information on how to use these utility programs is contained in the publication IBM OS Utilities. The following example illustrates the job control statements that might be used to scratch temporary data sets:

```
//SCR        JOB     ,SCRATCH,MSGLEVEL=1
//STEP1      EXEC    PGM=IEHPROGM
//SYSPRINT   DD      SYSOUT=A
//DD1        DD      UNIT=2311,DISP=OLD
//DD2        DD      UNIT=2311,DISP=OLD,   X
//                   VOLUME=SER=222222
                .
                .
                .
//SYSIN      DD        *
             SCRATCH   DSNAME=GOJOB.TEMP,    X
                       VOL=2311=222222,PURGE
             SCRATCH   VTOC,VOL=2311=222222,X
                       SYS,PURGE
                .
                .
                .
/*
```

In this example, the SYSPRINT DD statement specifies the output data set for the listing and the DD1 DD statement specifies the system residence volume. The other DD statements specify the volume serial number of the volumes that can be mounted on which the data sets have been written. These DD statements are needed to allocate the required devices. The first SCRATCH statement eliminates a data set for which DSNAME=&&TEMP had been specified on the DD statement, and the second SCRATCH statement eliminates all data sets on the volume for which no DSNAME parameter had been specified.

Note that the possibility of duplicate name definition also applies to cataloged procedures in which temporary data sets are used.

For those procedures that are executed often, the programmer may wish to include, at the beginning of his job, a procedure to scratch all temporary data sets.

Some techniques for increasing the efficiency of a COBOL program are described in this chapter. It is divided into seven ~~parts. The first four parts deal in~~ general with coding a COBOL program. The fifth is concerned with the Report Writer feature, the sixth with table handling, and the seventh with queue structure description.

GENERAL CONSIDERATIONS

Spacing the Source Program Listing

There are four statements that can be coded in any or all of the four divisions of a source program: SKIP1, SKIP2, SKIP3, and EJECT. These statements provide the user with the ability to control the spacing of a source listing and thereby improve its readability.

ENVIRONMENT DIVISION

CONFIGURATION SECTION

To take advantage of the new instruction set on the IBM System/370, the programmer should specify IBM-370 as the computer-name in the OBJECT-COMPUTER paragraph.

APPLY WRITE-ONLY Clause

To make optimum use of buffer space allocated when creating a standard sequential file with blocked V-mode records, the programmer may use the APPLY WRITE-ONLY clause for the file. Use of this option causes a buffer to be truncated only when the next record does not fit in the buffer. (If the APPLY WRITE-ONLY clause is not specified, the buffer is truncated when the maximum size record will not fit in the space remaining in the buffer.) When using APPLY WRITE-ONLY, all the WRITE statements must have FROM options. None of the subfields of the associated records may be referred to by procedure statements and they may not be the object of the DEPENDING ON option in an OCCURS clause.

QSAM Spanned Records

~~Except for APPLY WRITE-ONLY, ADVANCING,~~ POSITIONING, and APPLY RECORD-OVERFLOW, all the options for variable length record files apply to spanned records.

APPLY RECORD-OVERFLOW Clause

The APPLY RECORD-OVERFLOW clause makes more efficient use of direct access storage space by using the Track Overflow feature. If APPLY RECORD-OVERFLOW is specified, a record that does not fit on a track will be partially written on that track and the remainder will be written on the next available track.

The use of the APPLY RECORD-OVERFLOW option requires that Track Overflow be specified at system generation time.

APPLY CORE-INDEX Clause

To minimize processing time with indexed files accessed randomly, the programmer should use the APPLY CORE-INDEX clause. Use of this option causes the highest level index to be brought into core storage for input/output operations. This speeds processing by eliminating the extra time needed to search the index on the volume.

BDAM-W File Organization

The use of BDAM-W for file organization results in less system generated coding than for BDAM-D. When BDAM-D is used and a WRITE statement is issued, extra code must be generated to compare the contents of the ACTUAL KEY of the WRITE statement with the key of the preceding READ statement to determine whether the system should add or update a record. If the keys are the same the record is updated. If the keys are different the record is added.

BDAM-W eliminates this comparison step. The system adds a record when a WRITE statement is issued and updates a record when a REWRITE statement is issued.

DATA DIVISION

OVERALL CONSIDERATIONS

Prefixes

Assign a prefix to each level-01 item in a program, and use this prefix on every subordinate item (except FILLER) to associate a file with its records and work-areas. For example, MASTER is the prefix used here:

```
FILE SECTION.
FD  MASTER-INPUT-FILE
    .
    .
    .
01  MASTER-INPUT-RECORD.
    .
    .
    .
WORKING-STORAGE SECTION.
01  MASTER-WORK-AREA.
    05  MASTER-PAYROLL PICTURE 9(3).
    05  MASTER-SSNO PICTURE 9(9).
```

If files or work-areas have the same fields, use the prefix to distinguish between them. For example, if three files all have a date field, instead of DATE, DAT, and DA-TE, use MASTER-DATE, DETAIL-DATE, and REPORT-DATE. Using a unique prefix for each level-01 and all subordinate fields makes it easier for a person unfamiliar with the program to find fields in the program listing, and to know which fields are logically part of the same record or area.

When using the MOVE statement with the CORRESPONDING option and referring to individual fields, redefine or rename "corresponding" names with the prefixed unique names. This technique eliminates excessive qualifying. For example:

```
01  MST-WORK-AREA.
    05  SAME-NAMES.      (***)
        10  LAST-NAME  PIC...
        10  FIRST-NAME  PIC...
        10  PAYROLL  PIC...
        .
        .
        .
    05  DIFF-NAMES REDEFINES SAME-NAMES.
        10  MST-LAST-NAME  PIC...
        10  MST-FIRST-NAME  PIC...
        10  MST-PAYROLL  PIC...
01  RPT-WORK-AREA.
    05  SAME-NAMES.      (***)
        10  PAYROLL  PIC...
        10  FILLER  PIC...
        10  FIRST-NAME  PIC...
        10  FILLER  PIC...
        10  LAST-NAME  PIC...
        .
        .
        .
PROCEDURE DIVISION.
        .
        .
        .
    IF MST-PAYROLL IS EQUAL TO HDQ-PAYROLL
    AND MST-LAST-NAME
    IS NOT EQUAL TO PRRV-LAST-NAME
    MOVE CORRESPONDING
    MST-WORK-AREA
    TO RPT-WORK-AREA.
```

Note: Fields marked with a triple asterisk (***) in the foregoing listing must have exactly the same names for their subordinate fields in order to be considered corresponding. The same names must not be the redefining ones, or they will not be considered to correspond.

Level Numbers

The programmer should use widely incremented level numbers, i.e., 01, 05, 10, 15, etc., instead of 01, 02, 03, 04, etc., in order to allow room for future insertions of group levels. For readability, indent level numbers. Use level-88 numbers for codes. Then, if the codes must be changed, the Procedure Division coding for tests need not be changed.

FILE SECTION

RECORD CONTAINS Clause

The programmer should use the RECORD CONTAINS integer CHARACTERS clause in order to save himself as well as any future programmer the task of counting the data record description positions. Also, the compiler can then diagnose errors if the data record description conflicts with the RECORD CONTAINS clause.

COMMUNICATION SECTION

The Communication Section of a COBOL program must be specified if the program is to take advantage of the Teleprocessing Feature (TP). Through the inclusion of Communication Description (CD) entries, the programmer establishes communication between the COBOL object program and the Message Control Program (MCP).

CD Entries

When specified, the Communication Section must contain at least one CD entry. For example, a single CD entry would be sufficient for applications with either an input or an output message but not both. A COBOL TP program that is both to receive and to send messages must contain at least two[1] CD entries, as below.

```
CD cd-name FOR INPUT.
CD cd-name FOR OUTPUT.
```

The CD entry may instead be pre-written and included in the user-created library. The programmer may then include the entry in a COBOL program by means of a COPY statement.

```
CD cd-name COPY library-name.
```

The input CD contains such information as input queue and sub-queue names, message date and time, the source, the message text length, the end key, the message status key, and the queue depth. The output CD contains the text length, a status key, an error key, and the name of the output queue. For information about the CD formats possible, see the publication IBM OS Full American National Standard COBOL, Order No. GC28-6396.

----

[1]Multiple input and output CD entries may be specified.

Note: The required inclusion of the parameter DATE=YES in all input TPROCESS entries whose destination is a COBOL program results in the placing of the date and time of message entry in the input CD (see the section "Additional Interface Considerations" in the chapter entitled "Using the Teleprocessing Feature").

WORKING-STORAGE SECTION

Separate Modules

In a large program, the programmer should plan ahead for breaking the programs into separately compiled modules, as follows:

1. When employing separate modules, an attempt should be made to combine entries of each Working-Storage Section into a single level-01 record (or one level-01 record for each 32K bytes). Logical record areas can be indicated by use of level-02, level-03 etc., entries. A CALL statement with the USING option is more efficient when a single item is passed than when many level-01 and/or level-77 items are passed. When this method is employed, mistakes are more easily avoided.

2. Areas that do not have VALUE clauses should be separated from areas that do need VALUE clauses. VALUE clauses (except for level-88 items) are invalid in the Linkage Section and the Communication Section.

3. When the Working-Storage Section is one level-01 item with no VALUE clauses, the COPY statement can easily be used to include the item as the description of a Linkage Section in a separately compiled module.

4. See "Use of Segmentation Feature" for more information on how to modularize the Procedure Division of a COBOL program.

Locating the Working-Storage Section in Dumps

When any one or more of the options PMAP, CLIST, and DMAP are specified, both the location and the length (in hexadecimal) of the Working-Storage Section, if any, are provided (see the section "Options for the Compiler" in the chapter entitled "Job Control Procedures").

Alternatively, the programmer may locate this section in object-time dumps by including the following two statements in the program, in the order given:

```
77  FILLER PICTURE X(44), VALUE "PROGRAM
    XXXXXXXX WORKING-STORAGE BEGINS HERE".

01  FILLER PICTURE X(42), VALUE "PROGRAM
    XXXXXXXX WORKING-STORAGE ENDS HERE".
```

These two nonnumeric literals will appear in all dumps of the program, delineating the Working-Storage Section. The program-name specified in the PROGRAM-ID clause should replace the XXXXXXXX in the literal.

DATA DESCRIPTION

The Procedure Division operations that most often require adjustment of data items include the MOVE statement, the IF statement when used in a relation test, and arithmetic operations. Efficient use of data description clauses, such as REDEFINES, PICTURE, and USAGE, avoids the generation of extra code.

REDEFINES Clause

REUSING DATA AREAS: The main storage area can be used more efficiently by writing different data descriptions for the same data area. For example, the coding that follows shows how the same area can be used as a work area for the records of several input files that are not processed concurrently:

```
WORKING-STORAGE SECTION.
01  WORK-AREA-FILE1.
     (largest record description for FILE1)
.
.
.

01  WORK-AREA-FILE2 REDEFINES
    WORK-AREA-FILE1.
     (largest record description for FILE2)
.
.
.
```

ALTERNATE GROUPINGS AND DESCRIPTIONS: Program data can often be described more efficiently by providing alternate groupings or data descriptions for the same data. For example, a program refers to both a field and its subfields, each of which is more efficiently described with a different usage. This can be done with the REDEFINES clause as follows:

```
01  PAYROLL-RECORD.
    05  EMPLOYEE-RECORD PICTURE X(28).
    05  EMPLOYEE-FIELD REDEFINES
        EMPLOYEE RECORD.
        10  NAME          PICTURE X(23).
        10  NUMBERX       PICTURE S9(5) COMP.
    05  DATE-RECORD       PICTURE X(10).
```

As an example of different data descriptions specified for the same data, the following illustrates how a table (TABLEA) can be initialized:

```
    05  VALUE-A.
        10  A1 PICTURE S9(9) COMPUTATIONAL
            VALUE IS ZEROES.
        10  A2 PICTURE S9(9) COMPUTATIONAL
            VALUE IS 1.
        .
        .
        .

        10  A100 PICTURE S9(9)
            COMPUTATIONAL VALUE IS 99.

    05  TABLEA REDEFINES VALUE-A
        PICTURE S9(9) COMPUTATIONAL
         OCCURS 100 TIMES.
```

Note: Caution should be exercised when redefining a subscript; for if the value of the redefining data item is changed in the Procedure Division, no new calculation for the subscript is performed.

PICTURE Clause

DECIMAL-POINT ALIGNMENT: Procedure Division operations are most efficient when the decimal positions of the data items involved are aligned. If they are not, the compiler generates instructions to align the decimal positions before any operations involving the data items can be executed. This is referred to as scaling.

Assume, for example, that a program contains the following instructions:

```
WORKING-STORAGE SECTION.
77  A PICTURE S999V99.
77  B PICTURE S99V9.
.
.
.

PROCEDURE DIVISION.
.
.
    ADD A TO B.
```

Time and internal storage space are saved by defining B as:

```
77 B PICTURE S99V99.
```

If it is inefficient to define B differently, a one-time conversion can be done, as explained in "Data Format Conversion."

FIELDS OF UNEQUAL LENGTH: When a data item is moved to another data item of a different length, the following should be considered:

- If the items are external decimal items, the compiler generates instructions to insert zeros in the high-order positions of the receiving field when it is the larger.

- If the items are nonnumeric, the compiler generates instructions to insert spaces in the low-order positions of the receiving field (or the high-order positions if the JUSTIFIED RIGHT clause is specified. This generation of extra instructions can be avoided if the sending field is described with a length equal to or greater than the receiving field.

Use of Sign: The absence or presence of a plus or minus sign in the description of an arithmetic field often can affect the efficiency of a program. The following paragraphs discuss some of the considerations.

Decimal Items: The sign position in an internal or external decimal item can contain:

1. A plus or minus sign. If S is specified in the PICTURE clause, a plus or minus sign is inserted when either of the following conditions prevails:

    a. The item is in the Working-Storage Section and a VALUE clause has been specified.

    b. A value for the item is assigned as a result of an arithmetic operation during execution of the program.

    If an external decimal item is punched, printed, or displayed, an overpunch will appear in the low-order digit. In EBCDIC, the configuration for low-order zeros normally is a nonprintable character. Low-order digits of positive values will be represented by one of the letters A through I (digits 1 through 9); low-order digits of negative values will be represented by one of the letters J through R (digits 1 through 9).

2. A hexadecimal F. If S is not specified in the PICTURE clause, an F is inserted in the sign position when either of following conditions exists:

    a. The item is in the Working-Storage Section and a VALUE clause has been specified.

    b. A value for the item is developed during the execution of the program.

    An F is treated as positive, but is not an overpunch.

3. An invalid configuration. If an internal or external decimal item contains an invalid configuration in the sign position, and if the item is involved in a Procedure Division operation, the program will be abnormally terminated.

Items for which no S has been specified (unsigned items) are treated as absolute values. Whenever a value (signed or unsigned) is stored in, or moved in an elementary move to an unsigned item, a hexadecimal F is stored in the sign position of the unsigned item. For example, if an arithmetic operation involves signed operands and an unsigned result field, compiler-generated code will insert an F in the sign position of the result field when the result is stored.

For internal and external decimal items used as input, it is the user's responsibility to ensure that the input data is valid. The compiler does not generate a test to ensure that the configuration in the sign position is valid.

When a group item is involved in a move, the data is moved without regard to the level structure of the group items involved. The possibility exists that the configuration in the sign position of a subordinate numeric item may be destroyed. Therefore, caution should be exercised in moves involving group items with subordinate numeric fields or with other group operations such as READ or ACCEPT.

SIGN Clause

This clause, which specifies both the position and the mode of the operational sign for a numeric data description entry, is required only when an explicit description of the sign's properties is necessary. The SIGN clause may be specified for either a numeric data description entry whose PICTURE contains the character S or a group item that contains at least one such numeric data description entry.

The numeric data description entries to which the SIGN clause applies must be described, implicitly or explicitly, as USAGE IS DISPLAY. Only one SIGN clause may be associated with any given numeric data description entry.

The format of the SIGN clause is as follows:

$$\underline{\text{SIGN}}\ \underline{\text{IS}}\ \left\{\begin{matrix}\text{LEADING}\\ \text{TRAILING}\end{matrix}\right\}\ [\underline{\text{SEPARATE}}\ \text{CHARACTER}]$$

Use of the SEPARATE CHARACTER Option: The programmer can elect to consider the character S in the PICTURE character string as a separate character or not, as he chooses. If the SEPARATE CHARACTER option is specified:

- The position of the character S is not taken to be a digit position.
- The character S is counted in determining the size of the data item.
- The characters '+' and '-' are used for the positive and the negative operational signs, respectively.
- If neither the character '+' nor the character '-' is present in the data at object time, an error takes place and the program ABENDS.

Whether or not the SEPARATE CHARACTER option is in effect, the operational sign is assumed to be associated with either the LEADING or the TRAILING digit position, as specified, of the elementary numeric data item.

Table 25. Data Format Conversion

Usage	Bytes Required	Boundary Alignment Required	Typical Use	Converted for Arithmetic Operations	Special Characteristics
DISPLAY (external decimal)	1 per digit (except for V)	No	Input from cards, output to cards, listings	Yes	May be used for numeric fields up to 18 digits long. Fields over 15 digits require extra instructions if used in computations.
DISPLAY (external floating point)	1 per character (except for V)	No	Input from cards, output to cards, listings	Yes	Converted to COMPUTATIONAL-2 format via COBOL library subroutine.
COMP-3 (internal decimal)	1 byte per 2 digits plus 1 byte for the low-order digit and sign	No	Input to a report item Arithmetic fields Work areas	Sometimes when a small COMP-3 item is used with a small COMP item.	Requires less space than DISPLAY. Convenient form for decimal alignment. Can be used in arithmetic computations without conversion. Fields over 15 digits require a subroutine when used in computations.
COMP (binary)	2 if 1≤N≤4 4 if 5≤N≤9 8 if 10≤N≤18 where N is the number of 9s in the PICTURE clause	halfword fullword fullword	Subscripting Arithmetic fields	Sometimes for both mixed and unmixed usages	Rounding and testing for the ON SIZE ERROR condition are cumbersome if calculated result is greater than 9(9). Extra instructions are generated for binary computations if the SYMCHRONIZED clause is not specified. Fields of over 9 digits require more handling.
COMP-1 (internal floating point)	4 (short-precision)	fullword	Fractional exponentiation	No	Tends to produce less accuracy if more than 17 significant digits are required and if the exponent is big. Requires floating-point feature. Extra instructions are generated if the SYNCHRONIZED clause is not specified.
COMP-2 (internal floating point)	8 (long-precision)	double-word	Fractional exponentiation when more precision is required	No	Same as COMPUTATIONAL-1

USAGE Clause

This clause should be written at the highest level possible.

DATA FORMAT CONVERSION: Operations involving mixed, elementary numeric data formats require conversion to a common format. This usually means that additional storage is used and execution time is increased. The code generated must often move data to an internal work area, perform any necessary conversion, and then execute the indicated operation. Often, too, the result may have to be converted in the same way (see Table 25).

If it is impractical to use the same data formats throughout a program, and if two data items of different formats are frequently used together, a one-time conversion can be effected. For example, if A is defined as a COMPUTATIONAL item and B as a COMPUTATIONAL-3 item, A can be moved to a work area that has been defined as COMPUTATIONAL-3. This move causes the data in A to be converted to COMPUTATIONAL-3. Whenever A and B are used in a Procedure Division operation, reference can be made to the work area rather than to A. Using this technique, the conversion is performed only once, instead of each time an operation is performed.

The following eight cases show how data conversions are handled on mixed elementary items for names, data comparisons, and arithmetic operations. Moves to and from group items, without the CORRESPONDING option, as well as comparisons involving group items, are done without conversion.

Numeric DISPLAY to COMPUTATIONAL-3:

To Move Data: Converts DISPLAY data to COMPUTATIONAL-3 data.

To Compare Data: Converts DISPLAY data to COMPUTATIONAL-3 data.

To Perform Arithmetic Operations: Converts DISPLAY data to COMPUTATIONAL-3 data.

Numeric DISPLAY to COMPUTATIONAL:

To Move Data: Converts DISPLAY data to COMPUTATIONAL-3 data and then to COMPUTATIONAL data.

To Compare Data: Converts DISPLAY to COMPUTATIONAL-3 and then to COMPUTATIONAL or converts both DISPLAY and COMPUTATIONAL data to COMPUTATIONAL-3 data.

To Perform Arithmetic Operations: Converts DISPLAY data to COMPUTATIONAL-3 or COMPUTATIONAL data.

COMPUTATIONAL-3 to COMPUTATIONAL:

To Move Data: Moves COMPUTATIONAL-3 data to a work field and the converts COMPUTATIONAL-3 data to COMPUTATIONAL data.

To Compare Data: Converts COMPUTATIONAL data to COMPUTATIONAL-3 or vice versa, depending on the size of the field.

To Perform Arithmetic Operations: Converts COMPUTATIONAL data to COMPUTATIONAL-3 or vice versa, depending on the size of the field.

COMPUTATIONAL to COMPUTATIONAL-3:

To Move Data: Converts COMPUTATIONAL data to COMPUTATIONAL-3 data in a work field, then moves the work field.

To Compare Data: Converts COMPUTATIONAL to COMPUTATIONAL-3 data or vice versa, depending on the size of the field.

To Preform Arithmetic Operations: Converts COMPUTATIONAL to COMPUTATIONAL-3 data or vice versa, depending on the size of the field.

COMPUTATIONAL to Numeric DISPLAY:

To Move Data: Converts COMPUTATIONAL data to COMPUTATIONAL-3 data and then to DISPLAY data.

To Compare Data: Converts DISPLAY to COMPUTATIONAL or both COMPUTATIONAL and DISPLAY data to COMPUTATIONAL-3 data, depending on the size of the field.

To Perform Arithmetic Operations: Depending on the size of the field, converts DISPLAY data to COMPUTATIONAL data, or both DISPLAY and COMPUTATIONAL data to COMPUTATIONAL-3 data in which case the result is generated in a COMPUTATIONAL-3 work area and then converted and moved to the DISPLAY result field.

COMPUTATIONAL-3 to Numeric DISPLAY:

To Move Data: Converts COMPUTATIONAL-3 data to DISPLAY data.

To Compare Data: Converts DISPLAY data to COMPUTATIONAL-3 data. The result is generated in a COMPUTATIONAL-3 work area and is then converted and moved to the DISPLAY result field.

Numeric DISPLAY to Numeric DISPLAY:

To Perform Arithmetic Operations: Converts all DISPLAY data to COMPUTATIONAL-3 data. The result is generated in a COMPUTATIONAL-3 work area and is then converted to DISPLAY and moved to the DISPLAY result field.

External Floating-Point to Any Other: When an external floating-point item is to be used in an airthmetic operation or in data manipulation, precision errors may occur due to required conversions.

Internal Floating-Point to Any Other: When an item described as COMPUTATIONAL-1 or COMPUTATIONAL-2 (internal floating-point) is used in an operation with another data format, the item in the other data format is always converted to internal floating-point. If necessary, the internal floating-point result is then converted to the format of the other data item.

SYNCHRONIZED Clause

DATA FORMATS: As shown in Table 24, COMPUTATIONAL, COMPUTATIONAL-1, and COMPUTATIONAL-2 items have specific boundary alignment requirements. To ensure correct alignment, either the programmer or the compiler may have to add slack bytes, or the compiler must generate instructions to move the item to a correctly aligned work area when reference is made to the item.

The SYNCHRONIZED clause may be used at the elementary level to specify the automatic alignment of elementary items on their proper boundaries or at level-01 to synchronize all elementary items within the group. For COMPUTATIONAL items, if the PICTURE is in the range of S9 through S9(4), the item is aligned on a halfword boundary. If the PICTURE is in the range of S9(5) through S9(18), the item is aligned on a fullword boundary. For COMPUTATIONAL-1 items, the item is aligned on a fullword boundary. For COMPUTATIONAL-2 items, the item is aligned on a double-word boundary. The SYNCHRONIZED clause and slack bytes are fully discussed in the publication IBM OS Full American National Standard COBOL.

Special Considerations for DISPLAY and COMPUTATIONAL Fields

NUMERIC DISPLAY FIELDS: Zeros are not inserted into numeric DISPLAY fields by the instruction set. When numeric DISPLAY data is moved, the compiler generates instructions that insert any necessary zeros into the DISPLAY fields. When numeric DISPLAY data is compared, and one field is smaller than the other, the compiler generates instructions to move the smaller item to a work area where zeros are inserted.

COMPUTATIONAL FIELDS: COMPUTATIONAL fields can be aligned on either a halfword or fullword boundary. If an operation involves COMPUTATIONAL fields of different lengths, the halfword field is automatically expanded to a fullword field. Therefore, mixed halfword and fullword fields require no additional operations.

COMPUTATIONAL-1 AND COMPUTATIONAL-2 FIELDS: If an arithmetic operation involves a mixture of short-precision and long-precision fields, the compiler generates instructions to expand the short-precision field to a long-precision field before the operation is executed.

COMPUTATIONAL-3 FIELDS: The compiler does not have to generate instructions to insert high-order zeros for ADD and SUBTRACT statements that involve COMPUTATIONAL-3 data. The zeros are inserted by the instruction set.

Data Formats in the Computer

The various COBOL data formats and how they appear in the computer in EBCDIC (Extended Binary-Coded-Decimal Interchange Code) format are illustrated by the following examples. More detailed information about these data formats appears in the publication IBM OS Principles of Operation, Order No. A22-6821.

Numeric DISPLAY (External Decimal): Suppose the value of an item is -1234, and the PICTURE and USAGE are:

PICTURE 9999 DISPLAY.

or

PICTURE S9999 DISPLAY.

The item appears in the computer in the following forms respectively:

```
| F1 | F2 | F3 | F4 |
└────┴────┴────┴────┘
                \__/
                Byte

| F1 | F2 | F3 | D4 |
└────┴────┴────┴────┘
                \__/
                Byte
```

Hexadecimal F is treated arithmetically as plus in the low-order byte. The hexadecimal character D represents a negative sign.

COMPUTATIONAL-3 (Internal Decimal): Suppose the value of an item is +1234, and its PICTURE and USAGE are:

PICTURE 9999 COMPUTATIONAL-3.

or

PICTURE S9999 COMPUTATIONAL-3.

The item appears in the computer in the following forms, respectively:

```
| 01 | 23 | 4F |
└────┴────┴────┘
           \__/
           Byte

| 01 | 23 | 4C |
└────┴────┴────┘
           \__/
           Byte
```

Hexadecimal F is treated arithmetically as positive. The hexadecimal character C represents a plus sign.

Note: Since the low-order byte of an internal decimal number always contains a sign field, an item with an odd number of digits can be stored more efficiently than an item with an even number of digits. Note that a leading zero is inserted in the foregoing example.

COMPUTATIONAL (Binary): Suppose the value of an item is 1234, and its PICTURE and USAGE are:

PICTURE S9999 COMPUTATIONAL.

The item appears in the computer in the following form:

```
| 0000 | 0100 | 1101 | 0010 |
└──────┴──────┴──────┴──────┘
 ↑
 sign
position
```

A 0-bit in the sign position means the number is positive. Negative numbers are represented in two's complement form; thus, the sign position of a negative number will always contain a 1-bit.

For example -1234 would appear as follows:

```
| 1111 | 1011 | 0010 | 1110 |
└──────┴──────┴──────┴──────┘
 ↑
 sign
position
```

Binary Item Manipulation: A binary item is allocated storage ranging from one halfword to two words, depending on the number of 9s in its PICTURE. Table 26 is an illustration of how the compiler allocates this storage. Note that it is possible for a value larger than that implied by the PICTURE to be stored in the item. For example, PICTURE S9(4) implies a maximum value of 9,999, although it could actually hold the number 32,767.

Because most binary items are manipulated according to their allotted storage capacity, the programmer can ignore this situation. For the following reasons, however, there are some cases where he must be careful of his data:

1. When the ON SIZE ERROR option is used, the size test is made on the basis of the maximum value allowed by the picture of the result field. If a size error condition exists, the value of the result field is not altered and control is given to the imperative statements specified by the error option.

2. When a binary item is displayed or exhibited, the value used is a function of the number of 9s specified in the PICTURE clause.

3. When the actual value of a positive number is significantly larger than its picture value, a 1 could result in the sign position of the item, causing the item to be treated as a negative number in subsequent operations.

Table 27 illustrates three binary manipulations. In each case, the result field is an item described as PICTURE S9 COMPUTATIONAL. One halfword of storage has been allocated; and no ON SIZE ERROR option is involved. Note that if the ON SIZE ERROR option had been specified, it would have been executed for cases B and C.

Table 26. Relationship of PICTURE to Storage Allocation

PICTURE	Maximum Working Value	Assigned Storage
S9 through S9(4)	32,767	one halfword
S9(5) through S9(9)	2,147,483,647	one fullword
S9(10) through S9(18)	9,223,372,036,854,775,807	two fullwords

Table 27. Treatment of Varying Values in a Data Item of PICTURE S9

Case	Hexadecimal Result of Binary Calculation	Decimal Equivalent	Actual Decimal Value in Halfword of Storage	Display or Exhibit Value
A	0008	8	+8	8
B	000A	10	+10	0
C	C350	50000	-15536	6

COMPUTATIONAL-1 or COMPUTATIONAL-2 (Floating Point): Suppose the value of an item is +1234, and that its USAGE is COMPUTATIONAL-1, the item appears in the computer in the following form:

```
|0|100  0011|0100 1101 0010 0000 0000 0000|
└─┴─────────┴─────────────────────────────┘
 S 1       7 8                          31
```

S is the sign position of the number.

A 0-bit in the sign position indicates that the sign is plus.

A 1-bit in the sign position indicates that the sign is minus.

Bits 1 through 7 are the exponent (characteristic) of the number.

Bits 8 through 31 are the fraction (mantissa) of the number.

This form of data is referred to as floating-point. The example illustrates short-precision floating-point data (COMPUTATIONAL-1). In long-precision (COMPUTATIONAL-2), the fraction length is 56 bits. (For a detailed explanation of floating-point representation, see the publication IBM OS Principles of Operation.)

PROCEDURE DIVISION

A program can often be made more efficient or easier to debug in the Procedure Division with some of the techniques described below.

MODULARIZING THE PROCEDURE DIVISION

When the Procedure Division is modularized, programs are easier to maintain and document. In addition, modularization makes it simple to break down a program using the segmentation feature, thereby resulting in a more efficient segmented program. Modularization of the Procedure Division involves organizing it into at least three functional levels: a main-line routine, processing subroutines, and input/output subroutines.

Main-Line Routine

This routine should be short, simple, and contain all the major logical decisions of the program. This routine controls which second-level subroutines are executed and in what order. All second-level subroutines should be invoked from the main-line routine by PERFORM statements.

Processing Subroutines

These should be broken down into as many functional levels as necessary, depending on the complexity of the program. These must be completely closed subroutines, with one entry point and one exit point. The entry point should be the first statement of the subroutine. The exit point should be the EXIT statement. The processing subroutines can perform only lower level subroutines; return to the higher level subroutine (processing subroutine) must be made by a GO TO statement, which references the EXIT statement.

Input/Output Subroutines

These should be the lowest level subroutines, since all higher level subroutines should have access to them. There should be one OPEN subroutine and one CLOSE subroutine for the program, and only one functional (READ or WRITE) subroutine for each file. One READ or WRITE subroutine per file, which is always performed, has several advantages:

1. Coding can be added to count records on a file, transform blanks into zeros, check for 9s padding, etc.
2. Input and output files can be reformatted without changing the logic of the program.
3. DEBUG statements can be added during testing to create input or to DISPLAY formatted output, instead of having to create a test file.

INTERMEDIATE RESULTS

The compiler treats arithmetic statements as a succession of operations and sets up intermediate result fields to contain the results of these operations. Examples of such statements are the arithmetic statements, and statements containing arithmetic expressions. In the publication IBM OS Full American National Standard COBOL, the section "Appendix A: Intermediate Results" describes the algorithms used by the compiler to determine the number of places reserved for intermediate result fields.

Intermediate Results and Binary Data Items

If an operation involving binary operands requires an intermediate result greater than 18 digits, the compiler converts the operands to internal decimal before performing the operation. If the result field is binary, the result will be converted from internal decimal to binary.

If an intermediate result will not be greater than nine digits, the operation is performed most efficiently on binary data fields.

Intermediate Results and COBOL Library Subroutines

If a decimal multiplication operation requires an intermediate result greater than 30 digits, a COBOL library subroutine is used to perform the multiplication. The result of this multiplication is then truncated to 30 digits.

A COBOL library subroutine is used to perform division if:

1. the scaled divisor is equal to or greater than 15 digits.
2. the length of the scaled divisor plus the length of the scaled dividend is greater than 16 bytes. The lengths of the operands are internal decimal.
3. the scaled dividend is greater than 30 digits. (A scaled dividend is a number that has been multiplied by a power of ten in order to obtain the desired number of decimal places in the quotient.)

Intermediate Results Greater than 30 Digits

Whenever the number of digits in a decimal intermediate result is greater than 30, the field is truncated to 30 digits. A warning message will be generated at compile time, and program flow will not be interrupted at execution time. This truncation may cause a result to be incorrect.

If binary or internal decimal data is in accord with its data description, no interrupt can occur because of an overflow condition in an intermediate result. This is due to the truncation described in the preceding paragraph.

If the possibility exists that an intermediate result field may exceed 30 digits, truncation can be avoided by the specification of floating-point operands (COMPUTATIONAL-1 or COMPUTATIONAL-2); however, accuracy may not be maintained.

Intermediate Results and Floating-Point Data Items

If a floating-point operand has an intermediate result field in which exponent overflow occurs, the job will be abnormally terminated.

Intermediate Results and the ON SIZE ERROR Option

The ON SIZE ERROR option applies only to the final calculated results and not to intermediate result fields.

VERBS

CALL Statement

The CALL statement permits communication between a COBOL object program and one or more COBOL subprograms or other language subprograms. A called program may be entered either at the beginning of the Procedure Division or later in the program. When a subprogram is called, it may already be core resident and be link-edited with the main program, or it may be specified as dynamic and link-edited into a separate load module. Dynamic loading, via the CALL statement, enables the user to load a subprogram only when it is actually needed.

The first dynamic call to a subprogram brings in a fresh copy of that subprogram. Any subsequent calls to the same subprogram, by either the original caller or another subprogram in the same region/partition, cause a branch to the same copy of the subprogram in its last-used state, until the user deletes it (see the section on the "CANCEL Statement").

For examples of both static and dynamic CALL statements, see the section "Dynamic Subprogram Considerations" in the chapter entitled "Calling and Called Programs."

CANCEL Statement

The CANCEL statement permits dynamic deletion of COBOL subprograms from the COBOL processing environment. That is, a CANCEL statement issued for a subprogram that has been dynamically loaded causes the storage occupied by the subprogram to be freed. As a result, a subsequent call to the subprogram functions as if it were the first.

```
CANCEL CALLED-PROGRAM.
```

Note: A program other than the original caller may issue a CANCEL statement referring to a called program.

CLOSE Statement

There are two ways in which to use the CLOSE statement when closing several files:

```
CLOSE DETAIL-FILE MASTER-FILE.
```

or

```
CLOSE DETAIL-FILE.
CLOSE MASTER-FILE.
```

Each CLOSE statement for a file requires the use of a storage area that is directly proportional to the number of files being closed. Closing more than one file with the same statement is faster than when using a separate statement for each file. However, separate statements require less storage.

COMPUTE Statement

The use of the COMPUTE statement generates more efficient coding than does the use of individual arithmetic statements because the compiler can keep track of internal work areas and does not have to store the results of intermediate calculations. It is the user's responsibility, however, to insure that the data is defined with the level of significance required in the answer.

IF Statement

Nested and compound IF statements should be avoided as the logic is difficult to debug.

Performing an IF operation for an item greater than 256 bytes in length requires the generation of more instructions than are required for that of an IF operation for an item of 256 bytes or less.

Note: In teleprocessing applications, the MESSAGE condition determines whether or not one or more complete messages exist in a designated queue of messages. The COBOL programmer can include in an IF statement the condition:

```
[NOT] MESSAGE FOR cd-name
```

with an action to be performed when the condition is met.

When using compound IF statements, care must be taken to ensure that the message condition is tested, so that the QUEUE DEPTH field is actually updated. For example, according to the statement:

```
IF A = B AND MESSAGE FOR QUEUE-IN....
```

then when A is not equal to B, the message condition is not tested and the QUEUE DEPTH field for QUEUE-IN is not updated. (For further discussion of the message condition, see the publication IBM OS Full American National Standard COBOL.)

MOVE Statement

Performing a MOVE operation for an item greater than 256 bytes in length requires the generation of more instructions than are required for that of a MOVE operation for an item of 256 bytes or less.

When a MOVE statement with the CORRESPONDING option is executed, data items are considered CORRESPONDING only if their respective data names are the same, including all implied qualification, up to, but not including, the data-names used in the MOVE statement itself.

For example,

```
01  AA                        01  XX
    05  BB                        05  BB
        10  CC                        10  CC
        10  DD                        10  DD
    05  EE                        05  YY
        10  FF                        10  FF
```

The statement MOVE CORRESPONDING AA TO XX will result in moving CC and DD but not FF because FF of EE does not correspond to FF of YY).

Note: The other rules for MOVE CORRESPONDING, of course, must still be satisfied.

NOTE Statement

An asterisk (*) should be used in place of the NOTE statement, because there is the possibility that when NOTE is the first sentence in a paragraph, it will inadvertently cause the whole paragraph to be treated as part of the NOTE.

OPEN Statement

There are two ways in which to use the OPEN statement when opening several files:

```
OPEN INPUT INFILE UPDATES OUTPUT OUTFILE
```

or

```
OPEN INPUT INFILE
OPEN INPUT UPDATES
OPEN OUTPUT OUTFILE
```

Each OPEN statement for a file requires the use of a storage area that is directly proportional to the number of files being opened. Opening more than one file with the same statement is faster than using a separate statement for each file. However, separate statements require less storage.

PERFORM Verb

PERFORM is a useful verb if the programmer adheres to the following rules:

1. Always execute the last statement of a series of routines being operated on by a PERFORM statement. When branching out of the routine, make sure control will eventually return to the last statement of the routine. This statement should be an EXIT statement. Although no code is generated, the EXIT statement allows a programmer to recognize immediately the extent of a series of routines within the range of a PERFORM statement.

2. Always either PERFORM routine-name THRU routine-name-exit, or PERFORM section-name. A PERFORM

paragraph-name can cause trouble for the programmer trying to maintain the program. For example, if a paragraph must be broken into two paragraphs, the programmer must examine every statement to determine whether or not this paragraph is within the range of the PERFORM statement. Then all statements referencing the paragraph-name must be changed to PERFORM THRU statements.

READ INTO and WRITE FROM Options

Use READ INTO and WRITE FROM, and do all processing in the Working-Storage Section. This is suggested for two reasons:

1. Debugging is much simpler. Working-Storage areas are easier to locate in a dump than are buffer areas. And, if files are blocked, it is much easier to determine which record in a block was being processed when the abnormal termination occurred.

2. Trying to access a record area after the AT END condition has occurred (for example, AT END MOVE HIGH-VALUE TO INPUT-RECORD) can cause problems if the record area is only in the File Section.

Note: The programmer should be aware that additional time is used to execute the move operation involved in each READ INTO or WRITE FROM instruction.

RECEIVE Statement

The RECEIVE statement makes available to the COBOL program a message, a message segment, or part of a message or message segment, as well as information about that message from a queue maintained by the message control program (MCP). The following example of the RECEIVE statement is taken from the sample COBOL teleprocessing program shown in Figure 114:

```
RECEIVE CDNAME-IN MESSAGE INTO IDENT-REC.
```

SEND Statement

Specification of the SEND statement in the COBOL program causes a message, a message segment, or part of a message or message segment to be released to the message control program (MCP). The following example of the SEND statement is taken from the sample COBOL teleprocessing program shown in Figure 113:

```
SEND CDNAME-OUT FROM IDENT-SEND WITH EMI.
```

Notes:

- Although the COBOL program has access to a message only when the MCP has received it in entirety and placed it in a queue, once several messages have met this requirement the COBOL program can process messages from different MCP queues at the same time.

- If one execution of a RECEIVE statement (or a SEND statement) transmits only part of a message, subsequent executions of RECEIVE statements (or SEND statements) in that run unit are required for transmission of the rest of the message.

START Statement

The START statement must be executed before the READ statement for a given record if either of the following is true:

- Processing begins with other than the first record;

- Processing continues with a record other than the next sequential record.

There are two ways to use the START statement to begin processing a segment of a sequentially accessed indexed file at a specified key. The programmer may indicate either Method 1, to begin at a specific NOMINAL KEY that matches a RECORD KEY within the file, or Method 2, to start within the first record in a specific generic key class.

Method 1:

```
START file-name
     [INVALID KEY imperative-statement]
```

Method 2:

```
START file-name USING KEY data-name
     {EQUAL TO}
     {   =    } identifier
     [INVALID KEY imperative-statement]
```

where data-name is the data-name given in the RECORD KEY clause and identifier contains the generic key value for the request and may be any data item whose length is less than or equal to that of the RECORD KEY.

Note: For ISAM a problem may result with the generic key facility with binary key if the low-order byte of the search argument is binary zero.

STRING Statement

The STRING statement combines two or more subfields into a single field. When this statement is executed, characters from the sending item(s) are transferred to the receiving item in the same way that moves from alphanumeric to alphanumeric item(s) are effected. The example in Figure 66 illustrates the use of the STRING statement options available to the user. For a discussion of the formats possible with the STRING statement, see the publication *OS Full American National Standard COBOL*.

TRANSFORM Statement

The TRANSFORM statement generates more efficient code than the EXAMINE statement with the REPLACING BY option when only one character is being transformed. TRANSFORM, however, uses a 256-byte table.

UNSTRING Statement

The UNSTRING statement separates contiguous data in a sending field, placing it in multiple receiving fields. The example in Figure 68 illustrates the use of the UNSTRING statement options available to the user.

For a discussion of the formats possible with the UNSTRING statement, see the publication *IBM OS Full American National Standard COBOL*.

```
    STRING SNDFLD5 DELIMITED BY DLMTR
              SNDFLD6 DELIMITED BY SIZE

 *  Combine data in SNDFLD5 up to the delimiter indicated by DLMTR with all the data
 *  in another sending field (as indicated by the SIZE option of the STRING
 *  statement).

              INTO RCDFLD1 WITH POINTER POINTR

 *  Place the result in RCDFLD1 beginning at the relative location designated
 *  by POINTR.

              ON OVERFLOW GO TO OVERFLOW2.

 *  If RCDFLD1 is not large enough to accommodate the combined data-fields, or
 *  if the original contents of the pointer field were less than 1, execute a user-
 *  written checking routine called OVERFLOW2.
```
Figure 67. Using the STRING Statement

```
    UNSTRING SNDFLD

*   Separate the data in the sending area.

              DELIMITED BY DLMTR1
                 OR SPACES
                 OR ALL 'E'
              INTO RCFLD

*   When the character, or set of characters, marking the end of a section of the
*   sending area is found, move the isolated data into the data-receiving field.

              DELIMITER IN DELIM-IN

*   Move the delimiter found into the delimiter-receiving area DELIM-IN.

              COUNT IN COUNT-IN

*   Specify in COUNT-IN the number of characters placed in the RCFLD
*   data-receiving field.

              WITH POINTER POUNTR

*   Indicate the relative position in the SNDFLD sending area of the first
*   character to be examined.  At the end of the operation, POINTR contains a value
*   equal to the initial value plus the number of characters examined in the sending
*   field.

              TALLYING IN TALLY-IN

*   Record the number of data-receiving areas acted upon.  At the end of the
*   operation, TALLY-IN will contain a value equal to the initial value plus the
*   number of receiving areas acted upon.

              ON OVERFLOW
                 DISPLAY 'OVERFLOW CONDITION'
                 GO TO CHECK-ROUTINE.

*   If the data-receiving fields cannot accommodate the data being sent, or if
*   the original value of the pointer was less than 1 or greater than the size of the
*   sending field, execute a user-written checking routine.
```

Figure 68. Using the UNSTRING Statement

USING THE REPORT WRITER FEATURE

REPORT Clause in FD

A given report-name may appear in a maximum of two file description entries. The file description entries need not have the same characteristics. If the same report-name is specified in two file description entries, the report will be written on both files. For example:

```
ENVIRONMENT DIVISION.
    SELECT FILE-1  ASSIGN UR-1403-S-PRTOUT.
    SELECT FILE-2  ASSIGN UT-2400-S-SYSUT1.
    .
    .
    .
DATA DIVISION.
FD  FILE-1  RECORDING MODE F
            RECORD CONTAINS 121 CHARACTERS
            REPORT IS REPORT-A.
FD  FILE-2  RECORDING MODE V
            RECORD CONTAINS 101 CHARACTERS
            REPORT IS REPORT-A.
```

For each GENERATE statement, the records for REPORT-A will be written on FILE-1 and FILE-2, respectively. The records on FILE-2 will not contain columns 102 through 121 of the corresponding records on FILE-1.

Summing Technique

The object program can be made more efficient with respect to execution time by keeping in mind the fact that Report Writer source coding is treated as though the programmer had written the program in COBOL without the Report Writer feature. Therefore, a complex source statement or series of statements will generally be executed faster than simple statements that perform the same function. The example below shows two coding techniques for the Report Section of the Data Division. Method 2 uses the more complex statements.

```
RD...CONTROLS ARE YEAR MONTH WEEK DAYY
```

Method 1:

```
01   TYPE CONTROL FOOTING YEAR.
     05   SUM COST.
01   TYPE CONTROL FOOTING MONTH.
     05   SUM COST.
01   TYPE CONTROL FOOTING WEEK.
     05   SUM COST.
01   TYPE CONTROL FOOTING DAYY.
     05   SUM COST.
```

Method 2:

```
01   TYPE CONTROL FOOTING YEAR.
     05   SUM A.
01   TYPE CONTROL FOOTING MONTH.
     05   A SUM B.
01   TYPE CONTROL FOOTING WEEK.
     05   B SUM C.
01   TYPE CONTROL FOOTING DAYY.
     05   C SUM COST.
```

Method 2 will execute faster. One addition will be performed for each day, one more for each week, and one for each month. In Method 1, four additions will be performed for each day.

Use of SUM

Unless each identifier is the name of a SUM counter in a TYPE CONTROL FOOTING report group at an equal or lower position in the control hierarchy, the identifier must be defined in the File, Working-Storage or Linkage Sections, as well as in a TYPE DETAIL report group as a SOURCE item. A SUM counter is algebraically incremented just before presentation of the TYPE DETAIL report group in which the item being summed appears as a source item or the item being summed appeared in a SUM clause that contained an UPON option for this DETAIL report group. This is known as SOURCE-SUM correlation. In the following example, SUBTOTAL is incremented only when DETAIL-1 is generated:

```
FILE SECTION.
        .
        .
        .
    05  NO-PURCHASES        PICTURE 99.
        .
        .
        .
REPORT SECTION.
01  DETAIL-1 TYPE DETAIL.
    05  COLUMN 30           PICTURE 99 SOURCE
        NO-PURCHASES.
        .
        .
        .
01  DETAIL-2 TYPE DETAIL.
        .
        .
        .
01  DAY TYPE CONTROL FOOTING
    LINE PLUS 2.
        .
        .
        .
    05  SUBTOTAL COLUMN 30 PICTURE 999
        SUM NO-PURCHASES.
        .
        .
        .
01  MONTH TYPE CONTROL FOOTING
    LINE PLUS 2 NEXT GROUP
    NEXT PAGE.
```

SUM Routines

A SUM routine is generated by the Report Writer for each DETAIL report group of the report. The operands included for summing are determined as follows:

1. The SUM operand(s) also appears in a SOURCE clause(s) for the DETAIL report group.

2. The UPON detail-name option was specified in the SUM clause. In this case, all the operands are included in the SUM routine for only that DETAIL report group, even if the operand appears in a SOURCE clause in other DETAIL report groups.

When a GENERATE detail-name statement is executed, the SUM routine for that DETAIL report group is executed in its logical sequence. When a GENERATE report-name statement is executed and the report contains more than one DETAIL report group, the SUM routine is executed for each one. The SUM routines are executed in the sequence in which the DETAIL report groups are specified.

The following examples show the SUM routines that are generated by the Report

Writer. Example 1 illustrates how operands are selected for inclusion in the routine on the basis of simple SOURCE-SUM correlation. Example 2 illustrates how operands are selected when the UPON detail-name option is specified.

EXAMPLE 1: The following statements are coded in the Report Section:

```
   01  DETAIL-1 TYPE DE...
       05 ...   SOURCE A.
                .
                .
                .
   01  DETAIL-2 TYPE DE...
       05 ...   SOURCE B.
       05 ...   SOURCE C.
                .
                .
                .
   01  DETAIL-3 TYPE DE...
       05 ...   SOURCE B.
                .
                .
                .
   01  TYPE CF...
       05 SUM-CTR-1...SUM A, B, C.
                .
                .
                .
   01  TYPE CF...
       05 SUM-CTR-2...SUM B.
```

One SUM routine is generated for each DETAIL report group, as follows:

SUM Routine for DETAIL-1

```
   REPORT-SAVE
      ADD A TO SUM-CTR-1.
   REPORT-RETURN
```

SUM Routine for DETAIL-2

```
   REPORT-SAVE
      ADD B TO SUM-CTR-1.
      ADD C TO SUM-CTR-1.
      ADD B TO SUM-CTR-2.
   REPORT-RETURN
```

SUM Routine for DETAIL-3

```
   REPORT-SAVE
      ADD B TO SUM-CTR-1.
      ADD B TO SUM-CTR-2.
   REPORT-RETURN
```

EXAMPLE 2: In this example, the same coding is used as in Example 1, with one exception: the UPON detail-name option is used for SUM-CTR-1, as follows:

```
   01  TYPE CF...
       05  SUM-CTR-1...SUM A, B, C UPON
           DETAIL-2.
```

The following SUM routines would then be generated instead of those resulting from the calculations in Example 1.

SUM Routine for DETAIL-1

```
   REPORT-SAVE
   REPORT-RETURN
```

SUM Routine for DETAIL-2

```
   REPORT-SAVE
      ADD A TO SUM-CTR-1.
      ADD B TO SUM-CTR-1.
      ADD C TO SUM-CTR-1.
      ADD B TO SUM-CTR-2.
   REPORT-RETURN
```

SUM Routine for DETAIL-3

```
   REPORT-SAVE
      ADD B TO SUM-CTR-2.
   REPORT-RETURN
```

Output Line Overlay

The Report Writer output line is put together with an internal REDEFINES specification, indexed by integer-1. No check is made to prevent overlay on any line. For example:

```
    05  COLUMN 10           PICTURE X(23)
        VALUE "MONTHLY SUPPLIES REPORT".
    05  COLUMN 12           PICTURE X(9)
        SOURCE CURRENT-MONTH.
```

the length of 23 in column 10, followed by a specification for column 12 will cause field overlay.

Page Breaks

The Report Writer page break routine operates independently of the routines that are executed after any control breaks (except that a page break will occur as the result of a LINE NEXT PAGE clause). Thus, the programmer should be aware of the following facts:

1. A Control Heading is not printed after a Page Heading except for first generation. If the programmer wishes to have the equivalent of a Control Heading at the top of each page, he must include the information and data to be printed as part of the Page Heading. But since only one Page Heading may be specified for each report, he should be selective in considering his Control Heading because this "Control Heading" will be the same for each page, and may be printed at inopportune times (see "Control Footings and Page Format," in this chapter.)

2. GROUP INDICATE items are printed after page and control breaks. Figure 69 contains a GROUP INDICATE clause and shows the execution output.

```
REPORT SECTION.
    .
    .
    .
01  DETAIL-LINE TYPE IS DETAIL LINE
    NUMBER IS PLUS 1.
    05  COLUMN IS 2 GROUP INDICATE
        PICTURE IS A(9) SOURCE IS
        MONTHNAME OF RECORD-AREA (MONTH).
    .
    .
    .
          (Execution Output)
    .
    .
    .
-------------------------------------------

JANUARY    15    A00...
                 A02...

PURCHASES AND COST...

-------------------------------------------

JANUARY    21    A03...
                 A03...
```

Figure 69. Sample Showing GROUP INDICATE Clause and Resultant Execution Output

<u>WITH CODE Clause</u>

When more than one report is being written on a file and the reports are to be selectively written, a unique 1-character code must be given for each report. A mnemonic-name is specified in the RD-level entry for each report and is associated with the code in the Special-Names paragraph of the Environment Division.

<u>Note</u>: If a report is written with the CODE option, the report should not be written directly to a printer device.

This code will be written as the first character of each record that is written on the file. When the programmer wishes to write a report from this file, he needs merely to read a record, check the first character for the desired code, and have it printed if this code is found. The record should be printed starting from the third character, as illustrated in Figure 70.

Code	Control Character	Record	
1	2	3	n

Figure 70. Format of a Report Record When the CODE Clause is Specified

The following example shows how to create and print a report with a code of A. A Report Writer program contains the following statements:

```
ENVIRONMENT DIVISION.
.
.
.
SPECIAL-NAMES.   'A' IS CHR-A
                 'B' IS CHR-B.
.
.
.
DATA DIVISION.
FILE SECTION.
FD  RPT-OUT-FILE
    RECORDS CONTAIN 122 CHARACTERS
    LABEL RECORDS ARE STANDARD
    REPORTS ARE REP-FILE-A REP-FILE-B.
    .
    .
    .
REPORT SECTION.
RD  REP-FILE-A   CODE CHR-A...
.
.
.
RD  REP-FILE-B  CODE CHR-B...
.
.
.
```

The RPT-OUT-FILE must be written on a tape or mass storage device. A second program could then be used to print only the report with the code of A, as follows:

```
DATA DIVISION.
FD  RPT-IN-FILE
    RECORD CONTAINS 122 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS RPT-RCD.
01  RPT-RCD.
    05  CODE-CHR         PICTURE X.
    05  PRINT-PART.
        10  CTL-CHR      PICTURE X.
        10  RECORD-PART  PICTURE X(120).
FD  PRINT-FILE
    RECORD CONTAINS 121 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS PRINT-REC.
01  PRINT-REC.
    05  FILLER           PICTURE X(121).
.
.
PROCEDURE DIVISION.
.
.
LOOP.  READ RPT-IN-FILE AT END
         GO TO CONTINUE.
         IF CODE-CHR  = "A"
         WRITE PRINT-REC FROM
           PRINT-PART
         AFTER POSITIONING CTL-CHR
           LINES.
         GO TO LOOP.
CONTINUE.
         .
         .
```

Control Footings and Page Format

Depending on the number and size of Control Footings (as well as the page depth of the report), all of the specified Control Footings may not be printed on the same page if a control break occurs for a high-level control. When a page condition is detected before all required Control Footings are printed, the Report Writer will print the Page Footing (if specified), skip to the next page, print the Page Heading (if specified), and then continue to print Control Footings.

If the programmer wishes all of his Control Footings to be printed on the same page, he must format his page in the RD-level entry for the report (by setting the LAST DETAIL integer to a sufficiently low line number) to allow for the necessary space.

```
RD  EXPENSE-REPORT CONTROLS ARE FINAL,
    MONTH, DAYY
        .
        .
        .
01  TYPE CONTROL FOOTING DAYY
    LINE PLUS 1 NEXT GROUP
    NEXT PAGE.
        .
        .
        .
01  TYPE CONTROL FOOTING MONTH
    LINE PLUS 1 NEXT GROUP
    NEXT PAGE.
        .
        .
        .
(Execution Output)

EXPENSE REPORT
        .
        .
        .
January 31.........29.30
    (Output for CF DAYY)

January total.....131.40
    (Output for CF MONTH)
```

Note: The NEXT GROUP NEXT PAGE clause for the control footing DAY is not activated.

Floating First Detail Rule

The first presentation of a body group (PH, PF, CH, CF, or DE) that contains a relative line as its first line, will have its relative line spacing suppressed, and the first line will be printed on either the value of FIRST DETAIL or INTEGER PLUS 1 of a NEXT GROUP clause from the preceding page. For example:

A. If the following body group was the last to be printed on a page

   01 TYPE CF NEXT GROUP NEXT PAGE

   Then this next body group

   01 TYPE DE LINE PLUS 5

   would be printed on value of FIRST DETAIL (in PAGE clause).

B. If the following body group was the last to be printed on a page

   01 TYPE CF NEXT GROUP LINE 12

and after printing, line-counter = 40, then this next BODY GROUP

01 TYPE DETAIL LINE PLUS 5

would be printed on line 12 + 1 (i.e., line 13).

Report Writer Routines

At the end of the analysis of a report description entry (RD), the Report Writer routines are generated, based on the contents of the RD. Each routine has its own compiler-generated card number. Therefore, in the source listing, the last compiler-generated card number for an RD and that of the next source statement are not sequential.

TABLE HANDLING CONSIDERATIONS

Subscripts

If a subscript is represented by a constant and if the subscripted item has a fixed length, the location of the subscripted data item within the table or list is resolved at compile time.

If a subscript is represented by a data-name, the location is resolved at execution time. The most efficient format, in this case, is COMPUTATIONAL, with PICTURE size less than five integers.

The value contained in a subscript is an integer that represents an occurrence number within a table. Every time a subscripted data-name is referenced in a program, the compiler generates up to 16 instructions to calculate the correct displacement. Therefore, if a subscripted data-name is to be processed extensively, move the subscripted item to an unsubscripted work area, do all necessary processing, and then move the item back into the table. Even when subscripts are described as computational, subscripting takes time and core storage.

Index-Names

Index-names are compiler-generated items, one fullword in length, assigned storage in the TGT. An index-name is defined by the INDEXED BY clause. The value in an index-name represents an actualdisplacement from the beginning of the table that corresponds to an occurrence number in the table. Address calculation for a direct index takes a maximum of four instructions; address calculation for a relative index takes a few more. Therefore, the use of index-names in referencing tables is more efficient than the use of subscripts. The use of direct indexes is faster than the use of relative indexes.

Index-names can only be referenced in the PERFORM, the SEARCH, and the SET statements.

Index Data Items

Index data items are compiler-generated storage positions, one fullword in length, that are assigned storage within the COBOL program area. An index data item is defined by the USAGE IS INDEX clause. The programmer can use index data items to save values of index-names for later reference.

Great care must be used when setting values of index data items. Since an index data item is not part of any table, the compiler places the value contained in the index-name or other index data item into the index data item (see the example given in "SET Statement"). Index data items can only be referenced in SEARCH and SET statements.

OCCURS Clause

A _table element_ is represented by the subject of an OCCURS clause, and is equivalent to one level of a table. If indexing is to be used to reference a table element, and the Format 2 (SEARCH ALL) statement is also to be used, the KEY option _must_ be specified in the OCCURS clause. The table element must then be ordered upon the key(s) data-name(s) specified.

DEPENDING ON Option

If a data item described by an OCCURS clause with the DEPENDING ON _data-name_ option[1] is followed by nonsubordinate data

----

[1]For a discussion of the use of the OCCURS DEPENDING ON clause in a sort program, see "Sorting Variable-Length Records."

items, a change in the value of data-name during the course of program execution will have the following effects:

1. The size of any group described by or containing the related OCCURS clause will reflect the new value of data-name.

2. Whenever a MOVE to a field containing an OCCURS clause with the DEPENDING ON option is executed, the MOVE is made on the basis of the current contents of the object of the DEPENDING ON option.

3. The location of any nonsubordinate items following the item described with the OCCURS clause will be affected by the new value of data-name. If the user wishes to preserve the contents of these items, the following procedure can be used: prior to the change in data-name, move all nonsubordinate items following the variable item to a work area; after the change in data-name, move all the items back.

Note: The value of data-name may change because a move is made to it or to the group in which it is contained; or the value of data-name may change because the group in which it is contained is a record area that has been changed by execution of a READ statement.

For example, assume that the Data Division of a program contains the following coding:

```
01  ANYRECORD.
    05  A PICTURE S999 COMPUTATIONAL-3.
    05  TABLEA PICTURE S999 OCCURS 100
        TIMES DEPENDING ON A.
    05  GROUPB.

        (Subordinate data items.)

           (End of record.)
```

GROUPB items are not subordinate to TABLEA, which is described by the OCCURS clause. Assuming that WORKB is a work area with the same data structure as GROUPB, the following procedural coding could be used:

1. MOVE GROUPB TO WORKB

2. Calculate new value of A

3. MOVE WORKB TO GROUPB

The above statements can be avoided by putting the OCCURS clause with the DEPENDING ON option at the end of the record.

Note: Data-name can also change because of a change in the value of an item that redefines it. In this case, the group size and the location of nonsubordinate items as described in the two preceding paragraphs cannot be determined.

SET Statement

The SET statement is used to assign values to index-names and to index data items.

When the SET statement assigns to an index-name the value of a literal, identifier, or an index-name from another table element, it is set to an actual displacement from the beginning of the table element that corresponds to the occurrence number indicated by the second operand in the SET statement. The compiler performs all the necessary calculations. If the SET statement is used to assign an index-name to another index-name for the same table element, the compiler need make no conversion of the actual displacement value contained in the second operand.

However, when an index data item is set to another index data item or to an index-name, or when an index-name is set to an index data item, the compiler is unable to change any displacement value it finds, since an index data item is not part of any table. Thus, no conversion of values can take place. If the programmer forgets this, programming errors can occur.

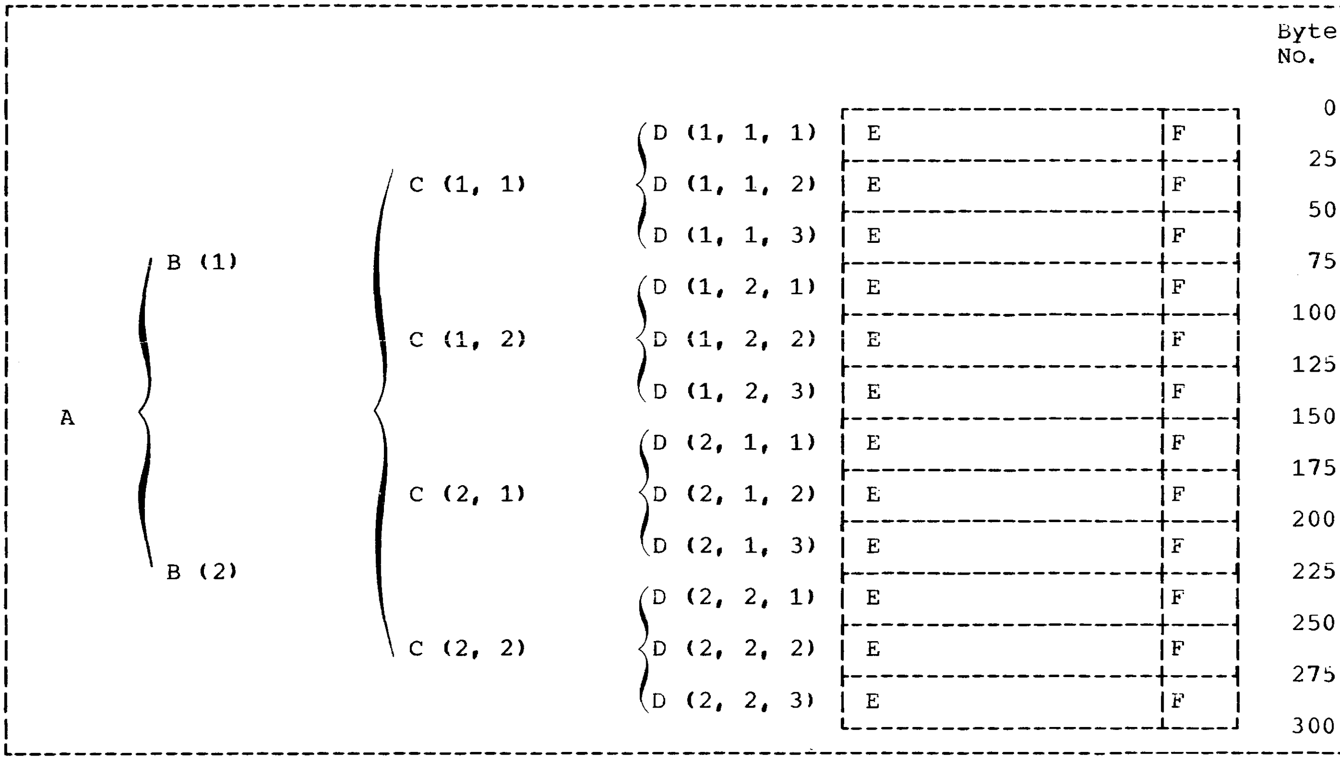


Figure 71. Storage Layout for Table Reference Example

For example, suppose that a table has been defined as:

```
01 A.
   02 B OCCURS 2 INDEXED BY I1, I5.
      03 C OCCURS 2 INDEXED BY I2, I6.
         04 D OCCURS 3 INDEXED BY I3, I4.
            05 E PIC X(20).
            05 F PIC 9(5).
```

Figure 71 shows how the table is laid out in main storage.

Now, suppose it is necessary to reference D (2,2,3). The following steps are <u>incorrect</u>:

```
SET I3 TO 2.
SET INDX-DATA-ITM TO I3.
SET I2, I1 TO INDX-DATA-ITM.
SET I3 UP BY 1.
MOVE D(I1, I2, I3) TO WORKAREA.
```

The value contained in I3 after the first SET statement is 25, which represents the beginning point (in bytes) of the second occurrence of D. When the second SET statement is executed, the value 25 is placed in INDX-DATA-ITM, and the third SET statement moves the value 25 into I2 and I1. The fourth SET statement increases the value in I3 to 50. The calculation for the address D (I1, I2, I3) would then be as follows:

```
(address of D(1,1,1)) + 25 + 25 + 50 =
  (address of D(1,1,1)) + 100
```

where D(1,1,1) represents the first occurrence of D. This is <u>not</u> the address of D (2,2,3).

The following steps will find the <u>correct</u> address:

```
SET I3 TO 2.
SET I2, I1 TO I3.
SET I3 UP BY 1.
```

In this case, the first SET statement places the value 25 in I3. Since the compiler is able to calculate the lengths of B and C, the second SET statement places the value 75 in I2, and the value 150 in I1. The third SET statement places the value 50 in I3. The correct address calculation will be:

```
(address of D(1,1,1)) + 150 + 75 + 50 =
    (address of D(1,1,1)) + 275.
```

The rules for the SET statement are shown in Table 28.

Table 28. Rules for the SET Statement

Receiving \ Sending	Index-name	Index Data Item	Identifier or Literal
Index-name	Set to value corresponding to occurrence number[1]	Move without conversion	Set to value corresponding to occurrence number
Index Data Item	Move without conversion	Move without conversion	--
Identifier	Set to occurrence number represented by index-name	--	--

[1]If index-name refer to the same table element move without conversion

SEARCH Statement

Only one level of a table (a table element) can be referenced with one SEARCH statement. Note that SEARCH statements cannot be nested, since an imperative-statement must follow the WHEN condition, and the SEARCH statement is itself conditional.

There are two formats for the SEARCH statement. Format 1, SEARCH, is used for a serial search. Format 2, SEARCH ALL, is used for a binary search.

Format 1 SEARCH statements perform a serial search of a table element. If the programmer knows that the "found" condition will come after some intermediate point in the table element, to speed up execution, he can use the SET statement to set the index-names at that point and search only part of the table element. If the table element is large, and must be searched from the first occurrence to the last, the use of Format 2 (SEARCH ALL) is more efficient than Format 1, since it uses a binary search technique; however, the table must then be ordered.

In Format 1, the VARYING option allows the programmer to:

- Vary an index-name other than the first index-name stated for this table element. Thus, with two SEARCH statements each using a different index-name, reference can be made to more than one value in the same table element for comparisons, etc.
- Vary an index-name from another table element. In this case, the first index-name specified for this table element is used for the search, and the index-name specified in the VARYING option is incremented at the same time. Thus, it is possible to step through two table elements at once.

In Format 1, the WHEN condition can be any relation condition, and can be multiple. If multiple WHEN conditions are stated, the implied logical connective is OR -- that is, if any one of the WHEN conditions is satisfied, the imperative-statement following the WHEN condition is executed. If all conditions of the SEARCH statement are to be satisfied before exiting from the search, a compound WHEN condition with an AND logical connective must be written.

In Format 2, the SEARCH ALL statement, the table must be ordered on the KEY(S) specified in the OCCURS clause. Any KEY may be specified in the WHEN condition, but all preceding data-names in the KEY option must also be tested. The test must be an "equal to" (=) condition, and the KEY data-name must be either the subject or object of the condition, or the name of a conditional variable with which the tested condition-name is associated. The WHEN condition can also be a compound condition, formed from one of the simple conditions listed above, with AND as the only logical connective. The KEY and its object of comparison must be compatible, as given in the rules of the relation test.

To write a series of statements that will search the three-dimensional table discussed in the section "The SET Statement", the programmer could write:

```
77 COMPARAND1 PIC X(5).
77 COMPARAND2 PIC 9(5).
01 A.
   05 B OCCURS 2 INDEXED BY I1 I5.
      10 C OCCURS 2 INDEXED BY I2 I6.
         15 D OCCURS 3 INDEXED BY I3, I4.
            20 E PIC X(5).
            20 F PIC 9(5).
                       .
                       .
                       .
    (initialize comparand1 and comparand2)
                       .
                       .
                       .
   PERFORM SEARCH-TEST1 THRU SEARCH-EXIT1
   VARYING I1 FROM 1 BY 1 UNTIL I1 GREATER
   THAN 2 AFTER I2 FROM 1 BY 1 UNTIL I2
   GREATER THAN 2.
ENTRY-NOENTRY1.  GO TO ERROR-RECOVERY1.
                       .
                       .
                       .
SEARCH-TEST1.  SET I3 TO 1.
   SEARCH D WHEN E (I1, I2, I3) =
      COMPARAND1 AND
      F (I1, I2, I3) = COMPARAND2
      SET I5 TO I1
      SET I6 TO I2
      SET I2 TO 3
      SET I1 TO 3
      ALTER ENTRY-NOENTRY1 TO PROCEED TO
      ENTRY-PROCESSING1.
   SEARCH-EXIT1.  EXIT.
                       .
                       .
                       .
   ERROR-RECOVERY1.
                       .
                       .
                       .
   ENTRY-PROCESSING1.
      MOVE E(I5, I6, I3) TO OUT-AREA1.
      MOVE F(I5, I6, I3) TO OUT-AREA2.
                       .
                       .
                       .
```

The PERFORM statement varies the indexed (I1 and I2) associated with table elements B and C; the SEARCH statement varies I3, which is associated with table element D.

The values of I1 and I2 that satisfy the WHEN conditions of the SEARCH statement are saved in I5 and I6. I1 and I2 are then both set to 3 using the SET statement, so that upon return from the SEARCH statement control will fall through the PERFORM statement to the GO TO statement.

Subsequent references to the desired occurrence of table elements E and F make use of the index-names I5 and I6 in which the correct value was saved.

For example, a user-defined table may be the following:

```
01   TABLE.
     05   ENTRY-IN-TABLEE OCCURS 90 TIMES
          ASCENDING KEY-1, KEY-2
          DESCENDING KEY-3
          INDEXED BY INDEX-1.
          10  PART-1   PICTURE 9(2).
          10  KEY-1    PICTURE 9(5).
          10  PART-2   PICTURE 9(6).
          10  KEY-2    PICTURE 9(4).
          10  PART-3   PICTURE 9(33).
          10  KEY-3    PICTURE 9(5).
```

A search of the entire table can be initiated with the following instruction:

```
SEARCH ALL ENTRY-IN-TABLEE AT
  END GO TO NOENTRY
WHEN KEY-1 (INDEX-1) = VALUE-1 AND
  KEY-2 (INDEX-1) = VALUE-2
  AND KEY-3 (INDEX-1) = VALUE-3
MOVE PART-1 (INDEX-1) TO
  OUTPUT-AREA.
```

The foregoing instructions will execute a search on the given array TABLE which contains 90 elements of 55 bytes and 3 keys. The primary and secondary keys (KEY-1 and KEY-2) are in ascending order whereas the least significant key (KEY-3) is in descending order. If an entry is found in which three keys are equal to the given values (i.e., VALUE-1, VALUE-2 VALUE-3) PART-1 of that entry will be moved to OUTPUT-AREA. If matching keys are not found in any of the entries in TABLE, the NOENTRY routine is entered.

If a match is found between a table entry and the given values, the index (INDEX-1) is set to a value corresponding to the relative position within the table of the matching entry. If no match is found, the index remains at the setting it had when execution of the SEARCH ALL statement began.

Compilation is faster if KEY(S) are tested in the SEARCH statement in the same order they appear in the KEY option.

Note that if KEY entries within the table do not contain valid values, then the results of the binary search will be unpredictable.

Building Tables

When reading in data to build an internal table:

1. Check to make sure the data does not exceed the space allocated for the table.

2. If the data must be in sequence, check the sequence.

3. If the data contains the subscript determining its position in the table, check the subscript for a valid range.

When testing for the end of a table, use a named value giving the item count, rather than using a literal. Then, if the table must be expanded, only one value need be changed, instead of all references to a literal.

QUEUE STRUCTURE CONSIDERATIONS

In a COBOL teleprocessing (TP) program, a CD FOR INPUT allows the specification of one through three levels of sub-queues from which data can be received; this allows the COBOL object program, at execution time, to make use of pre-defined queue structures, and to access all or parts of such structures. For TP programs, such queue structures are analogous in function and form to the File Description (FD) entry and its associated 01 record description for file processing programs. If pre-defined queue structures are used, each lowest level sub-queue name in the structure corresponds to a DD name; the associated DD card must specify a TPROCESS entry in the message control program (MCP) terminal table). Figure 72 shows the configuration of one queue structure such that queue A is made up of sub-queues B and C, sub-queue B is made up of sub-queues D and E, and sub-queue D is made up of sub-queues H and I (where sub-queue H contains messages Z1 and X2 and sub-queue I contains messages X3, X4, and X5), and so on.

During program execution, when the user wishes to receive a message from a queue (or sub-queue) he need not place the names of all sub-queues in the input CD; he need specify only the SYMBOLIC QUEUE name, which may be the name of a pre-defined queue structure, or he may specify that name plus one or more sub-queue names -- which allows him to access only that part of the entire structure that is needed. A COBOL object-time subroutine uses the name(s) placed in the input CD to determine which lowest-level sub-queue(s) and corresponding TCAM queue(s) can be used to fill the request.

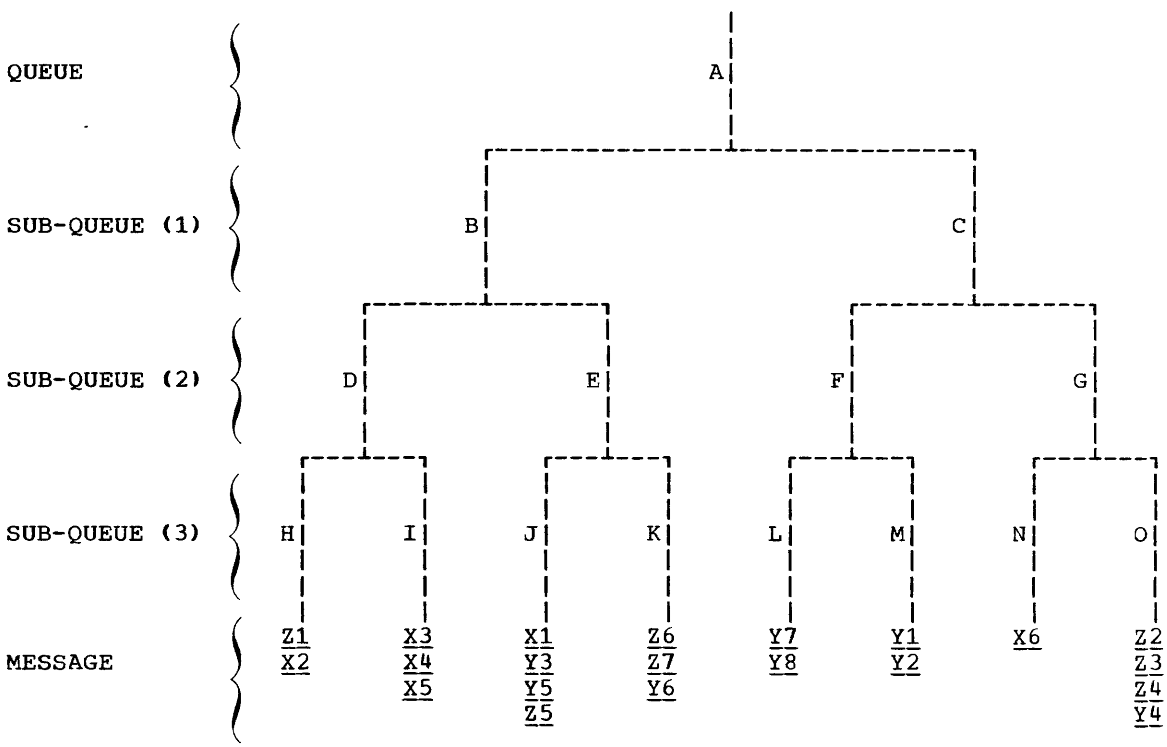


Figure 72. A Queue Structure with Three Levels of Sub-Queues

```
//BLDRDS     JOB  user information
//JOBLIB¹    DD   DSN=SYS1.COBLIB,UNIT=2314,VOL=SER=DC160,DISP=OLD
//SUBQS²     EXEC PGM=ILBOQSU,REGION=96K
//COBTPQD³   DD   DSN=SUBQPDS,UNIT=2314,VOL=SER=DC160,                 X
//                SPACE=(4000,(50,20,1)),DISP=(OLD,KEEP)
//SYSPRINT⁴  DD   SYSOUT=A
//SYSIN⁵     DD   *
//
```

QUEUE STRUCTURE DEFINITIONS FOR USE IN COBOL PROGRAMS WHICH PROCESS QUEUES AND SUB-QUEUES.

	Note: The parenthetical entries below are for illustrative purposes only, they may not appear in the program itself.
QUEUE IS A.	(FD clause)
SUB-QUEUE-1 IS B.	(01 entry)
SUB-QUEUE-2 IS D.	(02 entry)
SUB-QUEUE-3 IS H.	(03 entry)
SUB-QUEUE-3 IS I.	(03 entry)
SUB-QUEUE-2 IS E.	(02 entry)
SUB-QUEUE-3 IS J.	(03 entry)
SUB-QUEUE-3 IS K.	(03 entry)
SUB-QUEUE-1 IS C.	(01 entry)
SUB-QUEUE-2 IS F.	(02 entry)
SUB-QUEUE-3 IS L.	(03 entry)
SUB-QUEUE-3 IS M.	(03 entry)
SUB-QUEUE-2 IS G.	(02 entry)
SUB-QUEUE-3 IS N.	(03 entry)
SUB-QUEUE-3 IS O.	03 entry)

Notes:

1. The data-set name SYS1.COBLIB represents the Version 4 COBOL Library.

2. The utility program ILBOQSU (called the Queue Structure Description routine) creates a partitioned data set with one member for each complete queue structure defined. It has an alias name of BLDQS, which may be specified on the EXEC card instead.

3. The partitioned data set must be described on a DD card with the reserved name //COBTPQD. The SPACE parameter on this card must request allocation in terms of 4000-byte blocks.

4. The SYSPRINT DD statement defines the output message and listing data set.

5. The SYSIN DD statement defines the input to the program. The SYSIN data set must consist of 80-character records.

Figure 73. A Sample Queue Structure Descripion

In order to do this, the user must have previously defined his queue structures in a form that is acceptable to the COBOL object-time subroutine. A utility program that functions as the Queue Structure Description routine (included in the Version 4 Library) makes this possible. Input to the Queue Structure Description routine consists of a series of statements that define queue structures. The statements are written in a COBOL-like format, similar to an FD entry and its associated record description entry. The Queue Structure Description routine produces as output a partitioned data set with one member for each complete queue structure. The sample listing shown in Figure 73 provides the queue definition statements that correspond to this queue structure. At the right of each statement, in parenthesis, is each FD entry equivalent.

Each logical record in a queue structure description may include only a queue or sub-queue definition; it may not include, for example, the usual COBOL sequence number. (For a detailed description of the possible formats for input to the Queue Structure Description routine, see the Section "Rules for Queue Structure Description" in this chapter.)

ACCESSING QUEUE STRUCTURES THROUGH COBOL

Once the user has defined and stored the queue structures, COBOL TP programs can utilize these structures. At execution time, the partitioned data set is described on a DD card with the name COBTPQD. If, for example, the user wanted to access messages described in the queue structure defined in Figure 73, a DD card specifying the partitioned data set SUBQPDS, as below, would be required.

```
//COBTPQD DD DSN=SUBQPDS,DISP=OLD
```

Additional DD cards would be required to link the message control program terminal table entries and the lowest-level sub-queue names. (For a description of terminal table entries, see the section "Terminal and Line Control Areas" in the chapter "Using the Teleprocessing Feature".) The name of the DD card may be defined either as the sub-queue name itself (for example, as H, I, J, K, L, M, N, or O) or as a ddname that is equivalent to the lowest-level sub-queue name. This alternative approach permits the COBOL program to reuse SYMBOLIC SUB-QUEUE names without ambiguity. These two approaches are illustrated below.

<u>Method 1</u>: The DD card associated with the queue definition SUB-QUEUE-3 is H would be:

```
//H DD QNAME=Q1
```

<u>Method 2</u>: The DD card associated with the queue definition SUB-QUEUE-3 is H(FIRSTMSG) would be:

```
//FIRSTMSG DD QNAME=Q1
```

where Q1 is an entry in the terminal table

Before a RECEIVE statement is executed, the user places (via a MOVE statement) the needed queue and sub-queue name(s) in the CD entry. When the RECEIVE statement is executed, the RECEIVE subroutine checks for the presence of the partitioned data set describing these queue structures. If the data set is present, the RECEIVE subroutine invokes a Queue Analyzer routine, which searches the partitioned data set for a member corresponding to the name in the SYMBOLIC QUEUE field, reads that member into main storage, and uses it to validate the SYMBOLIC SUB-QUEUE name(s) in the COBOL program input CD entry. The Queue Analyzer routine then determines the first valid name for the structure specified and gives this name to the RECEIVE routine.

Names at the SUB-QUEUE-1 level take priority over names at the SUB-QUEUE-2 level. Names at the SUB-QUEUE-2 level take priority over names at the SUB-QUEUE-3 level. At any given level, names at the left take priority over, and are completely evaluated before, names at the right. (Taking advantage of this retrieval technique, the user can improve object-time performance by defining the most frequently used sub-queues at the left of the structure. Table 29 illustrates TCAM message retrieval.)

The RECEIVE subroutine then attempts to access the queue specified. If the DD card for this queue is not present, or if there are no messages in the associated MCP queue, the Queue Analyzer provides the RECEIVE routine with another valid name. The procedure is repeated until the RECEIVE routine accesses a message, or until there are no more queues to access.

During a RECEIVE operation, a COBOL program using queue structures need not specify all levels of sub-queues. The highest level (QUEUE) must be specified; that level plus a SUB-QUEUE-1 may also be specified; or all four levels may be specified. If a lower level is specified, then all higher levels must also be specified.

If the COBOL programmer wishes to access the next message in the queue structure,

Table 29. Sample Message Retrieval Options

Input CD	Message Returned by the MCP
CD CDNAME-IN FOR INPUT SYMBOLIC QUEUE IS data-name-1. (where data-name-1 contains 'A')	Message Z1
CD CDNAME-IN FOR INPUT SYMBOLIC QUEUE IS data-name-1. SYMBOLIC SUB-QUEUE-1 IS data-name-2. (where data-name-1 contains 'A' and data-name-2 contains 'C')	Message Y7
CD CDNAME-IN FOR INPUT SYMBOLIC-QUEUE IS data-name-1, SYMBOLIC-QUEUE-1 IS data-name-2, SYMBOLIC-QUEUE-2 IS data-name-3. (where data-name-1 contains 'A', data-name-2 contains 'B', and data-name-3 contains 'E')	Message X1
Note: Data-name-1, data-name-2, and so on, refer to the optional clauses of a queue structure defined under "Rules for Queue Structure Description" in this chapter.	

regardless of which sub-queue that message may be in, he specifies the queue name only, and initializes the sub-queue-names to SPACES. The Queue Analyzer, when supplying the message, returns to the COBOL object program any applicable sub-queue names via the data items in the associated input CD. If, however, the programmer wishes the next message in a given sub-queue, he must specify both the queue name and any applicable sub-queue names. Once a program has begun receiving any part of a message from a queue (or sub-queue), subsequent requests must specify both the queue-name and any applicable sub-queue-names until end of message is indicated. Table 29 illustrates the relationship between the information contained in the input CD at object time and the message(s) accessed when the RECEIVE statement is executed (where each example refers to the queue structure pictured in Figure 72).

Specifying ddnames with Elementary Sub-Queues

Suppose that an application program is written to accept TP messages as input to an inventory control process. Each of five different locations transmits data on four different parts. The diagram in Figure 74 illustrates the relationship between the input messages and the four different parts for each location.

Each elementary, or lowest-level, queue in the structure must specify the name of a DD card, which in turn names a TPROCESS entry. While the example, as shown in Figure 74, is not ambiguous (that is, INVENTORY.CHICAGO.PARTA is distinct from INVENTORY.LOS-ANGELES.PARTA), the elementary queues by themselves are not (that is, the elementary name PARTA, which corresponds to a ddname, can be any one of five different PARTA's). To eliminate this ambiguity, whenever there are duplicate names in the lowest level of a queue structure, the user must define ddnames in addition to the sub-queue names at the lowest level when he defines the structure to the Queue Structure Description routine. Then the object-time Queue Analyzer routine automatically associates the fully qualified queue structure names with the DD names required. Accordingly, in this example:

NEW-YORK.PARTA could have ddname DD1.

NEW-YORK.PARTB could have ddname DD2.

NEW-YORK.PARTC could have ddname DD3.

NEW-YORK.PARTD could have ddname DD4.

CHICAGO.PARTA could have ddname DD5.

CHICAGO.PARTB could have ddname DD6.

and so forth. In this way, each elementary queue has a unique designation; yet the COBOL program can refer to the sub-queue names without ambiguity.

```
      SUB-QUEUE-2                       SUB-QUEUE-1                    QUEUE

PARTA(DD1)  ----------------┐
                            |
PARTB(DD2)  ----------------┤          NEW-YORK
                            |-------------------------┐
PARTC(DD3)  ----------------┤                         |
                            |                         |
PARTD(DD4)  ----------------┘                         |
                                                      |
PARTA(DD5)  ----------------┐                         |
                            |                         |
PARTB(DD6)  ----------------┤          CHICAGO        |
                            |-------------------------┤
PARTC(DD7)  ----------------┤                         |
                            |                         |
PARTD(DD8)  ----------------┘                         |
                                                      |
PARTA(DD9)  ----------------┐                         |
                            |                         |
PARTB(DD10)-----------------┤          ST-LOUIS       |                        INVENTORY
                            |-------------------------+-------------------------------
PARTC(DD11)-----------------┤                         |
                            |                         |
PARTD(DD12)-----------------┘                         |
                                                      |
PARTA(DD13)-----------------┐                         |
                            |                         |
PARTB(DD14)-----------------┤          DENVER         |
                            |-------------------------┤
PARTC(DD15)-----------------┤                         |
                            |                         |
PARTD(DD16)-----------------┘                         |
                                                      |
PARTA(DD17)-----------------┐                         |
                            |                         |
PARTB(DD18)-----------------┤          LOS-ANGELES    |
                            ├-------------------------┘
PARTC(DD19)-----------------┤
                            |
PARTD(DD20)-----------------┘
```

Figure 74. Using ddnames with Queue Structures

Format

```
     (QUEUE      )  IS data
     (           )
     (Q          )

     (SUB-QUEUE-1)
{{   (           )  IS data-name-2[(ddname-)]}...
     (SUBQ1      )

     (SUB-QUEUE-2)
[[   (           )  IS data-name-3[(ddname)]]...
     (SUBQ2      )

     (SUB-QUEUE-3)
[    (           )  IS data-name-4[(ddname-3)]]...]}...
     (SUBQ3      )
```

Rules for Queue Structure Description

For each member of the partitioned data set, the input to the Queue Structure Description Routine must take the format above.

The clauses of the queue structure may be written free form; however, only one clause may appear on each 80-character record. ~~At least one sub-queue level must~~ be specified; no more than 200 sub-queue names may be specified in one queue structure.

The sub-queues at each level must be specified to the Queue Structure Description routine in left-to-right order. When the queue structure is referred to at object program execution time, names at a higher level take priority over names at a lower level. At a given level in the queue structure, names to the left take priority over names to the right.

A queue structure need not include all levels of sub-queues. However, if a lower level is included in one leg of a queue structure, then that leg must include all higher levels.

Each clause of the structure may optionally be followed by a period.

Data-name-1 is the name of the queue structure, and becomes the name of that member of the partitioned data set.

Data-name-2 though data-name-4 are sub-queue names within the data set member.

Note: A data-name cannot contain more than 12 characters.

Each data-name at the lowest (elementary) level of a leg of the queue structure may be a ddname; alternatively, each such data-name may be followed by a parenthesized ddname. If a parenthesized ddname follows a sub-queue name, the left parenthesis must immediately follow the sub-queue name with no intervening spaces. There must be no spaces between the parentheses and the ddname.

CALLING AND CALLED PROGRAMS

A COBOL program can refer to and pass control to other COBOL programs, or to programs written in other languages. A program in another language can refer to and pass control to a COBOL program. A program that refers to another program is a calling program. A program that is referred to is a called program. Control is returned from a called program to the first instruction following the calling sequence in the calling program.

A called program can also be a calling program; that is, a called program can, in turn, call another program. However, a called program cannot call the program that called it, an earlier calling program, or itself. In Figure 75, for instance, program A calls program B; program B calls program C. Therefore:

1. A is considered a calling program by B.
2. B is considered a called program by A.
3. B is considered a calling program by C.
4. C is considered a called program by B.

Control is returned in the same order of calling; that is, a called program (program C) returns control to its own calling program (program B), not to an earlier calling program (program A). Compiler-generated switches (e.g., ON and ALTER) are not reinitialized upon each entrance to the called program, that is, the program is in the last executed state unless it has been the object of a CANCEL statement.

Usually called and calling programs to be executed as a single job step are link-edited together; they must all be included in the same load module. However, with the COBOL dynamic call feature a programmer can request that a called program be link-edited into a separate module and called only if it is needed (see the section "Dynamic Subprogram Linkage", in this chapter).

This chapter describes the accepted linkage conventions for calling and called programs in both COBOL and assembler language and discusses how such programs are link-edited. An example is provided to illustrate the coding required to have proper interface between both COBOL and assembler language calling and called programs. In addition, it includes a discussion of overlay design in which different called programs may, at different times, occupy the same area in main storage. Another example is provided to illustrate one method of accomplishing program linkage using the dynamic overlay technique.

SPECIFYING LINKAGE

Whenever a program calls another program, a link must be established between the two. The calling program must state the entry point of the called program and must specify any identifiers to be passed. The called program must have an entry point and must be able to accept the identifiers. In addition, the called program must establish the linkage for the return of control to the calling program. See Figure 76 for an example of the linkage statements required in a typical calling/called situation.

```
              A                         B                         C
      ---------------           ---------------           ---------------
      |             |           |             |           |             |
      |             |           | Called      |           |             |
      | Calling     |           |  program of A|          | Called      |
      |  program of B|--------->|             |---------->|  program of B|
      |             |           | Calling     |           |             |
      |             |           |  program of C|          |             |
      |             |           |             |           |             |
      ---------------           ---------------           ---------------
```

Figure 75. Calling and Called Programs

LINKAGE IN A CALLING COBOL PROGRAM

A calling COBOL program must contain the following statement at the point at which another program is to be called:

```
CALL {literal-1   }
     {identifier-1}
  [USING identifier-list].
```

Literal-1 or the contents of identifier-1 must be either the name of the program that is being called or the name of an entry point in the called program. The first eight characters of literal-1 or identifier-1 are used to make the correspondence between the calling program and the called program. The identifier-list is one or more data-names, called identifiers and separated by blanks, that are to be passed to the called program.

If the called program is an assembler-language program, the identifier in the USING phrase may also be a file-name or a procedure-name. If the identifier in the USING phrase is a file-name, the COBOL compiler passes the address of the DCB for a queued file, or the address of the DECB for a basic file, as this entry of the identifier-list. This can be used to test bits in the DCB or DECB or to enter some options in the DCB. However, when changing a field of the DCB, precautions should be taken not to contradict the information in other fields or the information in the object code supplied by the compiler, job control language, or other sources. When the identifier in the USING phrase is a procedure-name, the value passed is the beginning address of the procedure. If no identifiers are passed, the USING clause is omitted.

LINKAGE IN A CALLED COBOL PROGRAM

A called COBOL program must contain two statements.

One of the following statements must be inserted to name the point where the program is to be entered:

```
ENTRY literal-1
  [USING identifier-list].

        or

PROCEDURE DIVISION [USING
  identifier-list].
```

The literal-1 or PROGRAM-ID is the name of the entry point in the called program. It is the same name that appears in the CALL statement of the program that calls this program that the compiler uses. The identifier-list is one or more data-names that correspond to the identifier-list of the CALL statement of the calling program. Each data name of the identifier-list must be defined in the Linkage Section of the Data Division and must have a level number of 01 or 77.

One of the following statements must be inserted at the point at which control is to be returned to the calling program:

```
GOBACK.

    or

EXIT PROGRAM.
```

The GOBACK or EXIT PROGRAM statement enables restoration of necessary registers and returns control to the point in the calling program immediately following the calling sequence.

Note: The GOBACK and EXIT PROGRAM statements may be used in a main program, with the result that any COBOL program can be used as either a calling or a called program, if written with this end in mind. If a GOBACK statement appears within the main program, control is returned immediately to the system; if an EXIT PROGRAM statement appears, it is simply regarded as a null instruction.

A called program may pass a completion code to its caller by storing a value in RETURN-CODE. The calling program may interrogate RETURN-CODE after a return is made from a called program to determine the completion code.

Note: RETURN-CODE may also be used to pass a completion code to the system at the end of a run unit.

Dynamic Subprogram Linkage

With the dynamic subprogram linkage feature, a called program need not be link-edited with the main program. It may instead be link-edited into a separate load module, so that at execution time it is loaded _if and only if_ it is called. Accordingly, the first dynamic call to a subprogram obtains a fresh copy of the subprogram. Subsequent calls to the same subprogram, by either the original caller or any other subprogram in the same region/partition, result in a branch to the same copy of the subprogram in its last-used state until the subprogram is cancelled the first call following a CANCEL statement results in a branch to a fresh copy of the subprogram.

Specification of the DYNAM option in the PARM field of the EXEC statement (see the section on "Compiler Options" in the chapter entitled "Job Control Procedures") makes all calls dynamic. If NODYNAM is in effect, through either user specification at compile time or as the default option, only CALL identifier statements are dynamic; when NODYNAM is in effect, CALL literal statements are static. (For a discussion of the formats possible with the CALL statement, see the publication IBM OS Full American National Standard COBOL.)

For an example of a COBOL program that takes advantage of the dynamic CALL/CANCEL feature, see Figure 76 in this chapter.

Notes:

1. When the dynamic CALL is used, the main program and all subprograms in one region/partition should take advantage of the COBOL Library Management Facility (see the "Libraries" chapter). Even when the DYNAM option is not specified, a program with CALL identifier or CANCEL identifier statements requires the Library Management Feature.

2. The USING option should be included in the CALL statement only if there is a USING option in the called entry point.

3. A segmented program may be called but only by its PROGRAM-ID or by an entry point within the root segment.

```
//CALLJOB        JOB   user information
//STEP1          EXEC     UCOBFCL,PARM.COB='DYNAM,RESIDENT'
//COB.SYSIN      DD    *
                 IDENTIFICATION DIVISION
                 PROGRAM-ID. SUBPROG1.
                 AUTHOR. J. SMITH
                 REMARKS.
                       THIS SUBPROGRAM IS CALLED BY THE MAIN PROGRAM.
                       IT ISSUES A MESSAGE TO INDICATE WHETHER IT IS
                       IN INITIAL OR LAST-USED STATE, AND THEN RETURNS
                       TO THE MAIN PROGRAM.
                 ENVIRONMENT DIVISION.
                 CONFIGURATION SECTION.
                 SOURCE-COMPUTER.  IBM-360.
                 OBJECT-COMPUTER.  IBM-360.
                 DATA DIVISION.
                 WORKING-STORAGE SECTION.
                 77  SWITCH  PIC 9 VALUE 0.
                 PROCEDURE DIVISION.
                       IF SWITCH=0 DISPLAY 'SUBPROG1 CALLED -- IN
                         INITIAL STATE'
                       GO TO RETURN-POINT.
                       DISPLAY 'SUPROG1 CALLED -- IN LAST-USED STATE'.
                 RETURN-POINT.
                       ADD 1 TO SWITCH.
                       EXIT PROGRAM.
/*
//LKED.SYSLMOD DD      DSN=SUBPROGS,UNIT=2314,VOL=SER=XXXXXX,
//                     DISP=(NEW,KEEP),SPACE=(TRK,(5,1,1))
/*
```

Note: When a subprogram is called dynamically, the (NAME and/or ALIAS) option of the linkage editor is used to identify the module that is accessed by an OS LOAD macro at execution time (see the section entitled "Link-editing COBOL Programs").

Figure 76. Sample Calling and Called Programs Using Dynamic CALL and CANCEL Statements (Part 1 of 3)

```
//CALLJOB2      JOB user information
//STEP1         EXEC UCOBFCL,PARM.COB='DYNAM,RESIDENT'
//COB.SYSIN     DD  *
                IDENTIFICATION DIVISION.
                PROGRAM-ID. SUBPROG2.
                AUTHOR. J. SMITH
                REMARKS.
                    THIS SUBPROGRAM IS CALLED BY THE MAIN PROGRAM.
                    IF IT IS IN INITIAL STATE, IT ISSUES A MESSAGE
                    TO THAT EFFECT AND RETURNS TO THE MAIN PROGRAM.
                    IF NOT, IT ISSUES A MESSAGE THAT IT IS IN THE
                    LAST-USED STATE, CANCELS SUBPROG1 VIA A CANCEL
                    IDENTIFIER, AND RETURNS TO THE MAIN PROGRAM.
                ENVIRONMENT DIVISION.
                CONFIGURATION DIVISION.
                SOURCE-COMPUTER. IBM-360.
                OBJECT-COMPUTER. IBM-360.
                DATA DIVISION.
                WORKING-STORAGE SECTION.
                77  SWITCH  PIC 9  VALUE 0.
                77  CANCL-ID  PIC X(8).
                PROCEDURE DIVISION.
                    IF SWITCH=0 DISPLAY 'SUBPROG2 CALLED -- IN INITIAL STATE'
                      GO TO RETURN-POINT.
                    DISPLAY 'SUBPROG2 CALLED -- IN LAST-USED STATE'.
                    DISPLAY 'SUBPROG2 CANCELLING SUBPROG1'.
                    MOVE 'SUBPROG1' TO CANCL-ID.
                    CANCEL CANCL-ID.
                RETURN-POINT.
                    ADD 1 TO SWITCH.
                    EXIT-PROGRAM.
/*
//LKED.SYSLMOD DD  DSN=SUBPROGS,UNIT=2314,VOL=SER=XXXXXX, DISP=OLD
/*
```

Figure 76. Sample Calling and Called Programs Using Dynamic CALL and CANCEL Statements (Part 2 of 3)

```
//CALLJOB3     JOB user information
//STEP1        EXEC UCOBFCLG,PARM.COB='DYNAM,RESIDENT'
//COB.SYSIN    DD  *
               IDENTIFICATION DIVISION.
               PROGRAM-ID. MAINPROG.
               AUTHOR. J. SMITH
               REMARKS.
                   THIS IS A MAIN PROGRAM.  IT CALLS SUBPROG1 AND
                   SUBPROG2 TWICE.  ON THE FIRST CALL, EACH SUBPROGRAM
                   SHOULD BE A FRESH COPY (THAT IS, IN INITIAL STATE).
                   ON THE SECOND CALL, EACH SUBPROGRAM SHOULD BE IN ITS
                   LAST-USED STATE.  WHEN SUBPROG2 IS CALLED THE SECOND
                   TIME, IT CANCELS SUBPROG1.  THEN MAINPROG CALLS
                   SUBPROG1 AGAIN, AND AGAIN A FRESH COPY OF THIS
                   SUBPROGRAM SHOULD BE MADE AVAILABLE.
                   THE OUTPUT FROM THIS RUN SHOULD READ AS FOLLOWS:

                   'BEGIN MAINPROG.
                   MAINPROG CALLING SUBPROG1.
                   SUBPROG1 CALLED -- IN INITIAL STATE.
                   MAINPROG CALLING SUBPROG2.
                   SUBPROG CALLED -- IN INITIAL STATE.
                   MAINPROG CALLING SUBPROG1.
                   SUBPROG CALLED -- IN LAST-USED STATE.
                   MAINPROG CALLING SUBPROG2.
                   SUBPROG2 CALLED -- IN LAST-USED STATE.
                   SUBPROG2 CANCELLING SUBPROG1.
                   MAINPROG CALLING SUBPROG1.
                   SUBPROG1 CALLED -- IN INITIAL STATE.
                   MAINPROG CANCELLING SUBPROG1 AND SUBPROG2.
                   END MAINPROG.'

               ENVIRONMENT DIVISION.
               CONFIGURATION SECTION.
               SOURCE-COMPUTER. IBM-360.
               OBJECT-COMPUTER. IBM-360.
               DATA DIVISION.
               WORKING-STORAGE SECTION.
               77  SWITCH  PIC 9 VALUE 0.
               77  CALLID  PIC X(8).
               PROCEDURE DIVISION.
                   DISPLAY 'BEGIN MAINPROG'.
               START-CALLS.
                   IF SWITCH IS LESS THAN 2 PERFORM CALL1.
                   PERFORM CALL2.
                   GO TO START-CALLS.
                   PERFORM CALL1.
                   DISPLAY 'MAINPROG CANCELLING SUBPROG1 AND SUBPROG2'.
                   CANCEL 'SUBPROG1','SUBPROG2'.
                   DISPLAY 'END MAINPROG'.
                   STOP RUN.
               CALL1.
                   MOVE 'SUBPROG' TO CALLID.
                   DISPLAY 'MAINPROG CALLING SUBPROG'.
                   CALL CALLID.
               CALL2.
                   MOVE 'SUBPROG2' TO CALLID.
                   DISPLAY 'MAINPROG CALLING SUBPROG2'.
                   CALL CALLID.
                   ADD 1 TO SWITCH.
/*
//GO.STEPLIB   DD  DSN=SUBPROGS,UNIT=2314,VOL=SER=XXXXXX,DISP=OLD
//GO.SYSOUT    DD  SYSOUT=A
/*
```

Figure 76. Sample Calling and Called Programs Using Dynamic CALL and CANCEL Statements (Part 3 of 3)

Correspondence of Identifiers in Calling and Called Programs

The number of data-names in the identifier list of a calling program must be the same as the number of data-names in the identifier list of the called program. There is a one-to-one correspondence; that is, the first identifier of the calling program is passed to the first identifier of the called program, the second identifier of the calling program is passed to the second identifier of the called program, and so forth.

Only the address of an identifier list is passed. Consequently, the data-name that is an identifier of the calling program and the data-name that is the corresponding identifier of the called program both refer to the same locations in main storage. The pair of names, however, need not be identical, but the data descriptions must be equivalent. For example, if an identifier of the calling program is a level-77 data-name of a character string of length 30, its corresponding identifier of the called program could also be a level-77 data-name of a character string of length 30, or the identifier of the called program could be a level-01 name with subordinate names representing character strings whose combined length is 30.

Although all identifiers of the called program in the ENTRY statement must be described with level numbers of 01 or 77, there is no such restriction made for identifiers of the calling program in the CALL statement. An identifier of the calling program may be a qualified name or a subscripted name. When a group item with a level number other than 01 is specified as an identifier of the calling program, proper word-boundary alignment is required if subordinate items are described as COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2. If the identifier of the calling program corresponds to a level-01 identifier of the called program, doubleword alignment is required.

FILE-NAME AND PROCEDURE-NAME ARGUMENTS

A calling COBOL program that calls an assembler-language program can pass file-names and procedure-names, in addition to data-names, as identifiers. In the actual identifier-list that the compiler generates, the procedure-name is passed as the address of the procedure. For a queued file, the file-name is passed as the address of the DCB (Data Control Block); for a basic file, the file-name is passed as the address of the DECB (Data Event Control Block).

LINKAGE IN A CALLING OR CALLED ASSEMBLER-LANGUAGE PROGRAM

In a COBOL program, the expansions of the linkage statement provide the save and return coding that is necessary to establish linkage between the calling and the called programs. Assembler-language programs must be prepared in accordance with the basic linkage conventions of the operating system. Table 30 shows the conventions for use of general registers as linkage registers.

Conventions Used in a Calling Assembler-Language Program

A calling assembler-language program must reserve a save area of 18 words, beginning on a fullword boundary, to be used by the called program for saving registers. It must load the address of this area into register 13. If the program is to pass identifiers, an identifier list must be prepared, and the address of the identifier list must be loaded into register 1. The calling program must load the address of the return point into register 14, and it must load the address of the entry point of the called program into register 15.

Table 30. Linkage Registers

Register Number	Register Use	Contents
1	Identifier	Address of the list that is passed to the called program.
13	Save Area	Address of an area (of 18 fullwords) to be used by the called program to save registers.
14	Return	Address of the location in the calling program to which control should be returned after execution of the called program.
15	Entry Point[1]	Address of the entry point in the called program to which control is to be transferred.

[1]Register 15 is also used as a return code register. The return code indicates whether or not any exceptional conditions occurred during execution of the called program.

The identifier list is a group of contiguous fullwords, each of which is an address of a data item to be passed to the called program. The identifier list must begin on a fullword boundary. The high-order bit of the last identifier, by convention, is set as a flag of one to indicate the end of the list. Figure 77 shows a portion of an assembler-language program that illustrates the conventions used in a calling program.

A GOBACK statement or a STOP RUN statement issued within a COBOL program will (always for STOP RUN, but only in a main program for GOBACK) reference the COBOL library subroutine ILBOSRV. Furthermore, the STOP RUN statement will end the run unit, which is assumed to begin with the highest-level COBOL program called. To circumvent this assumption, a higher-level assembler language program must call the COBOL library subroutine ILBOSTP0 before making any calls to other COBOL programs. This should be done as soon as possible after entry to the assembler-language program, as part of the program's initialization procedure.

Conventions Used in a Called Assembler-Language Program

A called assembler-language program must save the registers and store other pertinent information in the save area passed to it by the calling program (the layout of the save area is shown in Figure 79). A called program must also contain a return routine that (1) loads the address of the save area back into register 13, (2) restores the contents of other registers, loading the return address in register 14, and (3) optionally, sets flags in the high-order eight bits of word 4 of the save area to 1's to indicate that the return occurred. It can then branch to the address in register 14 to complete the return.

Figure 85 shows a portion of an assembler-language program that illustrates the conventions used in called programs that are also calling programs. Figure 86 shows the JCL suggested for compiling, link-editing, and executing a calling assembler-language program and a called COBOL program.

```
          LA    13,AREA           LOADS THE ADDRESS OF THIS PROGRAM'S SAVE AREA INTO
*                                 REGISTER 13.
          .
          .
          .
AREA      DS    18F               RESERVES 18 WORDS FOR THE SAVE AREA
          .
          .
          .
*  CALLING SEQUENCE
          LA    1,ARGLST          LOADS INTO REGISTER 1 THE ADDRESS OF THE IDENTIFIER
*                                 LIST TO BE PASSED.  TRANSFERS CONTROL TO THE ENTRY
          CALL  COBREGN2          POINT OF THE CALLED PROGRAM.  (THE CALL MACRO
*                                 INSTRUCTION GENERATES CODING THAT LOADS A V-TYPE
*                                 ADDRESS CONSTANT -- COBREGN2 -- INTO REGISTER 15 AND
*                                 PLACES INTO REGISTER 14 THE RETURN ADDRESS, THAT IS,
*                                 THE ADDRESS OF THE FIRST BYTE FOLLOWING THE MACRO
*                                 EXPANSION.
          .
          .
          .
*  PARAMETER LIST
          DS    0F                THIS PARAMETER LIST CONTAINS ONLY ONE ARGUMENT.
ARGLST    DC    X'80'             FIRST BYTE OF LAST ARGUMENT (ONLY
          DC    AL3(ARGUMENT)     ARGUMENT IN THIS PROGRAM) SETS BIT 0
ARGUMENT  DC    C'1'              TO 1.
```

Note: Since the calling program containing this coding could previously have been called by another program, it also could establish linkage between the save area it has received and the save area it passes to the called program. It would store in word three of the old save area the address of the new save area, and it would store in word two of the new save area the address of the old save area.

Figure 77. Sample Linkage Coding Used in a Calling Assembler-Language Program

COMMUNICATION WITH OTHER LANGUAGES

An American National Standard COBOL program may communicate at object time with programs written in other source program languages, such as COBOL F, PL/1, FORTRAN, and, as in the foregoing discussion, assembler language. The relatively few problems that may arise in using American National Standard COBOL with COBOL F usually have to do with slightly different boundary alignments, slack-byte insertion, different meanings for the same reserved word, and so on.

There is a greater disparity between American National Standard COBOL and FORTRAN, much of it stemming from the basic differences in the applications for which these languages were developed. (FORTRAN is process oriented and does comparatively little file processing; COBOL, on the other hand, is definitely file oriented and is not mathematically self-sufficient.) Care must be taken, therefore, in attempting to pass arguments between American National Standard COBOL and FORTRAN programs.

The use of COBOL and PL/I together presents such a large number of problems that a considerable amount of study is necessary to implement anything but the most basic application. For further information, see the publications IBM OS Linkage Editor and Loader, Order No. GC28-6538, and IBM OS PL/I (F) Programmer's Guide, Order No. C28-6594.

Abnormal terminations in non-COBOL programs calling COBOL programs compiled with either the STATE or the SYMDMP option (see the chapter entitled "Symbolic Debugging Features") cause generation of the following misinformation:

- Incorrect number for the statement responsible for the abnormal termination. The last COBOL statement in the called program executed before the return to the calling non-COBOL program is given in the "Last Card Number Executed" message.

- Incorrect PROGRAM-ID when such an abnormal termination occurs after return from the called COBOL program. The PROGRAM-ID message contains the user-specified name for the called COBOL program.

SAMPLE CALLING AND CALLED PROGRAMS

The following set of programs (Figure 76) contains a sample COBOL main-line program, COBMAIN, which calls COBOL and assembler-language programs using arguments that represent a data-item and a file-name.

Some of the called programs (COBOL1, COBOL1B, and ASSMPGM) are themselves calling programs. Program COBREGN0 is called by several programs, each of which enters at a different entry point within the program.

The assembler language program, ASSMPGM shown in Figure 78 (Part 6A), is illustrated in part in Figures 77 and 85, where sample linkage coding methods are demonstrated.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  COBMAIN.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-360-F50.
OBJECT-COMPUTER.  IBM-360-F50.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FILE-X ASSIGN TO UR-2540R-S-INFILE.
I-O-CONTROL.
DATA DIVISION.
FILE SECTION.
FD  FILE-X
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORD IS OMITTED.
01  IN-REC.
    05  TYPEN PIC X.
    05  HOLDER PIC X.
    05  FILLER PIC X(78).
WORKING-STORAGE SECTION.
77  SIGNAL PIC X(8).
PROCEDURE DIVISION.
    .
    .
    .
    OPEN INPUT FILE-X.
    READ FILE-X AT END GO TO CLOSE-FILE.
    .
    .
    .
    CALL 'COBOL1' USING IN-REC.
    .
    .
    .
    CALL 'COBREGN1' USING IN-REC.
    .
    .
    .
    CALL 'ASSMRTN' USING SIGNAL.
    .
    .
    .
CLOSE-FILE.  CLOSE FILE-X.
    .
    .
    .
    STOP RUN.
```

Figure 78. Sample Calling and Called Programs (Part 1 of 6)

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  COBOL1.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-360-F50.
OBJECT-COMPUTER.  IBM-360-F50.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
I-O-CONTROL.
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
77  TRANS-COBL PIC X(7).
LINKAGE SECTION.
01  PASS-REC.
    05  FILLER PIC X.
    05  TRANS-VALUE PIC X.
    05  FILLER PIC X(78).
PROCEDURE DIVISION USING PASS-REC.
    .
    .
    .
    CALL 'COBOL1A' USING TRANS-COB1.
    .
    .
    .
    CALL 'COBOLIB' USING TRANS-COB1.
    .
    .
    .
    GOBACK.
```

Figure 78. Sample Calling and Called Programs (Part 2 of 6)

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  COBOL1A.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-360-F50.
OBJECT-COMPUTER.  IBM-360-F50.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
I-O-CONTROL.
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
77  TRANS-COB1A PIC X(7).
PROCEDURE DIVISION USING TRANS-COB1A.
    .
    .
    .
    GOBACK.
```

Figure 78. Sample Calling and Called Programs (Part 3 of 6)

```
IDENTIFICATION DIVISION.
PROGRAM-ID.   COBOL1B.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-360-F50.
OBJECT-COMPUTER.  IBM-360-F50.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
I-O-CONTROL.
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
77  TRANS-COBREGN PIC X(7).
LINKAGE SECTION.
77  TRANS-COB1B PIC X(7).
PROCEDURE DIVISION USING TRANS-COB1B.
    .
    .
    .
    CALL 'COBREGN0' USING TRANS-COBREGN.
    .
    .
    .
    GOBACK.
```

Figure 78. Sample Calling and Called Programs (Part 4 of 6)

```
IDENTIFICATION DIVISION.
PROGRAM-ID.   COBREGN0.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-360-F50.
OBJECT-COMPUTER.  IBM-360-F50.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
I-O-CONTROL.
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
77  TRANS-COB PIC X(7).
77  TRANS-ASSM PIC X(4).
01  PASS-REC.
    05  FILLER PIC X.
    05  TRANS-VALUE PIC X.
    05  FILLER PIC X(78).
PROCEDURE DIVISION USING TRANS-COB.
    .
    .
    .
    GOBACK.
B.  ENTRY 'COBREGN1' USING PASS-REC.
    .
    .
    .
    GOBACK.
C.  ENTRY 'COBREGN2' USING TRANS-ASSM.
    .
    .
    .
    GOBACK.
```

Figure 78. Sample Calling and Called Programs (Part 5 of 6)

```
ASSMPGM   START 0
          PRINT NOGEN
          ENTRY ASSMRTN           ESTABLISHES ASSMRTN AS AN EXTERNAL NAME THAT CAN BE
*                                  REFERRED TO IN ANOTHER PROGRAM.
          USING ASSMRTN,15

*  SAVE ROUTINE
ASSMRTN   SAVE  (14,12)           STORES THE CONTENTS OF REGISTERS 14, 15, 0, AND 1
*                                   IN WORDS 4, 5, 6, AND 7 OF THE SAVE AREA.
*                                   THESE ARE CONVENTIONAL LINKAGE REGISTERS.
*                                   REGISTERS 2 THROUGH 12, WHICH ARE NOT
*                                   ACTUALLY USED FOR LINKAGE, ARE SAVED IN SUBSEQUENT
*                                   WORDS OF THE SAVE AREA.  THE EXPANDED CODE OF THE
*                                   SAVE MACRO INSTRUCTION USES REGISTER 13, WHICH
*                                   CONTAINS THE ADDRESS OF THE SAVE AREA, IN
*                                   EFFECTING THE STORAGE OF REGISTERS.
          LR    10,15
          DROP  15
          USING ASSMRTN,10
          LR    11,13             LOADS THE ADDRESS OF THE SAVE AREA INTO REGISTER 11,
*                                   WHICH WILL SUBSEQUENTLY BE USED TO REFER TO THE
*                                   SAVE AREA.

          LA    13,AREA           LOADS THE ADDRESS OF THIS PROGRAM'S SAVE AREA INTO
*                                   REGISTER 13.

          ST    13,8(11)          STORES THE ADDRESS OF THIS PROGRAM'S SAVE AREA INTO
*                                   WORD 3 OF THE SAVE AREA OF THE CALLING PROGRAM.

          ST    11,4(13)          STORES THE ADDRESS OF THE PREVIOUS SAVE AREA INTO
*                                   WORD 2 OF THIS PROGRAM'S SAVE AREA.
          B     PROCESS
AREA      DS    18F               RESERVES 18 WORDS FOR THE SAVE AREA.

PROCESS   L     2,0(1)            LOADS INTO REGISTER 2 THE ADDRESS OF THE IDENTIFIER-
*                                   LIST PASSED TO THE PROGRAM.  THE ADDRESS OF THE
*                                   IDENTIFIER-LIST IS ALWAYS PASSED IN REGISTER 1,
*                                   WHICH IS USED HERE AS THE BASE REGISTER TO GET THE
*                                   ADDRESS.  SUBSEQUENT REFERENCES TO THE IDENTIFIER
*                                   WILL USE REGISTER 2 AS THE BASE REGISTER FOR THAT
*                                   ADDRESS.  (IF A VARIABLE-LENGTH IDENTIFIER-LIST
*                                   COULD BE USED IN CALLING THIS PROGRAM, EACH
*                                   IDENTIFIER WOULD BE TESTED FOR A ONE IN THE
*                                   HIGH-ORDER BIT.)

          {User-written program statements}

*  CALLING SEQUENCE
          LA    1,ARGLST          LOADS INTO REGISTER 1 THE ADDRESS OF THE IDENTIFIER-
*                                   LIST TO BE PASSED.

          CALL  COBREGN2          TRANSFERS CONTROL TO THE ENTRY POINT OF THE CALLED
*                                   PROGRAM.  [THE CALL MACRO INSTRUCTION GENERATES
*                                   CODING THAT LOADS A V-TYPE ADDRESS CONSTANT --
*                                   COBREGN2 -- INTO REGISTER 15 AND PLACES INTO
*                                   REGISTER 14 THE RETURN ADDRESS (THAT IS, THE
*                                   ADDRESS OF THE FIRST BYTE FOLLOWING THE MACRO
*                                   EXPANSION)].

          {User-written program statements}
```

Figure 78. Sample Calling and Called Programs (Part 6A of 6)

```
*   CALLING SEQUENCE
            LA     1,ARGLST            LOADS INTO REGISTER 1 THE ADDRESS OF THE
                                       IDENTIFIER-
*                                        LIST TO BE PASSED.

            CALL   COBREGN2            TRANSFERS CONTROL TO THE ENTRY POINT OF THE CALLED
*                                        PROGRAM.  [THE CALL MACRO INSTRUCTION GENERATES
*                                        CODING THAT LOADS A V-TYPE ADDRESS CONSTANT --
*                                        COBREGN2 -- INTO REGISTER 15 AND PLACES INTO
*                                        REGISTER 14 THE RETURN ADDRESS (THAT IS, THE
*                                        ADDRESS OF THE FIRST BYTE FOLLOWING THE MACRO
*                                        EXPANSION)].

            {User-written program statements}

*   RETURN ROUTINE
            L      13,4(13)            LOADS THE ADDRESS OF THE PREVIOUS SAVE AREA
*                                        BACK INTO REGISTER 13.

            RETURN(14,12),T,RC=(15) THIS RETURN MACRO INSTRUCTION RESTORES THE SAVED
*                                        REGISTERS (14, 15, AND 0 THROUGH 12).  THE RETURN
*                                        ADDRESS IS RESTORED TO REGISTER 14, AND THE
*                                        EXPANSION INCLUDES A BRANCH TO THAT INSTRUCTION.
*                                        THE 'T' IN THE RETURN MACRO INSTRUCTION CAUSES
*                                        THE EIGHT HIGH-ORDER BITS OF WORD 4 OF THE SAVE
*                                        AREA TO BE SET TO ONES AS AN INDICATION THAT THE
*                                        RETURN HAS OCCURRED.  THE RC=(15) PARAMETER
                                         INDICATES THAT THIS PROGRAM IS PASSING A RETURN
                                         CODE IN REGISTER 15.

*   PARAMETER LIST
            DS     OF                  THIS PARAMETER LIST CONTAINS ONLY 1 ARGUMENT.
ARGLST      DC     X'80'
            DC     AL3(ARGUMENT)       FIRST BYTE OF LAST ARGUMENT (ONLY ARGUMENT IN
ARGUMENT    DC     C'1'                  THIS PROGRAM) SETS BIT 0 TO 1.
            END
```

Figure 78. Sample Calling and Called Programs (Part 6B of 6)

LINK-EDITING PROGRAMS

Each time an entry point is specified in a called program, an external name is defined (except when a program is compiled using the DYNAM and RESIDENT compiler options). An external name is a name that can be referred to by another separately compiled or assembled program. Each time an entry name is specified in a calling program, an external reference is defined except when a program is compiled using the DYNAM and RESIDENT compiler options. An external reference is a symbol that is defined as an external name in another separately compiled or assembled program. The linkage editor resolves external names and references and combines calling and called programs into a format suitable for execution together, i.e., as a single load module except when programs are compiled with dynamic CALL statements and/or the RESIDENT option (see the section entitled "Programs Compiled with the DYNAM and/or RESIDENT Options").

Load modules of both calling and called programs are used as input to the linkage editor. There are two kinds of input, primary and additional. Primary input consists of a sequential data set that contains one or more separately compiled object modules and/or linkage editor control statements. The primary input can contain object modules that are either calling or called programs or both. Additional input consists of object modules or load modules that are not part of the primary input data set but are to be included in the load module. The additional input may be in the form of (1) a sequential data set consisting of one or more object modules with or without linkage editor control statements, or (2) libraries containing object modules with or without linkage editor control statements, or (3) libraries consisting of load modules. Note that the secondary input (all libraries and/or data sets) must be composed of either all object modules or all load modules, but it cannot contain both types. The additional input is specified by

Word No.	Area No.	Contents
1	AREA	Used by COBOL.
2	AREA +4	Address (passed by the calling program) of the save area used by the calling program. This is the address of a save area that was passed to the called program by the program that called the called program.
3	AREA +8	Address (stored by the called program) of the next save area, that is, the save area that the called program provides for a program that it calls. The called program need not reserve a save area if it does not, in turn, call another program.
4	AREA +12	Return address (contents of register 14) stored by the called program.
5	AREA +16	Entry point address (contents of register 15) stored by the called program.
6	AREA +20	Contents of register 0 (stored by the called program).
7	AREA +24	Contents of register 1 (stored by the called program); that is, the address of the identifier list passed to the called program.
8 . . . 18	AREA +28 . . . AREA +68	Contents of registers 2 through 12 (stored by the called program).

Figure 79. Save Area Layout and Contents

linkage editor control statements in the primary input and a DD statement for each additional input data set. Additional input may contain either calling or called programs or both.

Note: Each additional input data set may itself contain external references or names and linkage editor control statements that specify more additional input.

SPECIFYING PRIMARY INPUT

The primary input data set is specified for linkage editor processing by the SYSLIN DD statement. The linkage editor must always have a primary input data set specified by a SYSLIN DD statement whether or not there are called or calling programs and even if the primary input data set contains only linkage editor control statements. The SYSLIN DD statement that specifies the primary input is discussed in "Linkage Editor Data Set Requirements" (see "Example of Linkage Editor Processing" for a discussion of how to specify a primary input data set that contains more than one object module along with linkage editor control statements).

SPECIFYING ADDITIONAL INPUT

Additional input data sets are specified by linkage editor control statements and a DD statement for each additional input data set.

The linkage editor control statements that specify additional input are INCLUDE and LIBRARY.[1] A primary input data set may consist entirely of such statements. The INCLUDE and LIBRARY statements may be placed before, between, or after object modules or other control statements in either primary or additional input data sets. One method of using these statements is shown in Figure 87.

Note: Additional input often contains members of libraries (see "Specifying Libraries as Additional Input" in "Libraries").

----

[1]The operation field in a linkage editor control statement must start after column 1. The operand field must be preceded by at least one blank.

INCLUDE Statement

The INCLUDE statement is used to include an additional input data set that is either a member of a library or a sequential data set. Its format is:

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

where ddname indicates the name of the DD statement that specifies the library or sequential data set, and member-name is the name of the library member that is to be included. Member-name is not used when the additional input data set is not a member of a partitioned data set.

LIBRARY Statement

The LIBRARY statement is used to include additional input that may be required to resolve external references.

The format is:

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

where ddname indicates the name of the DD statement that specifies the library, and member-name is the name of the member of the library.

The LIBRARY statement differs from the INCLUDE statement in that libraries specified in the LIBRARY statement are not searched for additional input until all other processing, except references reserved for the automatic library call, is completed by the linkage editor. Any additional module specified by an INCLUDE statement is incorporated immediately, whenever the INCLUDE statement is encountered.

ALIAS Statement

The ALIAS statement specifies additional names for the output library member, and can also display names of additional entry points. If a load module has more than one entry point or more than one CSECT and the user wishes to access that alternate entry at execution time via a dynamic CALL, he should specify an ALIAS with the same symbolic name as the desired entry point or CSECT.

Operation	Operand
ALIAS	$\left\{\begin{array}{l}\text{symbol}\\ \text{external name}\end{array}\right\} \left[\begin{array}{l}\text{,symbol}\\ \text{,external name}\end{array}\right]$

where symbol specifies an alternate name for the load module, and external name specifies a name that is defined as a control section name in the output module.

If the linkage-editor input includes an ALIAS statement, the symbolic name specified is identified with the relative location of the entry point or CSECT name that matches the ALIAS. If there is no matching entry point or CSECT name, the ALIAS is identified with relative location zero in the load module.

NAME Statement

The NAME statement specifies the name of the load module created from the preceding input modules, and serves as a delimiter for input modules, and serves as a delimiter for input to the load module. The NAME statement may be used to assign a symbolic name to a load module. This symbolic name is entered in the directory of the partitioned data set that contains the module, and allows the module to be accessed at execution time by an OS LOAD macro. A Load module name is always associated with relative location zero in the load module.

Operation	Operand
NAME	member-name [(R)]

where member-name specifies the name to be assigned to the load module that is created from the preceding input modules, and (R) indicates that this load module replaces an identically named module in the input module library. (If the module is not a replacement, the parenthesized value (R) should not be specified.)

If the linkage-editor input includes a NAME statement, the symbolic name specified is always identified with relative location zero in the load module.

PROGRAMS COMPILED WITH THE DYNAM AND/OR RESIDENT OPTIONS

In the usual called/calling situation, all references to any subprogram or library subroutines generated in an object program result in a V-type address constant (VCON) that must be resolved by the linkage editor. Therefore, at link-edit time, the modules referred to by VCONs are made a part of a single load module containing the object program and all required subprograms and library routines. When the object program is executed, all those required routines are present in the user region for the entire execution step, even though they may have been used only at the beginning of the main program and never invoked again. With dynamic linkage, on the other hand, the user can invoke a called program when it is needed and retain it for only the period needed.

Subprograms invoked through the CALL literal statement are dynamically loaded using the Operating System LOAD macro if DYNAM is specified. Before the CALL Subprogram is executed, linkage is effected for all COBOL library subroutines required by the subprogram. Similarly, use of the CANCEL statement makes it possible to dynamically delete subprograms at object time.

Figure 76 earlier in this chapter is an example of a job compiled with the DYNAM and RESIDENT options. Figures 80 through 83 in this section illustrate for called/calling programs the relationship between the possible combinations of the DYNAM/RESIDENT options and the identifier and literal options of the CALL and CANCEL statements. Figure 84 shows the JCL necessary for compiling, link-editing, and executing a calling COBOL program and a called COBOL program when both of the programs invoke the DYNAM and RESIDENT compiler options.

When a program is compiled with DYNAM and RESIDENT, no external references are generated. Therefore, while the program may refer to other modules, no references are resolved by the linkage editor. In such a case, the only input to the linkage editor is the program itself. Any module the program refers to must exist in load module form in a library that is available to the system at execution time.

The link-editing that takes place varies with the combinations of the DYNAM(NODYNAM) and RESIDENT(NORESIDENT) options in effect. What would seem to be the most representative link-edit situations are discussed in the sections that follow.

Specifying DYNAM/RESIDENT

When both DYNAM and RESIDENT are specified for the called/calling situation pictured in Figure 78, first the main program COBA is compiled and link-edited; then each of the two subprograms COBB and COBC is compiled and link-edited separately, thereby producing three modules. Then the main program is executed.

In this situation, all external references are dynamically resolved. Therefore, no VCONs are generated for the address of an external symbol that would be used in a static situation (that is, a CALL literal without the DYNAM option) to effect branches to other programs.

```
┌────┐           ┌────┐           ┌────┐
│    │CALL       │    │CALL       │    │
│COBA├──────────>│COBB├──────────>│COBC│
│    │literal    │    │literal    │    │
└────┘           └────┘           └────┘
```

Figure 80. CALL with DYNAM and RESIDENT

Specifying NODYNAM/RESIDENT

When NODYNAM and RESIDENT are specified for the called/calling situation pictured in Figure 81, a dynamic situation occurs because of the inclusion of CALL identifier in the calling programs. That is, because the name of the called subprogram is not available until execution time, a CALL identifier statement cannot be used in a static situation.

Moreover, when NODYNAM and NORESIDENT are either specified or implied by default, and a CALL identifier or CANCEL identifier statement occurs in the source program being compiled, the Library Management Feature is automatically in effect.

Note: A printed indication of the compiler options in effect appears in the statistics section of the compiler output. (For examples of compiler statistics, see the chapter entitled "Output.")

```
┌────┐          ┌────┐          ┌────┐
│    │CALL      │    │CALL      │    │
│COBA├─────────>│COBB├─────────>│COBC│
│    │identifier│    │identifier│    │
└────┘          └────┘          └────┘
```

Figure 81. CALL With NODYNAM and RESIDENT

In contrast with Figure 81, the called/calling situation pictured in Figure 82 invokes the CALL literal option. Again the programs are compiled in the order COBA, COBB, and COBC. The CALL literal statements included in programs COBA and COBB result in static calls that must be resolved by the linkage editor. However, with the COBOL Library Management Feature in effect, linkage to the library is dynamic. That is, the required COBOL object-time library subroutines are not link-edited, but linkage is effected dynamically at object time.

Note: When including both dynamic and static CALL statements in the same run unit, the programmer should not dynamically call any subprograms that are otherwise called statically. To do so might cause multiple copies of the called program to be created and, therefore, produce unpredictable results.

```
          CALL         CALL
          literal--->  literal--->
┌─────────┬─────────┬───────────────┐
│         │         │               │
│ COBA    │ COBB    │     COBC      │
│         │         │               │
└─────────┴─────────┴───────────────┘
```

Figure 82. CALL With NODYNAM and RESIDENT With CALL Literal Option

Specifying NODYNAM/NORESIDENT

For the called/calling situation pictured in Figure 83, the COBOL Library Management Feature is not in effect, and all CALL statements result in static calls that must be resolved by the linkage editor. One load module is produced for the programs COBA, COBB, COBC, and all of the necessary COBOL library subroutines.

The NODYNAM/NORESIDENT set of options should be used only when the user does not intend to use the CALL or CANCEL identifier statement or the Library Management Feature. If either a CALL identifier or a CANCEL identifier statement appears in any one program, the Library Management Feature is in effect for that program only. This situation may result in a duplication of subprograms and COBOL library subroutines within the user region/partition, thereby causing unpredictable results.

```
┌──────────┬────────────┬────────────┐
│          │            │            │
│ COBA     │ COBB       │ COBC       │
└────┬─────┴────────────┴──────┬─────┘
     │COBOL Library Subroutines│
     └─────────────────────────┘
```

Figure 83. CALL With NODYNAM and NONRESIDENT

```
//JOBY          JOB
//STEP1         EXEC    PGM=IKFCBL00,PARM='LOAD,DYNAM,RESIDENT'
                .
                .
                .
//SYSLIN        DD      DSNAME=&&LINKDS1,DISP=(MOD,PASS),UNIT=SYSSQ
//SYSIN         DD      *

                {Source module for COBMAIN, a calling COBOL program}

                CALL 'COBSUB'
/*
//STEP2         EXEC    PGM=IEWL
//SYSLMOD       DD      DSNAME=&&GOFILE,DISP=(MOD,PASS),UNIT=SYSSQ
//SYSLIN        DD      DSNAME=&&LINKDS1,DISP=(OLD,DELETE),UNIT=SYSSQ
//SYSIN         DD      *
  NAME          COBMAIN
/*
//STEP3         EXEC    PGM=IKFCBL00,PARM=LOAD,DYNAM,RESIDENT
//SYSLIN        DD      DSNAME=&&LINKDS2,DISP=(MOD,PASS),UNIT=SYSSQ
//SYSIN         DD      *

                {Source module for COBSUB, a called COBOL program}
/*
//STEP4         EXEC    PGM=IEWL
//SYSLMOD       DD      DSNAME=&&GOFILE,DISP=(MOD,PASS),UNIT=SYSSQ
//SYSLIN        DD      DSNAME=&&LINKDS2,DISP=(OLD,DELETE),UNIT=SYSSQ
//SYSIN         DD      *
  NAME          COBSUB
/*
//STEP5         EXEC    PGM=COBMAIN
//STEPLIB       DD      DSNAME=&&GOFILE,DISP=(OLD,DELETE),UNIT=SYSSQ
/*
```

Figure 84. Sample JCL for Called/Calling Programs Compiled with the DYNAM and RESIDENT Options

```
          .
          .
          .
          ENTRY ASSMRTN          ESTABLISHES ASSMRTN AS AN EXTERNAL NAME THAT CAN BE
*                                  REFERRED TO IN ANOTHER PROGRAM.
          .
          .
          .

*  SAVE ROUTINE
ASSMRTN SAVE (14,12)             STORES THE CONTENTS OF REGISTERS 14, 15, 0, AND 1
*                                  IN WORDS 4, 5, 6, AND 7 OF THE SAVE AREA.  THESE
*                                  ARE CONVENTIONAL LINKAGE REGISTERS.  REGISTERS 2
*                                  THROUGH 12, WHICH ARE NOT ACTUALLY USED FOR
*                                  LINKAGE, ARE SAVED IN SUBSEQUENT WORDS OF THE SAVE
*                                  AREA.  THE EXPANDED CODE OF THE SAVE MACRO
*                                  INSTRUCTION USES REGISTER 13, WHICH CONTAINS THE
*                                  ADDRESS OF THE SAVE AREA, IN EFFECTING THE STORAGE
*                                  OF REGISTERS.

          LR    11,13            LOADS THE ADDRESS OF THE SAVE AREA INTO REGISTER 11,
*                                  WHICH WILL SUBSEQUENTLY BE USED TO REFER TO THE
*                                  SAVE AREA.

          LA    13,AREA          LOADS THE ADDRESS OF THIS PROGRAM'S SAVE AREA INTO
*                                  REGISTER 13.

          ST    13,8(11)         STORES THE ADDRESS OF THIS PROGRAM'S SAVE AREA
*                                  INTO WORD 3 OF THE SAVE AREA OF THE CALLING
*                                  PROGRAM.

          ST    11,4(13)         STORES THE ADDRESS OF THE PREVIOUS SAVE AREA INTO
*                                  WORD 2 OF THIS PROGRAM'S SAVE AREA
          .
          .
          .

AREA      DS    18F'0'           RESERVES 18 WORDS FOR THE SAVE AREA AND
          .                        INITIALIZES THEM TO ZERO.
          .
          .

*  RETURN ROUTINE
          L     13,4(13)         LOADS THE ADDRESS OF THE PREVIOUS SAVE AREA BACK
*                                  INTO REGISTER 13.

          RETURN(14,12),T,RC=(15) THIS RETURN MACRO INSTRUCTION RESTORES THE SAVED
*                                  REGISTERS (14, 15, AND 0 THROUGH 12).  THE RETURN
*                                  ADDRESS IS RESTORED TO REGISTER 14, AND THE
*                                  EXPANSION INCLUDES A BRANCH TO THAT INSTRUCTION.
*                                  THE 'T' IN THE RETURN MACRO INSTRUCTION CAUSES THE
*                                  EIGHT HIGH-ORDER BITS OF WORD 4 OF THE SAVE AREA
*                                  TO BE SET TO ONES AS AN INDICATION THAT THE RETURN
*                                  HAS OCCURRED.  THE RC=(15) PARAMETER INDICATES
*                                  THAT THIS PROGRAM IS PASSING A RETURN CODE IN
*                                  REGISTER 15; THIS VALUE SHOULD BE SET TO ZERO
*                                  IF NONE IS WANTED.
```

Note: If the called program containing this coding did not call another program, it would not require a reserved save area (AREA) and the coding to store the save area's address.

Figure 85. Sample Linkage Coding Used in a Called Assembler-Language Program that Calls Another Program

```
//CALLPROG      JOB
//STEP1         EXEC     PGM=IKFCBL00,PARM=(LOAD,NODECK)
                .
                .
                .
//SYSLIN        DD       DSN=&&TEMPLIB1,UNIT=SYSSQ,DISP=(NEW,PASS),     X
//             SPACE=(TRK,(10,1))
//SYSIN         DD       *
               (Source module for COBSUB, a called COBOL program)
/*
//STEP2         EXEC     PGM=IEUASM,PARM=(LOAD,NODECK),                 X
//             COND=(9,LT,STEP1)¹
//SYSGO         DD       DSN=&&TEMPLIB1,UNIT=SYSSQ,DISP=(MOD,PASS)
//SYSIN         DD       *
               (Source module for ASSMMAIN, a calling assembler-
                language program)
/*
//STEP3         EXEC     PGM=IEWL,PARM=(LIST,XREF,LET),                 X
//             COND=((9,LT,STEP1),(5,LT,STEP2))
                .
                .
                .
//PROGLIB1      DD       DSN=&&TEMPLIB1,DISP=OLD
//SYSLIN        DD       *
                INCLUDE  PROGLIB1²
                ENTRY    ASSMMAIN³
/*
//STEP4         EXEC     PGM=*.STEP3.SYSLMOD,COND=((9,LT,STEP1),        X
//             (5,LT,STEP2),(5,LT,STEP3))
//SYSOUT        DD       SYSOUT=A
```

[1] This example was chosen to illustrate the testing of condition codes.
[2] See the discussion under the INCLUDE statement.
[3] Because the COBOL program is compiled first and the linkage editor cannot identify the proper entry point, the ENTRY statement must be included.

Figure 86. Sample Coding Used for a Calling Assembler-Language Program and a Called COBOL Program

LINKAGE EDITOR PROCESSING

The linkage editor first processes the primary input and any additional input specified by INCLUDE statements. All external references in the primary that refer only to other modules in the included input are resolved first. If there are still unresolved references after this input is processed, the automatic call library, which includes libraries specified by the SYSLIB DD statement and by the LIBRARY statements, is searched to resolve the references. The automatic call library generally will contain the COBOL library subroutines. (External references to these subroutines are generated by the COBOL compiler when statements in the source module require certain functions to be performed, such as some data conversions.)

If the additional input contains external references and/or linkage editor control statements, the references are resolved in the same way. Data sets specified by the INCLUDE statement are incorporated when the statement is encountered. Data sets specified by the LIBRARY statement are used only when there are unresolved references after all of the other processing is completed.

```
//JOBX         JOB
//STEP1        EXEC      PGM=IKFCBL00,PARM=LOAD
               .
               .
               .

//SYSLIN       DD        DSNAME=&&GOFILE,DISP=(MOD,PASS),UNIT=SYSSQ
//SYSIN        DD        *

               (Source module for COBMAIN)

/*
//STEP2        EXEC      PGM=IKFCBL00,PARM=LOAD
               .
               .
               .

//SYSLIN       DD        DSNAME=*.STEP1.SYSLIN,DISP=(MOD,PASS)
//SYSIN        DD        *

               (Source module for COBOL1)

/*
//STEP3        EXEC      PGM=IKFCBL00,PARM=LOAD
               .
               .
               .

//SYSLIN       DD        DSNAME=*.STEP2.SYSLIN,DISP=(MOD,PASS)
//SYSIN        DD        *

               (Source module for COBOL1A)

/*
//STEP4        EXEC      PGM=IEWL
               .
               .
               .

//SYSLIB       DD        DSNAME=SYS1.COBLIB,DISP=OLD
//SYSLMOD      DD        DSNAME=PGMLIB(CALPGM),DISP=NEW,UNIT=2311,SPACE= X
//                       (1024,(50,20,2)),VOLUME=SER=LIBPAK
//DBLIB        DD        DSNAME=DBJLIB,DISP=OLD
//ADDLIB       DD        DSNAME=MYLIB,DISP=OLD
//SYSLIN       DD        DSNAME=&&GOFILE,DISP=(OLD,DELETE)               X
//             DD        *
               INCLUDE   DBLIB(COBOL1B,ASSMPGM)
               LIBRARY   ADDLIB (COBREGN0)
/*
```

Figure 87. Specifying Primary and Additional Input to the Linkage Editor

<u>Example of Linkage Editor Processing</u>

Figure 87 shows the control statements for a job that separately compiles three source modules (one is a calling program and two are called programs) and places them in one data set as primary input for the linkage editor. The linkage editor then links them together with additional input (called programs that are members of the specified libraries) to form one load module.

STEP1 compiles a source module called COBMAIN, STEP2 compiles a source module called COBOL1, and STEP3 compiles a source module called COBOL1A. The object module from each step is placed in the sequential data set called &&GOFILE. (Since MOD and PASS are specified for &&GOFILE in the SYSLIN DD statement in STEP1, the object modules COBOL1 and COBOL1A are placed in the data set behind the object module COBMAIN.)

In STEP4, the linkage editor uses the &&GOFILE data set as primary input, and the cataloged libraries MYLIB, OBJLIB, and SYS1.COBLIB as additional input. (The INCLUDE and LIBRARY statements become part of the primary input through the DD * statement following the SYSLIN DD statement.

The object modules of the data set &&GOFILE and the members COBOL1B and ASSMPGM of OBJLIB are processed first. If there are unresolved references after this input is processed, the linkage editor searches the automatic call library, which includes the COBOL subroutine library and member COBREGNO of MYLIB, to resolve these references. OBJLIB is specified in the OBLIB DD statement and MYLIB in the ADDLIB DD statement.

After linkage editor processing is completed, a new library, PGMLIB, is created with CALPGM as a member. CALPGM contains COBMAIN, COBOL1, COBOL1A, COBOL1B, ASSMPGM, and, possibly, COBOL subroutines and COBREGNO.

OVERLAY STRUCTURES

If the called programs needed to execute one COBOL source program do not all fit into main storage at the same time, it is still possible to use them with the overlay technique or with the use of the segmentation feature. Called programs that do not need to be in main storage at the same time can be given the same relative storage address and then loaded at different times during execution when they are needed. In this way, the same storage space can be used for more than one called program. The use of segmentation is discussed in "Using the Segmentation Feature."

Considerations for Overlay

Assume that the six programs illustrated in Figure 78 have the following load module sizes:

Program	Module Size (in Bytes)
COBMAIN	11,000
COBOL1	4,000
COBOL1A	6,000
COBOL1B	5,000
COBREGNO	3,000
ASSMPGM	13,000

Through the linkage mechanism, CALL COBOL1..., all subprograms plus COBMAIN must be link-edited together to form one module 42,000 bytes in size. Therefore, COBMAIN would require 42,000 bytes of storage in order to be executed.

If the subprograms needed do not fit into main storage, the following three techniques of overlay are available to the COBOL programmer:

- Preplanned overlay using the linkage editor
- Dynamic overlay using macro instructions during execution
- Segmentation Feature

Note: The largest load module that can be processed by Fetch is 524,248 bytes. If a load module exceeds this limit, it should be divided.

Linkage Editing with Preplanned Overlay

The preplanned linkage editor facility permits the reuse of storage locations already occupied. By judiciously modularizing a program and using the linkage editor overlay facility, a program that is too large to fit into storage at one time can be executed.

In using the preplanned overlay technique, the programmer specifies to the linkage editor which subprograms are to overlay each other. The subprograms specified are processed as part of the program by the linkage editor, so they can be automatically placed in main storage for execution when requested by the program. The resulting output of the linkage editor is called an overlay structure.

It is possible, at linkage edit time, to set up an overlay structure by using the COBOL source language linkage statement and the linkage editor OVERLAY statement. These statements enable a user to call a subprogram that is not actually in storage The details for setting up the linkage editor control statements for accomplishing this procedure can be found in the publication IBM OS Linkage Editor and Loader.

In a linkage editor run, the programmer specifies the overlay points in a program by using OVERLAY statements. The linkage editor treats the entire input as one program, resolving all symbols and inserting tables into the program. These tables are used by the control program to bring the overlay subprograms into storage automatically when called.

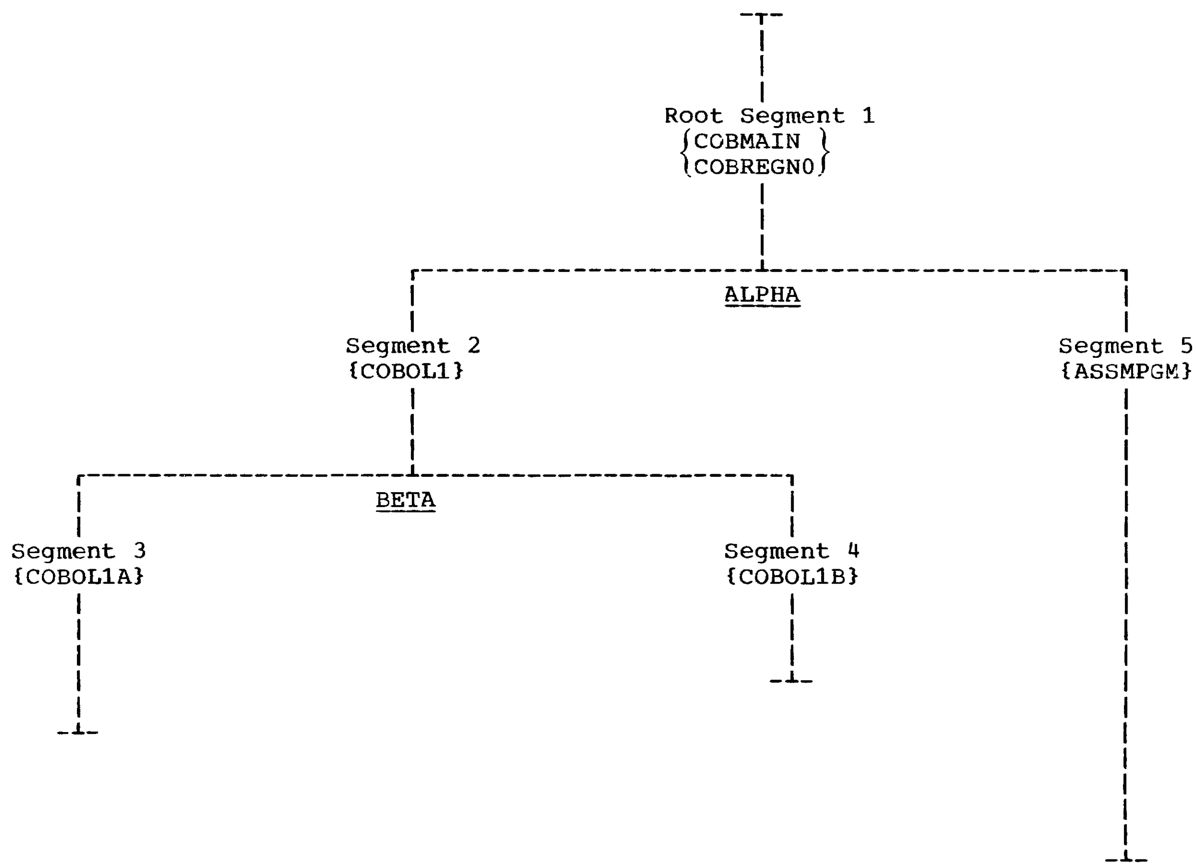


Figure 88. Overlay Tree Structure

Figure 88 is an overlay tree structure illustrating how the six programs in Figure 78 could be positioned in core at execution time using preplanned linkage editor overlay.

Figure 89 shows the deck arrangement required to achieve the overlay illustrated in Figure 88. The OVERLAY statements specify to the linkage editor that the overlay structure to be established is one in which the called programs of COBOL1 (COBOL1A and COBOL1B) overlay each other when called for execution, and that ASSMPGM and COBOL1 and its called program overlay each other when called.

Routine COBREGNO is placed with COBMAIN in the root segment of the overlay structure because it is called by three of the routines in the program, the largest of which is ASSMPGM. Utilizing COBREGNO as an individual overlay segment would not have resulted in a net decrease in the amount of core required for execution because the minimum amount of core needed would have to contain COBMAIN, ASSMPGM, and COBREGNO at the same time. Creating another overlay segment for COBREGNO would only have added to the amount of time required for program execution.

```
//OVERLAY     JOB       NY83937800,COSMO,MSGLEVEL=1
//STEP1       EXEC      PGM=IEWL,PARM='OVLY,LIST,XREF,LET'
//SYSLIB      DD        DSNAME=SYS1.COBLIB,DISP=SHR
//SYSPRINT    DD        SYSOUT=A
//SYSUT1      DD        UNIT=SYSDA,SPACE=(1024,(50,20))
//SYSLMOD     DD        DSNAME=&GODATA(RUN),DISP=(NEW,PASS),UNIT=SYSDA, X
//                      SPACE=(1024,(50,20,1))
//SYSLIN      DD        *

        {COBMAIN        object deck}
        {COBREGNO       object deck}

OVERLAY ALPHA
        {COBOL1         object deck}

OVERLAY BETA
        {COBOL1A        object deck}

OVERLAY BETA
        {COBOL1B        object deck}

OVERLAY ALPHA
        {ASSMPGM        object deck}
/*
```

Figure 89. Sample Deck for Linkage-Editor Overlay Structure

Dynamic Overlay Technique

In preparation for the dynamic overlay technique, each part of the program brought into storage independently should be processed separately by the linkage editor. (Hence, each part must be processed as a separate load module.) To execute the entire program, the programmer must:

1. Specify the main program in the EXEC statement.

2. Bring the separately processed load modules into storage when they are required, by using the appropriate supervisor linkage macro instructions. This is accomplished during execution.

The dynamic overlay technique can be used to overlay subprograms during execution. To accomplish dynamic overlay of subprograms, the programmer must write an assembler language subprogram that employs the LINK macro instruction to call each COBOL subprogram. For a detailed description of the LINK macro instruction, see the publication IBM OS: Supervisor and Data Management Macro Instructions.

In using the dynamic overlay technique, the main program communicates with the assembler language subprogram by using the COBOL language CALL statement. The CALL statement can be used to pass the name of the COBOL subprogram (to be linked) and the specified parameter list to the assembler language subprogram. This procedure is the same for each CALL used in the main program. Hence, each CALL results in linking with a subprogram through the assembler language subprogram.

When the COBOL subprogram is finished executing, it returns control to the assembler language subprogram, which in turn returns to the main program. The process is repeated for each CALL to the assembler-language subprogram.

Dynamic overlay requires that a programmer have detailed knowledge of the linkage conventions, assembler language, and the LINK macro instruction with its features and restrictions.

Figure 90 contains an example of a COBOL main program, PROGMAST, and an assembler language subprogram, LINKRTN. The two programs are link-edited together as a single load module. At execution time, the assembler-language subprogram dynamically fetches COBOL subprograms (OPN, BILL, CRDT TRNF, and LCK, none of which are shown in the example) for the main program using the LINK macro instruction. The COBOL subprograms are stored in a private library, DYNLINK.

The parameter list passed to LINKRTN contains three identifiers, TRANS-REC, COM-WORD, and SWITCH, two of which (TRANS-REC and SWITCH) are referenced by

LINKRTN, and two of which (TRANS-REC and COM-WORD) are referenced by the COBOL subprograms fetched. LINKRTN passes the same parameter list it receives to the COBOL subprograms fetched.

LINKRTN determines from identifier TRANS-REC which subprograms to fetch, and from SWITCH when to open and close the library DYNLINK.

Note: In structuring a program with either the preplanned overlay technique or the dynamic overlay technique, special consideration must be given to the presence of the TRANSFORM table and the class test tables, which are members of the COBOL object-time library (see "Appendix B: COBOL Library Subroutines"). The TRANSFORM table is link-edited with a COBOL program if the TRANSFORM statement is used. Similarly, one or more of the class test tables is present in a COBOL load module if a class test is performed or if the OCCURS DEPENDING ON option is used.

For these tables, which contain no executable code and are not branched to but are merely referenced, the compiler designates A-type address constants (ADCONs) and EXTRN references, rather than V-type address constants (VCONs). Accordingly, the overlay structure segment containing the table(s) must be either the root segment or a segment that is higher in the same leg as the segment containing the reference(s) to the table(s). This requirement has no effect on the COBOL segmentation feature (see the chapter entitled "Use of the Segmentation Feature"), since (1) all members of the object-time subroutine library are link-edited into the root segment, and (2) American National Standard COBOL subprograms may not be segmented.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  PROGMAST.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-360-F50.
OBJECT-COMPUTER.  IBM-360-F50.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FILE-Y ASSIGN TO UR-2540R-S-INFILE.
I-O-CONTROL.
DATA DIVISION.
FILE SECTION.
FD  FILE-Y
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORD IS OMITTED.
01  TRANS-REC.
    05  ACCOUNT-NUMBER PIC 9(10).
    05  TRANSACTION PIC 9(4).
    05  NAME PIC X(20).
    05  LOCATION PIC X(20).
    05  METER-READING PIC 9(6).
    05  DATE PIC 9(6).
    05  FILLER PIC X(8).
    05  AMOUNT PIC 9(6).
WORKING-STORAGE SECTION.
77  COM-WORD PIC X(12).
77  SWITCH PIC 9 VALUE ZERO.
PROCEDURE DIVISION.
    .
    .
    .
    OPEN INPUT FILE-Y.
B.  READ FILE-Y AT END GO TO END-RUN.
C.  CALL 'GETUM' USING TRANS-REC COM-WORD SWITCH.
    .
    .
    .
END-RUN.  CLOSE FILE-Y.
    MOVE 2 TO SWITCH.
    PERFORM C.
    STOP RUN.
```

Figure 90. Sample COBOL Main Program and Assembler-Language Subprogram Using Dynamic Overlay Technique (Part 1 of 3)

```
LINKRTN   START 0
          PRINT NOGEN
          ENTRY GETUM                   UPON ENTRY TO THIS PROGRAM, REGISTER 1 POINTS
*                                       TO A FIXED-LENGTH PARAMETER LIST OF THREE
*                                         WORDS.
*                                         THE FIRST WORD CONTAINS THE ADDRESS OF
*                                           RECORD TRANS-REC.
*                                         THE SECOND WORD CONTAINS THE ADDRESS OF
*                                           COM-WORD, TO WHICH THIS PROGRAM DOES NOT
*                                           REFER BUT WHICH IS USED BY ROUTINES THIS
*                                           PROGRAM LATER LINKS TO.
*                                         THE THIRD WORD CONTAINS THE ADDRESS OF
*                                           SWITCH USED BY THIS PROGRAM TO CHECK THE
*                                           STATUS OF THE PRIVATE LIBRARY DYNLINK
          USING GETUM,15
GETUM     SAVE  (14,12)
          LR    10,15
          DROP  15
          USING GETUM,10
          LR    11,13
          LA    13,SAVEAREA
          ST    13,8(11)
          ST    11,4(13)
          L     5,0(1)                  REGISTER 5 LOADED WITH ADDRESS OF TRANS-REC
          USING PARAMLST,5              REGTER 5 IS USED AS THE BASE REGISTER TO
*                                         REFERENCE TRANS-REC.
          B     OPENLIB
SAVEAREA  DS    18F

OPENLIB   L     6,8(1)                  REGISTER 6 LOADED WITH ADDRESS OF SWITCH.
          CLI   0(6),C'1'               CHECK SWITCH STATUS.
          BE    INITREG                 IF SWITCH = 1, DYNLINK IS ALREADY OPEN;
*                                         INITIALIZE REGISTERS.
          BH    CLOSLIB                 IF SWITCH > 1, DYNLINK IS NO LONGER NEEDED;
*                                         CLOSE DYNLINK.
          OPEN  (DYNLINK)               IF SWITCH = 0 THE FIRST TIME THROUGH, OPEN
*                                         DYNLINK.
          OI    0(6),C'1'               SET SWITCH SO THAT OPEN IS BYPASSED ON FUTURE
*                                         ENTRY.
*  TABLE LOOK-UP ROUTINE

INITREG   LA    2,RTNLST                INITIALIZE REGISTERS 2 AND 3 FOR LOOK-UP.
          LA    3,6
FINDRTN   CLC   TRANSACT,0(2)           TRANSACT CONTAINS THE TRANSACTION CODE THAT
*                                         DETERMINES WHICH ROUTINE TO FETCH.
          BE    GETRTN
          LA    2,12(0,2)
          BCT   3,FINDRTN
          MVC   ERRMSG+28(4),TRANSACT   PRODUCE ERROR MESSAGE IF TRANSACT CONTAINS
ERRMSG    WTO   'INVALID TRANSACTION'     AN INVALID CODE.
EXIT      L     13,4(13)
          SR    15,15                   SET REGISTER 15 TO ZERO.
          RETURN(14,12),T,RC=(15)       THE RC=(15) PARAMETER INDICATES THAT THIS
                                          PROGRAM IS PASSING A RETURN CODE IN REGISTER
                                          15.

*  DYNAMIC OVERLAY ROUTINE
GETRTN    L     1,24(11)                RESTORE REGISTER 1 TO ORIGINAL STATUS.
          LA    4,4(0,2)                PASS REGISTER 4 TO NAME OF ROUTINE TO BE
          LINK  EPLOC=(4),DCB=DYNLINK     FETCHED.  HAVE THE CONTROL PROGRAM
*                                         FETCH THE ROUTINE POINTED TO BY
*                                         REGISTER 4 FROM PRIVATE LIBRARY DYNLINK.
          B     EXIT
```

Figure 90. Sample COBOL Main Program and Assembler-Language Subprogram Using Dynamic Overlay Technique (Part 2 of 3)

```
CLOSLIB   CLOSE (DYNLINK)                  CLOSE PRIVATE LIBRARY.
          B     EXIT

          DS    0F
RTNLST    EQU   *                          AS THE TABLE SEARCHED BY THE TABLE LOOK-UP
*                                            ROUTINE, RTNLST CONTAINS A LIST OF ALL VALID
*                                            TRANSACTION CODES AND THE NAMES OF THE
*                                             ROUTINES FETCHED TO HANDLE THE TRANSACTIONS

          DC    C'0100'                    TRANSACTION CODE
          DC    CL8'OPN'                   ROUTINE NAME ASSOCIATED WITH ABOVE TRANSACTION
          DC    C'0200'
          DC    CL8'BILL'
          DC    C'0300'
          DC    CL8'CRDT'
          DC    C'0400'
          DC    CL8'TRNF'
          DC    C'0500'
          DC    CL8'LCK'

DYNLINK   EQU   *
          DCB   DDNAME=SYNLNKDD,DSORG=PO,MACRF=(R)
*                                          DCB TO DEFINE PRIVATE LIBRARY REFERRED TO IN
*                                            LINK MACRO INSTRUCTION.

PARAMLST  DSECT                            DSECT USED BY REGISTER 5 TO REFER TO TRANS-
*                                            REC.  THE RECORD DESCRIPTION CORRESPONDS TO
*                                            THAT OF TRANS-REC IN PROGMAST.
TRANSREC  DS    0CL80
ACCTNUM   DS    CL10
TRANSACT  DS    CL4
NAME      DS    CL20
LOCATION  DS    CL20
METERRD   DS    CL6
DATE      DS    CL6
          DS    CL8
AMOUNT    DS    CL6
          END
```

Note: Had a job or step library (requiring either a JOBLIB or STEPLIB DD statement in the job control for execution of the main program) been used instead of a private library (which for this example requires a DD statement named DYNLNKDD), responsibility for the opening and closing of the library would have been with the control program and not with LNKRTN.

The use of a private library reduces to a minimum the amount of search time needed to retrieve member modules from a library.

Figure 90. Sample COBOL Main Program and Assembler-Language Subprogram Using Dynamic Overlay Technique (Part 3 of 3)

LOADING PROGRAMS

The loader resolves external names and references and combines calling and called programs into a format suitable for execution as a single load module. For information on invoking the loader, see "Using the Cataloged Procedures."

When the dynamic call is used, all subprograms to be called dynamically must have been processed by the linkage editor. The loader may be used only to resolve references to subprograms invoked by static calls. Otherwise, load modules of both calling and called programs are used as input to the loader. There are two kinds of input, primary and additional. Primary input consists of one or more separately compiled object modules and/or load modules. Additional input consists of object modules or load modules that are not part of primary input data sets but are to be included in the load module. The additional input may be in the form of (1) libraries containing object modules, or (2) libraries containing load modules. Additional input may contain either calling or called programs or both.

SPECIFYING PRIMARY INPUT

The primary input data set is specified for loader processing by the SYSLIN DD statement. The loader must always have a primary input data set whether or not there are calling or called programs. The SYSLIN DD statement that specifies primary input is discussed in the section "Data Set Requirements."

SPECIFYING ADDITIONAL INPUT

Additional input data sets are specified by the SYSLIB DD statement. The SYSLIB DD statement is discussed in the section "Data Set Requirements."

Note: Neither the overlay facility nor the segmentation feature can be used with the loader.

Libraries are an integral part of the operating system. Some libraries have system-supplied names and system-supplied data. Other libraries have system-supplied names but may contain user-specified data. Still other libraries have both user-supplied names and user-supplied data.

Libraries, in general, are made up of partitioned data sets. Any library with a user-supplied name and user-supplied data is always a single partitioned data set, which is a collection of independent sets of sequentially organized data, called members. All of the members within a partitioned data set have the same characteristics as that of record format. When used to store programs, a partitioned data set containing load modules can contain only load modules; it cannot contain both load modules and object modules.

Each partitioned data set is headed by a directory of entries pointing to the members that make up the library. Each member has a unique member name. A partitioned data set must reside on a single mass storage device, but some libraries can consist of a concatenation of more than one partitioned data set.

Figure 91 shows the format of a library that is a single partitioned data set of four members. Space for the members of such a library and its directory is requested in the SPACE parameter of the DD statement when the library is created. Additional members can be added to a library at a later time. If additional space is required to store a member, allocation will be made in the amount specified by the secondary allocation in the SPACE parameter of the DD statement that was used when the library and its first member were created. Additional space cannot be allocated for the directory, however. Directory space is allocated for the entire library when the library is created. If the original allocation was not large enough, the IEHMOVE utility program can be used to expand the directory size. If the directory is filled, no additional members can be added to the library. Following is an example of a DD statement that might be used to create a library:

```
//DD1        DD DSNAME=FILELIB(FILE1),           X
//              DISP=(NEW,CATLG),                X
//              UNIT=2311,                       X
//              SPACE=(TRK,(40,10,3)),           X
//              VOLUME=SER=111111
```

This statement specifies that a library named FILELIB is to be created and cataloged in this job step. Its first member is named FILE1. Initial space allocated for data sets is to be 40 tracks, with additional allocation to be made, as necessary, in units of 10 tracks. In addition, space for three 256-byte records is to be allocated for the directory. The volume serial number is 111111.

A member of a partitioned data set can be replaced or deleted. The system actually accomplishes this by modifying or deleting the directory pointer to the member. The space occupied by the original member is not available for reuse until the MOVE or COPY control statement of the IEHMOVE utility program is used. The space previously occupied by the replaced or deleted member is thus made available. (For further details, see the publication IBM OS Utilities.)

KINDS OF LIBRARIES

A programmer can use libraries already provided by the system, or he can create libraries of his own. In addition, certain library names recognized by the system may be assigned to partitioned data sets provided by the system, by the programmer, or both. These libraries and their uses are discussed in the following paragraphs.

LIBRARIES PROVIDED BY THE SYSTEM

Link Library

The link library is a partitioned data set that contains load modules to be executed. Unless specified otherwise, a load module name in an EXEC statement is to be fetched from the link library. Operating system programs, such as the COBOL compiler, are usually contained in this library.

```
                 ┌─────────────┬─────────────┬─────────────┬─────────────┬──────────────┐
Directory        │ Entry for   │ Entry for   │ Entry for   │ Entry for   │              │
                 │ Member A    │ Member B    │ Member C    │ Member K    │   Note 1     │
                 ├─────────────┴─────────────┴─────────────┴─────────────┴──────┬───────┤
                 │                                                              │       │
                 │                              Member C                        │Note 2 │
                 ├───────────────────────────────────────────┬─────────────┬────┴───────┤
                 │                                           │             │            │
                 │                   Note 2                  │  Member B   │  Member K  │
                 ├───────────────────────────────────────────┴─────────────┴────────────┤
                 │                                                                      │
                 │                            Member K                                  │
Library          ├──────────────────────────────┬───────────────────────────────────────┤
Members          │                              │                                       │
                 │        Member K              │              Member A                 │
                 ├─────────────┬────────────────┴───────────────────────────────────────┤
                 │             │                                                        │
                 │ Member A    │                         Note 3                         │
                 ├─────────────┴────────────────────────────────────────────────────────┤
                 │                                                                      │
                 │                             Note 3                                   │
                 └──────────────────────────────────────────────────────────────────────┘
```

Notes:
1. Space available in directory.
2. Space available from deleted member after data set has been compressed.
3. Space available in library.

Figure 91. Format of a Library

The link library can be used by the programmer to store executable load modules at link-edit time. The technique for doing this is described in "Linkage Editor Data Set Requirements."

The link library is identified in a job control statement as SYS1.LINKLIB.

Procedure Library

The procedure library is a partitioned data set whose members are the cataloged procedures at an installation. They include the cataloged procedures provided by IBM. Procedures written at the installation can be added to the procedure library with the IEBUPDTE utility program (see "Using the Cataloged Procedures").

The system name for the procedure library is SYS1.PROCLIB.

Sort Library

The sort library is a partitioned data set that contains load modules from which the sort program is produced. It is identified by the name SYS1.SORTLIB (see "Using the Sort Feature").

COBOL Subroutine Library

The COBOL subroutine library is a partitioned data set that contains the COBOL library subroutines in load module form. These subroutines may be included in a COBOL load module or dynamically loaded to perform such functions as data conversion and double precision arithmetic. The COBOL programmer does not refer directly to these subroutines; calling sequences to them are generated at compile time from certain Procedure Division statements, and they are incorporated into the load module at link-edit time or loaded at program initialization time. A listing of subroutine names, functions, entry points, and size is given in Appendix B.

The system name for the COBOL subroutine library is SYS1.COBLIB.

LIBRARIES CREATED BY THE USER

A programmer can create members of the link library, the procedure library, and the job library. He can also create partitioned data sets for use in the copy library, the automatic call library, and the job library. In addition, he can create partitioned data sets to be used as libraries for additional input to the linkage editor, and he can create libraries whose members are source program entries.

Automatic Call Library

The automatic call library, defined by the SYSLIB DD statement in the link-edit job step, contains load modules or object modules that may be used as secondary input to the linkage editor. If the library contains object modules, it may also contain control statements. External symbols that are undefined after all primary input has been processed cause the automatic library call mechanism to search the automatic call library for modules that will resolve the references. The COBOL subroutine library must be specified for the automatic call library if any of the subroutines will be needed to resolve external references. Other partitioned data sets may be concatenated as shown in the following example:

```
//SYSLIB    DD DSNAME=SYS1.COBLIB,DISP=SHR
//          DD DSNAME=MYLIB,DISP=SHR
```

In this case, both the COBOL subroutine library and the partitioned data set named MYLIB are available to the automatic library call.

Note: If the partitioned data set named in the SYSLIB DD statement contains load modules, any data set concatenated with it must also be a load module partitioned data set. If the first contains object modules, the others must also contain object modules.

The linkage editor LIBRARY control statement has the effect of concatenating any specified member names with the automatic call library.

COBOL Copy Library

The COBOL copy library is a user-created library consisting of statements or entire COBOL programs frequently used by the programmer. The programmer can include these statements or programs into a program at compile time. He calls them with the COBOL COPY statement or BASIS card.

To enter or update source statements in the copy library, a utility program must be used. IEBUPDTE is the IBM-supplied utility program used to catalog procedures. A full discussion of the statements used in this program may be found in the publication IBM OS Utilities.

Entering Source Statements: Figure 92 illustrates the method to insert source statements into a copy library member.

The ./ ADD statement is a utility statement that copies CFILEA into the library called COPYLIB. CFILEA describes an FD entry. The NUMBER statement assigns a sequential numbering system to the statements in the library. The first statement is assigned number 10 and each

```
//CATALOG         JOB
//                EXEC   PGM=IEBUPDTE,PARM=(NEW)
//SYSUT2          DD     DSNAME=COPYLIB,UNIT=2311,              X
//                           DISP=(NEW,KEEP),                   X
//                           VOLUME=SER=111111,                 X
//                           SPACE=(TRK,(15,10,2)),             X
//                       DCB=(LRECL=80,BLKSIZE=80,RECFM=F)
//SYSPRINT        DD     SYSOUT=A
//SYSIN           DD     *
./                ADD    NAME=CFILEA,LEVEL=00,SOURCE=0,LIST=ALL
./                NUMBER NEW1=10,INCR=5
                    BLOCK CONTAINS 13 RECORDS
                    RECORD CONTAINS 120 CHARACTERS
                    LABEL RECORDS ARE STANDARD
                    DATA RECORD IS FILE-OUT.
./                ENDUP
/*
```

Figure 92. Entering Source Statements into the COPY Library

```
//UPDATE          JOB
//                EXEC    PGM=IEBUPDTE,PARM=(MOD)
//SYSUT1          DD      DSNAME=COPYLIB,UNIT=2311,               X
//                            DISP=(OLD,KEEP),                    X
//                            VOLUME=SER=111111,                  X
//                            DCB=(RECFM=F,BLKSIZE=80)
//SYSUT2          DD      DSNAME=COPYLIB,UNIT=2311,               X
//                            DISP=(OLD,KEEP),                    X
//                            VOLUME=SER=111111                   X
//SYSPRINT        DD      SYSOUT=A
//SYSIN           DD      *
./                CHANGE NAME=CFILEA,LEVEL=01,SOURCE=0,LIST=ALL
                    BLOCK CONTAINS 20 RECORDS                     00000010
./                ENDUP
/*
```

Figure 93. Updating Source Statements in a COPY Library

succeeding statement is incremented by 5. The entries following the utility statements are the actual source statements to be cataloged. The ENDUP statement signals the end of the entries to be inserted.

The same procedure can be used to catalog entire source programs.

__Updating Source Statements__: Figure 93 illustrates the method to update source statements in a copy library member inserted in the previous example.

SYSUT1 and SYSUT2 describe the data sets. Note that changes may be made on the same data set (identified on the DSNAME parameter). The utility statement CHANGE indicates that the new entry of CFILEA replaces the old entry. The sequence number of the altered statement must be supplied. This number, 00000010, is indicated in columns 73 through 80 of the replacement source statement. Note that, although in the insert example (see Figure 92 -- NUMBER statement) the number was coded as 10 without leading zeros, the program assigns an 8-character field to a sequence number and pads with leading zeros if necessary. When updating a sequence number in a library, these leading zeros must be included.

At compile time, COPYLIB is identified on a SYSLIB DD statement, as follows:

```
//SYSLIB  DD   DSNAME=COPYLIB,              X
//                 VOLUME=SER=111111,        X
//                 DISP=SHR,UNIT=2311
```

__Retrieving Source Statements__: Members of the cataloged library can be retrieved using the COPY statement or BASIS card.

__COPY Statement__

The COPY statement permits the programmer to include cataloged source statements in the Data or Environment Divisions. If the programmer wishes to retrieve the member, CFILEA, cataloged in the previous examples, he writes the statement:

```
FD FILEA COPY CFILEA
```

The compiler translates this instruction to read:

```
FD FILEA BLOCK CONTAINS 20 RECORDS
   RECORD CONTAINS 120 CHARACTERS
   LABEL RECORDS ARE STANDARD
   DATA RECORD IS FILE-OUT.
```

Note that CFILEA itself does not appear in the statement. CFILEA is a name identifying the entries. It acts as a header record but is not itself retrieved. The compiler source listing, however, will print out the COPY statement as the programmer wrote it.

The COPY statement also permits the programmer to include previously cataloged source statements into the Procedure Division.

Assume a procedure named DOWORK was cataloged with the following statements:

```
./  ADD            NAME=DOWORK,LEVEL=00,
                     SOURCE=0,LIST=ALL
./  NUMBER         SEQ1=400,INCR=10
       COMPUTE QTY-ON-HAND =
         TOTAL-USED-NUMBER-ON-HAND.
       MOVE-QTY-ON-HAND TO PRINT-AREA.
./  ENDUP
```

To retrieve the cataloged member, DOWORK, the programmer writes:

```
    paragraph-name.  COPY 'DOWORK'.
```

The statements included in the DOWORK procedure will immediately follow the paragraph-name, replacing the words COPY DOWORK.

BASIS Card

Frequently used source programs, such as a payroll program, can be inserted into the copy library. The BASIS card brings in an entire source program at compile time. Calling in a program eliminates the need for the programmer to handle a program each time he wants to compile it. The programmer may, however, alter any statement in the source program by referring to its COBOL sequence number with an INSERT or DELETE statement. The INSERT statement will add new source statements after the sequence number indicated. The DELETE statement will eliminate the statements indicated by the sequence numbers. The programmer may delete a single statement with one sequence number, or he may delete more than one statement, separating by a hyphen the first and last sequence numbers to be deleted.

Note: The COBOL sequence number is the 6-digit number that the programmer assigns in columns 1 through 6 of the source cards. This sequence number has nothing to do with the sequence numbers assigned in simulated columns 73 through 80 by the IEBUPDTE utility program. The sequence numbers assigned by IEBUPDTE are used to update source statements in the copy library. Changes made using these numbers are intended to be permanent changes. The COBOL sequence numbers are used to update COBOL source statements at compile time. Such changes are in effect for the one run only.

Assume that a company payroll program is kept as a source program in the copy library. The name of the program is PAYROLL. During a particular year, old age tax is taken out at a rate of two and a half percent each week for all personnel until earnings exceed $6600. The coding to accomplish this is shown in Figure 94.

Now, however, due to a change in the old age tax laws, tax is to be taken out until earnings exceed $7800 and a new percentage is to be placed. The programmer can code these changes as shown in Figure 95.

```
COBOL                                                                   IEBUPDTE
Sequence                                                                Sequence
Numbers                                                                 Numbers
000730              IF ANNUAL-PAY GREATER THAN 6600 GO TO PAY-WRITE.    00000105
000735              IF ANNUAL-PAY GREATER THAN 6600 - BASE-PAY GO TO LAST-TAX.  00000110
000740 TAX-PAYR.    COMPUTE TAX-PAY = BASE-PAY * .025                   00000115
000750              MOVE TAX-PAY TO OUTPUT-TAX.                         00000120
000760 PAY-WRITE.   MOVE BASE-PAY TO OUTPUT-BASE.                       00000125
000770              ADD BASE-PAY TO ANNUAL-PAY.                         00000130
   .                .                                                      .
   .                .                                                      .
   .                .                                                      .
000850              STOP RUN.                                           00000240
```

Figure 94. COBOL Statements to Deduct Old Age Tax

```
   .
   .
   .
BASIS PAYROLL
DELETE 000730-000740
000730                IF ANNUAL-PAY GREATER THAN 7800 GO TO PAY-WRITE.
000735                IF ANNUAL-PAY GREATER THAN 7800 - BASE-PAY GO TO LAST-TAX.
000740 TAX-PAYR.   COMPUTE TAX-PAY = BASE-PAY * .044.
```

Figure 95. Programmer Changes to Source Program

```
000730              IF ANNUAL-PAY GREATER THAN 7800 GO TO PAY-WRITE.
000735              IF ANNUAL-PAY GREATER THAN 7800 - BASE-PAY GO TO LAST-TAX.
000740 TAX-PAYR.    COMPUTE TAX-PAY = BASE-PAY * .044.
000750              MOVE TAX-PAY TO OUTPUT-TAX.
000760 PAY-WRITE.   MOVE BASE-PAY TO OUTPUT-BASE.
000770              ADD BASE-PAY TO ANNUAL-PAY.
   .                .
   .                .
   .                .
000850              STOP RUN.
```

Figure 96. Changed COBOL Statements to Source COPY Library Statements

The altered program will contain the coding shown in Figure 96.

Note that changes made through use of the INSERT and DELETE statements remain in effect for the one run only.

Note: If both the COPY statement and the BASIS card are used, the library containing the member specified in the BASIS card must be defined first. The COPY libraries concatenated with the BASIS library may be defined and referenced in any order (see "Appendix I: Checklist for Job Control Procedures"). For a discussion of special considerations when using BASIS with the BATCH option, see "Batch Compilation."

JOB Library

The job library consists of one or more partitioned data sets that contain load modules to be executed. It is specified by the JOBLIB DD statement that must precede the EXEC statement of the first step of a job. Partitioned data sets assigned to the job library are concatenated with the link library so that any load module is obtained automatically when its name appears in the PGM= parameter of the EXEC statement. The following statements illustrate how three partitioned data sets can be assigned to the job library:

```
//MYJOB    JOB ...
//JOBLIB   DD  DSNAME=MYLIB1,DISP=(OLD,PASS)
//         DD  DSNAME=MYLIB2,DISP=(OLD,PASS)
//         DD  DSNAME=MYLIB3,DISP=(OLD,PASS)
//STEP1    EXEC ...
            .
            .
            .
//STEP2    EXEC ...
            .
            .
            .
```

These statements specify that the job library containing the data sets MYLIB1, MYLIB2, and MYLIB3 is to be concatenated with the link library. When a load module is named in an EXEC statement in any step of the job, the directories of the job library will be searched for the name. When a job library is specified for a job, the link library is searched for a named load module only when the module is not found in the job library.

Partitioned data sets used in the job library can be created by specifying the partitioned data set name and the member name in the SYSLMOD DD statement when each member is processed by the linkage editor.

Additional Input to the Linkage Editor: Libraries of object modules (with or without linkage editor control statements) and libraries of load modules can be used as additional input to the linkage editor. Members are specified by use of the INCLUDE and LIBRARY linkage editor control statements.

A library of object modules and control statements can be created by use of the IEBUPDTE utility program.

A library of load modules can be created by use of the SYSLMOD DD statement in the linkage editor job step, as discussed in "Job Library."

SHARING COBOL LIBRARY SUBROUTINES

Use of the COBOL Library Management Feature makes it possible for all programs in the same or different regions/partitions to share one copy of the COBOL library subroutines. That is, the most economical use of main storage is made when the most frequently used COBOL library subroutines are placed in the MVT link pack area (LPA), or the MFT resident reusable routine (RRR) area, rather than in each region/partition. To make the most effective use of the Library Management Feature, and to use the IBM cataloged procedures whether or not Library Management is needed, the user should concatenate the COBOL subroutine library with the system link library.

The user may request the COBOL Library Management Feature at compile time, via the RESIDENT option (see the section "Options for the Compiler" in the chapter entitled "Job Control Procedures").

CONCATENATING THE SUBROUTINE LIBRARY

To concatenate the subroutine library with the link library, the user executes the IEBUPDTE utility program to add a member named LNKST00 to SYS1.PARMLIB, specifying the library desired (that is, either the entire COBOL subroutine library or a private library containing user-selected COBOL library subroutines). Note that the library containing the subroutines must be cataloged.

An installation that is planning to use the Library Management Feature will find it convenient to include frequently used COBOL library subroutines in the MVT LPA or the MFT RRR area. Infrequently used subroutines are then brought into the region/partition as required. To add COBOL subroutines to either of these areas, the user invokes the IEBUPDTE utility program to add a member named IEAIGGXX (see Note 2 in Figure 97) to SYS1.PARMLIB, specifying all names and aliases for the COBOL library subroutines to be included. Then, at an initial program load (IPL) time, the operator identifies the link list to the system, which subsequently places the identified COBOL subroutines in main storage in the LPA/RRR area.

Figure 97 illustrates how an installation can accomplish both these functions in one operation. The encircled letters in the figure refer to the JCL suggested A to concatenate the COBOL subroutine library (SYS1.COBLIB) with the system link library (SYS1.LINKLIB), and then B to place the user list of desired COBOL library subroutines and their aliases to the LPA/RRR. (For further information, see the publication OS Full American National Standard COBOL Compiler and Library, Version 4 Installation Reference Material.

Notes:

1. If the user does not wish to place any COBOL subroutines in the RRR/LPA area, he need not execute the portion of the IEBUPDTE utility program that adds IEAIGGXX to SYS1.PARMLIB shown above. He may still make use of the Library Management Feature. However, all required library subroutines will be loaded into his own region/partition when they are needed by one or more programs, and deleted when they are no longer needed. Thus, not all library subroutines needed by all programs in the region need be resident at the same time. In this case, however, the user must supply a job control card at execution time pointing to the COBOL subroutine library or to his own private library of COBOL subroutines. (For a discussion of the various COBOL library subroutines available to the programmer, see "Appendix B: COBOL Library Subroutines.")

```
//CATLG     JOB   user information
//          EXEC  PGM=IEBUPDTE,PARM=MOD
//SYSPRINT  DD    SYSOUT=A
//SYSUT1    DD    DSN=SYS1.PARMLIB,DISP=SHR
//SYSUT2    DD    DSN=SYS1.PARMLIB,DISP=SHR
//SYSIN     DD    *
      ./    REPL  NAME=LNKLST00,LIST=ALL
                       SYS1.LINKLIB,SYS1.COBLIB
      ./    ADD   NAME=IEAIGG01,LIST=ALL
                       SYS1962(562B1,NAME1,ALIAS1,...
/*
```

Notes

1. The name used in the JCL must identify the data set to be concatenated with the system link library, and is selected by the installation. (Note that this data set must be cataloged.)

2. The last two digits of this member-name can vary, but the digits specified here must also be specified in the RAM= parameter used at IPL time. For example, if IEAIGG02 were specified, 'RAM=02' would be required at IPL time.

3. The names and aliases of the COBOL library subroutine members to be made resident must be specified by the installation. The system searches the last name first; in this case, ALIAS1 is searched last. The user should, therefore, specify the most frequently used name last.

Figure 97. Concatenating the Subroutine Library

2. If one or more programs in a given region/partition request the COBOL Library Management Feature, then the main program and all subprograms in that region/partition must use it. Otherwise, the multiple copies of COBOL library subroutines resident at one time may cause unpredictable results.

CREATING AND CHANGING LIBRARIES

A programmer can create or change a partitioned data set in one of three ways: (1) through the use of DD statements, (2) through the use of utility programs, and (3) through the use of certain linkage editor control statements.

The DD statement can be used to create libraries as is discussed at the beginning of this chapter. In addition, DD statements can be used to add members to existing libraries, including the link library, and to retrieve members of existing libraries.

Utility programs can be used to create libraries such as those used in the copy library or as secondary input to the linkage editor. In addition, utility programs can be used to move, copy, and replace members of an existing library; to add, delete, and renumber the records within an existing library; and to assign sequence numbers to the records of a new library.

Linkage editor control statements can be used to make changes to members of a library of load modules. The name of a member can be changed or additional names can be specified. Additional entry points can be identified, existing entry points can be deleted, and portions of a load module can be deleted or replaced. For further information, see the publication IBM OS Linkage Editor and Loader.

A cataloged procedure is a set of job control statements placed in a partitioned data set called the procedure library (SYS1.PROCLIB). It can be retrieved from the library by using its member name in an EXEC statement of a job step in the input stream. Frequently used procedures, such as those used for compiling and linkage editing, can be cataloged to simplify their subsequent use.

A cataloged procedure can contain statements for the processing of an entire job, or it can contain statements to process one or more steps of a job, with the remaining steps defined by job control statements in the input stream. A job can use several cataloged procedures, each processing one or more of the job steps. A job can also call for execution of the same cataloged procedure in more than one job step.

This chapter describes the following:

- How to call cataloged procedures
- The types of cataloged procedures, including those supplied by IBM for use with COBOL source programs
- How to add procedures to the procedure library
- How to modify existing procedures for the current job step only
- How to override and add to cataloged procedures
- How to use the DDNAME parameter in cataloged procedures

CALLING CATALOGED PROCEDURES

A cataloged procedure is called by a job that appears in the input stream. The job must consist of a JOB statement and an EXEC statement that specifies the cataloged procedure name in the positional parameter (either procname or PROC=procname). For example:

```
//STEPQ EXEC COBUC
//STEPQ EXEC PROC=COBUC
```

Either of these EXEC statements could be used to call the IBM-supplied cataloged procedure COBUC to process the job step STEPQ.

A job step that calls for execution of a cataloged procedure can also contain DD statements that are applicable to the job steps of the cataloged procedure. A job that calls for execution of a cataloged procedure may, in other steps, call for execution of other cataloged procedures, call for other executions of the same cataloged procedure, or call directly for execution of load modules. The following example shows a job control procedure that calls both cataloged procedures and load modules.

```
//JOB1          JOB
//STEPA         EXEC    COBUC
//COB.SYSIN     DD      *

           (source module)

/*
//STEPL         EXEC    PGM=IEWL
                 .
                 .
                 .
     (DD statements for the linkage editor)
                 .
                 .
                 .
//STEPE         EXEC    PGM=*.STEPL.SYSLMOD
                 .
                 .
                 .
     (DD statements for user-defined files)
                 .
                 .
                 .
```

The IBM-supplied cataloged procedure COBUC for compilation is used to process STEPA. The COB.SYSIN DD statement is required to define the input to the compiler. The remaining statements in the procedure refer to execution of the linkage editor and the subsequent load module.

Data Sets Produced by Cataloged Procedures

Data sets produced during execution of a cataloged procedure can be used in subsequent job steps. They can also be called as follows:

```
//jobname     JOB  1234,J.SMITH
//STEPA       EXEC PROCED
//PROC1.SYSIN DD   *

          (source module)

/*
//stepname    EXEC PGM=*.STEPA.PROC2.SYSLMOD
                   .
                   .
                   .

      (DD statements for user-defined files)
                   .
                   .
                   .
```

The cataloged procedure PROCED is composed of two job steps, PROC1 and PROC2, that compile and linkage edit the source module.

TYPES OF CATALOGED PROCEDURES

The programmer can write his own procedures and catalog them, or he can use the five COBOL cataloged procedures provided by IBM.

PROGRAMMER-WRITTEN CATALOGED PROCEDURES

The programmer can write cataloged procedures, consisting of EXEC and DD statements, which incorporate job control procedures he uses frequently. For example, the programmer may wish to catalog an EXEC statement and the associated DD statements for a job step that specifies execution of a program. In this way, the DD statements need not be specified each time the program is executed.

In writing a procedure for cataloging, the programmer must follow these rules:

- Another cataloged procedure cannot be referred to, i.e., only the PGM=progname form in an EXEC statement can be used.

  Note, however, that a cataloged procedure may contain a DD statement that refers to a cataloged data set.

- SYSABEND or SYSUDUMP DD statements should not be cataloged because they cannot be overridden.

- The following statements cannot be used in a cataloged procedure:

  1. The JOB statement
  2. A DD statement with JOBLIB in the name field
  3. A DD statement with an * in the operand field
  4. A DD statement with DATA in the operand field
  5. The delimiter statement

Testing Programmer-Written Procedures

A procedure can be tested before it is placed in the procedure library by converting it into an in-stream procedure and executing it any number of times during a job. For further information about in-stream procedures, refer to the section "Testing a Procedure as an In-Stream Procedure".

Adding Procedures to the Procedure Library

The IEBUPDTE utility program is used to add procedures to the procedure library. A description of the use of this program is given in the publication IBM OS Utilities.

In Figure 98, two procedures are added to the procedure library (SYS1.PROCLIB). All control statements are in the input stream.

The first procedure is for a COBOL compilation. Mass storage volumes are specified for the four utility data sets, and 100 tracks are allocated for each utility data set. This cataloged procedure is named COBDA.

The second procedure is also for a COBOL compilation. Unlabeled tape volumes are specified for three utility data sets; for the fourth, SYSUT1, a mass storage device must be specified. This cataloged procedure is named COBTP.

Job control statements: the EXEC card specifies that the IEBUPDTE program is to be executed, and PARM=NEW is used because all data is read from one source, i.e., the input stream.

Utility statements: the ADD statement specifies the member name of the procedure, the level modification (00, first run) and the source of the modification (0, user-supplied). The NUMBER statement specifies the sequence numbers for records in the member. The first record of the cataloged procedure is numbered 00000010, and subsequent records are incremented by tens.

Note that leading zeros in the NUMBER statement are not necessary, as indicated in the example for the COBTP procedure.

IBM-SUPPLIED CATALOGED PROCEDURES

IBM distributes cataloged procedures with the operating system, which can be incorporated when the system is generated.

Five of the procedures are for use with COBOL programs.

1. COBUC provides for compilation.
2. COBUCL provides compilation and linkage editing.
3. COBULG provides linkage editing and execution.
4. COBUCLG provides for compilation, linkage editing, and execution.
5. COBUCG provides for compilation and loading.

These procedures may be used with any of the job schedulers released as part of the IBM Operating System. When parameters required by a particular scheduler are encountered by another scheduler that does not require those parameters, either they are ignored or alternative parameters are substituted automatically.

The five cataloged procedures are shown in Figures 99, 100, 101, 102, and 103. (Space allocations in these procedures are in terms of record lengths on the 2311 disk storage device.) Note that when DSNAME=&& is used in a DD statement the specified data set is given a unique name by the operating system, and it is assumed to be a temporary data set that will be deleted when the job is completed. If the data set is to be kept, the DD statement can be overridden with a permanent data set name, and the appropriate parameters can be specified.

```
Job           //ADPROC    JOB      1234,J.DUBOB
Control       //STEP1     EXEC     PGM=IEBUPDTE,PARM=NEW
Language      //SYSPRINT  DD       SYSOUT=A
Statements    //SYSUT2    DD       DSNAME=SYS1.PROCLIB,DISP=OLD
              //SYSIN     DD       DATA

Utility       ./          ADD      NAME=COBDA,LEVEL=00,SOURCE=0
Statements    ./          NUMBER   NEW1=00000010,INCR=00000010

              //COB       EXEC     PGM=IKFCBL00
              //SYSUT1    DD       UNIT=SYSDA,SPACE=(TRK,(100,10))
              //SYSUT2    DD       UNIT=SYSDA,SPACE=(TRK,(100,10))
First         //SYSUT3    DD       UNIT=SYSDA,SPACE=(TRK,(100,10))
Procedure     //SYSUT4    DD       UNIT=SYSDA,SPACE=(TRK,(100,10))
              //SYSPRINT  DD       SYSOUT=A
              //SYSPUNCH  DD       SYSOUT=B

Utility       ./          ADD      NAME=COBTP,LEVEL=00,SOURCE=0
Statements    ./          NUMBER   NEW1=10,INCR=10

              //COB       EXEC     PGM=IKFCBL00
              //SYSUT1    DD       UNIT=SYSDA,SPACE=(TRK,(100,10))
Second        //SYSUT2    DD       UNIT=2400,LABEL=(,NL)
Procedure     //SYSUT3    DD       UNIT=2400,LABEL=(,NL)
              //SYSUT4    DD       UNIT=2400,LABEL=(,NL)
              //SYSPRINT  DD       SYSOUT=A
              //SYSPUNCH  DD       SYSOUT=B

Delimiter     ./          ENDUP
Statements    /*
```

Figure 98. Example of Adding Procedures to the Procedure Library

Note: If the compiler options are not explicitly supplied with the procedure, default options established at the installation apply. The programmer can override these default options by using an EXEC statement that includes the desired options (see "Overriding and Adding to EXEC Statements" and "Overriding Cataloged Procedures Using Symbolic Parameters").

Procedure Naming Conventions

Procedure names begin with the abbreviated name of the processor program, which, in the case of the COBOL procedures, is COB.

The processor's abbreviated name is followed by the processor's level indicator (U) and then by C (compile), L (linkage edit), G (go -- i.e., execute), or combinations of them. Hence, procedure COBUC is a single-step procedure that compiles a program using the COBOL processor; COBUCLG is a 3-step procedure wherein the first step compiles a program using COBOL, the second step link-edits the output of the first step, and the third step executes the output of the linkage editor.

Step Names in Procedures

In a cataloged procedure, the step name is the same as the abbreviated processor name (LKED). The step that executes a compiled and link-edited program is named GO.

For example, in the procedure named COBUCLG, the first step is named COB, the second step is named LKED, and the third step is named GO.

Unit Names in Procedures

The two unit names used in IBM-supplied cataloged procedures are as follows:

SYSSQ any magnetic tape or mass storage device

SYSDA any mass storage device

A pool of units must be assigned to these unit names during the system generation procedure. For example, only 2311 Disk Storage Drives might be assigned to the SYSSQ name. Then again, both 2400 Magnetic Tape Units and 2311 Disk Storage Drives might be assigned to the SYSSQ name Once a pool of devices is assigned to these classes, device selection is done by the Job Scheduler.

Data Set Names in Procedures

When DSNAME=&&name is used in a DD statement, the specified data set is given a unique name by the scheduler, and it is assumed to be a temporary data set that will be deleted when the job terminates. If the data set is to be retained, the DD statement must be overridden with a permanent data set name and appropriate DISP parameters.

COBUC Procedure

The COBUC procedure is a single-step procedure to execute the COBOL compiler. It produces a punched object deck. Figure 99 shows the statements that make up the COBUC cataloged procedure.

The following DD statement must be supplied in the input stream:

```
//COB.SYSIN DD *    (or appropriate
                    parameters defining an
                    input data set)
```

If the DD * statement is used under MFT, the delimiter statement (/*) must follow the source module. Under MVT, the /* statement is not required.

COBUCL Procedure

The COBUCL procedure is a two-step procedure to compile and link-edit using the COBOL compiler. Figure 100 shows the statements that make up the cataloged procedure.

The COB job step produces an object module that is input to the linkage editor. Other object modules may be added as illustrated in Example 5 under "Using the DDNAME Parameter."

The following DD statement, indicating the location of the source module, must be supplied in the input stream:

```
//COB.SYSIN DD *    (or appropriate
                    parameters)
```

COBULG Procedure

The COBULG cataloged procedure is a two-step procedure to link-edit and execute the output of a COBOL compilation. Figure 101 shows the statements that make up the procedure.

The following DD statement indicating the location of the object module must be supplied in the input stream:

```
//LKED.SYSIN DD *  (or appropriate
                   parameters)
```

If the COBOL program refers to SYSIN in the execution step, the following DD statement must also be supplied and must be the last of the //GO, cards.

```
//GO.SYSIN  DD *   (or appropriate
                   parameters)
```

If the COBOL program refers to other data sets in the execution step such as user-defined files, DD statements that define these data sets must also be provided.

```
//COB        EXEC PGM=IKFCBL00,PARM='DECK,NOLOAD,SUPMAP',REGION=86K
//SYSPRINT   DD SYSOUT=A
//SYSPUNCH   DD SYSOUT=B
//SYSUT1     DD DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT2     DD DSNAME=&&SYSUT2,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT3     DD DSNAME=&&SYSUT3,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT4     DD DSNAME=&&SYSUT4,UNIT=SYSDA,SPACE=(460,(700,100))
```

Figure 99. Statements in the COBUC Procedure

```
//COB        EXEC PGM=IKFCBL00,REGION=86K
//SYSPRINT   DD SYSOUT=A
//SYSUT1     DD DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT2     DD DSNAME=&&SYSUT2,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT3     DD DSNAME=&&SYSUT3,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT4     DD DSNAME=&&SYSUT4,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSLIN     DD DSNAME=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,           X
//              SPACE=(80,(500,100))
//LKED       EXEC PGM=IEWL,PARM='LIST,XREF,LET',COND=(5,LT,COB),       X
//              REGION=96K
//SYSLIN     DD DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//           DD DDNAME=SYSIN
//SYSLMOD    DD DSNAME=&&GOSET,DISP=(NEW,PASS),UNIT=SYSDA,             X
//              SPACE=(1024,(50,20,1))
//SYSLIB     DD DSNAME=SYS1.COBLIB,DISP=SHR
//SYSUT1     DD UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),                     X
//              SPACE=(1024,(50,20))
//SYSPRINT   DD SYSOUT=A
```

Figure 100. Statements in the COBUCL Procedure

```
//LKED        EXEC PGM=IEWL,PARM='LIST,XREF,LET',REGION=96K
//SYSLIN      DD DDNAME=SYSIN
//SYSLMOD     DD DSNAME=&&GOSET(GO),DISP=(NEW,PASS),UNIT=SYSDA,            X
//                 SPACE=(1024,(50,20,1))
//SYSLIB      DD DSNAME=SYS1.COBLIB,DISP=SHR
//SYSUT1      DD DSNAME=&&SYSUT1,UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),        X
//                 SPACE=(1024,(50,20))
//SYSPRINT    DD SYSOUT=A
//GO          EXEC PGM=*.LKED.SYSLMOD,COND=(5,LT,LKED)
```

Figure 101. Statements in the COBULG Procedure

```
//COB         EXEC PGM=IKFCBL00,PARM=SUPMAP,REGION=86K
//SYSPRINT    DD SYSOUT=A
//SYSUT1      DD DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT2      DD DSNAME=&&SYSUT2,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT3      DD DSNAME=&&SYSUT3,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT4      DD DSNAME=&&SYSUT4,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSLIN      DD DSNAME=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,              X
//                 SPACE=(80,(500,100))
//LKED        EXEC PGM=IEWL,PARM='LIST,XREF,LET',COND=(5,LT,COB),          X
//                 REGION=96K
//SYSLIN      DD DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//            DD DDNAME=SYSIN
//SYSLMOD     DD DSNAME=&&GOSET(GO),DISP=(NEW,PASS),UNIT=SYSDA,            X
//                 SPACE=(1024,(50,20,1))
//SYSLIB      DD DSNAME=SYS1.COBLIB,DISP=SHR
//SYSUT1      DD UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),                        X
//                 SPACE=(1024,(50,20))
//SYSPRINT    DD SYSOUT=A
//GO          EXEC PGM=*.LKED.SYSLMOD,COND=((5,LT,COB),(5,LT,LKED))
```

Figure 102. Statements in the COBUCLG Procedure

```
//COB         EXEC PGM=IKFCBL00,PARM='LOAD',REGION=86K
//SYSPRINT    DD SYSOUT=A
//SYSUT1      DD DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT2      DD DSNAME=&&SYSUT2,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT3      DD DSNAME=&&SYSUT3,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT4      DD DSNAME=&&SYSUT4,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSLIN      DD DSNAME=&&LOADSET,DISP=(MOD,PASS),                         X
//               UNIT=SYSDA,SPACE=(80,(500,100))
//GO          EXEC PGM=LOADER,PARM='MAP,LET',COND=(5,LT,COB),REGION=106K
//SYSLIN      DD DSNAME=*.COB.SYSLIN,DISP=(OLD,DELETE)
//SYSLOUT     DD SYSOUT=A
//SYSLIB      DD DSNAME=SYS1.COBLIB,DISP=SHR
```

Figure 103. Statements in the COBUCG Procedure

COBUCLG Procedure

The COBUCLG procedure is a three-step procedure to compile, link-edit, and execute using the COBOL compiler. Figure 102 shows the statements that make up the procedure.

The COB job step produces an object module that is input to the linkage editor. Other object modules may be added as illustrated in Example 5 under "Using the DDNAME Parameter."

The following DD statement, indicating the location of the source module, must be supplied in the input stream:

```
//COB.SYSIN DD      * (or appropriate
                       parameters)
```

If the COBOL program refers to SYSIN, the following DD statement indicating the location of the input data set must also be supplied:

```
//GO.SYSIN  DD      * (or appropriate
                       parameters)
```

If the COBOL program refers to other data sets, DD statements that define these data sets must also be supplied.

COBUCG Procedure

The COBUCG procedure is a two-step procedure to compile, load, and execute using the COBOL compiler and OS loader. Figure 103 shows the statements that make up the procedure.

The COB job step produces an object module that is input to the loader.

The following DD statement, indicating the location of the source module, must be supplied in the input stream:

```
//COB.SYSIN DD      * (or appropriate
                       parameters)
```

If the COBOL program refers to SYSIN, the following DD statement indicating the location of the input data set must also be supplied:

```
//GO.SYSIN  DD      * (or appropriate
                       parameters)
```

If the COBOL program refers to other data sets, the DD statements that define these data sets must also be supplied.

MODIFYING EXISTING CATALOGED PROCEDURES

Existing cataloged procedures can be permanently modified by using the IEBUPDTE utility program described in the publication IBM OS Utilities.

OVERRIDING AND ADDING TO CATALOGED PROCEDURES

Any parameter in a cataloged procedure except the PGM=progname parameter in the EXEC statement can be overridden. Parameters or statements not specified in the procedure can also be added. When a cataloged procedure is overridden or added to, the changes apply only during one execution.

OVERRIDING AND ADDING TO EXEC STATEMENTS

An EXEC statement can be overridden or added to in one of two ways:

1. Specify, in the operand field of the EXEC statement calling the procedure, the keyword, the procedure step-name and the subparameters, for example:

   COND.procstep=(subparameters)

   If a multistep procedure is being modified, parameters in the calling EXEC statement must be specified step by step; i.e., the parameters for one step must be specified before those of the next step. If the return code of a cataloged procedure step is to be tested, the name of the step in the procedure (procstep) must be qualified by the name of the step that called for execution of the cataloged procedure (stepname).

2. Specify in the operand field of the EXEC statement calling the procedure only the keyword parameters and subparameters, for example:

   COND=(subparameters)

   If a multistep procedure is being called, the specified parameters (with the exception of PARM) apply to all steps in the procedure. The PARM keyword subparameters override the first EXEC statement and nullify any subsequent PARM keyword subparameters. The COND and ACCT parameters apply to all steps in the procedure. To override PARM parameters in job steps other than the first, the previous method can be used.

Note: A parameter in an EXEC statement cannot be partly overridden; it must be overridden in its entirety. Any parameter not overridden remains as originally defined.

Examples of Overriding and Adding to EXEC Statements

This section contains examples of overriding and adding to the EXEC statement. The procedures overridden or added to are the IBM procedures shown in Figures 99, 100, 101, 102 and 103.

Example 1: The following example shows the overriding of one parameter in the EXEC statement of the one procedure step in the IBM-supplied COBUC procedure. The statements appear in the input stream as follows:

```
//jobname    JOB  1234,J.SMITH
//STEPA      EXEC COBUC,PARM.COB='DECK,       X
//                NOLOAD,BUF=4000,            X
//                SIZE=9600'
//COB.SYSIN DD    *

          (source module)

/*
```

Note: In actual use the PARM.COB parameter cannot be continued in this manner. In the PARM parameter that is overridden, the DECK and NOLOAD options were specified. They are included again since the parameter must be overridden in its entirety. The information is here enclosed in single quotation marks, since subparameters that contain equal signs must be enclosed in this manner.

Example 2: The following example shows the overriding of two parameters and the adding of another in the EXEC statement of one procedure step of the IBM-supplied COBUCLG procedure. The statements appear in the input stream as shown:

```
//jobname     JOB  1234,J.SMITH
//STEPA       EXEC COBUCLG,COND.LKED=          X
//                 (9,LT,STEPA.COB),           X
//                 PARM.LKED=(MAP,LIST),       X
//                 ACCT=(1234)
//COB.SYSIN   DD   *

          (source module)

/*
```

Note: In actual use the COND.LKED and PARM.LKED parameters cannot be continued in this manner. For the linkage editor job step in the above example, the COND and PARM parameters have been overridden and the ACCT parameter added.

Example 3: The following example shows the overriding of individual parameters in more than one procedure step of the IBM-supplied COBUCLG procedure. The statements appear in the input stream as shown.

```
//jobname     JOB  1234,J.SMITH
//stepname    EXEC COBUCLG,PARM.LKED=OVLY,     X
//                 COND.GO=((5,EQ,             X
//                 stepname.COB),              X
//                 (5,EQ,stepname.LKED))
//COB.SYSIN   DD   *

          (source module)

/*
```

Note: In actual use the COND.GO statement cannot be continued in this manner. The PARM option OVLY replaces the PARM subparameters of the link-edit job step. The COND option EQ (equal to) replaces the option LT (less than) in the execution job step.

Note that all overriding parameters for one step of the procedure must be specified before those for the next step.

Example 4: The following example shows the overriding of parameters on all EXEC statements in the IBM-supplied COBUCLG procedure. The statements appear in the input stream as shown:

```
//jobname  JOB     1234,J.SMITH
//stepname EXEC    COBUCLG,                    X
//                 PARM=(LOAD,PMAP),           X
//                 COND=(3,LT),                X
//                 ACCT=(123456,DEPTQ)
//COB.SYSIN   DD    *

          (source module)

/*
```

The PARM options are added to the procedure step COB and nullify the PARM options in the LKED and GO steps. The COND and ACCT parameters apply to all steps in the procedure.

TESTING A PROCEDURE AS AN IN-STREAM PROCEDURE

A procedure can be tested before it is placed in the procedure library by converting it into an in-stream procedure and executing it any number of times during a job. In-stream procedures are described in detail in the publication IBM OS Job Control Language Reference.

An in-stream procedure is a series of job control language statements enclosed within a PROC statement and a PEND statement. The following example shows how to convert the COBUC procedure (Figure 99) into an in-stream procedure and execute it twice:

```
//CONVERT    JOB  1234,YOURNAME
//INSTREAM   PROC
//COB        EXEC PGM=IKFCBL00,PARM='DECK,  X
                  NOLOAD,SUPMAP',REGION=86K
//SYSPRINT   DD   SYSOUT=A
//SYSPUNCH   DD   SYSOUT=B
//SYSUT1     DD   DSNAME=&&SYSUT1,          X
                  UNIT=SYSDA,               X
                  SPACE=(460,(700,100))
//SYSUT2     DD   DSNAME=&&SYSUT2,          X
                  UNIT=SYSDA,               X
                  SPACE=(460,(700,100))
//SYSUT3     DD   DSNAME=&&SYSUT3,          X
                  UNIT=SYSDA,               X
                  SPACE=(460,(700,100))
//SYSUT4     DD   DSNAME=&&SYSUT4,          X
                  UNIT=SYSDA,               X
                  SPACE=(460,(700,100))
//ENDPROC    PEND
//           EXEC INSTREAM
//COB.SYSIN DD    *

                  (input data)

/*
//           EXEC INSTREAM
//COB.SYSIN DD    *

                  (input data)

/*
```

OVERRIDING AND ADDING TO DD STATEMENTS

A DD statement can be overridden or added to by using a DD statement whose name is composed of the procedure step-name that qualifies the ddname of the DD statement being overridden, as follows:

```
//procstep.ddname  DD  (appropriate
                        parameters)
```

Entire DD statements can also be added.

There are rules that must be followed when overriding or adding a DD statement within a step in a procedure.

- Overriding DD statements must be in the same order in the input stream as they are in the cataloged procedure.
- DD statements to be added must follow overriding DD statements.
- A DD statement with an * in the operand field terminates processing of subsequent DD statements in both the procedure and the input stream for the job step, but not necessarily for the job.

There are some special cases that should be kept in mind when overriding a DD statement.

- All parameters are overridden in their entirety, except for the DCB parameter. Within the DCB parameter, individual subparameters may be overridden.
- To nullify a keyword parameter (except the DCB parameter), write, in the overriding DD statement, the keyword and an equal sign followed by a comma. For example, to nullify the use of the UNIT parameter, specify UNIT=, in the overriding DD statement.
- A parameter can be nullified by specifying a mutually exclusive parameter. For example, the SPACE parameter can be nullified by specifying the SPLIT parameter in the overriding DD statement.
- The DUMMY parameter can be nullified by omitting it and specifying the DSNAME parameter in the overriding DD statement.
- To override DD statements in a concatenation of data sets, the programmer must provide one DD statement for each data set in the

concatenation. Only the first DD statement in the concatenation should be named. However, if a DD statement to be changed follows one (or more) DD statement(s) to be left intact, the first overriding statement(s) should have a blank operand.

- If the DDNAME=ddname parameter is specified in a cataloged procedure, it cannot be overridden; rather it can refer to a DD statement supplied at the time of execution.

Examples of Overriding and Adding to DD Statements

This section contains examples of overriding and adding to parameters in DD statements. The procedures overridden or added to are the IBM procedures shown in Figures 99, 100, 101, 102 and 103.

The DDNAME parameter is not used in these examples, although it can be useful with the cataloged procedures. The use of the DDNAME parameter is described in detail later in this chapter.

Example 1: The following example shows the overriding of DD statements in the IBM-supplied COBUCLG procedure.

```
//jobname       JOB  1234,J.SMITH
//stepname      EXEC COBUCLG
//COB.SYSLIN    DD   DSNAME=GOFILE
//COB.SYSIN     DD   *

          (source module)

/*
//LKED.SYSLIN   DD   DSNAME=*.COB.SYSLIN,     X
//                   DISP=(OLD,CATLG)
                  .
                  .
                  .
/*

        (other DD statements for
        user-defined files)
                  .
                  .
                  .
/*
```

The name of the data set in SYSLIN in the procedure step COB is changed to GOFILE. The name of the data set of SYSLIN in the procedure step LKED is changed to a reference to the SYSLIN DD statement in the COB procedure step, and the data set name GOFILE is cataloged.

Example 2: The following example shows the adding of DD statements to the IBM-supplied COBUCLG procedure. Note that if the statement DD * or the statement DD DATA is used, it must be the last to appear in a series of DD statements.

```
//jobname        JOB  1234,J.SMITH
//stepname       EXEC COBUCLG,                     X
//               PARM.COB=(DECK,LOAD,PMAP)
//COB.SYSPUNCH DD   SYSOUT=B
//COB.SYSIN      DD   *

             (source module)

/*
//GO.TRANSACT    DD   DSNAME=JUNE,DISP=OLD
                 .
                 .
                 .

          (other DD statements for
          user-defined files)
                 .
                 .
                 .
/*
```

Note: In the foregoing example TRANSACT is a cataloged data set. When a data set is cataloged, it is sufficient to refer to it by DSNAME and DISP=OLD.

The PARM.COB option DECK and the SYSPUNCH DD statement are added to obtain a punched object module. The PARM option PMAP is added to obtain a listing of the assembler language expansion of the source module.

Example 3: The following example shows overriding and adding to DD statements at the same time in the IBM-supplied COBUC procedure. Note that overriding statements must be in the same sequence as they appear in the procedure and must precede those statements being added.

```
//jobname        JOB  1234,J.SMITH
//stepname       EXEC COBUC,PARM.COB=(LOAD)
//COB.SYSUT2     DD   SPACE=,UNIT=SYSSQ
//COB.SYSLIN     DD   DSNAME=&&GOFILE,            X
//                      DISP=(MOD,PASS),          X
//                      UNIT=SYSSQ
//COB.SYSIN      DD   *

             (source module)

/*

          (subsequent job steps)
                 .
                 .
                 .
```

The device class on the COB.SYSUT2 DD statement is changed to SYSSQ, and the SPACE parameter is nullified. Therefore, mass storage devices cannot be allocated. Any tape volumes to be assigned must have standard labels. The COB.SYSLIN DD statement is changed so that it passes the object module to subsequent job steps.

Example 4: The following example shows how to concatenate a data set with a data set defined in the COBULG procedure.

```
//jobname       JOB   1234,J.SMITH
//stepname      EXEC  COBULG
                .
                .
                .
//LKED.SYSLIB   DD    [blank operand field]
//              DD    [parameters]
                .
                .
                .
/*
```

Instead of the blank operand field, parameters could have been used to override the SYSLIB statement; the data set defined by the unnamed DD statement would then be concatenated to the data set that was redefined by overriding.

Note that any number of libraries could be concatenated to the SYSLIB data set. For example:

```
//LKED.SYSLIB   DD
//              DD  DSNAME=USERLIB,DISP=OLD
//              DD  DSNAME=MYLIB,DISP=OLD
```

USING THE DDNAME PARAMETER

The DDNAME parameter is used to define a dummy data set that can assume the characteristics of an actual data set, defined by a subsequent DD statement within the step. If a matching DD statement is found, its characteristics, with the exception of its ddname, replace those of the statement using the DDNAME parameter. If a matching DD statement is not found within the step, the data set defined by the DDNAME parameter remains a dummy.

This section contains examples showing the use of the DDNAME parameter with cataloged procedures.

The rules for using the DDNAME parameter are as follows:

- A backward reference (e.g., *.ddname) to a DD statement referred to by a DDNAME parameter cannot be used because the statement that is referred to loses its identity.
- A backward reference to a statement containing a DDNAME parameter can be used, but only after the statement to which the DDNAME parameter refers has been encountered. If a backward reference is used before the dummy data set (defined by DDNAME) has been given real characteristics, these real characteristics will not be transferred to the DD statement that contains the backward reference. For example, if DCB=*.ddname is used (where ddname is the name of a statement containing an unresolved DDNAME parameter), the DCB fields that are transferred are blank.
- Unnamed DD statements can be placed after a statement containing the DDNAME parameter (indicating concatenation), but unnamed DD statements cannot be placed after a statement referred to by a DDNAME parameter.
- The DDNAME parameter can be used a maximum of five times in a step, but each DDNAME parameter must refer to a different statement.
- The DDNAME parameter cannot be used in a JOBLIB statement.

When using the DDNAME parameter, the programmer should also keep the following in mind:

- The name of the DD statement referred to does not replace the name of the referencing statement.
- If a statement that contains the DDNAME parameter is overridden, it is nullified.
- If overriding is performed with a statement that contains the DDNAME parameter, all parameters in the overridden statement are nullified.

The following DD statements:

```
//S1        EXEC PGM=progname
//D1        DD   DDNAME=D3
//D2        DD   (parameters X,Y,Z)
//D3        DD   (parameters U,T,V)
```

will result in the same data definition produced by the following statements:

```
//S1        EXEC PGM=progname
//D1        DD   (parameters U,T,V)
//D2        DD   (parameters X,Y,Z)
```

EXAMPLES OF USING THE DDNAME PARAMETER

Example 1: The following example shows how to override the first DD statement in a cataloged procedure with a DD * statement, and allow subsequent statements to be processed. Without the DDNAME parameter, replacing the first DD statement with a DD * statement would terminate processing of subsequent statements in the job step. The cataloged procedure (PROC3) is as follows:

```
//STEP1       EXEC PGM=progname
//DD1         DD   (any parameters except
                    DATA or *)
//DD2         DD   (any parameters except
                    DATA or *)
```

The job procedure in which the overriding takes place appears in the input stream as follows:

```
//JOB1        JOB  1234,J.SMITH
//S1          EXEC PROC3
//STEP1.DD1 DD     DDNAME=D1
//D1          DD   *
```

The STEP1.DD1 statement overrides the DD1 statement; the DD2 statement is processed; then the D1 statement is processed.

Example 2: The following example shows how to override the first DD statement in a cataloged procedure with a DD * statement and how to add a DD statement. The cataloged procedure (PROC3) is as follows:

```
//STEP1       EXEC PGM=progname
//DD1         DD   (any parameters except
                    DATA or *)
//DD2         DD   (any parameters except
                    DATA or *)
```

The job procedure in which the overriding takes place appears in the input stream as follows:

```
//JOB2        JOB  1234,J.SMITH
//S1          EXEC PROC3
//STEP1.DD1 DD     DDNAME=DD4
//STEP1.DD3 DD     (any parameters except
                    DATA or *)
//DD4         DD   *
```

The DD4 statement effectively overrides the DD1 statement, after the DD2 statement has been processed and the DD3 statement has been added.

Example 3: The following example shows how to concatenate a data set in the input stream with a data set defined by a DD statement in a cataloged procedure. The cataloged procedure (PROC3) is as follows:

```
//STEP1       EXEC PGM=progname
//DD1         DD   (any parameters except
                    DATA or *)
//DD2         DD   (any parameters except
                    DATA or *)
```

The job procedure in which the concatenation takes place appears in the input stream as follows:

```
//JOB3        JOB  1234,J.SMITH
//S1          EXEC PROC3
//STEP1.DD1 DD     (blank operand field)
//            DD   DDNAME=DD3
//DD3         DD   *
```

The data set in the input stream is concatenated with the data set defined by the DD1 statement after the DD2 statement has been processed.

Example 4: The following example shows how to concatenate a data set in the input stream with a data set defined by a DD statement in a cataloged procedure and how to add a DD statement. The cataloged procedure (PROC3) is as follows:

```
//STEP1       EXEC PGM=progname
//DD1         DD   (any parameters except
                    DATA or *)
//DD2         DD   (any parameters except
                    DATA or *)
```

The job procedure in which the concatenation takes place appears in the input stream as follows:

```
//JOB4        JOB  1234, J.SMITH
//S1          EXEC PROC3
//STEP1.DD2 DD     (blank operand field)
//            DD   DDNAME=DD4
//STEP1.DD3 DD     (any parameters except
                    DATA or *)
//DD4         DD   *
```

Example 5: The following example shows how the statement DD DDNAME=SYSIN in the IBM-supplied COBUCLG procedure can be used to add more object modules as input to the linkage editor. The statements appear in the input stream as follows:

```
//jobname      JOB    1234,J.SMITH
//stepname     EXEC   COBUCLG
               .
               .
               .
//COB.SYSIN    DD     *

      (source deck)

/*
//LKED.SYSIN   DD     *

      (first object module)
               .
               .
               .

      (last object module)

/*

(//GO. cards)
```

The COBUCLG procedure contains the following two statements in the linkage edit step:

```
//SYSLIN DD DSNAME=&&LOADSET,                 X
//          DISP=(OLD,DELETE)
//       DD DDNAME=SYSIN
```

The result of concatenating SYSIN with SYSLIN is that when SYSLIN (input to linkage editor) is read, SYSIN is also read and linked with it. For example, if ILBODSP0 is one of the object modules in the SYSIN stream, it will be linked with SYSLIN. The ILBODSP0 module from SYS1.COBLIB will not be used.

In order to use the IBM System/360 Operating System Sort/Merge program, Sort feature statements are written in the COBOL source program. These statements are described in the publication IBM OS Full American National Standard COBOL. The Sort/Merge program itself is described in the publication IBM OS Sort/Merge. In this publication, the system requirements when the Sort feature is used are discussed in "Machine Considerations."

DD statements must be written in the execution-time job steps of the procedure to describe the data sets used by the sort program. DD statements for data sets used during the sort process are described in the section "Sort DD Statements."

Note: The Sort/Merge Checkpoint Restart feature is available to the programmer who uses the COBOL SORT statement through the use of the RERUN statement.

SORT DD STATEMENTS

Three types of data sets can be defined for the sort program in the execution time job step: input, output, and work. In addition, data sets must be defined for the use of the system during the sorting operation.

SORT INPUT DD STATEMENTS

The input data set is associated with a ddname that appears as the ddname portion of the system-name in an ASSIGN clause in the COBOL source program. When the USING option is specified, the compiler will generate an input procedure that will open the data set, read the records, release the records and close the data set.

SORT OUTPUT DD STATEMENTS

The output data set is associated with a ddname that appears as the ddname portion of the system-name in an ASSIGN clause in the COBOL source program. When the GIVING option is specified, the compiler generates an output procedure that will open the data set, return the records, write the records, and close the data set.

SORT WORK DD STATEMENTS

~~The~~ sort program requires at least three ~~work~~ data sets. The ddname for each DD statement is in the form SORTWKnn, where nn is a decimal number. The ddnames for the required data sets must be SORTWK01, SORTWK02, and SORTWK03. Additional work data sets may be defined, but their ddnames must be consecutively numbered, beginning with 04.

SORTWKnn Data Set Considerations

Intermediate data sets (i.e., SORTWKnn data sets) for a sort may be assigned to either magnetic tape or mass storage devices. All of the intermediate storage for one sort must be assigned to the same device type. These may not be on both 7-track and 9-track tape units in the same sort. Any one of the following devices may be used for intermediate storage:

- IBM 2400-series Magnetic Tape Unit (7-track)
- IBM 2400-series Magnetic Tape Unit (9-track)
- IBM 2311 Disk Storage Drive
- IBM 2301 Drum Storage
- IBM 2305 Fixed Head Storage, Models 1 and 2[1]
- IBM 3330 Disk Storage[1]

The publication IBM OS Sort/Merge contains detailed information about these devices.

Since spanned records can be input to and output from the sorting operation, it is the user's responsibility to assign the sort work files to mass storage devices whose track sizes are larger than the logical record size of the records being sorted. An S-mode file whose logical record length is greater than its track size may be sorted by assigning the work files to a magnetic tape unit.

If data sets not involved in the sorting operation are assigned to tape units, these tape units may be used as sort work files by using the UNIT=AFF parameter. For example, if PAYROLL is specified as the

----

[1]The programmer should be sure that the sort program selected supports these new devices.

ddname of the ASSIGN clause in a SELECT statement, the tape unit assigned to PAYROLL could be used as a sort work file by using the following DD statement:

```
//PAYROLL DD UNIT=2400,...
//SORTWK02 DD UNIT=AFF=PAYROLL...
```

Input DD Statement

The input data set must reside on a physical device, a magnetic tape unit, a mass storage device, or in the system input stream. The following example shows DD statement parameters that could be used to define a cataloged input data set.

```
//INSORT    DD    DSNAME=INPT,                X
//                DISP=(OLD,DELETE)
```

These parameters cause the system to search the catalog for a data set named INPT (DSNAME parameter). When found, the data set is associated with the ddname INSORT and used by the sort program. The control program obtains the unit assignment and volume serial number from the catalog, and ~~displays a mounting message to the~~ operator. The DISP parameter indicates that the data set has already been created (OLD). It also indicates that the data set should be deleted (DELETE) after the current job step.

Output DD Statment

The output DD statement must define all of the characteristics of the output data set. The following example shows DD statement parameters that could be used to characterize an output data set:

```
//OUTSORT   DD    DSNAME=OUTPT,UNIT=2400,     X
//                DISP=(NEW,CATLG)
```

The DISP parameter indicates that the data set is unknown to the operating system (NEW) and that it should be cataloged (CATLG) under the name OUTPT (DSNAME parameter). The UNIT parameter specifies that the data set is on a 2400-series tape unit.

SORTWKnn DD Statements

SORTWKnn data sets may be contained on tape or mass storage volumes. When mass storage space is assigned, only the primary allocation is used by the sort, and it must be contiguous.

Note that the SORTWKnn data sets:

1. May not be on 7-track tape when the input data set is on 9-track tape.
2. May be on 7-track tape when the output data set is on 9-track tape.
3. Cannot use the data conversion feature if they are on 7-track tape. The TRTCH subparameter must reflect this.
4. May be on 9-track tape when the input data set is on 7-track tape.

SORTWKnn Example A: The following DD statement parameters could be used to define a tape intermediate storage data set:

```
//SORTWK01 DD    UNIT=2400,LABEL=(,NL),      X
//               VOLUME=SER=DUMMY
```

These parameters specify an unlabeled data set on a 2400-series tape unit. Since the DSNAME parameter is omitted, the system assigns a unique name to the data set. The omission of the DISP parameter causes the system to assume that the data set is new and that it should be deleted at the end of the current job step. The 2400-series tape units are explicitly of the 9-track format.

SORTWKnn Example B: The following DD statement parameters could be used to define a mass storage intermediate storage data set:

```
//SORTWK01 DD    UNIT=SYSDA,                 X
//               SPACE=(TRK,(200),,CONTIG)
```

These parameters specify a mass storage data set with a standard label (LABEL parameter default value). The SPACE parameter specifies that the data set is to be allocated 200 contiguous tracks. The system assigns a unique name to the data set and deletes it at the end of the job step.

ADDITIONAL DD STATEMENTS

The sort program requires two additional DD statements:

```
//SYSOUT   DD   SYSOUT=A
```

which defines the system output data set.

```
//SORTLIB  DD   DSNAME=SYS1.SORTLIB,       X
//              DISP=SHR
```

which defines the library containing the SORT modules.

Note: At system generation time, theprogrammer can designate that SORT diagnostic messages be printed either on the console or on the unit designated SYSOUT. If the system is generated to write SORT messages on SYSOUT, these messages may overprint any COBOL output assigned to SYSOUT. For example, if the programmer has selected SYSOUT on which to print a report in the output procedure associated with the execution of the COBOL SORT statement, any SORT messages will be interspersed within that report. If it is not possible to assign the SORT messages to the console, the programmer should assign his COBOL output to temporary files and print the reports at a later time.

SHARING DEVICES BETWEEN TAPE DATA SETS

A single tape unit may be assigned to two sort data sets when the data sets are one of the following pairs:

- The input data set and the first intermediate storage data set (SORTWK01).
- The input data set and the output data set.

The AFF subparameter of the UNIT parameter can be used to associate the input data set with either the SORTWK01 data set or the output data set. The subparameter can appear in the DD statement for SORTWK01 or output.

USING MORE THAN ONE SORT STATEMENT IN A JOB

More than one SORT statement may be used in a single program or in two or more programs that are combined into a single load module.

SORT PROGRAM EXAMPLE

The control cards in Figure 104 could be used with the sample program that illustrates the Sort feature. A description of the Sort Feature can be found in the publication IBM OS Full American National Standard COBOL.

```
//SORTEST      JOB   NY838670165,          X
//                   'J.SMITH',            X
//                   MSGLEVEL=1
//SORTJS3      EXEC  COBUCLG
//COB.SYSIN    DD    *
                .
                .
                .

      (COBOL source program)
                .
                .
                .
//GO.SORTWK01 DD     UNIT=2311,            X
//                   SPACE=(TRK,(200),     X
//                   ,CONTIG)
//GO.SORTWK02 DD     UNIT=2311,            X
//                   SPACE=(TRK,(200),     X
//                   ,CONTIG)
//GO.SORTWK03 DD     UNIT=2311,            X
//                   SPACE=(TRK,(200),     X
//                   ,CONTIG)
//GO.OUTSORT   DD    UNIT=183,             X
//                   LABEL=(,NL),          X
//                   VOLUME=SER=NONE
//GO.SYSOUT    DD    SYSOUT=A
//GO.SORTLIB   DD    DSNAME=SYS1.SORTLIB,  X
//                   DISP=SHR
//GO.INFILE    DD    UNIT=182,             X
//                   LABEL=(,NL),          X
//                   VOLUME=SER=DUMMY
```

Figure 104. Sort Feature Control Cards

The minimum number of SORTWKnn data sets are used; the sort operation can be optimized by using additional work data sets (see the publication IBM System/360 Operating System: Sort/Merge).

CATALOGING SORT DD STATEMENTS

Since repeated use of the Sort feature often involves the same execution time DD statements, the user may wish to catalog them (see "Using the Cataloged Procedures").

When using the COBOL RERUN feature, all SORT messages are written on the console.

LINKAGE WITH THE SORT/MERGE PROGRAM

Communication between the Sort/Merge program and the COBOL program is maintained by the COBOL library subroutine ILBOSRT0. The programmer must designate via the appropriate SORTLIB DD statement the Sort/Merge program he wishes to use.

If the INPUT PROCEDURE option of the SORT statement is specified, exit E15 of the Sort/Merge program is used. The return code indicating "insert records" is issued when a RELEASE statement is encountered, and the return code indicating "do not return" is issued when the end of the procedure is encountered.

If the OUTPUT PROCEDURE option is specified, exit E35 of the Sort/Merge program is used. The return code indicating "delete records" is issued when a RETURN statement is encountered, and the return code indicating "do not return" is issued when the end of the procedure is encountered. (For additional information, about the Sort/Merge program, see the publication IBM OS Sort/Merge).

Completion Codes

The Sort/Merge program returns a completion code upon termination. This code may be interrogated by the COBOL program. The codes are:

- 0 -- Successful completion of Sort/Merge
- 16 -- Unsuccessful completion of Sort/Merge

SUCCESSFUL COMPLETION: When a Sort/Merge application has been successfully executed, a completion code of zero is returned and the sort terminates.

UNSUCCESSFUL COMPLETION: If the sort, during execution, encounters an error that will not allow it to complete successfully, it returns a completion code of 16 and terminates. (Possible errors include an out-of-sequence condition or an input/output error that cannot be corrected.) The publication IBM OS Sort/Merge contains a detailed description of the conditions under which this termination will occur.

The returned completion code is stored in a special register called SORT-RETURN by the COBOL library subroutine; an unsuccessful termination of the sort may then be tested for and appropriate action specified. Note that the contents of SORT-RETURN will change with the execution of a SORT statement. The following is an example of the use of SORT-RETURN with the sort feature:

```
SORT SALES-RECORDS ON ASCENDING KEY
CUSTOMER-NUMBER, DESCENDING KEY DATE,
USING FN-1, GIVING FN-2.

IF SORT-RETURN NOT EQUAL TO ZERO,
DISPLAY 'SORT UNSUCCESSFUL' UPON
CONSOLE, STOP RUN.
```

If no references to SORT-RETURN are made in a program, an unsuccessful sort will generate the following message:

```
IKF888I- UNSUCCESSFUL SORT FOR SD
         SORT-FILE DDNAME
```

See the publication IBM OS Full American National Standard COBOL Version 4 Messages, for a description of action to be taken.

LOCATING SORT RECORD FIELDS

Records defined under a COBOL SD are assigned a BLL (Base Locator for Linkage Section), rather than a BL (Base Locator) as is done with other records. Location of a given data item in an object-time dump when the record in which it is contained references a BLL can be determined as follows:

1. From the compilation listing, determine:

   a. The displacement of the item (see Data Division Map).

   b. The relative address of the BLL CELLS (see the Memory Map Table).

   c. The BLL number.

2. From the dump, determine the relocation factor (USE/EP).

3. Add the relative address of the BLL CELLS to the relocation factor to obtain the absolute BLL CELLS address in the dump.

4. Each BLL is 4 bytes long; they are located in ascending sequence, beginning in the dump at the address computed in Step 3 BLL=1 is the first 4 bytes, BLL=2 is the second 4 bytes, etc. Find the appropriate 4 bytes.

5. The 4 bytes obtained in Step 4 contain the absolute base address of the desired record. Add the item's displacement to it to obtain the absolute address of the leftmost byte of the field in the dump.

LOCATING LAST RECORD RELEASED TO SORT BY AN INPUT PROCEDURE

For debugging purposes, it is sometimes useful to determine the last input record released to the Sort program. The following procedure should be used:

1. From the Data Division map, determine the BLL number of the SORT file being processed at the time of program termination. Assume it is BLLn.

2. From the Task Global Table map, determine the location of the BLL cells in the COBOL object program.

3. The nth BLL in the core dump will point to the last record released to SORT.

Note: This BLL is initialized when control is first transferred to the input procedure. Thus, if the program terminates before control ever goes to the input procedure, the BLL will not be initialized.

SORT/MERGE CHECKPOINT/RESTART

The CHECKPOINT/RESTART feature is available to the programmer using the COBOL SORT statement. In order to initiate a checkpoint, the programmer uses DD statements and the RERUN clause. The DD statement for use in taking a checkpoint is discussed in "Using the Checkpoint/Restart Feature."

The RERUN clause is used to indicate that checkpoints are written, at logical intervals determined by the sort program, during the execution of all SORT statements in the program. This RERUN clause is fully described in the publication IBM OS Full American National Standard COBOL.

EFFICIENT PROGRAM USE

The information you give the Sort/Merge program about the application it is to perform helps the sort and merge phases to produce a fast, efficient sort or merge. When you do not supply information such as data set size and record format, tne program must make assumptions, which, if incorrect, lead to ineffiency.

DATA SET SIZE

The most important information one can give is an accurate data set size using the SORT-FILE-SIZE special register. If the exact number of records in the input data set is known, that number should be used as the value. If the exact number is not known, an estimate should be made.

When the Sort/Merge program has accurate information about data set size, it can make the most efficient use of both main storage and intermediate storage. Unless the Sort/Merge Program Product (SM1) is installed, the SORT-FILE-SIZE special register has no effect when the sort work data sets are on disk. When SM1 is used, accurate specification of SORT-FILE-SIZE is the only way the SM1 performance benefits are reached.

MAIN STORAGE REQUIREMENTS

If the maximum amount of main storage to be used by the Sort/Merge program was not specified at system generation time for SM (the OS Sort/Merge Program -- Program No. 360S-SM-023) or at installation time for SM1 (the Program Product Sort/Merge -- Program No. 5734-SM1), the program assumes a maximum of 15,500 bytes. The sort program requests 12,000 bytes leaving 3500 bytes for system functions. Performance usually improves as the program is given more main storage. Approximately 44K bytes of main storage are needed for efficient execution of the sort/merge program, and performance increases as more main storage is made available.

If the amount of main storage was specified at system generation time, it is the programmer's responsibility to ensure that the Sort/Merge program has at least that much core storage available in addition to the space needed for Data Management and the COBOL program. If this amount of core storage is not available, the program will terminate abnormally.

The programmer may alter, dynamically within the COBOL program, the core storage default values for the Sort/Merge program. The SORT-CORE-SIZE special register may be used to communicate changes to the Sort-Merge program. In general, a positive value placed in SORT-CORE-SIZE denotes the amount of storage the programmer is allocating for use by the Sort/Merge program. For example, the statement "MOVE 30000 TO SORT-CORE-SIZE" means that 30000 bytes of storage are available to the Sort/Merge program.

Special considerations apply when SM1 is used. If the program product is installed with the CORE=MAX option, SM1 allocates all available core storage in a region for its own use. If an input procedure then attempts to open a file, an 80A abnormal termination may result if the necessary data management modules have not already been loaded, or are not resident in the link pack area (LPA), since no more space is available. Accordingly, if 30000 is moved to SORT-CORE-SIZE, COBOL communicates to SM1 that 30000 bytes of storage are available to it. There are, in addition, two other uses for SORT-CORE-SIZE.

If a negative value is placed in the special register prior to execution of the sort, SM1 uses the default CORE option specified at installation, but sets aside that absolute value before obtaining the CORE SIZE installed. Also, if ALL '9' (or +999999) is moved to SORT-CORE-SIZE prior to a sort operation, SM1 executes with the CORE=MAX option, regardless of the installed value, while reserving 6K bytes of main storage for use by the data management routines. (For additional information about the Sort Feature options, see the Program Product publication IBM OS Sort/Merge Programmer's Guide, Order No. SC33-4007.)

Changing the main storage allocation can be useful when a sort-merge application is to be run in a multiprogramming environment. By reducing the amount of main storage allocated to sort, so that other programs can have the storage they need to operate simultaneously, the performance of sort is impaired. However, if this allocation is increased, so that a large sort application runs more efficiently, the performance of other jobs sharing the multiprogramming environment is impaired, if not made altogether impossible.

SORT DIAGNOSTIC MESSAGES

The messages generated by the Sort Feature are listed in the publication IBM OS Sort/Merge and IBM OS Messages and Completion Codes. The identifying characters in a sort message are IER.

When the Sort/Merge program is installed, the user can elect to have messages sent to the printer, in which case a DD card with a ddname of SYSOUT must be included in the job step. If SM1 is used, the programmer can dynamically alter the ddname of the file on which SM1 is to write its messages. If SM1 has been installed with provision for routing its messages to the printer, then the programmer can place in the SORT-MESSAGE special register the ddname that SM1 is to substitute for SYSOUT, for message routing. For example, when the statement MOVE "SORTDDNM" TO SORT-MESSAGE is executed before an SM1 sort is initiated, then the SM1 sort writes its printer messages to the data set SORTDDNM rather than to SYSOUT. If SORT-MESSAGE is not referred to in the program, SYSOUT is the default value.

One technique for specifying the sort print file ddname would be to include source language and job control language statements as follows:

- Linkage Section

```
01 SORT-PARAMETERS.
05 PARAMETER-COUNT PIC 9(4) USAGE COMP.
05 SORT-DDNAME PIC X(8).
```

- Immediately preceding the sort operation

```
IF PARAMETER-COUNT IS NOT EQUAL TO 0
   MOVE SORT-DDNAME TO SORT-MESSAGE.
```

- On the EXEC card

```
//GOSTEP EXEC PGM=program-name,
     PARM='SORTDDNM'
```

Note: This technique of assigning a unique value to SORT-MESSAGE without modifying or recompiling the program can also be applied to the special registers SORT-CORE-SIZE, SORT-MODE-SIZE, and SORT-FILE-SIZE.

DEFINING VARIABLE-LENGTH RECORDS

If the input records used are of variable length, the record length that occurs most frequently in the input data set (modal length) should be put into the special register SORT-MODE-SIZE. This value is used to help define a data set based on a particular length. If a value is not specified, the SORT program assumes it is equal to the average of the maximum and minimum record lengths in the input data set. If, for example, the data set contains mostly small records and just a few long records, the SORT program would assume a high modal length and would allocate a larger record storage area than necessary. Conversely, if the data set contains just a few short records and many long records, the SORT program would assume a low modal length and might not allocate a large enough record storage area to sort data. For a complete discussion, see the publication IBM OS Sort/Merge Program.

SORTING VARIABLE-LENGTH RECORDS

Figure 105 illustrates one way to sort variable-length records described by the OCCURS clause with the DEPENDING ON option. If the FD's (file-name description) and the SD's (sort-file-name description) are defined as in this figure, where the record descriptions of the FD's and the SD correspond, possibilities for error arise. It is suggested, therefore, that the user consider the following:

1. Specification of the statement

   SORT SORT-FILE USING INPUT-FILE...

   would probably lead to incorrect results. This statement implies a READ ... INTO ... statement; that is, after INPUT-FILE has been read, the record is moved to AAA. However, because the user must set the length of this receiving field prior to moving A to AAA but cannot do so, the compiler may use an incorrect length that results in abnormal termination. Instead, the user should substitute an input procedure for the USING option, as in the section of code labeled PARA2B in the example.

2. Similarly, the statement

   SORT SORT-FILE... GIVING OUTPUT-FILE

   would probably yield incorrect results. Before OUTPUT-FILE is written out, the record is moved to AA. The correct length of this receiving field must be set before the move, but use of the GIVING option precludes this. To avoid error, the user should substitute an output procedure for the GIVING option, as in section PARA3B of the example.

TERMINATING A SORT OPERATION

The SORT-RETURN special register can also be used to terminate an SM1 sort operation. If the programmer places the value 16 in this special register at any point during an input or output procedure, the sort is terminated immediately after execution of the next RELEASE or RETURN statement.

Note: Once a value has been placed in a sort special register and the SORT statement has executed, this value is used (even if the special register is modified during the sort operation) until another sort is initiated. The one exception to this rule is in the use of the special register SORT-RETURN, which is set to zero at the beginning of each sort.

SORT FOR ASCII FILES

For sorting ASCII files, the normal EBCDIC collating sequence is provided unless the user specifies otherwise on a per sort basis.

To specify a sort using the ASCII collating sequence, the programmer must include the "C" organization entry in the ASSIGN clause for the file-name associated with the file to be sorted. No buffer offset may be given with the sort feature.

Note: The SM1 program is required for sorting a file using the ASCII collating sequence.

Part 1

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  VLSORT.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT ....
    SELECT ....
    SELECT ....
DATA DIVISION.
FILE SECTION.
FD  INPUT-FILE.
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS A.
01  A.
    02  B PIC 99.
    02  C OCCURS 1 TO 10 TIMES
        DEPENDING ON B.
        03  D PIC 99.
        03  E PIC XX.
FD  OUTPUT-FILE
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS AA.
01  AA.
    02  BB PIC 99.
    02  CC OCCURS 1 TO 10 TIMES
        DEPENDING ON BB.
        03  DD PIC 99.
        03  EE PIC XX.
SD  SORT-FILE
    DATA RECORD IS AAA.
01  AAA.
    02  BBB PIC 99.
    02  CCC OCCURS 1 TO 10 TIMES
        DEPENDING ON BBB.
        03  DDD PIC 99.
        03  EEE PIC XX.
```

Part 2

```
PROCEDURE DIVISION.
PAR1 SECTION.
      SORT SORT-FILE ASCENDING KEY BBB.
    INPUT PROCEDURE PAR2
    OUTPUT PROCEDURE PAR3.
            STOP RUN.
PAR2 SECTION.
PAR2A.
    OPEN INPUT INPUT-FILE.
PAR2B.
    READ INPUT-FILE AT END GO TO PAR2C.
    MOVE B TO BBB.
    RELEASE AAA FROM A.[1]
    GO TO PAR2B.
PAR2C.
    CLOSE INPUT-FILE.
PAR2-EXIT.
    EXIT.
PAR3 SECTION.
PAR3A.
    OPEN OUTPUT OUTPUT-FILE.
PAR3B.
    RETURN SORT-FILE AT END GO TO PAR3C.[2]
    MOVE BBB TO BB.
    WRITE AA FROM AAA.
    GO TO PAR3B.
PAR3C.
    CLOSE OUTPUT-FILE.
PAR3-EXIT.
    EXIT.
```

[1]When using a sort input procedure, the RELEASE ... FROM clause, which implies a MOVE and then a RELEASE, should always be preceded by a MOVE that sets the length of the receiving field (AAA, in this example).
[2]When using a sort output procedure, the RETURN ... INTO ... clause, which implies the RETURN and then a MOVE, should never be used. There is no way for the user to set the correct length of the receiving field.

Figure 105. Sorting Variable-Length Records Whose File-name Description and Sort-File-name Description Correspond

Segmentation provides a means of accomplishing object time overlay as a result of specifications made at the source language level. The programmer may divide the Procedure Division of a source program into sections. Through the use of a system of priority numbers, certain sections are designated as permanently resident in core and other sections as overlayable fixed segments and/or independent segments. Thus, a large program can be executed in a defined area of core storage by limiting the number of segments in the program that are permanently resident in core storage.

Note: The segmentation feature is not available when the loader is used.

Suppose that because of core storage limitations the program SAVECORE is segmented as shown in Figure 106. Only those segments having priority numbers less than the segment limit of 15 are designated as permanently resident.

Assuming that 12K is available for the program SAVECORE, Figure 107 shows that manner in which core storage would be utilized. Sections 3 and 6, and sections 5 and 7 are considered logical units since they have the same priority numbers. Sections 3 and 6 can be in core together, but section 7 cannot be there at the same time. Similarly, sections 5 and 7 can be in core together, but section 3 cannot be there at the same time.

Sections in the permanent segment (SECTION-1, SECTION-2, and SECTION-4) are those that must be available for reference at all times, or those to which reference is made frequently. They are distinguished here by the fact that they have been assigned priority numbers less than the segment limit.

Sections in the overlayable fixed segment are sections that are less frequently used. These sections are always made available in the state in which they were last used. They are distinguishable here by the fact that they have been assigned priority numbers greater than the segment limit, but less than 50.

Sections in the independent segment can overlay, and be overlaid by, either an overlayable fixed segment or another independent segment. Independent segments are those assigned priority numbers greater than 49 and less than 100. These independent segments are returned to their initial state when they are brought into core storage.

```
IDENTIFICATION DIVISION.

PROGRAM-ID.  SAVECORE.
.
.
.

ENVIRONMENT DIVISION.

OBJECT-COMPUTER.  IBM-360-H50
     SEGMENT-LIMIT IS 15.
.
.
.
DATA DIVISION.
.
.
.
PROCEDURE DIVISION.
SECTION-1  SECTION 8.
.
.
.
SECTION-2  SECTION 8.
.
.
.
SECTION-3  SECTION 16.
.
.
.
SECTION-4  SECTION 8.
.
.
.
SECTION-5  SECTION 50.
.
.
.
SECTION-6  SECTION 16.
.
.
.
SECTION-7  SECTION 50.
.
.
.
```

Figure 106. Segmentation of Program SAVECORE

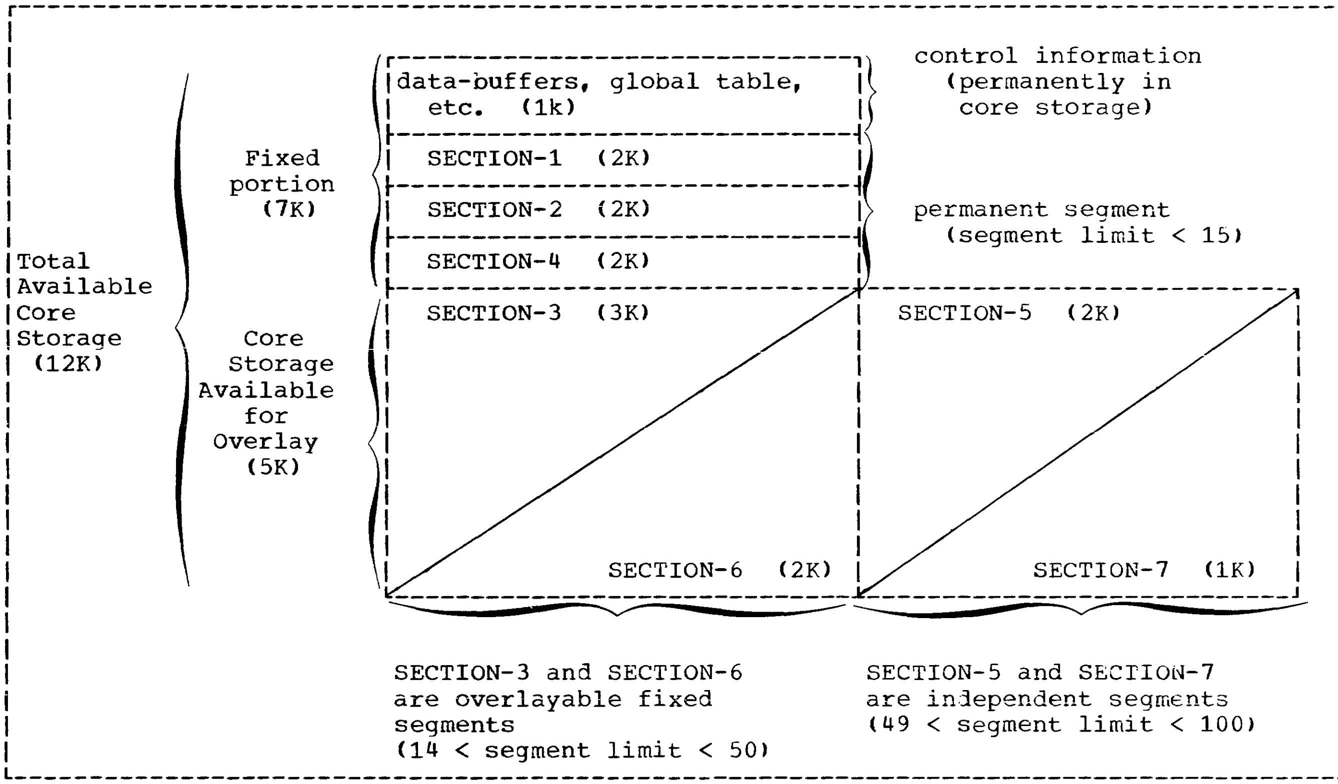


Figure 107. Storage Layout for SAVECORE

USING THE PERFORM STATEMENT IN A SEGMENTED PROGRAM

When the PERFORM statement is used in a segmented program, the programmer should be aware of the following:

- A PERFORM statement that appears in a section whose priority-number is less than the segment limit can have within its range only (a) sections with priority-numbers less than 50, and (b) sections wholly contained in a single segment whose priority-number is greater than 49.

  Note: As an extension to American National Standard COBOL, the OS Full American National Standard COBOL Compiler allows sections with any priority-number to fall within the range of a PERFORM statement.

- A PERFORM statement that appears in a section whose priority-number is equal to or greater than the segment limit can have within its range only (a) sections with the same priority-number as the section containing the PERFORM statement, and (b) sections with priority-numbers that are less than the segment limit.

  Note: As an extension to American National Standard COBOL, the OS Full American National Standard COBOL Compiler allows sections with any priority-number to fall within the range of a PERFORM statement.

- When a procedure-name in a segment with a priority-number less than the segment limit referred to by a PERFORM statement in a segment with a priority-number greater than the segment limit, the independent segment will be reinitialized upon exit from the PERFORM.

OPERATION

Execution of the object program begins in the root segment; i.e., the first segment in the permanent segment. If the program contains no permanent segments, or if the first section to be executed in the program is not part of the root segment, the compiler generates a dummy segment that will initiate the execution of the first overlayable or independent segment. All global tables, literals, and data areas are part of the root segment. Called object-time subroutines are also part of the root segment. Called subprograms are loaded with the fixed portion of the main program and assigned a priority of zero. Otherwise, the program executes just as if it were not segmented.

For a discussion on determining the priority of the last segment loaded into the transient area, see the section "Debugging a Segmented Program" in the chapter "Program Checkout".

COMPILER OUTPUT

The output produced by the compiler is an overlay structure consisting of multiple object modules preceded by linkage editor control statements. Segments whose priority is greater than the segment limit (or 49, if no SEGMENT-LIMIT clause is specified) consist of executable instructions only. The PMAP output is given in this sequence: all sections with priorities greater than the segment limit are listed first in ascending order by priority number, followed by the root segment.

Figure 108 shows the output of a sample segmentation program.

JOB CONTROL CONSIDERATIONS

In order to execute a segmented program, the programmer must specify OVLY in the parameter field of the linkage editor EXEC statement. Note that when using the IBM-supplied cataloged procedures, the user must respecify the LIST and LET parameters.

```
000060 IDENTIFICATION DIVISION.                                              00154790
000070 PROGRAM-ID.  SEG-SAMPLE.
000080 AUTHOR.  PROGRAMMER-NAME.
000090 REMARKS.                                                              00154820
000100     SPECIAL OPERATOR INSTRUCTIONS - NONE.                             00154830
000110     INPUT REQUIRED - NONE.                                            00154840
000120     PURPOSE                                                           00154850
000130         TO CREATE A SINGLE FILE ON DISK USING                         00154860
000140         QSAM/DTFSD, AND READ  IT BACK.                                00154870
000150         PROGRAM USES SEGMENTATION                                     00154880
000160         WITH FILE PROCESSING SPREAD OVER                              00154890
000170         THE PERMANENT, OVERLAYABLE FIXED,                             00154900
000180         AND INDEPENDENT SEGMENTS.                                     00154910
000190     EXPECTED RESULTS                                                  00154920
000200         START TEST SEG-SAMPLE
000210         (EACH SEGMENT DISPLAYS ITS SEGMENT NUMBER                     00154940
000220         AND FUNCTION)                                                 00154950
000230         END TEST SEG-SAMPLE SUCCESSFUL RUN
000240         SECTIONS WHILE WRITING APPEAR                                 00154970
000250         IN ORDER 80, 20, 30, 60, 40.                                  00154980
000260         SECTIONS WHILE READING APPEAR                                 00154990
000270         IN ORDER 80, 60, 30, 40, 20.                                  00155000
000280     ERROR INDICATIONS                                                 00155010
000290         **ERROR DISK SEQ I/O**                                        00155020
000300         **ERROR END OF EXTENT WRITING AFTER (RECORD)**                00155030
000310         **ERROR UNEXPECTED EOF READING AFTER                          00155040
000320             RECORD (RECNO)**                                          00155050
000330         **ERROR EOF NOT FOUND**                                       00155060
000340         **RECORD IS (RECNO)                                           00155070
000350           SHOULD BE (RECNO)**                                         00155080
000380         PROGRAM CONTAINS PERFORMS FROM BASE SECTION                   00155110
000390         TO PERMANENT, OVERLAYABLE FIXED, AND INDEPENDENT              00155120
000400         SEGMENTS.                                                     00155130
000410         ALSO CONTAINS PERFORMS FROM INDEPENDENT TO PERMANENT           00155140
000420         AND FROM OVERLAYABLE FIXED TO PERMANENT SEGMENTS.             00155150
000430         ALSO CONTAINS PERFORMS ENTIRELY WITHIN A SEGMENT IN           00155160
000440         IN EACH CATEGORY.                                             00155170
000450 ENVIRONMENT DIVISION.                                                 00155180
000460 CONFIGURATION SECTION.                                                00155190
000470 SOURCE-COMPUTER. IBM-360-40.                                          00155200
000480 OBJECT-COMPUTER. IBM-360-40                                           00155210
000490     MEMORY SIZE 64000 CHARACTERS                                      00155220
000500     SEGMENT-LIMIT IS 25.                                              00155230
000510 INPUT-OUTPUT SECTION.                                                 00155240
000520 FILE-CONTROL.                                                         00155250
000530     SELECT FILE-1 ASSIGN TO        DA-2311-S-DKSQ01A.                 00155260
000540 DATA DIVISION.                                                        00155270
000550 FILE SECTION.                                                         00155280
000560 FD  FILE-1                                                            00155290
000570     RECORDING MODE IS F                                               00155300
000580     LABEL RECORDS OMITTED
000590     DATA RECORD IS RECFD1.                                            00155310
000600 01  RECFD1 PICTURE X(83).                                             00155320
000610 WORKING-STORAGE SECTION.                                              00155330
000620 77  ERRORSW PIC A VALUE SPACE.                                        00155340
000630 77  ERCTFL  PIC S99 VALUE ZERO.                                       00155350
000640 77  MSGHDR  PIC X(22) VALUE '**ERROR DISK SEQ I/O**'.                 00155360
```

Figure 108. Sample Segmentation Program (Part 1 of 14)

```
000650 77  MSGEOX PIC X(36)                                            00155370
000660     VALUE '**ERROR END OF EXTENT WRITING AFTER '.               00155380
000670 77  MSGEOF  PIC X(37)                                           00155390
000680     VALUE '**ERROR UNEXPECTED EOF READING AFTER '.              00155400
000690 77  MSGNEF  PIC X(23) VALUE '**ERROR EOF NOT FOUND**'.          00155410
000700 01  REC1.                                                       00155420
000710     02  REC-ID.                                                 00155430
000720     03  REC-HD  PIC X(4) VALUE 'RECD'.                          00155440
000730     03  REC-NO  PIC S9(4) VALUE ZERO.                           00155450
000740     02  FILLER  PIC A(75) VALUE SPACES.                         00155460
000750     66  RECID RENAMES REC-ID.                                   00155470
000760 01  VER-REC.                                                    00155480
000770     02  VER-ID.                                                 00155490
000780     03  VER-HD  PIC X(4) VALUE 'RECD'.                          00155500
000790     03  VER-NO  PIC S9(4) VALUE ZERO.                           00155510
000800 PROCEDURE DIVISION.                                             00155520
000810 BASE-SECTION SECTION 0.                                         00155530
000820     DISPLAY 'START TEST SEG-SAMPLE'.
000830     OPEN OUTPUT FILE-1.                                         00155550
000840     PERFORM W-80-0 THRU W-80-9.                                 00155560
000850     PERFORM W-30-0 THRU W-30-9.                                 00155570
000860     PERFORM W-60-0 THRU W-60-9.                                 00155580
000870     PERFORM W-40-0 THRU W-40-9.                                 00155590
000880 BASE-50.                                                        00155600
000890     CLOSE FILE-1.                                               00155610
000900     OPEN INPUT FILE-1.                                          00155620
000910     PERFORM R-80-0 THRU R-80-9.                                 00155630
000920     GO TO R-60-0.                                               00155640
000930 BASE-60.                                                        00155650
000940     PERFORM R-40-0 THRU R-40-9.                                 00155660
000950     READ FILE-1 INTO REC1 AT END GO TO BASE-70.                 00155670
000960     DISPLAY MSGHDR DISPLAY MSGNEF                               00155680
000970     MOVE 'E' TO ERRORSW.                                        00155690
000980 BASE-70.                                                        00155700
000990     CLOSE FILE-1.                                               00155710
001000 BASE-90.                                                        00155720
001010     IF ERRORSW IS EQUAL TO 'E'                                  00155730
001020     DISPLAY 'END TEST SEG-SAMPLE UNSUCCESSFUL RUN' ELSE
001030     DISPLAY 'END TEST SEG-SAMPLE SUCCESSFUL RUN'.
001040     STOP RUN.                                                   00155760
001050 SECTION-20 SECTION 20.                                          00155770
001060 W-20-0.                                                         00155780
001070     DISPLAY 'SECTION 20 WRITE'.                                 00155790
001080     NOTE ENTERED BY PERFORM FROM W-80-0.                        00155800
001090     PERFORM W-21-0 THRU W-21-9 5 TIMES.                         00155810
001100 W-20-9.                                                         00155820
001110     EXIT.                                                       00155830
001120 W-21-0.                                                         00155840
001130     WRITE RECFD1 FROM REC1 INVALID KEY                          00155850
001140     DISPLAY MSGHDR                                              00155860
001150     DISPLAY MSGEOX RECID                                        00155870
001160     MOVE 'E' TO ERRORSW                                         00155880
001170     GO TO BASE-50.                                              00155890
001180     ADD 0001 TO REC-NO.                                         00155900
001190 W-21-9.                                                         00155910
001200     EXIT.                                                       00155920
001210 R-20-0.                                                         00155930
```

Figure 108. Sample Segmentation Program (Part 2 of 14)

```
001220        DISPLAY 'SECTION 20 READ'.                                 00155940
001230        NOTE ENTERED BY PERFORM FROM BASE-40.                      00155950
001240        PERFORM R-21-0 THRU R-21-9 5 TIMES.                        00155960
001250 R-20-9.                                                           00155970
001260        EXIT.                                                      00155980
001270 R-21-0.                                                           00155990
001280        READ FILE-1 INTO REC1 AT END                               00156000
001290        DISPLAY MSGHDR DISPLAY MSGEOF                              00156010
001300        ADD 4 TO ERCTFL MOVE 'E' TO ERRORSW                        00156020
001310        GO TO R-21-9.                                              00156030
001320        IF REC-ID IS NOT EQUAL TO VER-ID                           00156040
001330        DISPLAY MSGHDR DISPLAY 'EXPECTED ' VER-ID ' FOUND ' REC-ID 00156050
001340        ADD 1 TO ERCTFL MOVE 'E' TO ERRORSW                        00156060
001350        MOVE REC-ID TO VER-ID.                                     00156070
001360        ADD 1 TO VER-NO.                                           00156080
001370 R-21-9.                                                           00156090
001380        IF ERCTFL IS GREATER THAN 3                                00156100
001390        GO TO BASE-70.                                             00156110
001400 SECTION-30 SECTION 30.                                            00156120
001410 W-30-0.                                                           00156130
001420        DISPLAY 'SECTION 30 WRITE'.                                00156140
001430        NOTE ENTERED BY PERFORM FROM BASE-SECTION.                 00156150
001440        PERFORM W-31-0 THRU W-31-9 11 TIMES.                       00156160
001450 W-30-9.                                                           00156170
001460        EXIT.                                                      00156180
001470 W-31-0.                                                           00156190
001480        WRITE RECFD1 FROM REC1 INVALID KEY                         00156200
001490        DISPLAY MSGHDR                                             00156210
001500        DISPLAY MSGEOX RECID                                       00156220
001510        MOVE 'E' TO ERRORSW                                        00156230
001520        GO TO BASE-50.                                             00156240
001530        ADD 0001 TO REC-NO.                                        00156250
001540 W-31-9.                                                           00156260
001550        EXIT.                                                      00156270
001560 R-30-0.                                                           00156280
001570        DISPLAY 'SECTION 30 READ'.                                 00156290
001580        NOTE ENTERED BY GO TO FROM R-60-0.                         00156300
001590        PERFORM R-31-0 THRU R-31-9 11 TIMES.                       00156310
001600        GO TO BASE-60.                                             00156320
001610 R-31-0.                                                           00156330
001620        READ FILE-1 INTO REC1 AT END                               00156340
001630        DISPLAY MSGHDR DISPLAY MSGEOF                              00156350
001640        ADD 4 TO ERCTFL MOVE 'E' TO ERRORSW                        00156360
001650        GO TO R-31-9.                                              00156370
001660        IF REC-ID IS NOT EQUAL TO VER-ID                           00156380
001670        DISPLAY MSGHDR DISPLAY 'EXPECTED ' VER-ID ' FOUND ' REC-ID 00156390
001680        ADD 1 TO ERCTFL MOVE 'E' TO ERRORSW                        00156400
001690        MOVE REC-ID TO VER-ID.                                     00156410
001700        ADD 1 TO VER-NO.                                           00156420
001710 R-31-9.                                                           00156430
001720        IF ERCTFL IS GREATER THAN 3                                00156440
001730        GO TO BASE-70.                                             00156450
001740 SECTION-40 SECTION 40.                                            00156460
001750 W-40-0.                                                           00156470
001760        DISPLAY 'SECTION 40 WRITE'.                                00156480
001770        NOTE ENTERED BY PERFORM FROM BASE-SECTION.                 00156490
001780        PERFORM W-41-0 THRU W-41-9 17 TIMES.                       00156500
```

Figure 108. Sample Segmentation Program (Part 3 of 14)

```
001790 W-40-9.                                                            00156510
001800     EXIT.                                                          00156520
001810 W-41-0.                                                            00156530
001820     WRITE RECFD1 FROM REC1 INVALID KEY                             00156540
001830     DISPLAY MSGHDR                                                 00156550
001840     DISPLAY MSGEOX RECID                                           00156560
001850     MOVE 'E' TO ERRORSW                                            00156570
001860     GO TO BASE-50.                                                 00156580
001870     ADD 0001 TO REC-NO.                                            00156590
001880 W-41-9.                                                            00156600
001890     EXIT.                                                          00156610
001900 R-40-0.                                                            00156620
001910     DISPLAY 'SECTION 40 READ'.                                     00156630
001920     NOTE ENTERED BY PERFORM FROM BASE-60.                          00156640
001930     PERFORM R-41-0 THRU R-41-0 7 TIMES.                            00156650
001940     PERFORM R-20-0 THRU R-20-9.                                    00156660
001950 R-40-9.                                                            00156670
001960     EXIT.                                                          00156680
001970 R-41-0.                                                            00156690
001980     READ FILE-1 INTO REC1 AT END                                   00156700
001990     DISPLAY MSGHDR DISPLAY MSGEOF                                  00156710
002000     ADD 4 TO ERCTFL MOVE 'E' TO ERRORSW                            00156720
002010     GO TO R-41-9.                                                  00156730
002020     IF REC-ID IS NOT EQUAL TO VER-ID                               00156740
002030     DISPLAY MSGHDR DISPLAY 'EXPECTED ' VER-ID ' FOUND ' REC-ID     00156750
002040     ADD 1 TO ERCTFL MOVE 'E' TO ERRORSW                            00156760
002050     MOVE REC-ID TO VER-ID.                                         00156770
002060     ADD 1 TO VER-NO.                                               00156780
002070 R-41-9.                                                            00156790
002080     IF ERCTFL IS GREATER THAN 3                                    00156800
002090     GO TO BASE-70.                                                 00156810
002100 SECTION-60 SECTION 60.                                             00156820
002110 W-60-0.                                                            00156830
002120     DISPLAY 'SECTION 60 WRITE'.                                    00156840
002130     NOTE ENTERED BY PERFORM FROM BASE-SECTION.                     00156850
002140     PERFORM W-61-0 THRU W-61-9 13 TIMES.                           00156860
002150 W-60-9.                                                            00156870
002160     EXIT.                                                          00156880
002170 W-61-0.                                                            00156890
002180     WRITE RECFD1 FROM REC1 INVALID KEY                             00156900
002190     DISPLAY MSGHDR                                                 00156910
002200     DISPLAY MSGEOX RECID                                           00156920
002210     MOVE 'E' TO ERRORSW                                            00156930
002220     GO TO BASE-50.                                                 00156940
002230     ADD 0001 TO REC-NO.                                            00156950
002240 W-61-9.                                                            00156960
002250     EXIT.                                                          00156970
002260 R-60-0.                                                            00156980
002270     DISPLAY 'SECTION 60 READ'.                                     00156990
002280     NOTE ENTERED BY GO TO FROM BASE-50.                            00157000
002290     PERFORM R-61-0 THRU R-61-9 13 TIMES.                           00157010
002300     GO TO R-30-0.                                                  00157020
002310 R-61-0.                                                            00157030
002320     READ FILE-1 INTO REC1 AT END                                   00157040
002330     DISPLAY MSGHDR DISPLAY MSGEOF                                  00157050
002340     ADD 4 TO ERCTFL MOVE 'E' TO ERRORSW                            00157060
002350     GO TO R-61-9.                                                  00157070
```

Figure 108. Sample Segmentation Program (Part 4 of 14)

6

```
00229   002360        IF REC-ID IS NOT EQUAL TO VER-ID                                00157080
00230   002370        DISPLAY MSGHDR DISPLAY 'EXPECTED ' VER-ID ' FOUND ' REC-ID      00157090
00231   002380        ADD 1 TO ERCTFL MOVE 'E' TO ERRORSW                             00157100
00232   002390        MOVE REC-ID TO VER-ID.                                          00157110
00233   002400        ADD 1 TO VER-NO.                                                00157120
00234   002410 R-61-9.                                                                00157130
00235   002420        IF ERCTFL IS GREATER THAN 3                                     00157140
00236   002430        GO TO BASE-70.                                                  00157150
00237   002440 SECTION-80 SECTION 80.                                                 00157160
00238   002450 W-80-0.                                                                00157170
00239   002460        DISPLAY 'SECTION 80 WRITE'.                                     00157180
00240   002470        NOTE ENTERED BY PERFORM FROM BASE-SECTION.                      00157190
00241   002480        PERFORM  W-81-0  THRU W-81-9 7 TIMES.                           00157200
00242   002490        PERFORM W-20-0 THRU W-20-9.                                     00157210
00243   002500 W-80-9.                                                                00157220
00244   002510        EXIT.                                                           00157230
00245   002520 W-81-0.                                                                00157240
00246   002530        WRITE RECFD1 FROM REC1 INVALID KEY                              00157250
00247   002540        DISPLAY MSGHDR                                                  00157260
00248   002550        DISPLAY MSGEOX RECID                                            00157270
00249   002560        MOVE 'E' TO ERRORSW                                             00157280
00250   002570        GO TO BASE-50.                                                  00157290
00251   002580        ADD 0001 TO REC-NO.                                             00157300
00252   002590 W-81-9.                                                                00157310
00253   002600        EXIT.                                                           00157320
00254   002610 R-80-0.                                                                00157330
00255   002620        DISPLAY 'SECTION 80 READ'.                                      00157340
00256   002630        NOTE ENTERED BY PERFORM FROM BASE-50.                           00157350
00257   002640        PERFORM R-81-0 THRU R-81-9 17 TIMES.                            00157360
00258   002650 R-80-9.                                                                00157370
00259   002660        EXIT.                                                           00157380
00260   002670 R-81-0.                                                                00157390
00261   002680        READ FILE-1 INTO REC1 AT END                                    00157400
00262   002690        DISPLAY MSGHDR DISPLAY MSGEOF                                   00157410
00263   002700        ADD 4 TO ERCTFL MOVE 'E' TO ERRORSW                             00157420
00264   002710        GO TO R-81-9.                                                   00157430
00265   002720        IF REC-ID IS NOT EQUAL TO VER-ID                                00157440
00266   002730        DISPLAY MSGHDR DISPLAY 'EXPECTED ' VER-ID ' FOUND ' REC-ID      00157450
00267   002740        ADD 1 TO ERCTFL MOVE 'E' TO ERRORSW                             00157460
00268   002750        MOVE REC-ID TO VER-ID.                                          00157470
00269   002760        ADD 1 TO VER-NO.                                                00157480
00270   002770 R-81-8.
00271   002780        IF ERCTFL IS GREATER THAN 3                                     00157500
00272   002790        GO TO BASE-70.                                                  00157510
00273   002800 R-81-9.
00274   002810        EXIT.
```

7

INTRNL NAME	LVL	SOURCE NAME	BASE	DISPL	INTRNL NAME	DEFINITION	USAGE	R	O	Q	M
DNM=2-234	FD	FILE-1	DCB=01		DNM=2-234		QSAM				F
DNM=2-253	01	RECFD1	BL=1	000	DNM=2-253	DS 83C	DISP				
DNM=2-272	77	ERRORSW	BL=2	000	DNM=2-272	DS 1C	DISP				
DNM=2-292	77	ERCTFL	BL=2	001	DNM=2-292	DS 2C	DISP-NM				
DNM=2-308	77	MSGHDR	BL=2	003	DNM=2-308	DS 22C	DISP				
DNM=2-324	77	MSGEOX	BL=2	019	DNM=2-324	DS 36C	DISP				
DNM=2-340	77	MSGEOF	BL=2	03D	DNM=2-340	DS 37C	DISP				
DNM=2-356	77	MSGNEF	BL=2	062	DNM=2-356	DS 23C	DISP				
DNM=2-372	01	REC1	BL=2	080	DNM=2-372	DS 0CL83	GROUP				
DNM=2-389	02	REC-ID	BL=2	080	DNM=2-389	DS 0CL8	GROUP				
DNM=2-408	03	REC-HD	BL=2	080	DNM=2-408	DS 4C	DISP				
DNM=2-424	03	REC-NO	BL=2	084	DNM=2-424	DS 4C	DISP-NM				
DNM=2-440	02	FILLER	BL=2	088	DNM=2-440	DS 75C	DISP				
DNM=2-451	66	RECID	BL=2	080	DNM=2-451	DS 0CL8	GROUP				
DNM=2-469	01	VER-REC	BL=2	0D8	DNM=2-469	DS 0CL8	GROUP				
DNM=2-489	02	VER-ID	BL=2	0D8	DNM=2-489	DS 0CL8	GROUP				
DNM=3-000	03	VER-HD	BL=2	0D8	DNM=3-000	DS 4C	DISP				
DNM=3-016	03	VER-NO	BL=2	0DC	DNM=3-016	DS 4C	DISP-NM				

Figure 108. Sample Segmentation Program (Part 5 of 14)

MEMORY MAP

TGT	00218
SAVE AREA	00218
SWITCH	00260
TALLY	00264
SORT SAVE	00268
ENTRY-SAVE	0026C
SORT CORE SIZE	00270
RET CODE	00274
SORT RET	00276
WORKING CELLS	00278
SORT FILE SIZE	003A8
SORT MODE SIZE	003AC
PGT-VN TBL	003B0
TGT-VN TBL	003B4
VCONPTR	003B8
LENGTH OF VN TBL	003BC
LABEL RET	003BE
CURRENT PRIORITY	003BF
DBG R14SAVE	003C0
COBOL INDICATOR	003C4
A(INIT1)	003C8
DEBUG TABLE PTR	003CC
SUBCOM PTR	003D0
SORT DDNAME	003D4
UNUSED	003DC
DBG R11SAVE	003F0
UNUSED	003F4
PRBL1 CELL PTR	003F8
GENCBTBL PTR	003FC
UNUSED	00400
TA LENGTH	00401
UNUSED	00404
OVERFLOW CELLS	0040C
BL CELLS	0040C
DECBADR CELLS	00414
TEMP STORAGE	00418
TEMP STORAGE-2	00420
TEMP STORAGE-3	00420
TEMP STORAGE-4	00420
BLL CELLS	00420
VLC CELLS	00428
SBL CELLS	00428
INDEX CELLS	00428
SUBADR CELLS	00428
ONCTL CELLS	00428
PFMCTL CELLS	00428
PFMSAV CELLS	00450
VN CELLS	00498
SAVE AREA =2	004E0
SAVE AREA =3	004E0
XSASW CELLS	004E8
XSA CELLS	004E8

Figure 108. Sample Segmentation Program (Part 6 of 14)

```
LITERAL POOL (HEX)

005C8 (LIT+0)      1C4C3C00  4805EF48  00000005  000B0011  0007000D

     DISPLAY LITERALS (BCD)

005DC (LTL+20)   'START TEST SEG-SAMPLEEND TEST SEG-SAMPLE UNSUCCESSFUL RU'
00614 (LTL+76)   'NEND TEST SEG-SAMPLE SUCCESSFUL RUNSECTION 20 WRITESECTI'
0064C (LTL+132)  'ON 20 READEXPECTED  FOUND SECTION 30 WRITESECTION 30 REA'
00684 (LTL+188)  'DSECTION 40 WRITESECTION 40 READSECTION 60 WRITESECTION '
006BC (LTL+244)  '60 READSECTION 80 WRITESECTION 80 READ'

                    PGT                          00510

                 DEBUG LINKAGE AREA              00510
                 OVERFLOW CELLS                  00510
                 VIRTUAL CELLS                   00514
                 VIRTUAL EBCDIC NAMES            00528
                 PROCEDURE NAME CELLS            00550
                 GENERATED NAME CELLS            00550
                 DCB ADDRESS CELLS               0057C
                 VNI CELLS                       00580
                 LITERALS                        005C8
                 DISPLAY LITERALS                005DC
                 PROCEDURE BLOCK CELLS           006E4

 REGISTER ASSIGNMENT

  REG 6       BL =2
  REG 7       BL =1

WORKING-STORAGE STARTS AT LOCATION 00088 FOR A LENGTH OF 000E0.
```

Figure 108. Sample Segmentation Program (Part 7 of 14)

```
        ***************SEGMENT OF PTY 30****************

133     VERB  65
                 000000                       PN=016   EQU    *
134     VERB  66
                 000000                       PN=017   EQU    *
135     VERB  67
                 000000  58 F0 C 004                   L      15,004(0,12)         V(ILBODSP0)
                 000004  05 1F                         BALR   1,15
                 000006  0001                          DC     X'0001'
                 000008  10                            DC     X'10'
                 000009  000010                        DC     X'000010'
                 00000C  0C000156                      DC     X'0C000156'          LIT+158
                 000010  0000                          DC     X'0000'
                 000012  FFFF                          DC     X'FFFF'
137     VERB  68
                 000014  D2 03 D 258 D 294             MVC    258(4,13),294(13)    PSV=9          VN=06
                 00001A  41 00 B 02A                   LA     0,02A(0,11)          GN=019
                 00001E  50 00 D 294                   ST     0,294(0,13)          VN=06
                 000022  48 10 C 0C4                   LH     1,0C4(0,12)          LIT+12
                 000026  50 10 D 218                   ST     1,218(0,13)          PFM=3
                 00002A                       GN=019   EQU    *
                 00002A  58 E0 D 218                   L      14,218(0,13)         PFM=3
                 00002E  06 E0                         BCTR   14,0
                 000030  50 E0 D 218                   ST     14,218(0,13)         PFM=3
                 000034  12 EE                         LTR    14,14
                 000036  58 B0 C 1D8                   L      11,1D8(0,12)         PBL=2
                 00003A  47 40 B 048                   BC     4,048(0,11)          GN=054
                 00003E  58 F0 C 00C                   L      15,00C(0,12)         V(ILBOSGM1)
                 000042  05 EF                         BALR   14,15
                 000044  1E                            DC     X'1E'
                 000045  02                            DC     X'02'
                 000046  005A                          DC     X'005A'              PN=019
                 000048                       GN=054   EQU    *
                 000048  D2 03 D 294 D 258             MVC    294(4,13),258(13)    VN=06          PSV=9
138     VERB  69
                 00004E                       PN=018   EQU    *
139     VERB  70
140     VERB  71
                 00004E  58 00 D 290                   L      0,290(0,13)          VN=05
                 000052  58 F0 C 014                   L      15,014(0,12)         V(ILBOSGM0)
                 000056  05 EF                         BALR   14,15
                 000058  1E00                          DC     X'1E00'
                 00005A                       PN=019   EQU    *
141     VERB  72
                 00005A  D2 52 7 000 6 080             MVC    000(83,7),080(6)     DNM=2-253      DNM=2-372
                 000060  58 10 C 06C                   L      1,06C(0,12)          DCB=1
                 000064  18 21                         LR     2,1
                 000066  58 40 2 024                   L      4,024(0,2)
                 00006A  D2 02 4 019 C 04D             MVC    019(3,4),04D(12)                    GN=020+1
                 000070  58 10 C 06C                   L      1,06C(0,12)          DCB=1
                 000074  58 00 1 04C                   L      0,04C(0,1)
                 000078  58 F0 1 030                   L      15,030(0,1)
                 00007C  44 00 1 060                   EX     0,060(0,1)
                 000080  50 10 D 1F4                   ST     1,1F4(0,13)          BL =1
                 000084  58 70 D 1F4                   L      7,1F4(0,13)          BL =1
```

Figure 108. Sample Segmentation Program (Part 8 of 14)

```
                000088  96 01 4 01B                   OI    01B(4),X'01'
                00008C  47 F0 B 0CE                   BC    15,0CE(0,11)        GN=021
                000090                        GN=020  EQU   *
142     VERB  73
                000090  58 F0 C 004                   L     15,004(0,12)        V(ILBODSP0)
                000094  05 1F                         BALR  1,15
                000096  0001                          DC    X'0001'
                000098  00                            DC    X'00'
                000099  000016                        DC    X'000016'
                00009C  0D0001F8                      DC    X'0D0001F8'         BL =2
                0000A0  0003                          DC    X'0003'
                0000A2  FFFF                          DC    X'FFFF'
143     VERB  74
                0000A4  58 F0 C 004                   L     15,004(0,12)        V(ILBODSP0)
                0000A8  05 1F                         BALR  1,15
                0000AA  0001                          DC    X'0001'
                0000AC  00                            DC    X'00'
                0000AD  000024                        DC    X'000024'
                0000B0  0D0001F8                      DC    X'0D0001F8'         BL =2
                0000B4  0019                          DC    X'0019'
                0000B6  00                            DC    X'00'
                0000B7  000008                        DC    X'000008'
                0000BA  0D0001F8                      DC    X'0D0001F8'         BL =2
                0000BE  0080                          DC    X'0080'
                0000C0  FFFF                          DC    X'FFFF'
144     VERB  75
                0000C2  92 C5 6 000                   MVI   000(6),X'C5'        DNM=2-272
145     VERB  76
                0000C6  58 B0 C 1D4                   L     11,1D4(0,12)        PBL=1
                0000CA  47 F0 B 0DE                   BC    15,0DE(0,11)        PN=03
                0000CE                        GN=021  EQU   *
146     VERB  77
                0000CE  F2 73 D 200 6 084             PACK  200(8,13),084(4,6)  TS=01          DNM=2-424
                0000D4  FA 20 D 205 C 0B8             AP    205(3,13),0B8(1,12) TS=06          LIT+0
                0000DA  F3 32 6 084 D 205             UNPK  084(4,6),205(3,13)  DNM=2-424      TS=06
147     VERB  78
                0000E0                        PN=020  EQU   *
148     VERB  79
149     VERB  80
                0000E0  58 00 D 294                   L     0,294(0,13)         VN=06
                0000E4  58 F0 C 014                   L     15,014(0,12)        V(ILBOSGM0)
                0000E8  05 EF                         BALR  14,15
                0000EA  1E00                          DC    X'1E00'
                0000EC                        PN=021  EQU   *
150     VERB  81
                0000EC  58 F0 C 004                   L     15,004(0,12)        V(ILBODSP0)
                0000F0  05 1F                         BALR  1,15
                0000F2  0001                          DC    X'0001'
                0000F4  10                            DC    X'10'
                0000F5  00000F                        DC    X'00000F'
                0000F8  0C000166                      DC    X'0C000166'         LIT+174
                0000FC  0000                          DC    X'0000'
                0000FE  FFFF                          DC    X'FFFF'
152     VERB  82
                000100  D2 03 D 25C D 298             MVC   25C(4,13),298(13)   PSV=10         VN=07
                000106  41 00 B 116                   LA    0,116(0,11)         GN=022
                00010A  50 00 D 298                   ST    0,298(0,13)         VN=07
```

Figure 108. Sample Segmentation Program (Part 9 of 14)

```
                    00010E   48 10 C 0C4                  LH    1,0C4(0,12)           LIT+12
                    000112   50 10 D 21C                  ST    1,21C(0,13)           PFM=4
                    000116                       GN=022   EQU   *
                    000116   58 E0 D 21C                  L     14,21C(0,13)          PFM=4
                    00011A   06 E0                        BCTR  14,0
                    00011C   50 E0 D 21C                  ST    14,21C(0,13)          PFM=4
                    000120   12 EE                        LTR   14,14
                    000122   47 40 B 130                  BC    4,130(0,11)           GN=055
                    000126   58 F0 C 00C                  L     15,00C(0,12)          V(ILBOSGM1)
                    00012A   05 EF                        BALR  14,15
                    00012C   1E                           DC    X'1E'
                    00012D   02                           DC    X'02'
                    00012E   013E                         DC    X'013E'               PN=022
                    000130                       GN=055   EQU   *
                    000130   D2 03 D 298 D 25C            MVC   298(4,13),25C(13)     VN=07          PSV=10
153     VERB  83
                    000136   58 B0 C 1D4                  L     11,1D4(0,12)          PBL=1
                    00013A   47 F0 B 182                  BC    15,182(0,11)          PN=04
154     VERB  84
                    00013E                       PN=022   EQU   *
155     VERB  85
                    00013E   58 10 C 06C                  L     1,06C(0,12)           DCB=1
                    000142   18 21                        LR    2,1
                    000144   D2 02 2 021 C 051            MVC   021(3,2),051(12)                     GN=023+1
                    00014A   58 F0 1 030                  L     15,030(0,1)
                    00014E   05 EF                        BALR  14,15
                    000150   50 10 D 1F4                  ST    1,1F4(0,13)           BL =1
                    000154   58 70 D 1F4                  L     7,1F4(0,13)           BL =1
                    000158   D2 52 6 080 7 000            MVC   080(83,6),000(7)      DNM=2-372      DNM=2-253
                    00015E   47 F0 B 1A8                  BC    15,1A8(0,11)          GN=024
                    000162                       GN=023   EQU   *
156     VERB  86
                    000162   58 F0 C 004                  L     15,004(0,12)          V(ILBODSP0)
                    000166   05 1F                        BALR  1,15
                    000168   0001                         DC    X'0001'
                    00016A   00                           DC    X'00'
                    00016B   000016                       DC    X'000016'
                    00016E   0D0001F8                     DC    X'0D0001F8'           BL =2
                    000172   0003                         DC    X'0003'
                    000174   FFFF                         DC    X'FFFF'
156     VERB  87
                    000176   58 F0 C 004                  L     15,004(0,12)          V(ILBODSP0)
                    00017A   05 1F                        BALR  1,15
                    00017C   0001                         DC    X'0001'
                    00017E   00                           DC    X'00'
                    00017F   000025                       DC    X'000025'
                    000182   0D0001F8                     DC    X'0D0001F8'           BL =2
                    000186   003D                         DC    X'003D'
                    000188   FFFF                         DC    X'FFFF'
157     VERB  88
                    00018A   F2 71 D 200 6 001            PACK  200(8,13),001(2,6)    TS=01          DNM=2-292
                    000190   FA 10 D 206 C 0B9            AP    206(2,13),0B9(1,12)   TS=07          LIT+1
                    000196   F3 11 6 001 D 206            UNPK  001(2,6),206(2,13)    DNM=2-292      TS=07
157     VERB  89
                    00019C   92 C5 6 000                  MVI   000(6),X'C5'          DNM=2-272
158     VERB  90
                    0001A0   58 B0 C 1D8                  L     11,1D8(0,12)          PBL=2
```

Figure 108. Sample Segmentation Program (Part 10 of 14)

```
25

                    0001A4  47 F0 B 226                 BC    15,226(0,11)        PN=023
                    0001A8                      GN=024  EQU   *
159      VERB  91
                    0001A8  D5 07 6 080 6 0D8           CLC   080(8,6),0D8(6)     DNM=2-389      DNM=2-489
                    0001AE  47 80 B 214                 BC    8,214(0,11)         GN=025
160      VERB  92
                    0001B2  58 F0 C 004                 L     15,004(0,12)        V(ILBODSP0)
                    0001B6  05 1F                       BALR  1,15
                    0001B8  0001                        DC    X'0001'
                    0001BA  00                          DC    X'00'
                    0001BB  000016                      DC    X'000016'
                    0001BE  0D0001F8                    DC    X'0D0001F8'         BL =2
                    0001C2  0003                        DC    X'0003'
                    0001C4  FFFF                        DC    X'FFFF'
160      VERB  93
                    0001C6  58 F0 C 004                 L     15,004(0,12)        V(ILBODSP0)
                    0001CA  05 1F                       BALR  1,15
                    0001CC  0001                        DC    X'0001'
                    0001CE  10                          DC    X'10'
                    0001CF  000009                      DC    X'000009'
                    0001D2  0C000146                    DC    X'0C000146'         LIT+142
                    0001D6  0000                        DC    X'0000'
                    0001D8  00                          DC    X'00'
                    0001D9  000008                      DC    X'000008'
                    0001DC  0D0001F8                    DC    X'0D0001F8'         BL =2
                    0001E0  00D8                        DC    X'00D8'
                    0001E2  10                          DC    X'10'
                    0001E3  000007                      DC    X'000007'
                    0001E6  0C00014F                    DC    X'0C00014F'         LIT+151
                    0001EA  0000                        DC    X'0000'
                    0001EC  00                          DC    X'00'
                    0001ED  000008                      DC    X'000008'
                    0001F0  0D0001F8                    DC    X'0D0001F8'         BL =2
                    0001F4  0080                        DC    X'0080'
                    0001F6  FFFF                        DC    X'FFFF'
161      VERB  94
                    0001F8  F2 71 D 200 6 001           PACK  200(8,13),001(2,6)  TS=01          DNM=2-292
                    0001FE  FA 10 D 206 C 0B8           AP    206(2,13),0B8(1,12) TS=07          LIT+0
                    000204  F3 11 6 001 D 206           UNPK  001(2,6),206(2,13)  DNM=2-292      TS=07
161      VERB  95
                    00020A  92 C5 6 000                 MVI   000(6),X'C5'        DNM=2-272
162      VERB  96
                    00020E  D2 07 6 0D8 6 080           MVC   0D8(8,6),080(6)     DNM=2-489      DNM=2-389
                    000214                      GN=025  EQU   *
163      VERB  97
                    000214  F2 73 D 200 6 0DC           PACK  200(8,13),0DC(4,6)  TS=01          DNM=3-16
                    00021A  FA 20 D 205 C 0B8           AP    205(3,13),0B8(1,12) TS=06          LIT+0
                    000220  F3 32 6 0DC D 205           UNPK  0DC(4,6),205(3,13)  DNM=3-16       TS=06
164      VERB  98
                    000226                      PN=023  EQU   *
165      VERB  99
                    000226  F2 71 D 200 6 001           PACK  200(8,13),001(2,6)  TS=01          DNM=2-292
                    00022C  F9 10 D 206 C 0BA           CP    206(2,13),0BA(1,12) TS=07          LIT+2
                    000232  47 D0 B 23E                 BC    13,23E(0,11)        GN=026
166      VERB 100
                    000236  58 B0 C 1D4                 L     11,1D4(0,12)        PBL=1
                    00023A  47 F0 B 200                 BC    15,200(0,11)        PN=05

26

                    00023E                      GN=026  EQU   *
                    00023E  58 00 D 298                 L     0,298(0,13)         VN=07
                    000242  58 F0 C 014                 L     15,014(0,12)        V(ILBOSGM0)
                    000246  05 EF                       BALR  14,15
                    000248  1E00                        DC    X'1E00'
                    00024A  58 F0 C 00C                 L     15,00C(0,12)        V(ILBOSGM1)
                    00024E  05 EF                       BALR  14,15
                    000250  28                          DC    X'28'
                    000251  03                          DC    X'03'
                    000252  0000                        DC    X'0000'             PN=024
```

Figure 108. Sample Segmentation Program (Part 11 of 14)

CROSS-REFERENCE DICTIONARY

DEFN	REFERENCE								
000067									
000056	000123	000127	000131	000157	000161	000165	000193	000197	000201	000227
	000231	000235	000263	000267	000271					
000055	000090	000094	000109	000123	000127	000144	000157	000161	000178	000193
	000197	000214	000227	000231	000249	000263	000267			
000046	000076	000082	000083	000088	000092	000106	000121	000141	000155	000175
	000191	000211	000225	000246	000261					
000060	000122	000156	000192	000226	000262					
000058	000108	000143	000177	000213	000248					
000057	000089	000107	000122	000126	000142	000156	000160	000176	000192	000196
	000212	000226	000230	000247	000262	000266				
000062	000089									
000065										
000064	000125	000126	000128	000159	000160	000162	000195	000196	000198	000229
	000230	000232	000265	000266	000268					
000066	000111	000146	000180	000216	000251					
000053	000088	000106	000121	000141	000155	000175	000191	000211	000225	000246
	000261									
000068	000108	000143	000177	000213	000248					
000063	000088	000106	000121	000141	000155	000175	000191	000211	000225	000246
	000261									
000071										
000070	000125	000126	000128	000159	000160	000162	000195	000196	000198	000229
	000230	000232	000265	000266	000268					
000072	000129	000163	000199	000233	000269					
000069										

Figure 108. Sample Segmentation Program (Part 12 of 14)

PROCEDURE NAMES	DEFN	REFERENCE					
BASE-SECTION	000074						
BASE-50	000081	000110	000145	000179	000215	000250	
BASE-60	000086	000153					
BASE-70	000091	000088	000132	000166	000202	000236	000272
BASE-90	000093						
R-20-0	000114	000187					
R-20-9	000118	000187					
R-21-0	000120	000117					
R-21-9	000130	000117	000124				
R-30-0	000149	000223					
R-31-0	000154	000152					
R-31-9	000164	000152	000158				
R-40-0	000183	000087					
R-40-9	000188	000087					
R-41-0	000190	000186					
R-41-9	000200	000194					
R-60-0	000219	000085					
R-61-0	000224	000222					
R-61-9	000234	000222	000228				
R-80-0	000254	000084					
R-80-9	000258	000084					
R-81-0	000260	000257					
R-81-8	000270						
R-81-9	000273	000257	000264				
SECTION-20	000098						
SECTION-30	000133	000132					
SECTION-40	000167	000166					
SECTION-60	000203	000202					
SECTION-80	000237	000236					
W-20-0	000099	000242					
W-20-9	000103	000242					
W-21-0	000105	000102					
W-21-9	000112	000102					
W-30-0	000134	000078					
W-30-9	000138	000078					
W-31-0	000140	000137					
W-31-9	000147	000137					
W-40-0	000168	000080					
W-40-9	000172	000080					
W-41-0	000174	000171					
W-41-9	000181	000171					
W-60-0	000204	000079					
W-60-9	000208	000079					
W-61-0	000210	000207					
W-61-9	000217	000207					
W-80-0	000238	000077					
W-80-9	000243	000077					
W-81-0	000245	000241					
W-81-9	000252	000241					

Figure 108. Sample Segmentation Program (Part 13 of 14)

```
 F88-LEVEL LINKAGE EDITOR OPTIONS SPECIFIED LIST,OVLY
          DEFAULT OPTION(S) USED -   SIZE=(90112,12288)
IEW0000     INSERT SEG0SAMP
IEW0000     OVERLAY A
IEW0000     INSERT SEG0SA30
IEW0000     OVERLAY A
IEW0000     INSERT SEG0SA40
IEW0000     OVERLAY A
IEW0000     INSERT SEG0SA60
IEW0000     OVERLAY A
IEW0000     INSERT SEG0SA80
IEW0000     ENTRY SEG0SAMP
****RUN       DOES NOT EXIST BUT HAS BEEN ADDED TO DATA SET


START TEST SEG-SAMPLE
SECTION 80 WRITE
SECTION 20 WRITE
SECTION 30 WRITE
SECTION 60 WRITE
SECTION 40 WRITE
SECTION 80 READ
SECTION 60 READ
SECTION 30 READ
SECTION 40 READ
SECTION 20 READ
END TEST SEG-SAMPLE SUCCESSFUL RUN
```

Figure 108. Sample Segmentation Program (Part 14 of 14)

The IBM Operating System Checkpoint/ Restart feature is designed to be used with programs running for an extended period of time when interruptions may halt processing before the end of the job. The feature is available with both sequential and priority scheduling systems. The feature may be used when the programmer anticipates any type of interruption, i.e., interruptions caused by machine malfunctions, input/output errors, or intentional operator intervention, etc. It allows the interrupted program to be restarted at the job step or at a point other than at the beginning of the job step. The feature consists of two routines: Checkpoint and Restart.

The Checkpoint routine is invoked from the COBOL load module containing the user's program. It moves information stored in registers and in main storage into a checkpoint record at user-designated points during execution of the program. The programmer specifies these points using the COBOL RERUN clause in the Environment Division.

The Restart routine restarts an interrupted program. Restart can occur at the beginning of a job step, or at a checkpoint if a checkpoint record has been written. The checkpoint record will contain all information necessary to restart the program. Restart can be initiated at any time after the program was interrupted; that is, it may be run immediately after the interrupt has occurred, as an automatic restart, or at a later time convenient to the programmer, as a deferred restart.

The COBOL RERUN clause provides linkage to the system checkpoint routine. Hence, any cautions and restrictions on the use of the system Checkpoint/Restart feature also apply to the use of the RERUN clause.

The Checkpoint/Restart feature is fully described in the publication IBM OS Supervisor Services.

TAKING A CHECKPOINT

In order to initiate a checkpoint, the programmer uses job control statements and the COBOL RERUN clause. The programmer associates each RERUN clause with a particular COBOL file. The RERUN clause indicates that a checkpoint record is to be written onto a checkpoint data set whenever a specified number of records on that file are processed or when end of volume is reached while processing a file. The programmer decides when he wants the checkpoints taken as he codes the RERUN clause. The checkpoint records are written on the checkpoint data set defined by the DD statement and are referenced by system-name in the RERUN clause. The DD statement describes both a checkpoint data set and a checkpoint method.

Checkpoint records on ASCII-collated sorts can be taken, but the system-name indicating the checkpoint data set must not specify an ASCII file.

Note: If checkpoints are to be taken during a sorting operation, a DD statement called SORTCKPT must be added when the program is executed.

Checkpoint Methods

The programmer may elect to store single or multiple checkpoints.

Single: Only one checkpoint record exists at any given time. After the first checkpoint record is written, any succeeding checkpoint record overlays the previous one. This method is acceptable for most programs. It offers the advantage of saving space on the checkpoint data set and allows the programmer to restart his program at the latest checkpoint.

Multiple (multiple contiguous): Checkpoints are recorded and numbered sequentially. Each checkpoint is saved. This method is used when the programmer may wish to restart a program at a checkpoint other than the latest one taken.

DD STATEMENT FORMATS

The programmer records checkpoints on tape or direct access devices. Following are the DD formats to define checkpoint data sets.

For Tape:

```
//ddname  DD DSNAME=data-set-name,       X
//           VOLUME=SER=volser,          X
//           UNIT=deviceno,              X
                     NEW
//           DISP=(           ,PASS),    X
                     MOD
//           DCB=(TRTCH=C),LABEL=(,NL)
```

Note: The DCB parameter is necessary only for 7-track tape conversion; for 9-track tape it is not used.

For Mass Storage:

```
//ddname  DD DSNAME=data-set-name,       X
//           VOLUME=(PRIVATE,RETAIN,     X
//             SER=volser),              X
//           UNIT=deviceno,              X
//           SPACE=(subparms),           X
                     NEW
//           DISP=(           ,PASS),KEEP, X
                     MOD
```

where:

ddname
: is the same as the ddname portion of the system-name used in the COBOL RERUN clause to provide a link to the DD statement.

data-set-name
: is the name given to each particular data set used to write checkpoint records. This name identifies the checkpoint data set to the Restart procedure (see "Restarting a Program").

volser
: identifies the volume by serial number.

deviceno
: identifies the device. For tape it indicates the device number for 7-track or 9-track tape. For mass storage, it indicates the device number for disk or drum.

subparms
: specifies the amount of track space needed for the data set.

MOD
: is specified for the multiple contiguous checkpoint method.

NEW
: is specified for the single checkpoint method.

PASS
: is specified in order to prevent deletion of the data set at the successful completion of the job step unless it is the last step in the job. If it is the last step, the data set will be deleted with PASS.

KEEP
: is specified in order to keep the data set if the job step abnormally terminated and may be restarted.

The following listings are examples that define checkpoint data sets.

- To write single checkpoint records using tape:

```
//CHECKPT  DD DSNAME=CHECK1,                 X
//            VOLUME=SER=ND003,              X
//            UNIT=2400,DISP=(NEW,KEEP),     X
//            LABEL=(,NL)
              .
              .
              .
       ENVIRONMENT DIVISION.
              .
              .
              .
          RERUN ON UT-2400-S-CHECKPT EVERY
          5000 RECORDS OF ACCT-FILE.
```

- To write single checkpoint records using disk (note that more than one data set may share the same external-name):

```
//CHEK   DD   DSNAME=CHECK2,                 X
//            VOLUME=(PRIVATE,RETAIN,        X
//              SER=DB030,                   X
//            UNIT=2314,DISP=(NEW,KEEP),     X
//            SPACE=(TRK,300)
              .
              .
              .
       ENVIRONMENT DIVISION.
              .
              .
              .
          RERUN ON UT-2314-S-CHEK EVERY
          20000 RECORDS OF PAYCODE.
          RERUN ON UT-2314-S-CHEK EVERY
          30000 RECORD OF IN-FILE.
```

* To write multiple contiguous checkpoint records (on tape):

```
//CHEKPT  DD  DSNAME=CHECK3,                  X
//            VOLUME=SER=111111,              X
//            UNIT=2400,DISP=(MOD,PASS),      X
//            LABEL=(,NL)
              .
              .
              .
       ENVIRONMENT DIVISION.
              .
              .
              .
           RERUN ON UT-2400-S-CHEKPT EVERY
           10000 RECORDS OF PAY-FILE.
```

Note: A checkpoint data set must be sequential.

DESIGNING A CHECKPOINT

The programmer should design his checkpoints at critical points in his program so that data may be easily reconstructed. For example, in a program using mass storage files, changes to records in these files will replace previous information; thus the programmer should be sure he can identify previously processed records. Assume that a mass storage file contains loan records that periodically are updated for interest due. If a checkpoint is taken, records are updated, and then the program is interrupted, the records updated after the last checkpoint will be updated a second time in error unless the programmer controls this condition. (He may set up a date field for each record and update the date each time the record is processed. Then, after the restart, by investigating the date field he can determine whether or not the record was previously processed.) For efficient repositioning of a print file, the programmer should take checkpoints on that file only after printing the last line of a page. At system generation time, those ABEND codes for which the checkpoints are desired (DEFAULT) must be specified.

MESSAGES GENERATED DURING CHECKPOINT

The system checkpoint routine advises the operator of the status of the checkpoints taken by displaying informative messages on the console.

When a checkpoint has been successfully completed, the following message will be displayed:

```
[IHJ004I jobname (ddname,unit,volser)
        CHKPT checkid]
```

where checkid is the identification name of the checkpoint taken. Checkid is assigned by the control program as an 8-digit number. The first digit is the letter C, followed by a decimal number indicating the checkpoint. For example, checkid C0000004 indicates the fourth checkpoint taken in the job step.

RESTARTING A PROGRAM

The system Restart routine retrieves the information recorded in a checkpoint record, restores the contents of main storage and all registers.

The Restart routine can be initiated in one of two ways:

* Automatically at the time an interruption stopped the program
* At a later time as a deferred restart

The type of restart is determined by the RD parameter of the job control language.

RD Parameter

The RD parameter may appear on either the JOB or the EXEC statement. If coded on the JOB statement, the parameter overrides any RD parameters on the EXEC statement. If the programmer wishes to have his program restart automatically, he codes RD=R or RD=RNC. RD=R indicates that restart is to occur at the latest checkpoint. The programmer should specify the RERUN clause for at least one data set in his program in order to record checkpoints. If no checkpoint is taken prior to interruption, restart occurs at the beginning of the job step. RD=RNC indicates that no checkpoint is to be written and any restart will occur at the beginning of the job step. In this case, RERUN clauses are unnecessary; if any are present, they are ignored. If the RD parameter is omitted, the CHKPT macro instruction remains activated, and checkpoints may be taken during processing. If an interrupt occurs after the first checkpoint, automatic restart will occur. Thus, if the user does not want automatic restart, he should always include the RD parameter with a code of either RD=NR or RD=NC, both of which suppress the automatic restart procedure.

If the programmer wishes his program to be restarted on a deferred basis, he should code the RD parameter as RD=NR. This form of the parameter suppresses automatic restart but allows a checkpoint record to be written provided a RERUN clause has been specified. At restart time, the programmer may choose to restart his program at a checkpoint other than at the beginning of the job step.

The programmer may also elect to suppress both restart and writing checkpoints. By coding RD=NC, the programmer, in effect, is ignoring the features of the Checkpoint/Restart facility.

Automatic Restart

Automatic Restart occurs only at the latest checkpoint taken. (If no checkpoint was taken before interruption, Automatic Restart occurs at the beginning of the job step).

In order to restart automatically, a program must satisfy the following conditions.

- A program must request restart by using the RD parameter or by taking a checkpoint.
- An ABEND that terminated the job must return a code eligible to cause restart. (For further discussion on this requirement, see the publication IBM OS Supervisor Services.)
- The operator authorizes the restart, with the following procedure:

  The system displays the following message to request authorization of the restart:

  ```
  xxIEF225D SHOULD
       jobname.stepname.procstep
       RESTART [checkid]
  ```

  The operator must reply in the following form:

  ```
       REPLY xx, '{YES|NO|HOLD}'
  ```

  where YES authorizes restart, NO prevents restart, and HOLD defers restart until the operator issues a RELEASE command, at which time restart will occur. The HOLD option is applicable only in a multiprogramming environment.

Whenever automatic restart is to occur, the system will reposition all devices except unit-record machines.

Deferred Restart

Deferred restart may occur at any checkpoint, not necessarily the latest one taken.

The programmer requests a deferred restart by means of the RESTART parameter on the JOB card and a SYSCHK DD statement to identify the checkpoint data set. The formats for these statements are as follows:

```
//jobname JOB ,MSGLEVEL=1,                    X
//            RESTART=(request,[checkid])
//SYSCHK   DD DSNAME=data-set-name,           X
//            DISP=OLD,UNIT-deviceno,         X
//            VOLUME=SER=volser
```

where:

MSGLEVEL=1 (or MSGLEVEL=(1,y) where y is either 0 or 1)
: is required if restart is to occur in an MVT environment.

RESTART=(request,[checkid])
: identifies the particular checkpoint at which restart is to occur. Request may take one of the following forms:

  * to indicate restart at the beginning of the job

  stepname to indicate restart at the beginning of a job step

  stepname.procstep to indicate restart at a procedure step within the jobstep

checkid
: identifies the checkpoint where restart is to occur.

SYSCHK
: is the DDNAME used to identify a checkpoint data set to the control program. The SYSCHK DD statement must immediately precede the first EXEC statement of the resubmitted job, and must follow any JOBLIB statement.

data-set-name
: must be the same name that was used when the checkpoint was taken. It identifies the checkpoint data set

deviceno and volser
identify the device number and the volume serial number containing the checkpoint data set.

As an example illustrating the use of these job control statements, a restart of the GO step of a COBUCLG procedure, at checkpoint identifier (CHECKID) C0000003, might appear as follows:

```
//jobname JOB ,MSGLEVEL=1,                  X
//             RESTART=                     X
//               (stepname.GO,C0000003)
//SYSCHK   DD  DSNAME=CHEKPT,               X
//             DISP=OLD,UNIT=2400,          X
//             VOLUME=SER=111111
         .
         .
         .
```

{DD statements similar to original deck}

The Restart routine uses information from DD statements in the resubmitted job to reset files for use after restart; therefore, care should be taken with any DD statements that may affect the execution of the restarted job step. Attention should be paid to the following:

- During the original execution, a data set meant to be deleted at the end of a job step should conditionally be defined as PASS rather than DELETE in order to be available if an interruption forces a restart. If the restart is at the beginning of a step, a data set created in the original execution (defined as NEW on a DD statement) must be scratched prior to the restart. If the data set is not deleted, the DD statement must be changed to define it as OLD.

- At restart time, input data sets on cards should be positioned as they were at the time of the checkpoint. Input data sets on tape or direct access devices will be automatically repositioned by the system.

- At restart time, the EXEC statement parameters PGM and COND, and the DD statement parameters SUBALLOC and VOLUME=REF must not be used in steps following the restart step if they contain the form stepname or stepname.procstep referring to a step preceding the restart step. However, if these parameters are used, the preceding step referred to must be specified in the resubmitted deck.

When a deferred restart has been successfully completed, the system will display the following message on the console:

```
IHJ008I   jobname   RESTARTED
```

Control is then given to the user's program that executes in a normal manner.

CHECKPOINT/RESTART DATA SETS

If the RERUN clause was executed during the original execution of the processing program, checkpoint entries were written on a checkpoint data set. To resubmit a job for restart when execution is to be resumed at a particular checkpoint, an additional DD statement must be included. This DD statement describes the data set on which the checkpoint entry was written and it must have the ddname SYSCHK. The SYSCHK DD statement must immediately precede the first EXEC statement of the resubmitted job and must follow the DD statement named JOBLIB, if one is present.

For both deferred and automatic checkpoint/restart, if Direct SYSOUT Writer for the restarted job was active at the time the checkpoint was was taken, it must be available for the job to restart. For further information, see the publication <u>IBM OS Operator's Reference</u>, Order No. GC28-6691.

If the checkpoint data set is multivolume, the sequence number of the volume on which the checkpoint entry was written must be included in the VOLUME parameter. If the checkpoint data set is on a 7-track magnetic tape with nonstandard labels or no labels, the SYSCHK DD statement must contain DCB=(TRTCH=C,...).

Figure 109 illustrates a sequence of control statements for restarting a job.

```
//PAYROLL     JOB    MSGLEVEL=1,REGION=80K,RESTART=(STEP1,CHECKPT4)
//JOBLIB      DD     DSNAME=PRIV.LIB3,DISP=OLD
//SYSCHK      DD     DSNAME=CHKPTLIB,UNIT=2311,VOL=SER=456789,          X
//                   DISP=(OLD,KEEP)
//STEP1       EXEC   PGM=PROG4,TIME=5
```

Figure 109. Restarting a Job at a Specific Checkpoint Step

If a SYSCHK DD statement is present in a job and the JOB statement does not contain the RESTART parameter, the SYSCHK DD statement is ignored. If a RESTART parameter without the CHECKID subparameter (as in Figure 91) is included in a job, a SYSCHK DD statement must not appear before the first EXEC statement for a job.

Figure 110 illustrates the use of the RD parameter. Here, the RD parameter requests step restart for any abnormally terminated job step. The DD statement DDCKPNT defines a checkpoint data set. For this step, once a RERUN clause is executed, only automatic checkpoint restart can occur, unless a CHKPT cancel is issued.

Figure 111 illustrates those modifications that might be made to control statements before resubmitting the job for step restart. The job name has been changed to distinguish the original job from the restarted job. The RESTART parameter has been added to the JOB statement and indicates that restart is to begin with the first job step. The DD statement WORK originally assigned a conditional disposition of KEEP for this data set. If this step did not abnormally terminate during the original execution, the data set was deleted and no modifications need be made to this statement. If the step did abnormally terminate, the data set was kept. In this case, define a new data set as shown in Figure 111, or change the data set's status to OLD before resubmitting the job. A new data set has also been defined as the checkpoint data set.

Figure 112 illustrates those modifications that might be made to control statements before resubmitting the job for checkpoint restart.

```
//J1234     JOB   386,SMITH,MSGLEVEL=1,RD=R
//S1        EXEC  MYPROG
//INDATA    DD    DSNAME=INVENT,UNIT=2400,DISP=OLD,VOLUME=SER=91468,   X
//                LABEL=RETPD=14
//REPORT    DD    SYSOUT=A
//WORK      DD    DSNAME=T91468,DISP=(,,KEEP),UNIT=SYSDA,              X
//                SPACE=(3000,(5000,500)),VOLUME=(PRIVATE,RETAIN,,6)
//DDCKPNT   DD    UNIT=2400,DISP=(MOD,PASS,CATLG),DSNAME=C91468
```

Figure 110. Using the RD Parameter

```
//J3412     JOB   386,SMITH,MSGLEVEL=1,RD=R,RESTART=*
//S1        EXEC  MYPROG
//INDATA    DD    DSNAME=INVENT,UNIT=2400,DISP=OLD,VOLUME=SER=91468,   X
//                LABEL=RETPD=14
//REPORT    DD    SYSOUT=A
//WORK      DD    DSNAME=S91468,DISP=(,,KEEP),UNIT=SYSDA,              X
//                SPACE=(3000,(5000,500)),VOLUME=(PRIVATE,RETAIN,,6)
//DDCHKPNT  DD    UNIT=2400,DISP=(MOD,PASS,CATLG),DSNAME=R91468
```

Figure 111. Modifying Control Statements Before Resubmitting for Step Restart

```
//J3412      JOB    386,SMITH,MSGLEVEL=1,RD=R,RESTART=(*.C0000002)
//S1         EXEC   MYPROG
//SYSCHK     DD     DSNAME=C91468,DISP=OLD
//INDATA     DD     DSNAME=INVENT,UNIT=2400,DISP=OLD,                    X
//                  VOLUME=SER=91468,LABEL=RETPD=14
//REPORT     DD     SYSOUT=A
//WORK       DD     DSNAME=T91468,DISP=(,,KEEP),UNIT=SYSDA,              X
//                  SPACE=(3000,(5000,500)),VOLUME=(PRIVATE,RETAIN,,6)
//DDCKPNT    DD     UNIT=2400,DISP=(MOD,KEEP,CATLG),DSNAME=C91468
```

Figure 112. Modifying Control Statements Before Resubmitting for Checkpoint Restart

The job name has been changed to distinguish the original job from the restarted job. The RESTART parameter has been added to the JOB statement and indicates that restart is to begin with the first step at the checkpoint entry named C0000002. The DD statement DDCKPNT originally assigned a conditional disposition of CATLG for the checkpoint data set. If this step did not abnormally terminate during the original execution, the data set was kept. In this case, the SYSCHK DD statement must contain all of the information necessary to retrieve the checkpoint data set. If the job did abnormally terminate, the data set was cataloged. In this case, the only parameters required on the SYSCHK DD statement, as shown in Figure 112, are the DSNAME and DISP parameters.

A teleprocessing environment consists of a central computer[1], remote or local[2] stations, and communication lines between such stations and the central computer. Use of the Teleprocessing Feature (TP) enables the COBOL programmer to create device-independent programs for teleprocessing applications.

Teleprocessing applications require a special, user-written assembler-language program that controls the flow of data between the central computer and the remote stations. This message control program (MCP) also performs such additional tasks required only in a TP environment as dial-up, polling, (or contacting each remote station), and synchronization, as well as such device-dependent tasks as character translation and insertion of control characters.

The MCP consists of routines that identify the teleprocessing network to the operating system, establish line control between the computer and the various kinds of stations, and process messages in a way tailored to meet the needs of the user. A "message" is the data flowing either from a remote station to the central computer or from the central computer to a remote station. Each unit of data representing a message is terminated by a control character. An MCP is required in a teleprocessing system operating under TCAM.

Depending on the needs of the installation, one or more COBOL programs may be required to process the contents of the messages. An example of a job needing no application program is message switching, an operation consisting only of forwarding messages unaltered (except for such processing as the MCP may perform) to one or more other stations.

The MCP itself can perform limited processing (for example, examination of the first portion of a message to determine certain routine information and message code translation). Further, the MCP can obtain the time of day a message is received from a station and transmit this information to a COBOL program. It can also check the input messages to determine whether an error message should be sent to the designated station.

This section describes the flow of a single-segment message through a system operating under TCAM, from the time it is entered at the remote station to its transmission to a destination station. Figure 113 outlines the flow of a message segment through a TCAM system. The encircled numerals in the flow diagram correspond to the steps listed in the description that follows.

Because of the possible variety of both message types and destinations, it is often helpful for the user to precede the message "text" with a message "header" so that the user can transmit to the MCP information essential to handling the text. It is the user who determines which part of the message is the header and which part is the text.

Steps 1 and 2: The input message is prepared at the remote station and entered on the line. The message may be keyed in, or it may be entered from a card or tape reader. The originating station enters the message via a communication line, the transmission control unit, and the multiplexor channel.

Step 3: The message enters the central computer and is stored, together with the internally generated buffer prefix, in a main storage buffer. As message data fills the buffer, TCAM inserts the necessary control information in the prefix. Before the message characters are placed in the first buffer, TCAM may reserve space in the buffer for later insertion of the time, date, and sequence number for the message, and for the screen control character for the IBM 2260 and 2265 remote display complexes, if appropriate. Once a buffer is filled with the first segment of the message, the MCP controls the flow of the buffer through the teleprocessing network. The heart of the MCP consists of the message handlers (MH) constructed by the user to process messages from the various lines or line groups.

----

[1]A System/360, at least a Model 40, or a System/370 model with a minimum of 128K bytes of main storage.

[2]A station whose control unit is connected directly to a computer data channel by a local cable.

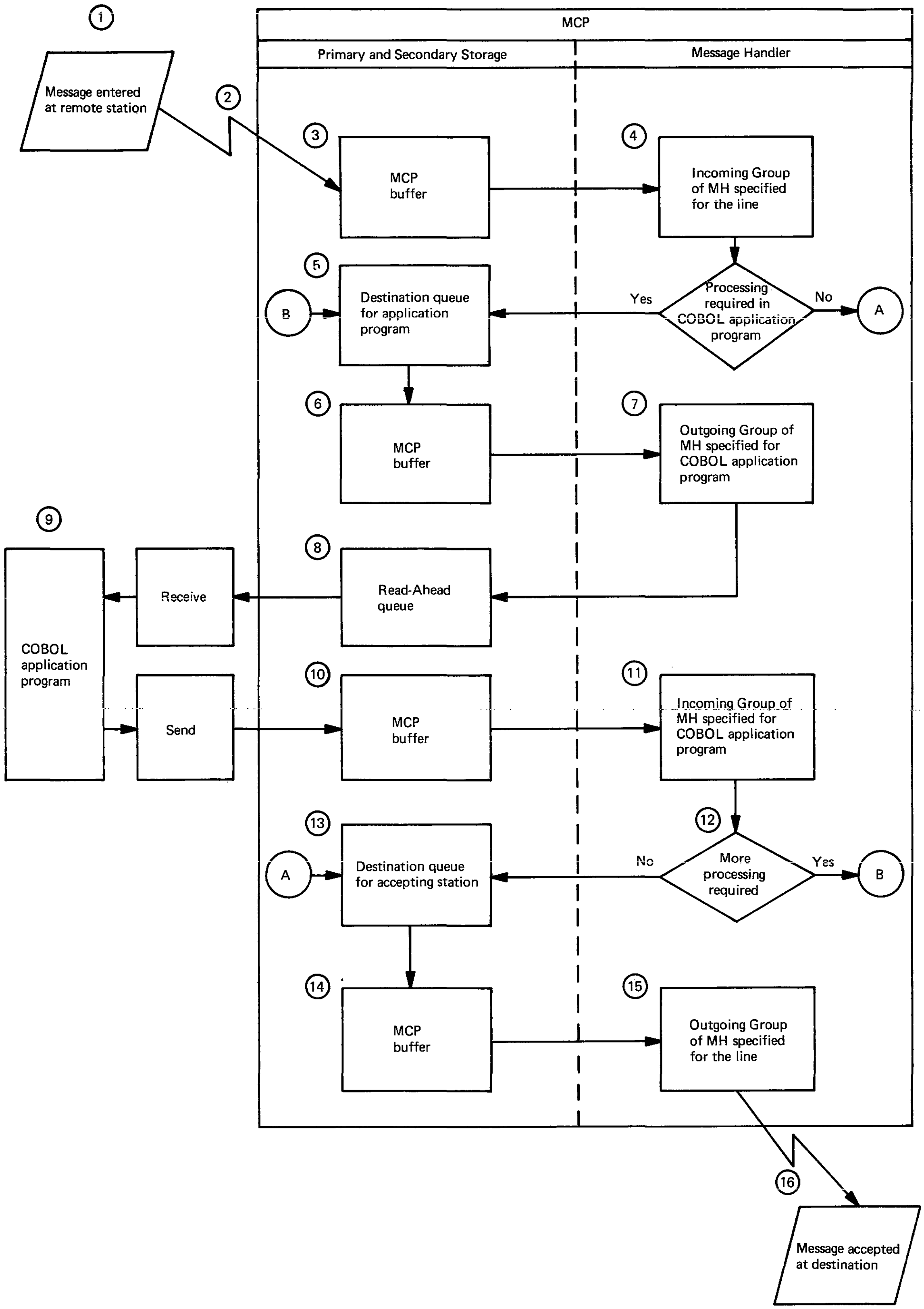


Figure 113. Message Flow Between Remote Stations and a COBOL Program

Step 4: The incoming message is routed to the incoming group of the MH specified for the line (by the MH= operand of the DCB macro for the line group in which the line is included). The message is passed, a buffer at a time, through the incoming group, which performs such user-selected functions on the message header as origin checking, and input sequence-number checking. Similarly, such functions may be performed for the message segment as translating the segment from line code to EBCDIC and causing an error message to be sent to the originating station when the incoming group detects any user-specified error in the segment. In performing its functions, the incoming group of the MH scans and processes header fields based on the relative order of the individual MH macro instructions. The incoming group then routes the message to the destination queue.

Step 5: After processing by the incoming group, the message is placed on a destination queue for either the COBOL program, for processing, or an accepting station. (If no message processing is necessary, the next action performed is that described in Step 13.) All messages requiring text processing are routed to the destination queue for the COBOL program that processes that type of message. The user controls this routing via the message header by placing the name of the destination queue for the COBOL program in a destination field of the message header or by MH macro instructions such as MSGTYPE that may be used to direct messages of a particular type to a particular queue.

Steps 6, 7, and 8: The message from a destination queue for a COBOL program is placed in a main-storage buffer; the outgoing group of an MH (the MH is created especially for the application program and is assigned to it by the MH= operand of the PCB macro in the MCP) places it on the read-ahead queue, a special queue that allows overlap of MCP and application program processing of messages queued for a particular destination.

Step 9: Each time the COBOL program issues a RECEIVE statement, TCAM passes message data from the read-ahead queue to a user-specified work area in the COBOL program. As the message data is moved to the work area, TCAM removes the header or text prefix from the buffer. After receiving the message data, the COBOL program processes it as required and then generates a response message, if any is to be returned to a station. The destination queues act as buffers between the COBOL TP program and the remote stations. Thus, the COBOL TP program can accept messages from MCP destination queuse and place these messages in MCP destination queues as if the queues were sequential files within a conventional COBOL program. (The sample COBOL program TESTTP1, shown in Figure 118, reads a sequential file and then sends each record to a destination queue, creating a TCAM data set for the COBOL TP program TESTTP2, shown in Figure 119, making it possible to test a COBOL TP program without terminals.)

Steps 10 and 11: When the COBOL program issues a SEND statement, TCAM moves the data from the work area into an MCP buffer before it is handled by the incoming group of the MH designed for the COBOL program. A header or text buffer prefix is created when data is moved to the buffer, as for other incoming messages. As the message data fills the buffer, TCAM inserts control information in the prefix field. The response message generated by an application program can be any user-selected length. After the buffer is filled, the message is handled by the incoming group of the MH assigned to the application program by the MH= operand of the PCB macro instruction that provides an interface between the MCP and the COBOL program.

Step 12: If further processing of the message is required in another application program, the message is queued for that destination (and Steps 5 through 11 are repeated). If however, no other application program processing is needed, the processed message is placed on the destination queue for an accepting station. The destination is that specified by the COBOL programmer in the file referenced by the SYMBOLIC DESTINATION clause of the output CD. It may be for an application program or a station.

Step 13: The destination queue for an accepting station, like the destination queue for an application program, is a part of the message queues data set. TCAM obtains message segments from the destination queue on a first-ended first-out (FEFO) basis within priority groups.

Steps 14 and 15: The message segment is placed in a buffer, and the outgoing group of the MH specified for the line processes the message. The MH performs such user-selected functions as converting the code of the message to the transmission code for the station (if necessary), inserting the time and data in the header, logging messages, and updating message counts. These operations are performed in the buffers that receive the message segments from the destination queue.

Step 16: TCAM transmits the message, minus the header and text prefixes, to the appropriate station.

WRITING A MESSAGE CONTROL PROGRAM

The COBOL programmer can write a message control program (MCP) designed specifically for his teleprocessing needs using telecommunications access methods (TCAM) macro instructions. Using a group of TCAM macro instructions, the user follows in general the coding requirements and restrictions of any other assembler-language macro instruction. Guidelines for writing an MCP are contained in the IBM OS Telecommunications Access Method (TCAM) Programmer's Guide and Reference Manual. The user must tailor these general statements to meet the needs of the installation.

The sample message control program that appears in Figure 114 in this chapter is a hypothetical program designed for specific COBOL applications. The needs of the user will undoubtedly vary from installation to installation. Nevertheless, the sample MCP together with the sample COBOL programs TESTTP1 and TESTTP2 (shown in Figures 118 and 119) can serve as an excellent example of COBOL programs and an MCP written for teleprocessing applications.

FUNCTIONS OF THE MESSAGE CONTROL PROGRAM

Depending on the requirements of the installation, the user can create an MCP to perform any of the following functions:

- Enable and disable communication lines
- Invite terminals to transmit messages
- Receive messages from terminals
- Dynamically assign buffers to incoming messages
- Handle messages on the basis of user-specified priorities
- Perform message-editing functions for incoming messages
- Determine the appropriate destination queue for a message and route the message to that queue
- Queue the message in the appropriate destination queue
- Place response messages generated by application programs on queues for subsequent transmission
- Retrieve messages from destination queues and prepare them for transmission to remote stations
- Perform message-editing functions for outgoing messages
- Take periodic checkpoints of the system
- Provide operator-to-system communications through system control terminals
- Initiate corrective action when an error or unusual condition is detected
- Cancel incoming messages containing errors
- Reroute messages with erroneous control information to a special queue
- Transmit error messages

However, not all of these functions are required of an MCP. Many of the optional TCAM macros allow the user to write an MCP that includes functions that would otherwise have to be executed by the COBOL program. There are, nevertheless, some functions the MCP must always provide and in so doing follow certain conventions. These requirements are discussed under "User Tasks."

USER TASKS

Guidelines for writing an MCP are contained in the publication IBM OS Access Method (TCAM) Programmer's Guide and Reference Manual. The user must tailor these general statements to meet the specific needs of his installation. For example, a message can be transmitted from one terminal to another, from a terminal to an application program, or from one application program to another. Moreover, the message may contain any one of several types of data.

Regardless of the specific requirements of the user, the MCP writer must always be concerned with four major tasks, as follows:

- Defining the core storage buffers used by the MCP for handling, queueing, and transferring message data between communication lines and queueing devices.

- Defining the data sets referred to by the MCP, and providing for their activation and deactivation.

- Defining the various terminal and line control areas used by the MCP (that is, the operating procedures and signals by which a teleprocessing system is controlled).

- Defining the message handlers (the sets of routines that examine and process control information in message headers, prepare message segments for forwarding to their destination, and route messages to their proper destination).

In carrying out each of these tasks, the user codes a variety of assembler-language macros in a specified order. Some of these macros must be included in every MCP; others the user specifies according to the needs of his installation. Required as well as optional macros are illustrated in the sample MCP given in Figure 114. The encircled numerals in the discussion that follows refer to sections of code that are similarly labeled in the figure.

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                              F15OCT70   5/03/72

                                    1 ***
                                    2 *
                                    3 * MESSAGE CONTROL PROGRAM
                                    4 *
000000                              5 MCP      CSECT
                                    6          PRINT NOGEN
                                    7 *
                                    8 * IN THE FOLLOWING MACRO--
                                    9 *        PROGID MAY BE OMITTED--IF USED, IT IS PLACED AT THE
                                   10 *          BEGINNING OF THE EXECUTABLE CODE IN THE MCP
                                   11 *        DISK=YES IS THE ASSUMED OPERAND--IF NO MESSAGE QUEUES DATA
                                   12 *          SETS ARE ON DISK,CODE DISK=NO
                                   13 *        CPB= USED IN READING FROM AND WRITING TO DISK--NEEDED IF
                                   14 *          DISK=YES--NO. DEPENDS ON NO. OF LINES, AMOUNT OF MESSAGE
                                   15 *          TRAFFIC AND SIZE OF BUFFER UNITS
                                   16 *        CIB=NO. OF COMMAND INPUT BLOCKS--BUFFER-LIKE AREAS USED TO
                                   17 *          CONTAIN OPERATOR CONTROL MESSAGES FROM SYSTEM CONSOLE--
                                   18 *          FREED ONCE A MESSAGE PROCESSED--2 ASSUMED AND MAX. IS 255
                                   19 *        PRIMARY=SYSCON--THIS IS ASSUMED AND SPECIFIES THE SYSTEM
                                   20 *          CONSOLE AS THE PRIMARY OPERATOR CONTROL TERMINAL FOR
                                   21 *          ENTERING AND ACCEPTING OPERATOR CONTROL MESSAGES--IF A
                                   22 *          TERMINAL IS SPECIFIED, IT MUST BE ON A NON-SWITCHED LINE
                                   23 *          AND BE ABLE TO ACCEPT AND ENTER MESSAGES
                                   24 *        CONTROL=--USED TO IDENTIFY OPERATOR CONTROL MESSAGES TO SYSTEM
                                   25 *          WHEN RECEIVED FROM OTHER THAN SYSTEM CONSOLE--0 IS DEFAULT
                                   26 *          AND IS VALID ONLY IF ALL OPERATOR COMMANDS ARE TO BE
                                   27 *          ENTERED FROM SYSTEM CONSOLE
                                   28 *        KEYLEN=--SIZE OF BUFFER UNIT--BETWEEN 33 AND 255--
                                   29 *          CAN ALSO SPECIFY BY UNITSZ= RATHER THAN KEYLEN=
                                   30 *        LNUNITS=--NO. OF BUFFER UNITS TO BE USED IN BUILDING BUFFERS
                                   31 *          FOR INCOMING AND OUTGOING MESSAGE SEGMENTS--IF TOO FEW ARE
                                   32 *          SPECIFIED, INCOMING MESSAGE DATA MAY BE LOST--TOO MANY
                                   33 *          WASTES STORAGE SPACE
                                   34 *        MSUNITS=--NEEDED IF HAVE MAIN STORAGE MESSAGE QUEUES DATA SET
                                   35 *          --NO. OF BUFFER UNITS ASSIGNED TO THIS DATA SET--IF NO DISK
                                   36 *          BACK-UP IS SPECIFIED, MESSAGE SEGMENTS MAY BE LOST IF NOT
                                   37 *          ENOUGH UNITS
                                   38 *        MSMAX=--PERCENTAGE OF UNITS IN MAIN STORAGE MESSAGE QUEUES
                                   39 *          DATA SET WANT USED BEFORE BIT IN ERROR RECORD SET--
                                   40 *          70 ASSUMED
                                   41 *        MSMIN=--PERCENTAGE OF UNITS IN MAIN STORAGE MESSAGE QUEUES
                                   42 *          DATA SET WANT UNUSED BEFORE BIT SET NOTIFYING NO LONGER
                                   43 *          CROWDED--MUST BE LESS THAN MSMAX--
                                   44 *          50 ASSUMED
                                   45 *          (NOTE--THIS BIT ALWAYS SET IF SPECIFIED PERCENTAGE OF UNITS
                                   46 *          UNUSED)
                                   47 *        DLQ=--OPTIONAL--USED TO SPECIFY A TERMINAL TO RECEIVE MESSAGES
                                   48 *          HAVING INVALID DESTINATIONS AS DETERMINED BY FORWARD MACRO
                                   49 *        INTVAL=--AN OPERATOR CONTROL MESSAGE TELLS TCAM TO ENTER THIS
                                   50 *          DELAY TO MINIMIZE UNPRODUCTIVE POLLING--WHEN ALL MULTIPOINT
                                   51 *          LINES ARE INACTIVE, THE INTERVAL COMMENCES--LINES TO
                                   52 *          SWITCHED STATIONS AND NONSWITCHED CONTENTION LINES LEFT
                                   53 *          ACTIVE--THE OPERATOR COMMAND IS A MODIFY COMMAND REFERRED
                                   54 *          TO AS 'INTERVAL'--THE NO. SPECIFIES THE NO. OF SECONDS
                                   55 *        STARTUP=-IF THIS OPERAND IS OMITTED, THE USER WILL BE GIVEN
```

Figure 114. A Message Control Program for Teleprocessing Application (Part 1 of 20)

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                              F15OCT70   5/03/72

                                    56 *             THE OPPORTUNITY TO SPECIFY IT AT INITIALIZATION TIME AND
                                    57 *             HE MAY ALSO CHANGE OTHER INTRO OPERANDS--CY MEANS ALWAYS
                                    58 *             A COLD START--W SPECIFIES A WARM START AFTER A QUICK OR
                                    59 *             FLUSH CLOSEDOWN AND A CONTINUATION AFTER A SYSTEM FAILURE
                                    60 *             --W INDICATES THE CONTINUATION RESTART WILL INCLUDE FULL
                                    61 *             SCANNING OF THE QUEUES--WY IS THE SAME AS W EXCEPT NO
                                    62 *             SCANNING OF THE QUEUES FOR ALREADY SENT MESSAGES IS DONE--
                                    63 *             --A CHECKPOINT DATA SET IS NEEDED FOR ANYTHING BUT A COLD
                                    64 *             START--ALSO, IF DD CARD FOR CHECKPOINT DATA SET SPECIFIES
                                    65 *             DISP=NEW, WILL GET A COLD START REGARDLESS
                                    66 *           OLTEST=IF DO NOT WISH ON-LINE TEST FACILITY--CODE 0
                                    67 *           FEATURE= THE DEFAULTS ARE DIAL, 2741, AND TIMER--SINCE WE DO
                                    68 *             NOT HAVE A 2741 TERMINAL, WE ARE CODING TO INDICATE THIS
                                    69 *           LINETYP= STSP SPECIFIES START-STOP LINES ONLY, BISC SPECIFIES
                                    70 *             BSC LINES ONLY, MINI SPECIFIES ALL TERMINALS ARE IBM 1050
                                    71 *             ON LEASED LINES, BOTH IS DEFAULT AND INDICATES ALL TYPES
                                    72 *             OF LINES ARE SUPPORTED--IF THE LINES IN THE SYSTEM DO NOT
                                    73 *             FALL UNDER THE 'BOTH' CATEGORY, SPACE IS SAVED BY CODING
                                    74 *             THIS OPERAND
                                    75 *           DTRACE -- PUT IN FOR TESTING ONLY                          TEST *
                                    76 *
                              (1)   77             INTRO PROGID=MCP,DISK=YES,CPB=10,CIB=2,PRIMARY=SYSCON,         X
                                                         CONTROL=TCAM,KEYLEN=100,LNUNITS=20,MSUNITS=50,MSMAX=75,  X
                                                         MSMIN=50,DLQ=T1,INTVAL=1200,STARTUP=W,OLTEST=0,          X
                                                         FEATURE=(DIAL,NO2741,TIMER),LINETYP=BOTH,                X
                                                         DTRACE=700
                                   309 *
                                   310 * TEST IF INTRO MACRO WORKED SUCCESSFULLY
000512 12FF                        311             LTR   15,15
000514 4780 D520          00528    312             BZ    OPENFILE            YES
                                   313 ABEND  ABEND 123,DUMP                 INTRO OR AN OPEN FAILED
                                   321 *
                                   322 * THE MESSAGE QUEUES DATA SET MUST BE OPENED FIRST IF IT RESIDES ON
                                   323 * DISK--A MAIN STORAGE MESSAGE QUEUES DATA SET IS NOT OPENED
                              (2)  324 OPENFILE OPEN  (MSGQ,(INOUT)) (a)
000532 9110 D738    00740          330             TM    MSGQ+48,X'10'       CHECK IF OPEN SUCCESSFUL
000536 47E0 D510          00518    331             BNO   ABEND               BRANCH IF NOT
                                   332 *
                                   333 * IF THE CHECKPOINT DATA SET IS USED, IT MUST BE OPENED NEXT
                                   334             OPEN  (CHKPT,(INOUT)) (b)
000546 9110 D764    0076C          340             TM    CHKPT+48,X'10'      CHECK IF OPEN SUCCESSFUL
00054A 47E0 D510          00518    341             BNO   ABEND               BRANCH IF NOT
                                   342 *
                                   343 * OPEN LINE GROUP DATA SETS--LINES WILL BE ACTIVATED SINCE IDLE NOT
                                   344 * SPECIFIED
                                   345 * NOTE--WE ARE NOT CHECKING FOR OPEN ERRORS FOR THE LINES--SINCE THERE
                                   346 * IS PROBABLY NO NEED TO STOP THE SYSTEM IF SOME OF THE LINES ARE NOT
                                   347 * WORKING--MESSAGES WILL BE PRINTED ON THE SYSTEM CONSOLE FOR LINES
                                   348 * THAT ARE NOT WORKING--
                                   349 * IF A LINE BECOMES OPERATIONAL DURING A RUN, IT CAN THEN BE STARTED
                                   350 * BY THE VARY COMMAND USED TO START A LINE WHICH IS OPENED AS IDLE
                                   351             OPEN  (LN1050,(INOUT),LNTWX,(INOUT)) (c)
                                   359 *
                                   360 * OPEN LOG DATA SET (d)
                                   361 *
```

Figure 114. A Message Control Program for Teleprocessing Applications (Part 2 of 20)

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT    SOURCE STATEMENT                                              F15OCT70   5/03/72

                                    362           OPEN  (MSGLOG,(OUTPUT))
00056A 9110 D7FC      00804         368           TM    MSGLOG+48,X'10'       CHECK IF OPEN SUCCESSFUL
00056E 47E0 D510            00518   369           BNO   ABEND                 BRANCH IF NOT
                                    370 *                                                                TEST***
                                    371           OPEN  (DUMP,(OUTPUT)) (e)   FOR SNAPS
00057E 9110 D854      0085C         377           TM    DUMP+48,X'10'         CHECK IF OPEN SUCCESSFUL
000582 47E0 D510            00518   378           BNO   ABEND                 BRANCH IF NOT
                                    379 *
                                    380 * ISSUE THE FOLLOWING BETWEEN THE OPENING AND CLOSING OF THE DATA SETS
                                 (3)381           READY
                                    398 *                                                                TEST***
                                    399           SNAP  DCB=DUMP,PDATA=ALL                                      *
                                    411 *
                                 (4)412 * CLOSE DATA SETS
                                    413           CLOSE (LN1050,,LNTWX) (a)   LINE GROUP DATA SETS
                                    421 *                                                                TEST***
                                    422           CLOSE (DUMP,DISP) (b)       SNAP DATA SET                     *
                                    428 *
                                    429           CLOSE (MSGLOG,DISP) (c)     LOG DATA SET
                                    435 *
                                    436 * ALWAYS CLOSE CHECKPOINT DATA SET NEXT TO LAST
                                    437           CLOSE (CHKPT,DISP) (d)
                                    443 *
                                    444 * THE MESSAGE QUEUES DATA SET MUST ALWAYS BE CLOSED LAST
                                    445           CLOSE (MSGQ,DISP) (e)
                                    451 *
                                    452 * RETURN TO OS SUPERVISOR
00060E 58DD 0004            00004   453           L     13,4(13)          PICK UP ADDRESS OF SYSTEM SAVE AREA SAVED
                                    454 *                                 IN IEDSAVE1--ADDRESS OF IEDSAVE1 WAS PUT
                                    455 *                                 IN REG. 13 WHICH WAS MADE BASE REGISTER
                                 (5)456           RETURN  (14,12),RC=0
                                    460 *
```

Figure 114. A Message Control Program for Teleprocessing Applications (Part 3 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                              F15OCT70   5/03/72

                                  462 ****
                                  463 *
                                  464 * DATA DEFINITIONS--PROCESS CONTROL BLOCKS AND DATA CONTROL BLOCKS
                                  465 *
                                  466 ****
                                  467 *
                                  468 * PCB--PROCESS CONTROL BLOCK--USED TO COMMUNICATE BETWEEN THE MCP
                                  469 * AND AN APPLICATION PROGRAM--
                                  470 * ONE PCB IS NEEDED FOR EACH ACTIVE APPLICATION PROGRAM
                                  471 *
                                  472 * IN THE FOLLOWING MACRO--
                                  473 *   MH= GIVES THE SYMBOLIC ADDRESS OF THE MESSAGE HANDLER FOR THIS
                                  474 *     APPLICATION PROGRAM
                                  475 *   BUFSIZE= SPECIFIES SIZE OF BUFFERS TO HANDLE MESSAGES FOR
                                  476 *     APPLICATION PROGRAM
                                  477 *   BUFIN= INITIAL NO. OF BUFFERS INTO WHICH USERS WRITE WORK AREA
                                  478 *     EMPTIED--OPTIMUM NO. IS ENOUGH FOR ALL OF WORK AREA--BETWEEN
                                  479 *     2 AND 15--2 ASSUMED
                                  480 *   BUFOUT= INITIAL NO. OF BUFFERS THAT MAY BE FILLED IN ANTICIPATION
                                  481 *     OF A READ--BETWEEN 2 AND 15--2 ASSUMED
                                  482 *   RESERVE=NO. OF BYTES TO RESERVE FOR INSERTION OF CHARS. BY DATETIME
                                  483 *     AND SEQUENCE MACROS FOR MESSAGES COMING FROM APPLICATION PROGRAMS
                                  484 *   DATE=YES--THIS IS NEEDED FOR ALL PCB ENTRIES FOR A COBOL PROGRAM.
                                  485 *     THIS WILL MAKE THE DATE AND TIME AVAILABLE SO IT MAY BE PLACED
                                  486 *     IN THE COBOL PROGRAM INPUT CD--(IT IS ALSO NEEDED ON AN INPUT
                                  487 *     TPROCESS ENTRY)
                                  488 *
                                  489 * PROCESS CONTROL BLOCK FOR COBOL PROGRAM RUNNING WITH TERMINALS
                                  490 *         a
                              (6) 491 PCBLK     PCB  MH=MHTRMAPP,BUFSIZE=100,BUFIN=2,BUFOUT=5,RESERVE=21,      X
                                                     DATE=YES
                                  526 *
                                  527 * PROCESS CONTROL BLOCK FOR COBOL PROGRAMS THAT SIMULATE TERMINAL
                                  528 * INPUT DATA--USED FOR TESTING WITHOUT TERMINALS
                                  529 *         b
                                  530 PCBLK1    PCB  MH=MHAPPAPP,BUFSIZE=100,BUFIN=2,BUFOUT=5,DATE=YES
                                  563 *
                                  564 * PROCESS CONTROL BLOCK FOR COBOL PROGRAMS TESTING MESSAGES SENT TO
                                  565 * DESTINATIONS DEFINED BY A QUEUE STRUCTURE
                                  566 *
                                  567 * IT USES THE SAME MH THAT PCKBLK1 USES
                                  568 *         c
                                  569 PCBLK2    PCB  MH=MHAPPAPP,BUFSIZE=100,BUFIN=2,BUFOUT=5,DATE=YES
                                  602 *
                                  603 * DCBS
                                  604 *
                                  605 * DCB FOR MESSAGE QUEUES DATA SET
                                  606 *  IN THE FOLLOWING MACRO--
                                  607 *        OPTCD=R SPECIFIES REUSABLE DISK--IF NON-REUSABLE,SPECIFY L
                                  608 *        THRESH= SHOULD PROBABLY BE USED IF NON-REUSABLE DISK--
                                  609 *          SPECIFIES PERCENTAGE OF RECORDS TO BE USED BEFORE A FLUSH
                                  610 *          CLOSEDOWN INITIATED--A CERTAIN PERCENTAGE ASSUMED
                              (7) 611 MSGQ      DCB  DSORG=TQ,MACRF=(G,P),DDNAME=QFILE,OPTCD=R
                                  643 *         a
                                  644 * DCB FOR THE CHECKPOINT DATA SET
```

Figure 114. A Message Control Program for Teleprocessing Applications (Part 4 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                          F15OCT70    5/03/72

                                 645 CHKPT      DCB   DSORG=TQ,MACRF=(G,P),DDNAME=CFILE,OPTCD=C
                                 677 *             b
                                 678 * DCB FOR THE 1050 LINE GROUP
                                 679 *    IN THE FOLLOWING MACRO--
                                 680 *          CPRI=R INDICATES THAT RECEIVE HAS PRIORITY OVER SENDING--
                                 681 *               S INDICATES THAT SENDING HAS PRIORITY OVER RECEIVING--
                                 682 *               E INDICATES EQUAL PRIORITY--
                                 683 *            FOR SWITCHED LINES, S MUST BE SPECIFIED
                                 684 *          BUFIN=NO. OF BUFFERS TO ASSIGN INITIALLY FOR RECEIVING FOR
                                 685 *            EACH LINE--1 ASSUMED--15 MAXIMUM
                                 686 *          BUFOUT=NO. OF BUFFERS TO ASSIGN INITIALLY FOR SENDING FOR
                                 687 *            EACH LINE--2 ASSUMED--15 MAXIMUM
                                 688 *          BUFMAX=MAX. NO. OF BUFFERS TO BE USED FOR DATA TRANSFER FOR
                                 689 *            EACH LINE IN LINE GROUP--NO LESS THAN LARGER OF BUFIN AND
                                 690 *            BUFOUT--15 MAXIMUM
                                 691 *          BUFSIZE=BUFFER SIZE IN BYTES USED FOR ALL LINES IN THIS LINE
                                 692 *            GROUP--SIZE SHOULD BE A MULTIPLE OF THE BUFFER UNIT SIZE
                                 693 *            SPECIFIED IN KEYLEN= OPERAND OF INTRO MACRO-- (MAY BE
                                 694 *            OVERRIDDEN ON A STATION BASIS BY BUFSIZE= OPERAND OF THE
                                 695 *            TERMINAL MACRO)
                                 696 *          INVLIST=NAMES OF INVITATION LISTS FOR LINES OF LINE GROUP
                                 697 *            --INVITATION LIST NAMES ARE SPECIFIED ACCORDING TO THE
                                 698 *            ASCENDING RELATIVE LINE NOS. OF THE LINES IN THE GROUP
                                 699 *          MH=ADDRESS OF MESSAGE HANDLER
                                 700 *          PCI=SPECIFIES IF AND HOW A PROGRAM-CONTROLLED INTERRUPTION
                                 701 *            TO BE USED FOR BUFFER ALLOCATION AND DEALLOCATION--1ST
                                 702 *            SUBOPERAND REFERS TO RECEIVING AND 2ND TO SENDING--
                                 703 *            N SPECIFIES NO PCIS--R SPECIFIES AFTER 1ST BUFFER, COMPLETED
                                 704 *            BUFFER DEALLOCATED--A IS ASSUMED AND  SPECIFIES AFTER 1ST
                                 705 *            BUFFER, COMPLETED BUFFER DEALLOCATED AND ANOTHER BUFFER IS
                                 706 *            ALLOCATED
                                 707 *          RESERVE=NO. OF BYTES TO RESERVE FOR INSERTION OF CHARS. BY
                                 708 *            DATETIME AND SEQUENCE MACROS
                                 709 *          TRANS=TRANSLATION TABLE
                                 710 *          SCT=SPECIAL CHARACTERS TABLE
                                 711 *          (IF CPRI=R AND NON-SWITCHED LINE, NEED INTVL= OR NO MESSAGES
                                 712 *          ARE SENT--INTVL=NO. OF SECONDS TO DELAY AFTER PASS THRU
                                 713 *          INVITATION LIST--NO LARGER THAN 255--TOO SHORT A DELAY CAUSES
                                 714 *          MESSAGES TO ACCUMULATE)
                                 715 *                 c
                                 716 LN1050     DCB   DSORG=TX,MACRF=(G,P),CPRI=S,DDNAME=LN1,BUFIN=2,           X
                                                      BUFOUT=4,BUFMAX=4,BUFSIZE=100,INVLIST=(LIST1050),         X
                                                      MH=MH1050,PCI=(A,A),RESERVE=21,TRANS=105F,SCT=105F
                                 753 *
                                 754 * DCB FOR THE TWX LINE--SEE DESCRIPTION OF OPERANDS BEFORE DCB FOR
                                 755 * 1050--LN1050
                                 756 *                 d
                                 757 LNTWX      DCB   DSORG=TX,MACRF=(G,P),CPRI=S,DDNAME=LN2,BUFIN=2,           X
                                                      BUFOUT=4,BUFMAX=4,BUFSIZE=100,INVLIST=(LISTTWX),          X
                                                      MH=MHTWX,PCI=(A,A),RESERVE=21,TRANS=TTYC,SCT=TTYC
                                 794 *
                                 795 * DCB FOR LOG DATA SET
                                 796 *    IN THE FOLLOWING MACRO--                                             *
                                 797 *          BLKSIZE=--THE VALUE SHOULD BE THE SAME AS IN KEYLEN OPERAND OF
                                 798 *            INTRO MACRO                                                  *
```



```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                          F15OCT70    5/03/72

                                 799 *          NCP=--MAX. NO. OF BUFFER UNITS THAT MAY APPEAR IN A BUFFER     *
                                 800 *                 e
                                 801 MSGLOG     DCB   DSORG=PS,MACRF=(W),DDNAME=LOGFILE,BLKSIZE=100,RECFM=F,   *X
                                                      NCP=2                                                     *
                                 852 *                                                                  TEST***
                             8   853 * DCB FOR SNAPS                                                           *
                                 854 DUMP       DCB   DSORG=PS,RECFM=VBA,MACRF=(W),LRECL=125,DDNAME=LRDUMP,    *X
                                                      BLKSIZE=882                                               *
```

Figure 114. A Message Control Program for Teleprocessing Applications (Part 5 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                            F15OCT70    5/03/72

                                  906 ****
                                  907 *
                                  908 * TERMINAL AND LINE CONTROL--DEFINES  TERMINAL TABLE ENTRIES AND THE
                                  909 * INVITATION LISTS FOR EACH LINE
                                  910 *
                                  911 ****
                                  912 *
                                  913 * DEFINE THE TERMINAL TABLE
                                  914 *          LAST= NAME OF LAST ENTRY IN TABLE
                                  915 *          MAXLEN= NUMBER OF CHARACTERS IN LONGEST NAME
                                  916 *
                              8   917            TTABLE  LAST=D1,MAXLEN=5
                                  945 *                                                                       NOTE*
                                  946 * IF ANY OPTION MACROS ARE NEEDED, THEY GO HERE--DATA GOES IN ENTRIES *
                                  947 * USING THE OPDATA= OPERAND OF THE TERMINAL OR TPROCESS ENTRIES
                                  948 *
                                  949 * ENTRY FOR 1050 TERMINAL
                                  950 *   IN THE FOLLOWING MACRO--
                                  951 *          QBY= T SPECIFIES THAT OUTGOING MESSAGES ARE TO BE QUEUED BY
                                  952 *            TERMINAL--USE L IF BY LINE
                                  953 *            --MUST QUEUE BY TERMINAL IF A SWITCHED STATION OR A
                                  954 *            BUFFERED TERMINAL
                                  955 *          DCB= DCBNAME FOR LINE
                                  956 *          RLN=RELATIVE LINE NO. WITHIN THE LINE GROUP OF THIS LINE
                                  957 *          TERM=SPECIFIES TYPE OF TERMINAL
                                  958 *          QUEUES=MR SPECIFIES MESSAGE QUEUES KEPT IN MAIN STORAGE WITH
                                  959 *            BACKUP ON REUSABLE DISK
                                  960 *          ADDR=6213 IS A9 IN 1050 CODE--USED WHEN COMPUTER HAS MESSAGE
                                  961 *            TO SEND--9 IS CODE FOR ANY OUTPUT DEVICE
                                  962 *          ALTDEST=IS NEEDED BECAUSE THIS IS REUSABLE DISK--NEEDED SO
                                  963 *            MESSAGE IS NOT DISCARDED AT ZONE CHANGEOVER
                                  964 *          NTBLKSZ= THE NO. OF CHARS. BETWEEN INSERTION OF EOB CHARS.
                                  965 *            IN OUTPUT MSG. WHEN MSGFORM CODED IN OUTHDR
                                  966 *                   a
                              9   967 T1         TERMINAL QBY=T,DCB=LN1050,RLN=1,TERM=1050,QUEUES=MR,           X
                                                       ADDR=6213,ALTDEST=T1,NTBLKSZ=(120)
                                 1001 *
                                 1002 * DEFINE ENTRY FOR THE SWITCHED TWX LINE WHICH CAN BE USED BEFORE AN
                                 1003 * ORIGIN MACRO IS ISSUED TO IDENTIFY THE STATION
                                 1004 *          UTERM=YES IDENTIFIES THIS AS SUCH AN ENTRY
                                 1005 * THIS MACRO MUST PRECEDE ALL TERMINAL MACROS FOR STATIONS ON LINE
                                 1006 * IN THE FOLLOWING MACRO--
                                 1007 *          ALWAYS SPECIFY DCB NAME,RELATIVE LINE NO., TERMINAL TYPE,
                                 1008 *            AND QUEUES
                                 1009 *          --ADDR= MIGHT BE CODED IF STATION HAD ADDRESSING CHARS.--IF
                                 1010 *          USED, ALL STATIONS ON LINE MUST HAVE IDENTICAL ADDRESSING
                                 1011 *          CHARACTERS
                                 1012 *                   b
                                 1013 T2A        TERMINAL UTERM=YES,DCB=LNTWX,RLN=1,TERM=3335,QUEUES=MR
                                 1041 *
                                 1042 * TERMINAL ENTRY FOR TWX TERMINAL--SEE DESCRIPTION OF MOST OF OPERANDS
                                 1043 * PRECEDING TERMINAL MACRO FOR 1050
                                 1044 * IN ADDITION--
                                 1045 *          DIALNO= SPECIFIES TELEPHONE NO. OF STATION AND MUST BE
                                 1046 *            SPECIFIED FOR SWITCHED STATIONS--CODE 'NONE' IF NO AUTO
```

Figure 114. A Message Control Program for Teleprocessing Applications (Part 6 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                              F15OCT70    5/03/72
                                 1047 *          CALL FEATURE
                                 1048 *        ADDR= IS NOT GIVEN SINCE THIS STATION IS ON A SWITCHED LINE
                                 1049 *        NTBLKSZ IS NOT USED FOR TWX TERMINALS
                                 1050 *        CINTVL= NO. OF SECONDS BEFORE COMPUTER SHOULD CALL STATION
                                 1051 *          --NOT NEEDED IF NO AUTO CALL FEATURE
                                 1052 *                      c
                                 1053 T2       TERMINAL  QBY=T,DCB=LNTWX,RLN=1,TERM=3335,QUEUES=MR,              X
                                                     DIALNO=NONE,ALTDEST=T2
                                 1076 *
                                 1077 * TPROCESS ENTRIES
                                 1078 *
                                 1079 * IN THE FOLLOWING MACROS--
                                 1080 *        PCB= NAME OF PROCESS CONTROL BLOCK--ALL TPROCESS
                                 1081 *          ENTRIES FOR THE SAME APPLICATION PROGRAM MUST HAVE THE SAME
                                 1082 *          PCB
                                 1083 *        QUEUES= IS THE SAME AS FOR A TERMINAL MACRO--HOWEVER, BY
                                 1084 *          OMITTING, USER SPECIFIES THAT THIS ENTRY IS USED FOR PUTS &
                                 1085 *          WRITES FROM APPLICATION PROGRAM
                                 1086 *        ALTDEST= FOR OUTPUT,GIVES WHERE REPLIES TO OPERATOR MSGS. SENT
                                 1087 *          IF WERE ENTERED FROM AN APPLICATION PROGRAM--NOT APPLICABLE
                                 1088 *          TO COBOL--
                                 1089 *          ONLY NEEDED FOR INPUT QUEUES IF REUSABLE DISK QUEUEING
                                 1090 *        RECDEL= SPECIFIES CHARACTER USED TO DENOTE END OF RECORD
                                 1091 *        DATE=YES--THIS IS NEEDED FOR ALL INPUT TPROCESS ENTRIES
                                 1092 *          FOR A COBOL PROGRAM.  THIS WILL MAKE THE DATE AND TIME
                                 1093 *          AVAILABLE SO IT MAY BE PLACED IN THE COBOL PROGRAM
                                 1094 *          INPUT CD.
                                 1095 *
                                 1096 * INPUT TPROCESS ENTRY FOR COBOL PROGRAM RUNNING WITH TERMINALS
                              10 1097 *
                                 1098 PIN      TPROCESS  PCB=PCBLK,QUEUES=MR,ALTDEST=PIN,RECDEL=FF,DATE=YES   a
                                 1127 *
                                 1128 * OUTPUT TPROCESS ENTRY FOR COBOL PROGRAM RUNNING WITH TERMINALS
                                 1129 *
                                 1130 POUT     TPROCESS  PCB=PCBLK,RECDEL=FF
                                 1156 *
                                 1157 * THE FOLLOWING TWO INPUT TPROCESS ENTRIES ARE FOR COBOL PROGRAMS
                                 1158 * THAT SIMULATE TERMINAL INPUT DATA--USED FOR TESTING WITHOUT
                                 1159 * TERMINALS
                                 1160 *
                                 1161 P1       TPROCESS  PCB=PCBLK1,QUEUES=MR,ALTDEST=P1,RECDEL=FF,DATE=YES  b
                                 1187 *
                                 1188 P2       TPROCESS  PCB=PCBLK1,QUEUES=MR,ALTDEST=P2,RECDEL=FF,DATE=YES  c
                                 1214 *
                                 1215 * OUTPUT TPROCESS ENTRY FOR THESE COBOL PROGRAMS
                                 1216 *
                                 1217 POUT1    TPROCESS  PCB=PCBLK1,RECDEL=FF   d
                                 1243 *
                                 1244 * THE FOLLOWING SIX INPUT TPROCESS ENTRIES ARE FOR COBOL QUEUE
                                 1245 * STRUCTURE TEST PROGRAMS
                                 1246 *
                                 1247 PQ1      TPROCESS  PCB=PCBLK2,QUEUES=MR,ALTDEST=PQ1,RECDEL=FF,DATE=YES  e
                                 1273 *
                                 1274 PQ2      TPROCESS  PCB=PCBLK2,QUEUES=MR,ALTDEST=PQ2,RECDEL=FF,DATE=YES  f
                                 1300 *
```

Figure 114. A Message Control Program for Teleprocessing Applications (Part 7 of 20)

```
                                                                                          PAGE    9

LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                                  F15OCT70    5/03/72

                                  1301 PQ3       TPROCESS  PCB=PCBLK2,QUEUES=MR,ALTDEST=PQ3,RECDEL=FF,DATE=YES
                                  1327 *
                                  1328 PQ4       TPROCESS  PCB=PCBLK2,QUEUES=MR,ALTDEST=PQ4,RECDEL=FF,DATE=YES
                                  1354 *
                                  1355 PQ5       TPROCESS  PCB=PCBLK2,QUEUES=MR,ALTDEST=PQ5,RECDEL=FF,DATE=YES
                                  1381 *
                                  1382 PQ6       TPROCESS  PCB=PCBLK2,QUEUES=MR,ALTDEST=PQ6,RECDEL=FF,DATE=YES
                                  1408 *
                                  1409 * OUTPUT TPROCESS ENTRY FOR COBOL QUEUE STRUCTURE TEST PROGRAMS
                                  1410 *
                                  1411 PQOUT     TPROCESS  PCB=PCBLK2,RECDEL=FF
                                  1437 *
                                  1438 *
                                  1439 * DISTRIBUTION LIST ENTRY --
                                  1440 *
                                  1441 * IN THE FOLLOWING MACRO --
                                  1442 *         LIST = NAMES OF TERMINAL OR TPROCESS ENTRIES IN THE
                                  1443 *           TERMINAL TABLE
                                  1444 *
                                  1445 *         THE LIST SHOULD NOT INCLUDE A TPROCESS ENTRY FOR A
                                  1446 *         COBOL APPLICATION PROGRAM
                                  1447 *
                                  1448 *         TYPE= D SPECIFIES THIS IS A DISTRIBUTION LIST ENTRY
                                  1449 *               C WOULD SPECIFY A CASCADE LIST ENTRY
                                  1450 *               DISTRIBUTION LISTS INDICATE A MESSAGE FORWARDED TO THEM
                                  1451 *               WILL BE SENT TO ALL NAMES IN THE LIST
                                  1452 *
                                  1453 *               WITH CASCADE LISTS, MESSAGES WILL BE SENT TO THE QUEUE
                                  1454 *               SPECIFIED IN THE LIST WITH THE FEWEST NO. OF MESSAGES
                                  1455 *
                                  1456 * 1050 AND TWX--USED BY MESSAGE PROCESSING PROGRAM
                                  1457 *
                               11 1458 D1        TLIST  LIST=(T1,T2),TYPE=D  a
                                  1479 *
                                  1480 *
                                  1481 * INVITATION LISTS
                                  1482 * SHOULD ALWAYS BE SPECIFIED FOLLOWING THE MACROS DEFINING THE TERMINAL
                                  1483 * TABLE
                                  1484 *
                                  1485 * LIST FOR 1050 LINE--
                                  1486 *         ORDER= ENTRIES FOR STATIONS ON LINE IN THE ORDER TO BE POLLED
                                  1487 *           T1 SPECIFIES A STATION ON THE LINE DEFINED BY A TERMINAL
                                  1488 *             MACRO
                                  1489 *           + SPECIFIES THE TERMINAL IS INITIALLY ACTIVE, - WOULD
                                  1490 *             SPECIFY IT WAS INITIALLY INACTIVE
                                  1491 *         6215=A0 IN 1050 CODE--A IS THE STATION ADDRESS--0 ASKS FOR
                                  1492 *             INPUT FROM ANY INPUT COMPONENT
                                  1493 *
                               12 1494 LIST1050 INVLIST  ORDER=(T1+6215)  a
                                  1503 *
                                  1504 * LIST FOR TWX LINE--
                                  1505 *   SINCE A TERMINAL MACRO WITH UTERM=YES WAS DEFINED FOR THIS LINE,
                                  1506 *   THIS MACRO NAME IS USED RATHER THAN THE ONE FOR THE TWX STATION
                                  1507 *
                                  1508 * THIS IS A SWITCHED LINE WHICH DOES NOT HAVE THE AUTO-CALL FEATURE--
```

```
                                                                                          PAGE   10

LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                                  F15OCT70    5/03/72

                                  1509 * THE COMPUTER NEVER ASKS FOR THE ID SEQUENCE FROM THE TWX TERMINAL
                                  1510 * UNLESS THE AUTO-CALL FEATURE IS PRESENT
                                  1511 *
                                  1512 * IF AUTO-CALL FEATURE IS NOT PRESENT OR TWX TERMINAL DOES NOT HAVE
                                  1513 * AN ID SEQUENCE FOR AN ANSWER-BACK, OMIT THE ID SEQUENCE CHARS. IN
                                  1514 * THE INVLIST MACRO
                                  1515 *
                                  1516 * IF AN ID SEQUENCE IS USED FOR THE TWX--IT IS SUGGESTED THE
                                  1517 * FOLLOWING CHARACTERS BE USED -- CR LF IDCHARS CR LF XON--IN LINE
                                  1518 * CODE
                                  1519 *
                                  1520 * THE CPUID OPERAND IS NEEDED FOR TWX TERMINALS--IT WILL PRINT AT
                                  1521 * TERMINAL WHEN CONNECTION IS MADE
                                  1522 *
                                  1523 LISTTWX  INVLIST  ORDER=(T2A+),CPUID=TWXSEQ  b
                                  1532 *
                                  1533 * REFERENCED BY LISTTWX AS CPUID OPERAND
                                  1534 * -- SUGGESTED USE NULL CR LF RUBOUT IDCHARS CR LF XON
                                  1535 * CPUID IS -- COBOL
0009DB 0C                         1536 TWXSEQ   DC    X'0C'           12 CHARACTERS
0009DC 01B151FFC3F343F3           1537          DC    X'01B151FFC3F343F333B15189'
                                  1538 *
```

Figure 114. A Message Control Program for Teleprocessing Applications (Part 8 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                          F15OCT70   5/03/72

                                 1540 ****
                                 1541 *
                                 1542 * MESSAGE HANDLERS--MH'S
                                 1543 *
                                 1544 * THE HEADER RECEIVED FROM THE TERMINAL IS--
                                 1545 *        POSSIBLE LINE FORMAT CHARS.--CR,LF,NL
                                 1546 *        $
                                 1547 *        BLANK
                                 1548 *        MSGTYPE--1 CHAR.
                                 1549 *        BLANK
                                 1550 *        SOURCE--2 CHARS.
                                 1551 *        BLANK
                                 1552 *        EOF FIELD--F IF END OF A GROUP OF MESSAGES
                                 1553 *                 --ANY OTHER CHAR. (EXCEPT BLANK) IF NOT
                                 1554 *        BLANK
                                 1555 *        ACTION CODE FOR APPLICATION PROGRAM--2 CHARS.
                                 1556 *        BLANK
                                 1557 *        PUNCTUATION MARK--PERIOD
                                 1558 *
                                 1559 ****
                                 1560 ***
                                 1561 *
                                 1562 * MESSAGE HANDLER FOR INPUT FROM AND OUTPUT TO 1050 TERMINAL
                                 1563 *
                                 1564 * THE FOLLOWING MACRO IS REQUIRED AND MUST BE FIRST
                                 1565 *        LC= IS THE ONLY REQUIRED OPERAND--
                                 1566 *          OUT SAYS TO REMOVE LINE CONTROL CHARS.
                                 1567 *          IN SAYS NOT TO REMOVE LINE CONTROL CHARS.
                                 1568 *        STOP= SAYS WHEN EOB ERROR FOUND AND RETRY COUNT EXHAUSTED,
                                 1569 *          ONLY THAT PORTION OF MESSAGE RECEIVED OR SENT CONTINUES
                                 1570 *          THRU MH--USER MAY CHECK ERROR RECORD BITS IN INMSG OR OUTMSG
                                 1571 *        CONT= SAYS THAT AFTER RETRY, SET BIT IN ERROR RECORD--BUT
                                 1572 *          CONTINUE TRANSMISSION
                                 1573 * IF NEITHER STOP NOR CONT SPECIFIED,NO EOB CHECKING PERFORMED
                                 1574 *
                            13   1575 MH1050   STARTMH  LC=OUT,CONT=YES
                                 1596 *
                                 1597 * THE FOLLOWING MACRO IS REQUIRED AS THE FIRST MACRO IN ANY INCOMING
                                 1598 * GROUP
                            14   1599          INHDR
                                 1613 *
                                 1614 * THE FOLLOWING MACRO TRANSLATES FROM LINE CODE TO EBCDIC--MACROS
                                 1615 * FOLLOWING THIS WILL ACT UPON CHARACTERS IN EBCDIC--IT WILL CAUSE
                                 1616 * ENTIRE MESSAGE TO BE TRANSLATED EVEN THOUGH IN INHDR GROUP
                            15   1617          CODE
                                 1641 *
                                 1642 * LOG INCOMING HEADERS--USE DCBNAME AS OPERAND
                                 1643 *
                            16   1644          LOG   MSGLOG
                                 1655 *
                                 1656 * SET SCAN POINTER TO $
                            17   1657          SETSCAN C'$'
                                 1673 *
                                 1674 * PROCESS THE REMAINDER OF THE HEADER ACCORDING TO THE MSGTYPE FIELD
                                 1675 * SPECIFIED NEXT IN THE HEADER--IF THE NEXT FIELD MATCHES THE CHARACTER
```

Figure 114. A Message Control Program for Teleprocessing Applications (Part 9 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                         F15OCT70    5/03/72

                                 1676 * SPECIFIED IN THE OPERAND, THE MACROS SPECIFIED BETWEEN IT AND THE
                                 1677 * NEXT MSGTYPE MACRO ARE EXECUTED AND CONTROL IS THEN PASSED TO THE
                                 1678 * NEXT DELIMITER--IN THIS CASE INBUF -IF THEY DO NOT MATCH, CONTROL
                                 1679 * PASSES TO THE NEXT MSGTYPE MACRO WHERE THE TEST IS AGAIN MADE
                                 1680 *
                           (18)  1681 * IF MSGTYPE IS 1, THIS MESSAGE SHOULD BE FORWARDED TO THE 1050
                                 1682          MSGTYPE  C'1'
                                 1698 *
                                 1699 * SCAN POINTER IS AT SOURCE FIELD--SINCE THIS IS A NON-SWITCHED STATION
                                 1700 * --ORIGIN VERIFIES THAT THE SOURCE FIELD CONTAINS THE SYMBOLIC NAME
                                 1701 * OF THE STATION THAT WAS INVITED TO SEND THE MESSAGE--IF NOT, ERROR
                                 1702 * BIT IN ERROR RECORD FOR MESSAGE IS SET TO 1
                           (19)  1703          ORIGIN
                           (20)  1716          FORWARD  DEST=C'T1'
                                 1734 *
                                 1735 * IF MSGTYPE IS 2, THIS MESSAGE SHOULD BE FORWARDED TO TWX TERMINAL--
                                 1736 * SEE COMMENTS UNDER MSGTYPE 1 FOR OTHER MACROS
                                 1737          MSGTYPE  C'2'
                                 1755          ORIGIN
                                 1765          FORWARD  DEST=C'T2'
                                 1780 *
                                 1781 * IF MSGTYPE IS 5, THIS MESSAGE SHOULD BE FORWARDED TO THE COBOL
                                 1782 * APPLICATION PROGRAM--
                                 1783 * SEE COMMENTS UNDER MSGTYPE 1 FOR OTHER MACROS
                                 1784          MSGTYPE  C'5'
                                 1802          ORIGIN
                                 1812          FORWARD  DEST=C'PIN'
                                 1827 *
                                 1828 * IF MSGTYPE IS 6, THE SOURCE FIELD HAS BEEN OMITTED--UNNECESSARY TO
                                 1829 * ISSUE AN ORIGIN FOR A NON-SWITCHED LINE--SEND MESSAGE TO THE COBOL
                                 1830 * APPLICATION PROGRAM
                                 1831          MSGTYPE  C'6'
                                 1849          FORWARD  DEST=C'PIN'
                                 1864 *
                                 1865 * IF THE MSGTYPE IS ANYTHING ELSE, IT IS INVALID--SET THE USER ERROR
                                 1866 * BIT WITH THE TERRSET MACRO--IN THE INMSG GROUP, WE WILL CANCEL MSG.--
                                 1867 * ISSUE FORWARD MACRO ANYWAY SINCE REQUIRED
                                 1868          MSGTYPE
                                 1873          FORWARD  DEST=C'T1'
                           (21)  1888          TERRSET
                                 1895 *
                                 1896 * THE MACROS IN THE FOLLOWING SUBGROUP ARE EXECUTED FOR EVERY BUFFER
                                 1897 * OF THE MESSAGE
                           (22)  1898          INBUF
                                 1903 *
                                 1904 * SPECIFY THE MAXIMUM NO. OF CHARACTERS ALLOWED IN AN INCOMING MESSAGE
                                 1905 * --THIS MACRO ALSO CHECKS IF THE INPUT BUFFER IS FILLED WITH IDENTICAL
                                 1906 * CHARACTERS, USUALLY AN INDICATION OF STATION MALFUNCTION--SETS A
                                 1907 * BIT IN ERROR RECORD FOR EITHER CONDITION
                           (23)  1908          CUTOFF  900
                                 1919 *
                                 1920 * INSERT X'FF' FOR EVERY NL AND LF CHARACTER--X'FF' IS THE RECDEL CHAR.
                                 1921 * SPECIFIED IN THE TPROCESS MACROS--IF A MESSAGE WERE ALWAYS BEING
                                 1922 * FORWARDED TO AN APPLICATION PROGRAM, WE COULD USE DELIMIT INSTEAD
                                 1923 * OF XL1'FF'
```

Figure 114. A Message Control Program for Teleprocessing Applications (Part 10 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                        F15OCT70   5/03/72

                           (24)  1924          MSGEDIT ((RA,XL1'FF',XL1'15'),(RA,XL1'FF',XL1'25'))
                                 1953 *
                                 1954 * THE INMSG SUBGROUP IS SPECIFIED AFTER OTHER SUBGROUPS--IT IS EXECUTED
                                 1955 * AFTER AN ENTIRE MESSAGE OR BLOCK HAS BEEN PROCESSED--NO EXECUTABLE
                                 1956 * USER-WRITTEN CODE SHOULD BE INCLUDED IN THIS SUBGROUP
                           (25)  1957          INMSG
                                 1965 *
                                 1966 * CANCELMG CAUSES IMMEDIATE CANCELLATION OF MESSAGE IF ANY ERRORS
                                 1967 * SPECIFIED BY ITS MASK OCCUR--IF USED, IT MUST BE 1ST MACRO UNDER
                                 1968 * INMSG--AN ERRORMSG MACRO MAY THEN NOTIFY OF THE ERROR--
                                 1969 * CANCELMG IF THE USER ERROR BIT IS SET INDICATING THE MSGTYPE FIELD
                                 1970 * WAS INVALID--BIT20
                           (26)  1971          CANCELMG  X'0000080000'
                                 1979 *
                                 1980 * IN THE FOLLOWING ERROR MESSAGES, THE 1ST FIELD IS THE MASK CORRE-
                                 1981 * SPONDING TO THE BITS IN THE ERROR RECORD,DEST= IS ALWAYS T1 FOR THE
                                 1982 * 1050 TERMINAL AND THE DATA= IS THE ERROR MESSAGE THAT IS SENT--
                                 1983 * THE MESSAGE INCLUDES THE HEADER OF THE MESSAGE IN ERROR AND THE
                                 1984 * ERROR MESSAGE
                                 1985 *
                                 1986 * THE LAST CHARACTER OF THE MESSAGE IS NL--SO THE CARRIAGE WILL BE
                                 1987 * RETURNED WITH A LINE FEED AT THE END OF THE PRINTING OF THE MESSAGE
                           (27)  1988          ERRORMSG  X'8000000000',DEST=C'T1',                               X
                                                     DATA=C'E ERROR IN PROCESSING HEADER '
                                 2004          ERRORMSG  X'4000000000',DEST=C'T1',                               X
                                                     DATA=C'E INVALID ORIGIN IN HEADER '
                                 2016          ERRORMSG  X'0200000000',DEST=C'T1',                               X
                                                     DATA=C'E INSUFFICIENT BUFFERS FOR INCOMING MESSAGE '
                                 2028          ERRORMSG  X'0100000000',DEST=C'T1',                               X
                                                     DATA=C'E MESSAGE TOO LONG '
                                 2040 *
                                 2041 * THE FOLLOWING ERROR MESSAGE SHOULD ONLY OCCUR WITH MAIN STORAGE
                                 2042 * QUEUEING WITH OR WITHOUT DISK BACKUP
                                 2043          ERRORMSG  X'0040000000',DEST=C'T1',                               X
                                                     DATA=C'E PERCENTAGE OF BUFFER UNITS IN BUFMAX ARE USED-SX
                                                     LOW DOWN '
                                 2055 *
                                 2056          ERRORMSG  X'0002000000',DEST=C'T1',                               X
                                                     DATA=C'E FORWARDED TO INVALID DESTINATION '
                                 2068          ERRORMSG  X'0000400000',DEST=C'T1',                               X
                                                     DATA=C'E INVALID STATION ID AT CONNECT TIME '
                                 2080          ERRORMSG  X'0000200000',DEST=C'T1',                               X
                                                     DATA=C'E TERMINAL IS IN HOLD STATUS '
                                 2092          ERRORMSG  X'0000080000',DEST=C'T1',                               X
                                                     DATA=C'E MSGTYPE CODE IN HEADER INVALID '
                                 2104          ERRORMSG  X'000000E000',DEST=C'T1',                               X
                                                     DATA=C'E A HARDWARE ERROR HAS OCCURRED '
                                 2116 *
                                 2117 * INEND IS REQUIRED AS LAST DELIMITER MACRO OF INCOMING GROUP
                           (28)  2118          INEND
                                 2122 *
                                 2123 ***
                                 2124 *
                                 2125 * OUTGOING GROUP OF MESSAGE HANDLER FOR 1050 TERMINAL
                           (29)  2126          OUTHDR
```

Figure 114. A Message Control Program for Teleprocessing Applications (Part 11 of 20)

```
        2132 *
        2133 * THE FOLLOWING MACRO CAUSES EOT LINE CONTROL CHARACTERS TO BE INSERTED
        2134 * IN EACH OUTGOING MESSAGE--SINCE NTBLKSZ=(BLKSIZE) CODED IN THE
        2135 * TERMINAL MACRO--IT ALSO INSERTS EOB CHARS.--THIS PARAMETER COULD
        2136 * ALSO BE PLACED AS AN OPERAND OF THIS MACRO TO OVERRIDE THE NO.
        2137 * SPECIFIED IN THE TERMINAL MACRO
   30   2138          MSGFORM
        2149 *
        2150 * SINCE ERROR MESSAGES ARE SENT TO THIS TERMINAL--AND THESE COULD
        2151 * INCLUDE THOSE FOR THE APPLICATION TO APPLICATION PROGRAM WHICH
        2152 * WILL NOT HAVE A HEADER AND CANNOT BE PROCESSED AS A NORMAL OUTPUT
        2153 * MESSAGE TO THIS TERMINAL--CHECK 1ST CHARACTER FOR AN E--THE 1ST
        2154 * CHAR. OF EVERY ERRORMSG--IF NOT E WILL SKIP TO NEXT MSGTYPE MACRO--
        2155 * IF E, WILL PROCESS TO NEXT MSGTYPE MACRO AND THEN SKIP TO NEXT
        2156 * DELIMITER--OUTBUF
        2157          MSGTYPE   C'E'
        2173 *
        2174 * SET SCAN POINTER BACK TO BEGINNING OF BUFFER AND INSERT NL CHARACTER
        2175 * AT BEGINNING OF MESSAGE--IDLES WILL BE INSERTED AFTER NL IN OUTBUF
        2176          SETSCAN  1,POINT=BACK
        2187          MSGEDIT  ((I,XL1'15',SCAN))
        2202 *
        2203 * USE MSGTYPE WITH BLANK OPERAND TO PROCESS OTHER MESSAGES
        2204          MSGTYPE
        2209 *
        2210 * INSERT NL CHARACTER AT BEGINNING OF MESSAGE--IDLES WILL BE INSERTED
        2211 * AFTER NL IN OUTBUF
        2212          MSGEDIT  ((I,XL1'15',SCAN))
        2224 *
        2225 * SET THE SCAN POINTER TO THE PERIOD IN THE HEADER AND INSERT DATE,
        2226 * TIME, AND SEQUENCE NO.--INSERTED IN EBCDIC SO DO BEFORE CODE
        2227          SETSCAN C'.'
        2240 *
        2241 * IF NO OPERAND--BOTH DATE AND TIME ARE INSERTED--SPACE MUST BE
        2242 * RESERVED BY MEANS OF THE RESERVE= OPERAND OF DCB FOR LINE--THE DATE
        2243 * IS IN FORM--(BLANK)YY.DDD--7 CHARS.--TIME IN FORM--
        2244 * (BLANK) HH.MM.SS--9 CHARACTERS
   31   2245          DATETIME
        2261 *
        2262 * SEQUENCE IN AN OUTHDR SUBGROUP INSERTS SEQUENCE NO. IN FORM--
        2263 * (BLANK)NNNN--5 CHARS.--SPACE MUST BE RESERVED BY MEANS OF RESERVE=
        2264 * OPERAND OF DCB FOR LINE
   32   2265          SEQUENCE
        2275 *
        2276 * LOG OUTGOING HEADERS--USE DCBNAME AS OPERAND--PUT MACRO AFTER
        2277 * INSERTION OF DATE, TIME, AND SEQUENCE NOS. SO THESE WILL APPEAR
        2278 * IN LOGGED HEADER
        2279 *
        2280          LOG   MSGLOG
        2288 *
        2289 * THE MACROS IN THE FOLLOWING SUBGROUP ARE EXECUTED FOR EVERY BUFFER
        2290 * OF THE MESSAGE
   33   2291          OUTBUF
        2296 *
        2297 * INSERT NL CHAR. FOR EVERY X'FF' CHAR. IN MESSAGE--X'FF' IS THE
```

Figure 114. A Message Control Program for Teleprocessing Applications (Part 12 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                              F15OCT70   5/03/72

                                 2298 * RECDEL CHAR. SPECIFIED IN THE TPROCESS MACROS
                                 2299          MSGEDIT  ((RA,XL1'15',XL1'FF'))
                                 2317 *
                                 2318 * INSERT 13 IDLE CHARS. AFTER EVERY NL CHARACTER PLACED IN MESSAGE
                                 2319          MSGEDIT  ((I,(X'17',13),XL1'15'))
                                 2336 *
                                 2337 * TRANSLATE THE MESSAGE FROM EBCDIC TO LINE CODE--IF ISSUED IN A
                                 2338 * SUBGROUP AND ANY SEGMENTS OF A MESSAGE PROCESSED BY THAT SUBGROUP,
                                 2339 * THE ENTIRE MESSAGE IS TRANSLATED
                                 2340          CODE
                                 2349 *
                                 2350 * THE OUTMSG SUBGROUP IS SPECIFIED AFTER OTHER SUBGROUPS IN OUTGOING
                                 2351 * GROUP--IT IS EXECUTED ONLY AFTER AN ENTIRE BLOCK OR MESSAGE HAS BEEN
                                 2352 * SENT
                            34   2353          OUTMSG
                                 2362 *
                                 2363 * THE HOLD MACRO SUSPENDS TRANSMISSION TO A STATION EITHER FOR A TIME
                                 2364 * INTERVAL (IF SPECIFIED) OR UNTIL RELEASED BY AN OPERATOR CONTROL
                                 2365 * MESSAGE--IF NOT USED, MESSAGES THAT CANNOT BE TRANSMITTED ARE
                                 2366 * TREATED AS THOUGH THEY HAVE BEEN TRANSMITTED--ALSO, A HOLD OPERATOR
                                 2367 * CONTROL MESSAGE HAS NO EFFECT IF THERE IS NO HOLD MACRO--
                                 2368 * BITS BEING TESTED BY MASK ARE FOR HARDWARE ERRORS
                            35   2369          HOLD  X'000000E000'
                                 2381 *
                                 2382 * IN THE FOLLOWING ERROR MESSAGES, THE 1ST FIELD IS THE MASK CORRE-
                                 2383 * SPONDING TO THE BITS IN THE ERROR RECORD,DEST= IS ALWAYS T1 FOR THE
                                 2384 * 1050 TERMINAL AND THE DATA= IS THE ERROR MESSAGE THAT IS SENT--
                                 2385 * THE MESSAGE INCLUDES THE HEADER OF THE MESSAGE IN ERROR AND THE
                                 2386 * ERROR MESSAGE
                                 2387 *
                                 2388 * THE LAST CHARACTER OF THE MESSAGE IS NL--SO THE CARRIAGE WILL BE
                                 2389 * RETURNED WITH A LINE FEED AT THE END OF THE PRINTING OF THE MESSAGE
                                 2390          ERRORMSG  X'8000000000',DEST=C'T1',                             X
                                                     DATA=C'E ERROR IN PROCESSING HEADER '
                                 2402 *
                                 2403 * THE FOLLOWING ERROR MESSAGE SHOULD ONLY OCCUR WITH MAIN STORAGE
                                 2404 * QUEUEING WITH OR WITHOUT DISK BACKUP
                                 2405          ERRORMSG  X'0040000000',DEST=C'T1',                             X
                                                     DATA=C'E PERCENTAGE OF BUFFER UNITS IN BUFMAX ARE USED-SX
                                                     LOW DOWN '
                                 2417 *
                                 2418          ERRORMSG  X'0000400000',DEST=C'T1',                             X
                                                     DATA=C'E INVALID STATION ID AT CONNECT TIME '
                                 2430          ERRORMSG  X'0000200000',DEST=C'T1',                             X
                                                     DATA=C'E TERMINAL IS IN HOLD STATUS '
                                 2442          ERRORMSG  X'000000E000',DEST=C'T1',                             X
                                                     DATA=C'E A HARDWARE ERROR HAS OCCURRED '
                                 2454 *
                                 2455 * OUTEND REQUIRED AS LAST DELIMITER MACRO OF OUTGOING GROUP
                            36   2456          OUTEND
                                 2460 *
                                 2461 * A LTORG SHOULD BE CODED AFTER LAST DELIMITER OF EACH MH IF MCP HAS
                                 2462 * MORE THAN 1 MH
000E60                           2463          LTORG
                                 2464 *
```

Figure 114. A Message Control Program for Teleprocessing Applications (Part 13 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                    F15OCT70    5/03/72

                                 2466 ***
                                 2467 *
                                 2468 * MESSAGE HANDLER FOR INPUT FROM AND OUTPUT TO TWX TERMINAL
                                 2469 *
                                 2470 * THE FOLLOWING MACRO IS REQUIRED AND MUST BE FIRST
                                 2471 *          LC= IS THE ONLY REQUIRED OPERAND--
                                 2472 *            OUT SAYS TO REMOVE LINE CONTROL CHARS.
                                 2473 *            IN SAYS NOT TO REMOVE LINE CONTROL CHARS.
                                 2474 *
                                 2475 MHTWX    STARTMH  LC=OUT
                                 2489 *
                                 2490 * THE FOLLOWING MACRO IS REQUIRED AS THE FIRST MACRO IN ANY INCOMING
                                 2491 * GROUP
                                 2492          INHDR
                                 2503 *
                                 2504 * THE FOLLOWING MACRO TRANSLATES FROM LINE CODE TO EBCDIC--MACROS
                                 2505 * FOLLOWING THIS WILL ACT UPON CHARACTERS IN EBCDIC--IT WILL CAUSE
                                 2506 * ENTIRE MESSAGE TO BE TRANSLATED EVEN THOUGH IN INHDR GROUP
                                 2507          CODE
                                 2527 *
                                 2528 * LOG INCOMING HEADERS--USE DCBNAME AS OPERAND
                                 2529 *
                                 2530          LOG   MSGLOG
                                 2538 *
                                 2539 * SET SCAN POINTER TO $
                                 2540          SETSCAN C'$'
                                 2553 *
                                 2554 * PROCESS THE REMAINDER OF THE HEADER ACCORDING TO THE MSGTYPE FIELD
                                 2555 * SPECIFIED NEXT IN THE HEADER--IF THE NEXT FIELD MATCHES THE CHARACTER
                                 2556 * SPECIFIED IN THE OPERAND, THE MACROS SPECIFIED BETWEEN IT AND THE
                                 2557 * NEXT MSGTYPE MACRO ARE EXECUTED AND CONTROL IS THEN PASSED TO THE
                                 2558 * NEXT DELIMITER--IN THIS CASE INBUF -IF THEY DO NOT MATCH, CONTROL
                                 2559 * PASSES TO THE NEXT MSGTYPE MACRO WHERE THE TEST IS AGAIN MADE
                                 2560 *
                                 2561 * IF MSGTYPE IS 1, THIS MESSAGE SHOULD BE FORWARDED TO THE 1050
                                 2562          MSGTYPE  C'1'
                                 2578 *
                                 2579 * SCAN POINTER IS AT SOURCE--ISSUE ORIGIN--SINCE THIS IS A SWITCHED
                                 2580 * LINE,ORIGIN WILL CHECK VALIDITY OF FIELD AND IDENTIFY THE CALLING
                                 2581 * STATION TO TCAM
                                 2582          ORIGIN
                                 2592          FORWARD  DEST=C'T1'
                                 2607 *
                                 2608 * IF MSGTYPE IS 2, THIS MESSAGE SHOULD BE FORWARDED TO TWX TERMINAL--
                                 2609 * SEE COMMENTS UNDER MSGTYPE 1 FOR OTHER MACROS
                                 2610          MSGTYPE  C'2'
                                 2628          ORIGIN
                                 2638          FORWARD  DEST=C'T2'
                                 2653 *
                                 2654 * IF MSGTYPE IS 5, THIS MESSAGE SHOULD BE FORWARDED TO THE COBOL
                                 2655 * APPLICATION PROGRAM--
                                 2656 * SEE COMMENTS UNDER MSGTYPE 1 FOR OTHER MACROS
                                 2657          MSGTYPE  C'5'
                                 2675          ORIGIN
                                 2685          FORWARD  DEST=C'PIN'
```

Figure 114. A Message Control Program for Teleprocessing Applications (Part 14 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                              F15OCT70   5/03/72

                                 2700 *
                                 2701 * IF MSGTYPE IS 6, THE SOURCE FIELD HAS BEEN OMITTED (IN ORDER FOR
                                 2702 * THE COBOL PROGRAM TO CHECK THAT THE LINE NAME--T2A--RATHER THAN THE
                                 2703 * STATION NAME--T2--IS GIVEN AS SOURCE)--THE MESSAGE IS TO BE SENT TO
                                 2704 * THE COBOL APPLICATION PROGRAM
                                 2705            MSGTYPE  C'6'
                                 2723            FORWARD  DEST=C'PIN'
                                 2738 *
                                 2739 * IF THE MSGTYPE IS ANYTHING ELSE, IT IS INVALID--SET THE USER ERROR
                                 2740 * BIT WITH THE TERRSET MACRO--IN THE INMSG GROUP, WE WILL CANCEL MSG.--
                                 2741 * ISSUE FORWARD MACRO ANYWAY SINCE REQUIRED
                                 2742            MSGTYPE
                                 2747            FORWARD  DEST=C'T1'
                                 2762            TERRSET
                                 2769 *
                                 2770 * THE MACROS IN THE FOLLOWING SUBGROUP ARE EXECUTED FOR EVERY BUFFER
                                 2771 * OF THE MESSAGE
                                 2772            INBUF
                                 2777 *
                                 2778 * SPECIFY THE MAXIMUM NO. OF CHARACTERS ALLOWED IN AN INCOMING MESSAGE
                                 2779 * --THIS MACRO ALSO CHECKS IF THE INPUT BUFFER IS FILLED WITH IDENTICAL
                                 2780 * CHARACTERS, USUALLY AN INDICATION OF STATION MALFUNCTION--SETS A
                                 2781 * BIT IN ERROR RECORD FOR EITHER CONDITION
                                 2782            CUTOFF  900
                                 2790 *
                                 2791 * DELETE EVERY CR CHAR. AND INSERT X'FF' FOR EVERY LF CHAR.--X'FF'
                                 2792 * IS THE RECDEL CHARACTER SPECIFIED IN THE TPROCESS MACROS (IF MESSAGES
                                 2793 * WERE ALWAYS GOING TO AN APPLICATION PROGRAM, WE COULD USE DELIMIT
                                 2794 * INSTEAD OF XL1'FF')
                                 2795            MSGEDIT  ((RA,CONTRACT,XL1'26'),(RA,XL1'FF',XL1'15'))
                                 2818 *
                                 2819 * THE INMSG SUBGROUP IS SPECIFIED AFTER OTHER SUBGROUPS--IT IS EXECUTED
                                 2820 * AFTER AN ENTIRE MESSAGE OR BLOCK HAS BEEN PROCESSED--NO EXECUTABLE
                                 2821 * USER-WRITTEN CODE SHOULD BE INCLUDED IN THIS SUBGROUP
                                 2822            INMSG
                                 2830 * CANCELMG CAUSES IMMEDIATE CANCELLATION OF MESSAGE IF ANY ERRORS
                                 2831 *
                                 2832 * SPECIFIED BY ITS MASK OCCUR--IF USED, IT MUST BE 1ST MACRO UNDER
                                 2833 * INMSG--AN ERRORMSG MACRO MAY THEN NOTIFY OF THE ERROR--
                                 2834 * CANCELMG IF THE USER ERROR BIT IS SET INDICATING THE MSGTYPE FIELD
                                 2835 * WAS INVALID--BIT20
                                 2836            CANCELMG  X'0000080000'
                                 2841 *
                                 2842 * IN THE FOLLOWING ERROR MESSAGES, THE 1ST FIELD IS THE MASK CORRE-
                                 2843 * SPONDING TO THE BITS IN THE ERROR RECORD,DEST= IS ALWAYS T1 FOR THE
                                 2844 * 1050 TERMINAL AND THE DATA= IS THE ERROR MESSAGE THAT IS SENT--
                                 2845 * THE MESSAGE INCLUDES THE HEADER OF THE MESSAGE IN ERROR AND THE
                                 2846 * ERROR MESSAGE
                                 2847 *
                                 2848 * THE LAST CHARACTER OF THE MESSAGE IS NL--SO THE CARRIAGE WILL BE
                                 2849 * RETURNED WITH A LINE FEED AT THE END OF THE PRINTING OF THE MESSAGE
                                 2850            ERRORMSG  X'8000000000',DEST=C'T1',                           X
                                                       DATA=C'E ERROR IN PROCESSING HEADER '
                                 2862            ERRORMSG  X'4000000000',DEST=C'T1',                           X
                                                       DATA=C'E INVALID ORIGIN IN HEADER '
```

Figure 114. A Message Control Program for Teleprocessing Applications (Part 15 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT    SOURCE STATEMENT                                                  F15OCT70   5/03/72
                                 2874           ERRORMSG  X'0200000000',DEST=C'T1',                                  X
                                                     DATA=C'E INSUFFICIENT BUFFERS FOR INCOMING MESSAGE '
                                 2886           ERRORMSG  X'0100000000',DEST=C'T1',                                  X
                                                     DATA=C'E MESSAGE TOO LONG '
                                 2898 *
                                 2899 * THE FOLLOWING ERROR MESSAGE SHOULD ONLY OCCUR WITH MAIN STORAGE
                                 2900 * QUEUEING WITH OR WITHOUT DISK BACKUP
                                 2901           ERRORMSG  X'0040000000',DEST=C'T1',                                  X
                                                     DATA=C'E PERCENTAGE OF BUFFER UNITS IN BUFMAX ARE USED-SX
                                                     LOW DOWN '
                                 2913 *
                                 2914           ERRORMSG  X'0002000000',DEST=C'T1',                                  X
                                                     DATA=C'E FORWARDED TO INVALID DESTINATION '
                                 2926           ERRORMSG  X'0000400000',DEST=C'T1',                                  X
                                                     DATA=C'E INVALID STATION ID AT CONNECT TIME '
                                 2938           ERRORMSG  X'0000200000',DEST=C'T1',                                  X
                                                     DATA=C'E TERMINAL IS IN HOLD STATUS '
                                 2950           ERRORMSG  X'0000080000',DEST=C'T1',                                  X
                                                     DATA=C'E MSGTYPE CODE IN HEADER INVALID '
                                 2962           ERRORMSG  X'000000E000',DEST=C'T1',                                  X
                                                     DATA=C'E A HARDWARE ERROR HAS OCCURRED '
                                 2974 *
                                 2975 * INEND IS REQUIRED AS LAST DELIMITER MACRO OF INCOMING GROUP
                                 2976           INEND
                                 2980 *
                                 2981 ***
                                 2982 *
                                 2983 * OUTGOING GROUP OF MESSAGE HANDLER FOR TWX TERMINAL
                             37  2984           OUTHDR
                                 2990 *
                                 2991 * INSERT CR LF RUBOUT AT BEGINNING OF MESSAGE
                                 2992           MSGEDIT  ((I,XL3'261507',SCAN))
                                 3004 *
                                 3005 * THE FOLLOWING MACRO CAUSES EOT LINE CONTROL CHARACTERS TO BE INSERTED
                                 3006 * IN EACH OUTGOING MESSAGE
                                 3007           MSGFORM
                                 3014 *
                                 3015 * SET THE SCAN POINTER TO THE PERIOD IN THE HEADER AND INSERT DATE,
                                 3016 * TIME, AND SEQUENCE NO.--INSERTED IN EBCDIC SO DO BEFORE CODE
                                 3017           SETSCAN C'.'
                                 3030 *
                                 3031 * IF NO OPERAND--BOTH DATE AND TIME ARE INSERTED--SPACE MUST BE
                                 3032 * RESERVED BY MEANS OF THE RESERVE= OPERAND OF DCB FOR LINE--THE DATE
                                 3033 * IS IN FORM--(BLANK)YY.DDD--7 CHARS.--TIME IN FORM--
                                 3034 * (BLANK) HH.MM.SS--9 CHARACTERS
                                 3035           DATETIME
                                 3048 *
                                 3049 * SEQUENCE IN AN OUTHDR SUBGROUP INSERTS SEQUENCE NO. IN FORM--
                                 3050 * (BLANK)NNNN--5 CHARS.--SPACE MUST BE RESERVED BY MEANS OF RESERVE=
                                 3051 * OPERAND OF DCB FOR LINE
                                 3052           SEQUENCE
                                 3059 *
                                 3060 * LOG OUTGOING HEADERS--USE DCBNAME AS OPERAND--PUT MACRO AFTER
                                 3061 * INSERTION OF DATE, TIME, AND SEQUENCE NOS. SO THESE WILL APPEAR
                                 3062 * IN LOGGED HEADER
```

Figure 114. A Message Control Program for Teleprocessing Applications (Part 16 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                              F15OCT70   5/03/72

                                 3063 *
                                 3064          LOG   MSGLOG
                                 3072 *
                                 3073 * THE MACROS IN THE FOLLOWING SUBGROUP ARE EXECUTED FOR EVERY BUFFER
                                 3074 * OF THE MESSAGE
                                 3075          OUTBUF
                                 3079 *
                                 3080 * INSERT CR LF RUBOUT FOR EVERY X'FF' CHAR. IN MESSAGE--X'FF' IS
                                 3081 * THE RECDEL CHAR. SPECIFIED IN THE TPROCESS MACROS
                                 3082          MSGEDIT  ((RA,XL3'261507',XL1'FF'))
                                 3100 *
                                 3101 * TRANSLATE THE MESSAGE FROM EBCDIC TO LINE CODE--IF ISSUED IN A
                                 3102 * SUBGROUP AND ANY SEGMENTS OF A MESSAGE PROCESSED BY THAT SUBGROUP,
                                 3103 * THE ENTIRE MESSAGE IS TRANSLATED
                                 3104          CODE
                                 3113 *
                                 3114 * THE OUTMSG SUBGROUP IS SPECIFIED AFTER OTHER SUBGROUPS IN OUTGOING
                                 3115 * GROUP--IT IS EXECUTED ONLY AFTER AN ENTIRE BLOCK OR MESSAGE HAS BEEN
                                 3116 * SENT
                                 3117          OUTMSG
                                 3126 *
                                 3127 * THE HOLD MACRO SUSPENDS TRANSMISSION TO A STATION EITHER FOR A TIME
                                 3128 * INTERVAL (IF SPECIFIED) OR UNTIL RELEASED BY AN OPERATOR CONTROL
                                 3129 * MESSAGE--IF NOT USED, MESSAGES THAT CANNOT BE TRANSMITTED ARE
                                 3130 * TREATED AS THOUGH THEY HAVE BEEN TRANSMITTED--ALSO, A HOLD OPERATOR
                                 3131 * CONTROL MESSAGE HAS NO EFFECT IF THERE IS NO HOLD MACRO--
                                 3132 * BITS BEING TESTED BY MASK ARE FOR HARDWARE ERRORS
                                 3133          HOLD  X'000000E000'
                                 3139 *
                                 3140 * IN THE FOLLOWING ERROR MESSAGES, THE 1ST FIELD IS THE MASK CORRE-
                                 3141 * SPONDING TO THE BITS IN THE ERROR RECORD,DEST= IS ALWAYS T1 FOR THE
                                 3142 * 1050 TERMINAL AND THE DATA= IS THE ERROR MESSAGE THAT IS SENT--
                                 3143 * THE MESSAGE INCLUDES THE HEADER OF THE MESSAGE IN ERROR AND THE
                                 3144 * ERROR MESSAGE
                                 3145 *
                                 3146 * THE LAST CHARACTER OF THE MESSAGE IS NL--SO THE CARRIAGE WILL BE
                                 3147 * RETURNED WITH A LINE FEED AT THE END OF THE PRINTING OF THE MESSAGE
                                 3148          ERRORMSG  X'8000000000',DEST=C'T1',                           X
                                                     DATA=C'E ERROR IN PROCESSING HEADER '
                                 3160 *
                                 3161 * THE FOLLOWING ERROR MESSAGE SHOULD ONLY OCCUR WITH MAIN STORAGE
                                 3162 * QUEUEING WITH OR WITHOUT DISK BACKUP
                                 3163          ERRORMSG  X'0040000000',DEST=C'T1',                           X
                                                     DATA=C'E PERCENTAGE OF BUFFER UNITS IN BUFMAX ARE USED-SX
                                                     LOW DOWN '
                                 3175 *
                                 3176          ERRORMSG  X'0000400000',DEST=C'T1',                           X
                                                     DATA=C'E INVALID STATION ID AT CONNECT TIME '
                                 3188          ERRORMSG  X'0000200000',DEST=C'T1',                           X
                                                     DATA=C'E TERMINAL IS IN HOLD STATUS '
                                 3200          ERRORMSG  X'000000E000',DEST=C'T1',                           X
                                                     DATA=C'E A HARDWARE ERROR HAS OCCURRED '
                                 3212 *
                                 3213 * OUTEND REQUIRED AS LAST DELIMITER MACRO OF OUTGOING GROUP
                                 3214          OUTEND
```



```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                              F15OCT70   5/03/72

                                 3218 *
                                 3219 * A LTORG SHOULD BE CODED AFTER LAST DELIMITER OF EACH MH IF MCP HAS
                                 3220 * MORE THAN 1 MH
001270                           3221          LTORG
                                 3222 *
```

Figure 114. A Message Control Program for Teleprocessing Applications (Part 17 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                              F15OCT70    5/03/72

                                 3224 ***
                                 3225 *
                                 3226 * MESSAGE HANDLER FOR INPUT FROM AND OUTPUT TO APPLICATION PROGRAM
                                 3227 * RUNNING WITH TERMINALS
                                 3228 *
                                 3229 * THE FOLLOWING MACRO IS REQUIRED AND MUST BE FIRST
                                 3230 *          LC= IS A REQUIRED OPERAND--PUT'IN'SINCE NO LINE CONTROL
                                 3231 *            CHARACTERS TO REMOVE
                                 3232 MHTRMAPP STARTMH  LC=IN
                                 3246 *
                                 3247 * THE INCOMING GROUP HANDLES MESSAGES COMING FROM AN APPLICATION
                                 3248 * PROGRAM--THE MESSAGES WILL SUBSEQUENTLY BE PROCESSED BY THE OUTGOING
                                 3249 * GROUP FOR THE DESTINATION TERMINAL
                                 3250 *
                                 3251 * THE INHDR DELIMITER IS REQUIRED AND IS ALWAYS 1ST MACRO
                             38  3252          INHDR
                                 3263 *
                                 3264 * LOG INCOMING HEADERS--USE DCBNAME AS OPERAND
                                 3265 *
                                 3266          LOG    MSGLOG
                                 3274 *
                                 3275 * THE FORWARD MACRO IS REQUIRED IN EACH INHDR SUBGROUP--
                                 3276 * THE OPERAND DEST=PUT SAYS TO FORWARD TO THE DESTINATION SPECIFIED
                                 3277 * IN THE PREFIX TO THE APPLICATION PROGRAM WORK AREA
                                 3278          FORWARD  DEST=PUT
                                 3286 *
                                 3287 * THE INMSG SUBGROUP IS SPECIFIED AFTER OTHER SUBGROUPS IN AN INCOMING
                                 3288 * GROUP--IT IS EXECUTED AFTER AN ENTIRE MESSAGE OR BLOCK HAS BEEN
                                 3289 * PROCESSED
                                 3290          INMSG
                                 3298 *
                                 3299 * IN THE FOLLOWING ERROR MESSAGES, THE 1ST FIELD IS THE MASK CORRE-
                                 3300 * SPONDING TO THE BITS IN THE ERROR RECORD, DEST= IS ALWAYS T1 FOR THE
                                 3301 * 1050 TERMINAL AND THE DATA= IS THE ERROR MESSAGE THAT IS SENT--
                                 3302 *
                                 3303 * THE LAST CHARACTER OF THE MESSAGE IS NL--SO THE CARRIAGE WILL BE
                                 3304 * RETURNED WITH A LINE FEED AT THE END OF THE PRINTING OF THE MESSAGE
                                 3305          ERRORMSG  X'0200000000',DEST=C'T1',                           X
                                                     DATA=C'E INSUFFICIENT BUFFERS FOR INCOMING MESSAGE '
                                 3317 *
                                 3318 * THE FOLLOWING ERROR MESSAGE SHOULD ONLY OCCUR WITH MAIN STORAGE
                                 3319 * QUEUEING WITH OR WITHOUT DISK BACKUP
                                 3320          ERRORMSG  X'0040000000',DEST=C'T1',                           X
                                                     DATA=C'E PERCENTAGE OF BUFFER UNITS IN BUFMAX ARE USED-SX
                                                     LOW DOWN '
                                 3332          ERRORMSG  X'0002000000',DEST=C'T1',                           X
                                                     DATA=C'E FORWARDED TO INVALID DESTINATION '
                                 3344 *
                                 3345 * INEND IS REQUIRED AS LAST DELIMITER OF INCOMING GROUP
                                 3346          INEND
                                 3350 *
                                 3351 ***
                                 3352 *
                             39  3353 * OUTGOING GROUP HANDLES MESSAGES BEING SENT TO APPLICATION PROGRAM
                                 3354          OUTHDR
```

Figure 114. A Message Control Program for Teleprocessing Applications (Part 18 of 20)

```
                                                                                       PAGE  22

LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                        F15OCT70    5/03/72

                                 3360 *
                                 3361 * DELETE ANY CHARS. SUCH AS CR,LF WHICH APPEAR BEFORE $ IN HEADER
                                 3362          MSGEDIT  ((R,CONTRACT,SCAN,C'$'))
                                 3377 *
                                 3378 * SET SCAN POINTER OVER 2 NON-BLANK CHARS.--$ AND MSGTYPE FIELD--SO
                                 3379 * IT POINTS TO BEFORE SOURCE FIELD
                                 3380          SETSCAN  2
                                 3388 *
                                 3389 * INSERT SEQUENCE NO.--FORMAT IS (BLANK)NNNN--5 CHARS--SPACE MUST BE
                                 3390 * RESERVED BY MEANS OF RESERVE= OPERAND OF DCB FOR LINE
                                 3391          SEQUENCE
                                 3398 *
                                 3399 * LOG OUTGOING HEADERS--USE DCBNAME AS OPERAND--PUT MACRO AFTER
                                 3400 * INSERTION OF SEQUENCE NO. SO THIS WILL APPEAR IN LOGGED HEADER
                                 3401 *
                                 3402          LOG   MSGLOG
                                 3410 *
                                 3411 * SET SCAN POINTER OVER 2 NON-BLANK CHARS. (SOURCE FIELD) SO IT POINTS
                                 3412 * TO EOF FIELD
                                 3413          SETSCAN  2
                                 3421 *
                                 3422 * SETEOF IS USED TO IDENTIFY THE LAST MESSAGE OF A GROUP OF MESSAGES
                                 3423 * TO THE APPLICATION PROGRAM--IT CAUSES THE NEXT READ/CHECK AFTER
                                 3424 * THIS COMPLETE MESSAGE HAS BEEN RECEIVED TO PASS TO AN APPLICATION
                                 3425 * PROGRAM EODAD ROUTINE--(THE COBOL PROGRAM WOULD RECEIVE AN ETI
                                 3426 * INDICATION)
                                 3427          SETEOF  C'F'
                                 3444 *
                                 3445 * NO OUTMSG SUBGROUP WILL BE EXECUTED FOR A MESSAGE BEING TRANSFERRED
                                 3446 * FROM A TPROCESS QUEUE TO AN APPLICATION PROGRAM--
                                 3447 * SO OMIT OUTMSG IN THIS MESSAGE HANDLER
                                 3448 *
                                 3449 *
                                 3450 * OUTEND IS REQUIRED AS LAST DELIMITER OF OUTGOING GROUP
                                 3451          OUTEND
                                 3462 *
                                 3463 * A LTORG SHOULD BE CODED AFTER LAST DELIMITER OF EACH MH IF MCP HAS
                                 3464 * MORE THAN 1 MH
001390                           3465          LTORG
                                 3466 *
```

Figure 114. A Message Control Program for Teleprocessing Applications (Part 19 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                               F15OCT70   5/03/72

                                 3468 ***
                                 3469 *
                                 3470 * MESSAGE HANDLER FOR COBOL PROGRAMS THAT SIMULATE TERMINAL INPUT DATA
                                 3471 * --USED FOR TESTING WITHOUT TERMINALS
                                 3472 *
                                 3473 * THE FOLLOWING MACRO IS REQUIRED AND MUST BE FIRST
                                 3474 *        LC= IS A REQUIRED OPERAND--PUT'IN'SINCE NO LINE CONTROL
                                 3475 *          CHARACTERS TO REMOVE
                                 3476 MHAPPAPP STARTMH  LC=IN
                                 3490 * THE INCOMING GROUP HANDLES MESSAGES COMING FROM AN APPLICATION
                                 3491 * PROGRAM--THE MESSAGES WILL SUBSEQUENTLY BE PROCESSED BY THE OUTGOING
                                 3492 * GROUP WHEN THE APPLICATION PROGRAM READS THEM BACK
                                 3493 *
                                 3494 * THE INHDR DELIMITER IS REQUIRED AND IS ALWAYS 1ST MACRO
                          40     3495          INHDR
                                 3506 *
                                 3507 * THE FORWARD MACRO IS REQUIRED IN EACH INHDR SUBGROUP--
                                 3508 * THE OPERAND DEST=PUT SAYS TO FORWARD TO THE DESTINATION SPECIFIED
                                 3509 * IN THE PREFIX TO THE APPLICATION PROGRAM WORK AREA
                                 3510          FORWARD  DEST=PUT
                                 3518 *
                                 3519 * THE INMSG SUBGROUP IS SPECIFIED AFTER OTHER SUBGROUPS IN AN INCOMING
                                 3520 * GROUP--IT IS EXECUTED AFTER AN ENTIRE MESSAGE OR BLOCK HAS BEEN
                                 3521 * PROCESSED
                                 3522          INMSG
                                 3530 *
                                 3531 * IN THE FOLLOWING ERROR MESSAGES, THE 1ST FIELD IS THE MASK CORRE-
                                 3532 * SPONDING TO THE BITS IN THE ERROR RECORD, DEST= IS ALWAYS T1 FOR THE
                                 3533 * 1050 TERMINAL AND THE DATA= IS THE ERROR MESSAGE THAT IS SENT--
                                 3534 *
                                 3535 * THE LAST CHARACTER OF THE MESSAGE IS NL--SO THE CARRIAGE WILL BE
                                 3536 * RETURNED WITH A LINE FEED AT THE END OF THE PRINTING OF THE MESSAGE
                                 3537          ERRORMSG  X'0200000000',DEST=C'T1',                             X
                                                     DATA=C'E INSUFFICIENT BUFFERS FOR INCOMING MESSAGE '
                                 3549 *
                                 3550 * THE FOLLOWING ERROR MESSAGE SHOULD ONLY OCCUR WITH MAIN STORAGE
                                 3551 * QUEUEING WITH OR WITHOUT DISK BACKUP
                                 3552          ERRORMSG  X'0040000000',DEST=C'T1',                             X
                                                     DATA=C'E PERCENTAGE OF BUFFER UNITS IN BUFMAX ARE USED-SX
                                                     LOW DOWN '
                                 3564          ERRORMSG  X'0002000000',DEST=C'T1',                             X
                                                     DATA=C'E FORWARDED TO INVALID DESTINATION '
                                 3576 *
                                 3577 * INEND IS REQUIRED AS LAST DELIMITER OF INCOMING GROUP
                                 3578          INEND
                                 3582 *
                                 3583 ***
                                 3584 *
                                 3585 * OUTGOING GROUP HANDLES MESSAGES BEING SENT TO APPLICATION PROGRAM
                                 3586 *
                                 3587 * NO OUTMSG SUBGROUP WILL BE EXECUTED FOR A MESSAGE BEING TRANSFERRED
                                 3588 * FROM A TPROCESS QUEUE TO AN APPLICATION PROGRAM--
                                 3589 * SO OMIT OUTMSG IN THIS MESSAGE HANDLER
                                 3590 *
                                 3591 *
```



```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                               F15OCT70   5/03/72

                                 3592 * OUTEND IS REQUIRED AS LAST DELIMITER OF OUTGOING GROUP
                                 3593          OUTEND
                                 3604 *
                                 3605 * A LTORG SHOULD BE CODED AFTER LAST DELIMITER OF EACH MH IF MCP HAS
                                 3606 * MORE THAN 1 MH
001420                           3607          LTORG
                                 3608 *
                                 3609          END
```

Figure 114. A Message Control Program for Teleprocessing Applications (Part 20 of 20)

Defining the Buffers

User-defined areas of main storage receive any and all messages entering a TCAM network. Such areas, known as buffers, are used for handling, queueing, and transferring message segments between all lines and queueing media, and between queueing media and COBOL work areas.

In order to understand how the buffers are defined, it is necessary to distinguish between buffer units and buffers. TCAM has one buffer unit pool that contains buffer units of one size. Buffer units are the basic building blocks from which buffers are constructed (that is, buffer units are linked together to form buffers). Therefore, even though the buffers for line groups and for the application program may differ in size, each size specified should be a multiple of the size specified for a buffer unit, in order to use space optimally.

Three operands of the INTRO macro, 1, describe the TCAM buffer unit pool. As in the sample program shown in Figure 114, the operands that define the size of buffer units and specify the number assigned are KEYLEN, LNUNITS, and MSUNITS. The operands BUFSIZE, BUFIN, and BUFOUT, given in the DCB for line groups, 7c, and in the PCB, 6, for an application program, specify the buffer size and the number of buffers to be assigned initially for a receiving or sending operation. The manner in which the PCI= operand of the DCB for a line group, 7c, is coded greatly affects the number coded for LNUNITS in the INTRO macro and the numbers coded for the BUFIN and BUFOUT operands of the DCB for the line group.

Activating and Deactivating the Message Control Program

The TCAM message control program is assembled, link-edited, and executed like any other program running under an OS system. The macros INTRO, OPEN, and READY, issued as a group, make up the data-set initialization and activation section of the message control program.

Orderly deactivation of the TCAM system must stop incoming and outgoing message traffic and create a checkpoint record. The user must ensure that the data sets for any application program using TCAM as its access method are closed before the MCP enters its deactivation section, which closes the MCP data sets. (It is suggested that the headers of messages transmitted to the COBOL programs contain a code that signals the COBOL program to go to the STOPRUN statement.) Finally, the MCP coding must return control to the OS supervisor.

INTRO Macro: As the first macro executed in the message control program, INTRO, 1 establishes standard entry linkage, chains save areas, provides addressability, and saves the start parameter list pointer. (A description of the operands in the INTRO macro precedes the macro itself in the sample program.)

Note: The message below is issued if at least one of the following operands is omitted from the INTRO macro: STARTUP=, KEYLEN=, LNUNITS= , and (if DISK=YES is coded in the INTRO macro) CPB=.

00 IED002A SPECIFY TCAM PARAMETERS

The user may then enter the additional required parameters, changing certain other operands as desired.

OPEN Macro: The OPEN macros, 2, complete the initialization of the TCAM data sets and activate them for use. The TCAM data sets that must be activated in the MCP by OPEN macros are those for the message queues, a, checkpoint, optionally b, the line groups, c, and the message log, optionally d. If a snap dump is used, the user must also open the data set for snap, e.

READY Macro: The READY macro, 3, must be the last instruction in the initialization and activation section of the MCP. When READY has executed, the system is ready to handle message traffic.

CLOSE Macro: An optional snap dump of the program begins the deactivation section. Then the first CLOSE macro instruction, 4, is executed. This deactivation section is not executed until all data sets in TCAM application programs have been closed. In the example, the user closes the line group data sets, a, first; next the snap data set, b; then the message log data set, c; the checkpoint data set next to last, d; and finally the message queues data set, e.

Note: The data sets may be closed in any order provided that the checkpoint data set and the message queues data set are closed in the order indicated.

RETURN Macro: The assembler-language Load instruction is issued to restore register 13 with the address of the system save area, and the RETURN macro, 5, is issued to return control to the OS supervisor.

Defining the MCP Data Sets and Process Control Blocks

The user must provide information that serves as an interface between the message control program and the application program. This information is contained in process control blocks (PCB) and is generated by the PCB macro.

The message control program must also describe the MCP data sets to be used. Two of the four possible types of data sets usually required by every message control program are the line group data set, if there are lines, and the message queues data set, if there are disk queues. The operation of the MCP requires that either a message queues data set or a line group data set be opened. A user employing main storage queueing for application-to-application program processing (who, therefore, does not need either of these data sets) must, nevertheless, open a dummy line to meet this requirement. An error message will be issued at the system console because no hardware is attached, but this message can be ignored.

If the user does not open a line and, therefore, does not need either a DCB for a line group or a TERMINAL entry, the assembly of the MCP, nevertheless, generates an error message for the undefined symbol of IEDQSTCS. The user can either define this symbol in this program with a dummy label or ignore the severity level of 8 in the link-edit step. The symbol IEDQSTCS need not be correctly defined when the user is running only application-to-application programs.

Either or both of the other two types of data sets -- the checkpoint data set and the log data set -- may be specified if needed. To describe data sets to the system, the user (via a DCB macro) defines a data control block (DCB) for each data set cited.

PCB Macro: A process control block, created through specification of the PCB macro, is required in the MCP for each active application program. The PCB macro is similar to the DCB for the line groups in that it specifies the name of the message handler to be used for messages being sent by or received from an application program, as well as buffer requirements. The TPROCESS macro (see the discussion under "Defining Terminal and Line Control Areas") refers to the name of the PCB macro.

In the sample program given in Figure 114 are three process control blocks -- PCBLK, a, for a COBOL program running with terminals; PCBLK1, b, for COBOL programs that simulate the sending of messages from a remote terminal; and PCBLK2, c, for testing COBOL programs that take advantage of the queue structure feature. Having these three control blocks makes it possible for the COBOL program running with terminals to run at the same time as one of the other COBOL programs. In the example in this chapter, the TESTTP1 program simulates a terminal sending messages to the TESTTP2 program.

DCB Macro: A data control block, created through specification of the DCB macro, 7, is required for each data set referred to by the MCP. In the sample message control program, data control blocks are defined as follows:

- The message queues DCB macro, which defines a data control block for a message queues data set, MSGQ a.
- The checkpoint DCB macro, which defines a checkpoint data set if the checkpoint facility is to be used, CHKPT b.
- The line group DCB macro, which defines a line group data set, must be specified for each line group in the system. In the sample MCP, two line group data sets are defined -- the 1050 line group, named LN1050 c, and the TWX line group, named LNTWX d.
- The log DCB marco, which defines data sets for messages or message segments, should be specified for each secondary storage device on which messages or message segments may be logged. In the sample program, only one log DCB defining the MSGLOG data set, e, is specified.
- The snap dump DCB macro, which defines the data set for a snap dump, should be specified only if the user wants a snap dump. In the sample program, the DUMP data set is defined, f.

Defining Terminal and Line Control Areas

In writing an MCP, the user must provide information that identifies the remote stations, specifies their characteristics to the system, and tells how they are to be handled. Line control is the scheme of operating procedures and signals by which a teleprocessing system is controlled.

Line control concerns itself with such tasks as establishing contact between a sending and a receiving station, directing a message to a specific station on a multistation line, handling priorities when two stations try to send at the same time, and performing a user-specified action when a station fails to respond to a message.

Several TCAM macros are available to the user for identifying stations and specifying how message transmission is to be handled. The TCAM macros used in the sample message control program given in Figure 113 -- TTABLE, TERMINAL, INVLIST, TLIST, and TPROCESS -- are described below. Two additional macros -- OPTION, which reserves space for an option field, and LOGTYPE, needed only for logging entire messages -- are also available to the COBOL user.

TTABLE Macro: The TTABLE macro, 8, defines the start and the end of the terminal table, needed to provide information about each station and application program.

TERMINAL Macro: The TERMINAL macro, 9, specified three times in the sample program, must be coded for each station that can accept messages (as well as for some terminals that can only enter messages), each group of non-switched terminals equipped with the hardware group-code feature, and each switched line to stations that do not uniquely identify themselves after calling the computer.

Specification of the TERMINAL macro places a station or line name and associated information in this terminal table. TERMINAL produces a single entry, a group entry, or a line entry. In the example, the T1 entry, a, provides information about the 1050 terminal, the T2A entry, b, information about the switched TWX line, and the T2 entry, c, information about the TWX terminal on this line.

Notes:

1. The "UTERM=YES" specification in the TERMINAL macro for the switched TWX line creates an entry for the line. This gives the program the control information it needs to handle stations that call this line. After the station is identified by means of the ORIGIN macro in the MH, the program then refers to the TERMINAL entry for the station.

2. All TERMINAL macros for lines in a line group must be arranged in ascending relative line numbers. The TERMINAL macro for a particular line must immediately precede all TERMINAL macros for stations on that line. In the sample MCP, there is only one line per line group and one terminal per line, but this need not be true.

TPROCESS Macro: By placing the name of a queue for an application program, as well as associated information, in the terminal table, the TPROCESS macro, 10, helps connect a COBOL program with the message control program.

The user must specify one TPROCESS macro for each destination queue from which a COBOL program is to receive messages and at least one that is used when messages are sent by a COBOL program. (That is, one output TPROCESS entry is required for each application program running simultaneously.) The output TPROCESS entry is not the name of a queue. In the sample program, for example, twelve TPROCESS entries are specified. The PIN entry, a, identifies an input destination queue for a COBOL program running with terminals; POUT identifies an output process entry.

Similarly, the P1, c, and P2, d, entries identify input destination queues for COBOL programs that simulate terminal input data, and the POUT1, e, entry identifies an output process entry for such COBOL programs. The PQ1, PQ2, PQ3, PQ4, PQ5, PQ6, and PQOUT TPROCESS entries are used for COBOL programs that employ the queue structure feature.

Note: Because the PIN and POUT entries in the example refer to one process control block (PCBBLK) and the P1, P2, and POUT1 entries refer to another process control block (PCBBLK1), a program running with terminals can run concurrently with another program. This is alco true of the PQ entries, which refer to PCBLK2.

TLIST Macro: An instruction that places the name of a list of a single, a group, or a process entry in the terminal table, the TLIST macro, 11, must be specified for each such list to be created. This list can be specified as either a distribution list or a cascade list. When a message is sent to a distribution list, the same message is sent to all locations on the list. When a message is sent to a cascade list, the message is transmitted to the listed destination with the fewest messages enqueued. In the sample message control program, the TLIST entry D1, (a), represents a distribution list entry. The list should not include a TPROCESS entry for a COBOL application program.

INVLIST Macro: An instruction that creates an invitation list entry containing the invitation characters for the stations on the line (in the order in which they are to be invited to send messages), the INVLIST macro, 12, must be issued for each line in the system. However, one INVLIST macro suffices for all output-only lines to stations that do not use invitation sequences. Two INVLIST entries -- LIST1050, a , and LISTTWX, b -- appear in the sample program.

Note: Either a parameter of + in the INVLIST macro or an operator control command (see the section "Using TCAM Service Facilities" in this chapter) must initially activate a station for entering messages.

In the entry LIST1050, for example, 'T1 + 6215' indicates that the IBM 1050 terminal identified as T1 is active for entering messages. (6215 is the IBM 1050 transmission code representation of the polling characters A0 in hexadecimal notation.) Accordingly, the symbol 'T2A+' in the LISTTWX entry indicates an initially active line. (Note: The terminal name for the line, not the station, must be used.) For a TWX station, the '+' character would be followed by an ID sequence instead of the polling character used in the LIST1050 example. In the example, no ID sequence is given. The (CPUID) = operand in the INVLIST macro for the TWX terminal is required.

Designing the Message Handler

The major section in a message control program is the group of message handlers (MH), made up of sets of routines that examine and process control information in message headers (see Table 31) and perform the functions necessary in preparing message segments for forwarding to their destinations. There is usually a message handler for each line group or active application program. Each message handler usually contains both an incoming and an outgoing group.

A message may consist of two parts -- the header, or control, portion and the text portion -- depending on the application. The sample message control program shown in Figure 114 contains four message handlers, as listed below. Three of these message handlers are based on a message header containing the information described in the comments that immediately precede the first sample message handler, 13 . The fourth message handler in the sample MCP, MHAPPAPP, handles messages with no headers.

- A message handler (MH1050) for input from and output to the IBM 1050 Data Communications System Terminal.

- A message handler (MHTWX) for input from and output to the Teletypewriter Exchange (TWX).

- A message handler (MHTRMAPP) for input from and output to an application program running with terminals.

- A message handler (MHAPPAPP) for input from and output to an application program that simulates terminal input data. This type of message handler can be used for testing without terminals or for handling messages sent from one application program to another, as in the sample COBOL programs TESTTP1 (see Figure 116) and TESTTP2 (see Figure 117).

Two kinds of macro instructions that may be included in a message handler are functional macros and delimiter macros. Functional macros perform the specific operations necessary for messages directed to the message handler. Delimiter macros classify and identify sequences of functional macro instructions and then direct control to the appropriate sequence. Table 31 shows some of the functional macros that can be used with the delimiter macros in the incoming group and the outgoing group of the message handler. All of these macros are included in the sample message handler in Figure 114.

To decide which macro to place in which group, the user must understand which group is executed when. This is discussed in the description associated with Figure 113. The steps executed by a message handler are shown at the right-hand side of this figure. When messages are received from stations, the incoming group of a message handler for the line is executed before the outgoing group. However, when messages are sent to application programs, the outgoing group of the message handler for the application program is executed first. The decision boxes shown in Figure 113 are determined by the destination specified in the required FORWARD macro of a message handler (that is, if the destination is the name of a TPROCESS entry, processing is required in a an application program; if, however, the destination is the name of a TERMINAL macro, no more processing is required).

Table 31. Macros that can be coded in a Message Handler

Groups	Subgroups	Delimiter Macros	Functional Macros
Incoming Group		STARTMH*	
	Inheader Subgroup	INHDR*	CODE LOG SETSCAN MSGTYPE ORIGIN FORWARD TERRSET
	Inbuffer Subgroup	INBUF	CUTOFF MSGEDIT
	Inmessage Subgroup	INMSG	CANCELMG ERRORMSG
		INEND*	
Outgoing Group	Outheader Subgroup	OUTHDR	MSGFORM MSGTYPE MSGEDIT SETSCAN DATETIME SEQUENCE LOG SETEOF
	Outbuffer Subgroup	OUTBUF	MSGEDIT CODE
	Outmessage Subgroup	OUTMSG	HOLD ERRORMSG
		OUTEND*	

*The STARTMH macro is always required. If the message handler is to handle incoming messages, the INHDR, INEND, and OUTEND macros are also required. If the message handler is to handle outgoing messages, the OUTEND macro is required.

Note: For descriptions of other macros that can be coded in an MCP, see the publication IBM OS Telecommunications Access Method (TCAM) Programmer's Guide and Reference Manual.

A discussion of sample message handlers for terminal line groups appears below. For discussions of the MHTRMAPP and MHAPPAPP message handlers, see the sections "A Message Handler for an Application Program Running with Terminals" and "A Message Handler for an Application Program that Simulates Input Data."

A MESSAGE HANDLER FOR THE TERMINAL LINE GROUPS: Because the message handlers for the 1050 line and the TWX line are similar (except for the difference in line control characters and the use of the 1050 for error messages), the description of the message handler for the 1050 (MH1050) given below should also suffice for the TWX line group (MHTWX).

The Incoming Group: The first macro in the MH1050 message handler is STARTMH, 13 , in which the LC=OUT operand specifies that line control characters are to be removed. The first macro in the INHDR, 14 , subgroup (CODE), 15 , translates the incoming messages to EBCDIC. Then the LOG macro, 16 , records the header on the log data set. Even though the CODE macro is part of the INHDR subgroup, all buffers of the message are translated from line code to EBCDIC -- not just the first (header) buffer. In the normal case, unless the line code is EBCDIC, the CODE macro should be placed first, as in this example. A CODE macro must be issued before an ORIGIN macro, since the name in the header is checked against the terminal names, which are in EBCDIC. The name in the header, therefore, cannot be located unless it has first been translated. The same translation requirements apply to such macros as SETSCAN, 17 , in the example. In this case, if the C'$' in the message were not first translated to EBCDIC, the C'$' would have to be specified in line code.

The SETSCAN macro, 17 , sets the scan pointer to "$" in the header, and the MSGTYPE macros, 18 , that follow check the character in the next field (with fields separated by at least one blank character) for one of the four codes that represent possible message destinations. If the scan yields a match between a field in the incoming message and the code for one of the MSGTYPE macros, the macros between this MSGTYPE macro and the next MSGTYPE macro are executed. Control is then given to the next subgroup (INBUF), 22 . When a MSGTYPE match is found, the ORIGIN macro, 19 , is issued. The FORWARD macro, 20 , which is always required, transmits the message to the destination specified.

If there is no match with any of the operands specified in the MSGTYPE macros, the last MSGTYPE macro, which has a blank operand field, is executed. The required FORWARD macro follows, and the TERRSET macro, 21 , sets the user error bits in the error record for the message.

In the INBUF subgroup, 22 , the CUTOFF macro, 23 , limits the size of the incoming messages and checks for station malfunction. The insertion of the RECDEL character by the MSGEDIT macro, 24 , allows for record delimiters in the message, needed when the COBOL program reads in segment mode. The INMSG subgroup, 25 , checks the error bits in the error record for this message and either cancels the message via the CANCELMG macro, 26 , and/or sends an error message to the 1050 terminal using the ERRORMSG macro, 27 . The INEND macro, 28 , a required delimiter macro, signifies the end of the incoming groups.

The Outgoing Group: The macros discussed below, known as the outgoing group, are executed when messages are transmitted to the 1050 terminal. In the OUTHDR subgroup, 29 , the MSGFORM macro, 30 , causes line control characters to be inserted in the outgoing message. (Unless the user provides line control characters himself, this macro must be coded.) The MSGTYPE macro determines the type of message, so that a message can be processed either as an ordinary message or as an error message.

For every error message, the SETSCAN macro returns the scan pointer to the beginning of the message, and the MSGEDIT macro inserts the "NL" character before the message text. Processing of error messages resumes in the OUTBUF subgroup, 33 , of the message handler.

For the non-error messages, the MSGEDIT macro also inserts "NL" at the beginning of the message. Then the SETSCAN macro sets the scan pointer to the period at the end of the message header so that pertinent information can be inserted there. The DATETIME macro, 31 , records in the message being sent the date and time this macro is executed. The SEQUENCE macro, 32 , inserts a sequence number, and the LOG macro records the control information contained in the message header.

In the OUTBUF subgroup, 33 , of this message handler, the MSGEDIT macro inserts an "NL" character for every record delimiter character in the message. Because in the incoming group the RECDEL character is inserted for every "NL" and "LF" character, for a message that is simply transmitted from one terminal to another the message handler appears to send the same line control characters it receives. For a message sent by a COBOL program, on the other hand, wether or not record delimiter characters remain depends on the mode specified in the RECEIVE or SEND statement. (That is, when the programmer receives a message in segment mode, the record delimiter character is removed; when the programmer receives a message in message mode, the record delimiter is not removed. Accordingly, when the programmer sends a message in segment mode, the record delimiter character is added; when the programmer sends a message in message mode, the record delimiter is not added.) The next MSGEDIT macro inserts 13 idle characters after every "NL" character placed in the message, to allow the terminal sufficient time to return its carriage before receiving the next line. Finally, the CODE macro translates from EBCDIC to line code when no more handling is required with macros that operate in EBCDIC.

Like the INMSG subgroup (see "The Incoming Group"), the OUTMSG subgroup, 34 , checks the error bit in the error record for the message and transmits error messages, if any, to the 1050 terminal. The HOLD macro, 35 , is invoked only if there are hardware errors. Accordingly, a terminal placed in HOLD status is not released until an operator control message is issued. The OUTEND macro, 36 , signifies the end of the outgoing group.

A MESSAGE HANDLER FOR AN APPLICATION PROGRAM RUNNING WITH TERMINALS: The MHTRMAPP message handler handles messages transmitted by a terminal for the application program that is sending and receiving messages from terminals. Like the message handler discussed earlier, MHTRMAPP includes both an incoming group and an outgoing group.

In this message handler, because messages are sent to the application program from a terminal, the outgoing group headed by the OUTHDR macro, 37 , is executed first. The first macro (MSGEDIT) deletes any characters (for example, "NL", "CR", or "LF") that have preceded "$" in the header. This step is necessary because of the application program's expectation of receiving a fixed-length header beginning with "$". The next macro (SETSCAN) sets the scan pointer over the "$" and the MSGTYPE field. Then the SEQUENCE macro numbers the messages sent to the application program, and the LOG macro records the information contained in the message header.

The next SETSCAN macro sets the scan pointer over the source field in the header so that it points instead to the EOF field. The SETEOF macro identifies the last message in a data file being processed by an application program. If the character specified at the location pointed to by the scan pointer (and given as an operand in the SETEOF macro) is "F", the first RECEIVE statement issued by the COBOL program after receipt of the message causes the MCP to enter an application program EODAD routine. As far as the COBOL user is concerned, this section sets the "ETI" indicator in the field referred to by the END KEY clause in the input communication description (CD). The OUTMSG subgroup is not included in this message handler because it is not executed for messages sent to an application program. Nevertheless, the OUTEND delimiter macro signifies the end of the outgoing group.

The macros in the incoming group of this message handler, headed by the INHDR macro, 38 , are executed when messages are received from the COBOL program. The LOG macro records the information contained in the header, and the FORWARD macro, which is always required, specifies "DEST=PUT" as the message destination. This will cause the message to be forwarded to the destination the COBOL program has indicated in the output CD. The INMSG subgroup that follows checks to see whether sufficient buffer units are available for the message and verifies that the destination specified is valid. The INEND delimiter macro then specifies the end of the incoming group.

A MESSAGE HANDLER FOR AN APPLICATION PROGRAM THAT SIMULATES TERMINAL INPUT DATA: The MHAPPAPP message handler is for messages having no header. As a result, the only macro in the outgoing group is the delimiter macro OUTEND, 39 , which is always required.

The incoming group contains both the INHDR, 40 , subgroup, containing the required FORWARD macro, and the INMSG subgroup, which checks for availability of sufficient buffer units and verifies that the destination specified is valid. The required INEND delimiter macro is present.

PUTTING THE MCP TOGETHER

This section names the parts of the MCP described earlier, explaining how to arrange them in relation to one another and how to assemble, link-edit, and execute a TCAM MCP. The five sections of an MCP include those previously discussed -- an activation and deactivation section, a data set definition section, a terminal and line control area section, a message handler section -- and an optional user routine section (that is, user subroutines called by a message handler, as well as exit routines referred to by the INTRO macro, by DCB macros, and by the STARTMH macro). The only stipulation about ordering these sections is that the activation and deactivation section must come first.

ASSEMBLING, LINK-EDITING, AND EXECUTING AN MCP

The assembly, link-edit, and execution steps of a TCAM MCP are similar to these steps for any other problem program running under OS. The job control statements given below for these three steps are guidelines only.

Assembling an MCP

A typical control card sequence for assembling a TCAM MCP is as follows:

```
//ASSEMBLY  JOB   MSGLEVEL=1
//STEP1     EXEC ASMFC
//ASM.SYSIN DD    *

{MCP Source Deck}
```

Link-Editing an MCP

The following is a typical control card sequence for link-editing an MCP:

```
//LINKEDIT  JOB   MSGLEVEL=1
//STEP1     EXEC  PGM=IEWL,PARM='XREF,LIST,
//                LET',REGION=96K
//SYSPRINT  DD    SYSOUT=A
//SYSUT1    DD    UNIT=SYSDA,
//                SPACE=(1024,(200,20))
//SYSLMOD   DD    DSNAME=SYS1.TCAMLIB,
                  DISP=OLD
//SYSLIB    DD    DSNAME=SYS1.TELCMLIB,     X
                  DISP=OLD
//          DD    DSNAME=SYS1.MACLIB,       X
//                DISP=OLD
//SYSLIN    DD    *

{MCP Object Module}

            NAME  TCAMPROG(R)
```

Note: In this example, the MCP load module is to be placed in a user-created private library called SYS1.TCAMLIB.

Executing an MCP

The TCAM MCP is normally executed as the highest-priority task in the highest-priority partition or region in the system. It may have an equal priority, but it should never be assigned a lower priority. A typical control card sequence for executing an MCP is the following:

```
//EXECMCP   JOB  'EXECUTE MCP',MSGLEVEL=1,  X
//               PRTY=12
//GOSTEP    EXEC PGM=TCAMPROG,REGION=100K
//STEPLIB   DD   DSNAME=SYS1.TCAMLIB,       X
//               DISP=SHR
//DD1050    DD   UNIT=025
//          DD   UNIT=026
//          DD   UNIT=027
//DD2740    DD   UNIT=015
//          DD   UNIT=016
//          DD   UNIT=017
//QFILE     DD   DSNAME=MSGQ,DISP=OLD
//LOGFILE   DD   DSNAME=LOGF,DISP=OLD
//SYSABEND  DD   SYSOUT=A
```

Notes:

1. In this example, the MCP has two line group data sets, each containing three lines; no checkpoint facility is included. (For a discussion of the DD cards for a checkpoint data set, see the section "Defining the Checkpoint Data Set.")

2. The QFILE DD statement is for a message queues data set residing on disk; QFILE is the name specified in the DDNAME= operand of the DCB macro for this data set, and MSGQ is the name of the data set specified by the DSNAME operand of the IEDQDATA DD statement for the IEDQXA utility used to preformat disk message queues data sets residing on disk (see the section "Defining the Message Queues Data Sets").

3. If the data set is not cataloged, the UNIT= and VOLUME= operands must be included in the DD statement for the disk message queues data set.

4. The //LOGFILE DD card must be included if the LOG data set is to be used.

Defining the Checkpoint Data Set: One DD statement that may or may not catalog the data set must be issued for the checkpoint data set. However, if it is not cataloged, the user should allocate the data set by specifying DISP=(NEW,KEEP) as in the example and subsequent uses of the data set must contain the UNIT= and VOL=SER=keyword operands, given below.

```
//CFILE      DD     DSNAME=CPDS,UNIT=2314,   X
//                  VOL=SER,DB197,           X
//                  SPACE=(TRK,(5)),         X
//                  DISP=(NEW,KEEP)
```

After a checkpoint data set is set up and the MCP has terminated normally, the programmer should replace the DD card described above with one of the following type:

```
//CFILE      DD     DSNAME=CPDS,DISP=OLD,
                    VOL=SER=DB197,UNIT=2314
```

Defining Line Group Data Sets: The user must include in his job control statements at least one DD statement for each line group data set, but he has two options for handling these definitions.

1. If a UNITNAME macro is issued for a line group at system generation time, then a single DD statement may be issued for this line group at MCP execution time. For example, a UNITNAME macro could be issued to define a group of lines as follows:

   ```
   UNITNAME    UNIT=(040,041)
   NAME=GROUPLINE
   ```

   Where the two numerals in the UNIT=operand parameter represent the hardware addresses of two lines in a line group. At execution time for the MCP, the following DD statement might be issued for this line group:

   ```
   //LNS  DD  UNIT=(GROUPLINE,2)
   ```

   Where the line group data set would be made up of two lines defined by the UNITNAME macro.

2. A DD statement may be issued for each line in a line group, as in the DD cards for line group DD1050 and line group DD2740 in the sample JCL statements given in section "Executing an MCP."

3. The following DD cards were used to execute the sample message control program shown in Figure 112.

```
//LN1  DD  UNIT=040 (for the 1050 terminal)
//LN2  DD  UNIT=041 (for the TWX terminal)
```

Defining the Message Queues Data Sets: The number of message queues data sets required for an MCP depends on the types of queues, which depend on the application. TCAM supports three types of data sets -- a main storage data set, a reusable disk set, and a nonreusable disk data set. (For checklists governing specification of the three types of message queues data sets, see the publication IBM OS Telecommunications Access Method (TCAM) Programmer's Guide and Reference Manual.)

TCAM expects the disk message queues (both reusable and nonreusable) to be totally preformatted. The COBOL user should engage the IEDQXA utility routine to perform this task prior to initially of a set of job control statement used to invoke this routine.

Note: The value given in the KEYLEN parameter must be the same as that specified in the KEYLEN operand of the INTRO macro (see the section "Defining the Buffers").

```
//JOBNAME  JOB  user information
//FORMATQ  EXEC PGM=IEDQXA
//SYSPRINT DD   SYSOUT=A
//IEDQDATA DD   DSN=MSGQ,DISP=(,CATLG),       X
//              SPACE=(CYL,(5,5),,CONTIG) X
//              UNIT=(2311,1),               X
//              VOL=SER=333333,              X
//              DCB=(,KEYLEN=100)
/*
```

WRITING A TCAM-COMPATIBLE COBOL PROGRAM

Two of the chief processing applications for which COBOL programs can be written are inquiry processing and processing collected data. An inquiry-processing COBOL program receives messages from stations, processes the data, and then sends replies to the originating stations. Depending on the inquiry, the COBOL program may transmit either the information requested or a message stating that this information is unavailable and telling when it can be provided. The COBOL program that simply processes data collected by a message control program can either operate concurrently with the collection of data by the MCP or be loaded and initiated at a later time.

The sample COBOL TP program TESTTP2 (shown in Figure 119) represents an application of processing data. This program accepts messages transmitted from a remote station, formats the message, and then transmits each complete message to the destination specified. The COBOL program TESTTP1 (shown in Figure 118) simulates terminal input data. The user can, therefore, test an installation-written COBOL TP program by running it with the sample MCP and TESTTP1.

TESTING A COBOL TP PROGRAM

Depending on the status of an installation's teleprocessing system, the user can code any one of three sets of JCL to run a teleprocessing job. A system that is fully operational has a message control program with a user-designated message handler for each type of teleprocessing situation expected, as well as remote terminal hook-ups. The user whose system is only partially developed or is still in the design stage may, nevertheless, wish to test COBOL teleprocessing programs using BSAM.

Accordingly, the JCL shown in Figure 115 is for a strictly BSAM situation (that is, for a teleprocessing program that is to be run without either an MCP or hardware); the JCL shown in Figure 116 is for a quasi-terminal situation (that is, with MCP but without hardware); and the JCL shown in Figure 117 is for a teleprocessing job running with a remote terminal. For both the non-terminal and the quasi-terminal situation an input data set must be created. To run a COBOL teleprocessing program with a terminal hook-up, only the teleprocessing program itself is needed.

```
//TESTTP1       JOB     user information
//              EXEC    UCOBFCLG
//COB.SYSIN     DD      *

                {Source deck for TESTTP1 program (Figure 118)}
/*
//GO.TSTTP      DD¹     UNIT=2400,LABEL=(,NL),VOL=SER=NI195,DCB=(LRECL=50,BLKSIZE=50,     X
//                      RECFM=F,DEN=2)
//GO.COBTPOUT DD²       UNIT=2314,VOL=SER=231400,DSN=&1,DISP=(NEW,PASS),SPACE=(CYL,(1,2))
//TESTTP2       JOB     user information
//              EXEC    UCOBFCLG
//COB.SYSIN     DD      *

                {Source deck for TESTTP2 program (Figure 119)}
/*
//GO.Q1         DD³     DSN=P1,VOL=SER=231400,UNIT=2314,DISP=(OLD,PASS)
//GO.COBTPOUT DD⁴       DSN=P2,VOL=SER=231400,UNIT=2314,DISP=(NEW,PASS),SPACE=(CYL,(1,2))
//DUMPIT        JOB     user information
//              EXEC⁵   PGM=IMASP=AP
//SYSLIB        DD      DSNAME=data set to be printed,UNIT=2314,VOL=SER=231400,           X
//                      DISP=OLD,DCB=DSORG=PS
//SYSPRINT      DD      SYSOUT=A
//SYSIN         DD      *
                ABSDUMP ALL
/*
```

Notes:

1. Input sequential file with records of 50 characters each (BSAM JCL).

2. Output data set that simulates sending messages to a terminal named 'P1'.

3. Input data set that simulates reading messages from a terminal named 'P1'.

4. Output data set that simulates sending messages to a terminal named 'P2'.

5. This job prints out the records in the simulated data set. For further information, see the publication IBM OS Service Aids, Order No. GC28-6719.

Figure 115. Sample JCL for Running a Teleprocessing Job without Hardware.

```
//TESTTP1       JOB    user information
//              EXEC   UCOBFCLG
//COB.SYSIN     DD     *

                {Source deck for TESTTP1 program (Figure 118)}
/*
//GO.TSTTP      DD¹    UNIT=2400,LABEL=(,NL),VOL=SER=NI195,DCB=(LRECL=50,BLKSIZE=50,     X
//                     RECFM=F,DEN=2)
//GO.COBTPOUT DD²      QNAME=POUT1
//TESTTP2       JOB    user information
//              EXEC   UCOBFCLG
//COB.SYSIN     DD     *

                {Source deck for TESTTP2 program (Figure 119)}
/*
//GO.Q1         DD³    QNAME=P1
//GO.COBTPOUT DD⁴      QNAME=POUT1
//DUMPIT        JOB    user information
//              EXEC   IEDQXC
//DISQ01        DD⁵    DSN=MSGQ,VOL=SER=DB197,UNIT=2314,DISP=SHR
//SYSPRINT      DD     SYSOUT=A
/*
```

Notes:

1. Input sequential file with records of 50 characters each. (This is the same JCL as in BSAM.)

2. Output is sent to an MCP message queue named 'P1', which is defined for processing by a COBOL program.

3. ~~Input is received from the MCP message queue named 'P1'.~~

4. Output is sent to an MCP message queue named 'P2', which is defined for processing by a COBOL program.

5. This job prints out records in the MSGQ queue. For further information, see the publication IBM OS Telecommunications Access Method (TCAM) Programmer's Guide and Reference Material, Order No. GC30-2024.

Figure 116. Sample JCL for Running a Teleprocessing Job in a Quasi-Terminal Environment.

```
//TESTTP2       JOB  user information
//              EXEC UCOBFCLG
//COB.SYSIN     DD   *
                {Source deck for TESTTP2 program (Figure 119)}
/*
//GO.Q1         DD¹  QNAME=P1
//GO.COBTPOUT DD²    QNAME=POUT1
```

Notes:

1. The input is received from the MCP message queue 'P1'.

2. The output is sent to an MCP message queue defined for a terminal.

Figure 117. Sample JCL for Running a Teleprocessing Job with a Remote Terminal.

COMMUNICATING BETWEEN A COBOL PROGRAM AND THE MCP

The TCAM message control program routes messages between a COBOL teleprocessing program and remote stations. Because the MCP performs the input/output operations necessary for the COBOL teleprocessing program, the user must establish an interface between these two programs by doing the following:

- Defining the interface
- Activating the interface
- Transferring messages between the COBOL program and the MCP
- Deactivating the interface

In each of the sections that follow, both COBOL statements and TCAM macros, as well as their relationship, are described as appropriate. The encircled numerals in this discussion refer to the sections similarly labeled in the sample COBOL teleprocessing program TESTTP2 shown in Figure 119.

Defining the Interface

The Communication Section in the COBOL program and the PCB and TPROCESS macros in the message control program set up the interface between the two programs.

Defining Input and Output Data Sets: At execution time, one DD statement must be provided for each SYMBOLIC QUEUE name specified in an input CD. A prototype of such a statement is

```
//ddname    DD QNAME=procname
```

where "procname" is the name of the process entry in the terminal table to which this entry refers (see the section "Defining Terminal and Line Control Areas").

As in the following example from the sample program TESTTP2 (shown in Figure 92.7), the COBOL user should specify the SYMBOLIC QUEUE names for the ddnames and the corresponding TPROCESS entry names for the procname. In TESTTP2, one input queue and one output queue are defined. The DD card for the input queue is:

```
//Q1  DD  QNAME=P1
```

Similarly, the COBOL programmer must provide a DD card for the process entry defined in the MCP to send messages from the COBOL program. As in the example that follows, the ddname must be COBTPOUT, and the QNAME must be the name of a TPROCESS entry for an output process entry. The MCP should have a queue defined as P2, but no DD statement is needed for this queue.

```
//COBTPOUT  DD QNAME=POUT1
```

The user should notice that these destination queues are among those specified in the message control program via the TPROCESS macro. For examples of these TPROCESS entries, see the discussion the "TPROCESS Macro" under "Defining Terminal and Line Control Areas".

In a COBOL TP program, the user can specify one through three levels of subqueues from which data can be received. This feature allows the COBOL object program, at execution time, to make use of pre-defined queue structures, and to access all or parts of such structures. If pre-defined queue structure are used, each lowest level subqueue name in the structure corresponds to a TCAM queue and must, therefore, have an associated DD card pointing to a TPROCESS entry in the MCP terminal table. Each subqueue must be defined in the communication description (CD) of the COBOL source program and have been defined earlier in a queue structure description (see the sections "Queue Structure Considerations" and "Communication Section" in the chapter on "Programming Techniques").

If the user wishes to access the next message in the queue structure, regardless of which sub-queue that message may be in, he specifies the queue name only and initializes the sub-queue names to SPACES. The MCP, when supplying the message, returns to the COBOL object program any applicable sub-queue names via the data items in the associated input CD. If, however, the programmer desires the next message in a given sub-queue, he must specify both the queue name and any applicable sub-queue names. Once a program has begun receiving any part of a message from a queue (or sub-queue), subsequent requests must specify all applicable names until end of message (EMI) is indicated.

```
001010 IDENTIFICATION DIVISION.
001020 PROGRAM-ID.
001030     TESTTP1.
001080 DATE-COMPILED. DEC  9,1971
001100 REMARKS.        THE SAMPLE COBOL TELEPROCESSING PROGRAM THAT
           FOLLOWS SERVES AS A SIMPLE ILLUSTRATION OF THE COBOL TELE-
           PROCESSING FEATURE.  THIS PROGRAM READS IN A FILE OF 50-
           CHARACTER MESSAGES, TRANSMITTING THEM ONE BY ONE TO THE
           SPECIFIED DESTINATION.
001160
001170 ENVIRONMENT DIVISION.
001180 INPUT-OUTPUT SECTION.
001190 FILE-CONTROL.
001200     SELECT MASTER-FILE
001210     ASSIGN TO UT-2400-S-TSTTP.
002010 DATA DIVISION.
002020 FILE SECTION.
002030 FD  MASTER-FILE
002040     RECORDING MODE IS F
002050     LABEL RECORDS ARE STANDARD
002060     DATA RECORD IS RECORD1.
002070 01  RECORD1  PIC  X(50).
003010 WORKING-STORAGE SECTION.
003110
003120 01  IDENT-SEND.
003130     02  I-SEND   PIC   X(50).
      *  SET UP A WORK AREA OF 50 CHARACTERS
003150
      *  THE COMMUNICATION SECTION MUST BE SPECIFIED IN A COBOL PROGRAM
      *  THAT IS TO UTILIZE THE COBOL TELEPROCESSING FEATURE.  THE
      *  COMMUNICATION DESCRIPTION (CD) ENTRIES THAT APPEAR IN THIS
      *  GROUP OF SOURCE STATEMENTS ESTABLISH THE INTERFACE BETWEEN THE
      *  COBOL OBJECT PROGRAM AND THE MESSAGE CONTROL PROGRAM (MCP).
004010 COMMUNICATION SECTION.
004120 CD  CDNAME-OUT FOR OUTPUT
004130     TEXT LENGTH IS TEXTLNTH-OUT
      *  SPECIFY LENGTH OF OUTPUT MESSAGE.
004140     STATUS KEY IS STATKY-OUT
      *  PROVIDE INFORMATION ON MESSAGE STATUS.
004150     ERROR KEY IS ERRKY
      *  PROVIDE ERROR INFORMATION.
004160     SYMBOLIC DESTINATION IS SYMDES.
      *  SPECIFY OUTPUT QUEUE.
004170
005010 PROCEDURE DIVISION.
005020 START-JOB.
005030     DISPLAY 'BEGIN TESTTP1'.
      *  START THE COBOL TELEPROCESSING PROGRAM.
005040     OPEN INPUT MASTER-FILE.
      *  OPEN THE INPUT FILE.
005045 READ-ROUTINE.
005050     READ MASTER-FILE INTO IDENT-SEND
005060         AT END GO TO END-ROUTINE.
      *  PLACE INPUT RECORDS IN A WORK AREA UNTIL END OF FILE IS
      *  REACHED.
006010 SEND-ROUTINE1.
006020     MOVE 'P1' TO SYMDES.
      *  SET UP OUTPUT DESTINATION.
006040     MOVE 50 TO TEXTLNTH-OUT.
      *  IDENTIFY MESSAGE LENGTH AS 50.
006060     SEND CDNAME-OUT FROM IDENT-SEND WITH EMI.
      *  TRANSMIT A COMPLETE MESSAGE.
006070     PERFORM CHECK-SEND THRU CHECK-EXIT.
006080     GO TO READ-ROUTINE.
```

Figure 118. Creating a TCAM Data Set for Testing without Terminals (Part 1 of 2)

```
       *  EXECUTE USER-WRITTEN CODE FOR CHECKING ON THE SUCCESSFUL
       *  COMPLETION OF MESSAGE TRANSMISSION.  IF END OF FILE IS
       *  REACHED, GO TO END-OF-JOB ROUTINE.  OTHERWISE, GET THE NEXT
       *  RECORD.
  008010 CHECK-SEND.
  008020*
  008021*
  008022*
  008030*  USER CHECKING ROUTINE FOR DETERMINING THE
  008040*  SUCCESSFUL COMPLETION OF THE SEND.
  008050*
  008160
  008170 CHECK-EXIT.
  008180     EXIT.
**008180*
  008190
  011110 END-ROUTINE.
  011111     CLOSE MASTER-FILE.
       *  CLOSE THE INPUT FILE.
  011150     DISPLAY 'SUCCESSFUL END OF TESTTP1'.
       *  TERMINATE THE PROGRAM.
  011160     STOP RUN.
```

Figure 118. Creating a TCAM Data Set for Testing without Terminals (Part 2 of 2)

```
    001010 IDENTIFICATION DIVISION.
    001020 PROGRAM-ID.
    001030     TESTTP2.
    001080 DATE-COMPILED. DEC  9,1971
    001100 REMARKS.       THE SAMPLE COBOL TELEPROCESSING PROGRAM THAT
               FOLLOWS SERVES AS A SIMPLE ILLUSTRATION OF THE COBOL TELE-
               PROCESSING FEATURE.  THIS PROGRAM SETS UP A DESTINATION
               FOR INCOMING MESSAGES, AND THEN READS THEM, ONE BY ONE,
               INTO A WORK AREA.  THE PROGRAM BUILDS 50-CHARACTER MESSAGES
               AND SENDS THEM TO THE MCP WITH THE END-OF-MESSAGE (EMI)
               INDICATOR.  WHEN ALL THE INCOMING MESSAGES HAVE BEEN PRO-
               CESSED, THE MESSAGE 'SUCCESSFUL END OF TESTTP2' IS PRINTED
               ON THE CONSOLE, AND THE PROGRAM IS TERMINATED.
    001120
    001130
    001170 ENVIRONMENT DIVISION.
    001180 CONFIGURATION SECTION.
    001190 INPUT-OUTPUT SECTION.
    001200
    002010 DATA DIVISION.
           WORKING-STORAGE SECTION.
    003110
    003120 01  IDENT-SEND.
(1) 003130     02  I-SEND   PIC  X(50).
    003160 01  IDENT-REC.
    003170     02  I-REC  PIC  X(50).
    003190
          *  THE COMMUNICATION SECTION MUST BE SPECIFIED IN A COBOL PROGRAM
          *  THAT IS TO UTILIZE THE COBOL TELEPROCESSING FEATURE.  THE
          *  COMMUNICATION DESCRIPTION (CD) ENTRIES THAT APPEAR IN THIS
          *  GROUP OF SOURCE STATEMENTS ESTABLISH THE INTERFACE BETWEEN THE
          *  COBOL OBJECT PROGRAM AND THE MESSAGE CONTROL PROGRAM (MCP).
           COMMUNICATION SECTION.
    004120 CD  CDNAME-OUT FOR OUTPUT
    004130     TEXT LENGTH IS TEXTLNTH-OUT
          *  SPECIFY LENGTH OF OUTPUT MESSAGE.
    004140     STATUS KEY IS STATKY-OUT
          *  PROVIDE INFORMATION ON OUTPUT MESSAGE STATUS.
    004150     ERROR KEY IS ERRKY
          *  PROVIDE ERROR INFORMATION.
    004160     SYMBOLIC DESTINATION IS SYMDES.
          *  SPECIFY OUTPUT QUEUE.
    004170
  **004020 CD  CDNAME-IN FOR INPUT
    004030     SYMBOLIC QUEUE IS SYMQ
          *  IDENTIFY INPUT MESSAGE QUEUE.
    004040     MESSAGE DATE IS MSGDATE
    004050     MESSAGE TIME IS MSGTIME
          *  PROVIDE DATE AND TIME OF RECEIPT OF MESSAGE.
    004060     SYMBOLIC SOURCE IS SYMSOURCE
          *  IDENTIFY THE MESSAGE SOURCE.
    004070     TEXT LENGTH IS TEXTLNTH-IN
          *  SPECIFY THE EXPECTED LENGTH OF INPUT MESSAGE.
    004080     END KEY IS ENDKY
          *  PROVIDE CODE FOR ACTIVATING END-OF-JOB ROUTINE.
          ** FOR A RECEIVE MESSAGE:
          *  A CODE OF 3 INDICATES END OF TRANSMISSION (ETI).
          *  A CODE OF 2 INDICATES END OF MESSAGE (EMI).
          *  A CODE OF 0 INDICATES RECEIPT OF LESS THAN A MESSAGE.
          ** FOR A RECEIVE SEGMENT:
          *  A CODE OF 3 INDICATES END OF TRANSMISSION (ETI).
          *  A CODE OF 2 INDICATES END OF MESSAGE (EMI).
          *  A CODE OF 1 INDICATES END OF SEGMENT (ESI)
          *  A CODE OF 0 INDICATES RECEIPT OF LESS THAN A SEGMENT.
```

Figure 119. A COBOL Program That Processes TCAM Messages
(Part 1 of 2)

```
        ** HIERARCHY -- 0, ESI, EMI, ETI-WHEN MORE THAN ONE CONCURRENTLY-
         *    HIGH LEVEL APPEARS.
  004090      STATUS KEY IS STATKY-IN
         *  PROVIDE INFORMATION ON INPUT MESSAGE STATUS.
  004100      QUEUE DEPTH IS QDEPTH.
         *  SPECIFY DEPTH OF INPUT QUEUE.
  004110
**002100
          PROCEDURE DIVISION.
2             DISPLAY 'BEGIN TESTTP2'.
          RECV-DATA.
  009040      MOVE 'Q1' TO SYMQ.
         *  SET UP INPUT DESTINATION.
3 009050      RECEIVE CDNAME-IN MESSAGE INTO IDENT-REC
  009055          NO DATA GO TO END-ROUTINE.
         *  ACCEPT INPUT MESSAGES, ONE BY ONE, AS ON A SEQUENTIAL FILE.
         *  WHEN ALL MESSAGES HAVE BEEN PROCESSED, INVOKE END-OF-JOB
         *  ROUTINE.
  009060 CHECK-RECEIVE.
  009070*
  009080*  USER CHECKING ROUTINE FOR DETERMINING THE
  009090*  SUCCESSFUL COMPLETION OF THE RECEIVE.
  009100*
  009110 PROCESS-DATA.
  009120*
  009130*  USER ROUTINE TO BUILD MESSAGE TO BE SENT.
  009140*
**006010 SEND-ROUTINE1.
  006020      MOVE 'P2' TO SYMDES.
         *  SET UP OUTPUT DESTINATION.
         *  NOTE:  FOR THE NON-TERMINAL AND PARTIAL TERMINAL SITUATIONS,
         *  'P2' SHOULD BE SPECIFIED AS THE SYMBOLIC DESTINATION.  FOR
         *  A COBOL PROGRAM RUNNING WITH TERMINALS, 'T1' SHOULD BE
         *  SPECIFIED.
  006040      MOVE 50 TO TEXTLNTH-OUT.
         *  SPECIFY LENGTH OF OUTPUT MESSAGES.
4 006060      SEND CDNAME-OUT FROM IDENT-SEND WITH EMI.
         *  TRANSMIT FORMATTED MESSAGE, WITH THE CODE FOR A COMPLETE
         *  MESSAGE.
  006070      PERFORM CHECK-SEND THRU CHECK-EXIT.
         *  INVOKE USER-WRITTEN ROUTINE FOR CHECKING MESSAGE TRANSMISSION.
         *  ACCEPT THE NEXT MESSAGE FROM THE INPUT QUEUE.
  006090      GO TO RECV-DATA.
  007120
  008010 CHECK-SEND.
  008020*
  008030*  USER CHECKING ROUTINE FOR DETERMINING THE
  008040*  SUCCESSFUL COMPLETION OF THE SEND.
  008050*
  008170 CHECK-EXIT.
  008180      EXIT.
  008190
  011110 END-ROUTINE.
  011150      DISPLAY 'SUCCESSFUL END OF TESTTP2'.
5 011160      STOP RUN.
```

Figure 119. A COBOL Program That Processes TCAM Messages (Part 2 of 2)

Defining Process Control Blocks: In the MCP the user must also code a process control block (PCB) for each active application program running with the MCP. The PCB macro specifies the name of the PCB process control block generated by the macro. The process control block is referred to in the TPROCESS macro (see "Defining the MCP Data Sets and Process Control Blocks").

Activating the Interface

The COBOL programmer coding a program for a teleprocessing application initializes work areas, 1 , and activates the COBOL program as for any other OS application. In this application, the job begins with the use of the DISPLAY statement "BEGIN TESTTP2," 2. The COBOL programmer need not be concerned with how the interface is activated. The interface is activated when the first RECEIVE or SEND statement is issued.

Transferring Messages between the COBOL Program and the MCP

TCAM enables the application programmer to obtain messages from the MCP and to return response messages to the MCP. Specifically, the COBOL programmer can use either the RECEIVE statement or the SEND statement to transfer data between the MCP and the COBOL program, depending on the direction of the flow of data.

The RECEIVE Statement: This COBOL source statement causes transmission of message data from an input queue to a user-specified work area in the COBOL program. In the sample COBOL teleprocessing program shown in Figure 119, the RECEIVE statement, 3 , transfers data from the input queue referred to by SYMQ to a work area. The COBOL sentence before the RECEIVE statement is "MOVE 'Q1' TO SYMQ," so the data is received from Q1.

The SEND Statement: The COBOL source statement causes data from the COBOL program to be placed in an output queue for subsequent transmission. Accordingly, when the outgoing message has been formatted, the sample SEND statement, 4 , transmits it to the output destination referred to by SYMDES. The end-of-message indicator (EMI) signals a complete message. The first sentence in the paragraph labeled "SEND-ROUTINE1" is "MOVE 'P2' TO SYMDES," so the data is sent to P2.

Notes:

- For an additional example of the format of the RECEIVE statement and the SEND statement, see the section "Procedure Division" in the chapter on "Programming Techniques".
- The amount of data transferred from the MCP to a COBOL program by a single RECEIVE statement, or transferred from an application program to the MCP by a single SEND statement, is called a "work unit". Each work unit is processed in a user-designated work area in the COBOL program.

Deactivating the Interface

As in all American National Standard COBOL programs, the teleprocessing application user returns control to the system by issuing a STOP RUN statement, 5.

Note: So that the COBOL program can give control to the STOP RUN statement, the MCP writer should include in the message header a special code for the COBOL program. Although the sample MCP (Figure 114) has an action code field which includes such a code in the section of comments immediately preceding the MH1050 message handler, 13 Figure 119 gives control to the STOP RUN statement only when there is no more data. This technique is acceptable for a COBOL program that receives a fixed amount of data, i.e., a program that is not continually looping waiting for data.

Additional Interface Considerations

The information that follows is a summary of miscellaneous recommendations and/or restrictions that apply to the communication between the message control program and the COBOL application program.

1. The parameter DATE=YES must be coded in all input TPROCESS entries whose destination is a COBOL program and the parameter is also required in the PCB macro referenced by the TPROCESS macro. Inclusion of this parameter causes the date and time of message entry to be placed in the MESSAGE DATE and MESSAGE TIME clauses of the input CD (see "Communication Section" in the chapter entitled "Programming Techniques").

2. The RECDEL= parameter must be coded in the TPROCESS macro of the MCP if the COBOL programmer is to accept (via the RECEIVE statement) or transmit (via the SEND statement) data in SEGMENT mode. The user may either include in the incoming message the delimiter specified in this parameter or insert it via a MSGEDIT macro (see the section "Designing the Message Handler" in this chapter).

3. The INITIATE macro cannot be used in a message handler for messages whose destination is a COBOL program. This macro would cause the MCP to transmit segments of a message to a destination queue before receiving the complete message. American National Standard COBOL, on the other hand, assumes that a complete message has been enqueued.

4. American National Standard COBOL removes the last character of a message if it is X'37' (which is the EBCDIC representation for the EOT character). This is the last character of a message from a terminal that has been translated in the MH of the MCP via the CODE macro, or that is not processed in conversational mode (which would have been specified by coding CONV=YES in the STARTMH macro).

5. An execution of the RECEIVE statement with the SEGMENT option results in the setting of the ESI (end of segment) indicator if end of segment is reached. When end of segment is also end of message, an end key of 2 indicating EMI is given. If the last two characters in the message are an end segment indicator and the end of message character, the user will receive the indication first. Another RECEIVE will be necessary to receive the EMI indication. The RECEIVE from the EMI indication will set the TEXT LENGTH field of the input CD to zeros.

6. For a message transmitted from a COBOL program to the location specified in the SYMBOLIC DESTINATION clause of an output CD, the FORWARD macro in the inheader subgroup of the MH for the COBOL program must specify DEST=PUT as its operand.

USING TCAM SERVICE FACILITIES

TCAM allows for a variety of services in support of a COBOL teleprocessing system. Some of these services are provided automatically; others the user must specify. Some of the TCAM services are the following: operator control, error recovery, checkpoint/restart, message logging, debugging aids, and an on-line test feature. All of these TCAM aids are discussed in the publication <u>IBM OS Telecommunications Access Method (TCAM) Programmer's Guide and Reference Manual</u>. Some of these TCAM services have already been discussed in this chapter. This section briefly describes the operator control facility.

OPERATOR CONTROL

The TCAM operator control facility enables the user to examine or alter the status of a TP network simply by entering a series of specified operator commands. These commands may be entered from the system console or remote stations.

Use of the operator control facility is made possible through operands of the INTRO and TERMINAL macros, discussed under "User Tasks" in this chapter. The INTRO macro contains an operand PRIMARY= that identifies the primary control station. The INTRO macro also specifies the single set of control characters that identify all operator commands. In the sample MCP shown in Figure 114 the INTRO command includes the PRIMARY=SYSCON operand, where SYSCON is the default, and the CONTROL=operand.

<u>Note</u>: The CONTROL= operand is needed only when operator control messages are to be received from sources other than the system console. This operand is included in the example to show how it is specified.

The MCP writer may specify a terminal name rather than SYSCON, provided that the terminal is on a nonswitched line and is able both to enter and to accept messages. If a station other than the system console is to be the primary operator station, "SECTERM=YES" must be specified in the station's TERMINAL macro. This operand of the TERMINAL macro is also used to specify other stations as secondary operator control stations.

Operator command fields must be in the order indicated below and be separated from one another by at least one blank character.

control-chars operation operand [nextline]ending

where:

control-chars
: indicates a character string of one-to-eight nonblank characters identifying a command as an operand.

Note: The "control chars" field must be specified only with commands entered from a station and must not be specified in commands entered from the system console.

operation
: indicates one of five operation types -- HALT, HOLD, RELEASE, DISPLAY, and VARY -- discussed under "Specifying Operator Commands". (There is also a MODIFY operation not discussed here.)

operand
: consists of one or more operands, the most commonly used of which are statname, address, grpname, and rln. These operands determine which functional operator command is associated with the operation type specified. (For some sample operands, see Table 32 in this chapter.)

[nextline]
: ensures that the reply will start on the next line. The "nextline" subfield must be followed immediately by the "ending" subfield.

Note: The "nextline" subfield is specified only at terminals; it may not be used at the system console.

ending
: indicates the end of a message and must be used by all sources entering an operator command. Depending on where the commands are being entered, TCAM has provided end-of-message signals as follows:

- EOB, for system console
- EOT, for start-stop stations
- ETX/EOT, for BSC stations.

These signals are further described in the publication IBM OS Telecommunications Access Method (TCAM) Programmer's Guide and Reference Material.

SPECIFYING OPERATOR COMMANDS

Five sample operator commands that the COBOL TP user may want to use are the following: SYSCLOSE, SUSPXMIT, RESXMIT, INTRCEPT, and STARTLINE. These commands are described briefly below; their formats are given in Table 32. A general discussion of command formats is included under "Operator Control".

For additional information about these and other possible operator commands, see the publication IBM OS Telecommunications Access Method (TCAM) Programmer's Guide and Reference Manual.

SYSCLOSE Command

Initiates either a quick or a flush closedown of the system. In a "quick" closedown, message traffic for each line is stopped as soon as any messages currently being sent or received have been completed. In a "flush" closedown, incoming message traffic is suspended as in a quick closedown, and queued outgoing messages are sent to their destination before closedown is completed.

SUSPXMIT Command

Suspends transmission to a specified station. An intercepted station may still enter messages; only traffic to the station is suspended.

RESMXIT Command

Releases intercepted messages queued either for a specified station or for the line on which the specified station is located.

INTRCEPT Command

Requests display of all stations in the system that are intercepted (that is, those stations to which transmission has been suspended by a HOLD macro or a SUSXMIT operator command).

STARTLINE Command

Causes transmission either to begin or to resume on a particular line (or all the lines) in a line group.

Table 32. Operator Command Formats

Command	Control Characters	Operation	Operand	[nextline]ending
SYSCLOSE		HALT Z	TP, QUICK [1] FLUSH	EOB
SUSPXMIT		HOLD H	TP=statname[2]	EOB
RESMXIT		RELEASE A	TP=statname[2]	EOB
INTRCEPT		DISPLAY D	TP,INTER	EOB
STARTLINE		VARY V	(grpname,[3]rln[4]),ONTP grpname[3] address[5]	EOB

Notes:

1. The user selects either QUICK or FLUSH, depending on the type of closedown desired.

2. The replacement for "statname" should be the name of the TERMINAL macro.

3. The "grpname" field should be the same as the DDNAME=operand field of the DCB for the line group.

4. The "rln" field should contain the relative line number of the line within the group.

5. The address is the physical line number.

This chapter contains information concerning system requirements for the COBOL compiler, execution time, and the sort feature. Additional information for use in estimating the main and auxiliary storage requirements is contained in the publication IBM OS Full American National Standard COBOL Compiler and Library, Version 4, Installation Reference Material.

MINIMUM MACHINE REQUIREMENTS FOR THE COBOL COMPILER

The basic system requirements for use of the COBOL compiler are:

- A System/360 (at least a Model 40) or a System/370 model[1], with a minimum of 80K (81,920) bytes of main storage available to the compiler, and the standard and decimal instruction sets. The floating-point instruction set is required if floating-point data items ~~and fractional exponents are used in~~ the program.

  At least 80K (81,920) bytes should be allocated in the SIZE option of the EXEC job control card that requests execution of the compiler. If less than this is specified, the system assumes the default value of 80K. If more storage is allocated, the compiler will run more efficiently.

  Notes: Before deciding on a value for the SIZE option, the programmer should consider all of the following:

  1. The value of compiler data set SPACE parameters. Given limited storage under MFT, if the primary space allocation for compiler data sets is too small and secondary extents are needed, the system must often use the compiler linkage area for the respective data extent block. Such action often results in either an 80A abnormal termination, if the space limitations are encountered when an attempt is made to load a compiler phase, or diagnostic message IKF0020I-D, if more extensive core has to be allocated for table space for compiler processing.
  2. The size and/or complexity of the program to be compiled. A large or complex program requires more table space than a small or simple one. Accordingly, this table space must be reflected in the SIZE parameter chosen. (For further discussion of table requirements, see "Table Handling Considerations.")
  3. The blocking factors used for compiler data sets. The SIZE parameter (and BUF parameter) ~~reflect the increased buffer size~~ needed to handle blocked compiler data sets.

- Compiler Work Files -- Five utility data sets named SYSUT1, SYSUT2, SYSUT3, SYSUT4, and SYSUT5 (if the SYMDMP option is specified). At least one mass storage device, such as an IBM 2311 Disk Storage Drive, for residence of the operating system and SYSUT1. Both the operating system and SYSUT1 may reside on the same volume. The data sets SYSUT2, SYSUT3, SYSUT4, and SYSUT5 (if the SYMDMP option is specified) can reside on tape or on mass storage. If they reside on tape, there must be a tape volume for each data set. If they reside on mass storage, there must be enough space on the volume to accommodate the data sets.

- A device, such as the 1052 Printer-Keyboard, for direct operator communication.

- A device, such as a card reader or a tape unit, for the job input stream.

- A printer or tape unit for the system output file.

----

[1]A System/370 model may be substituted for a System/360 model for compilation regardless of other considerations. If, however, IBM-370 is specified as the computer-name in the OBJECT-COMPUTER paragraph, a System/370 model must be used for object program execution.

MULTIPROGRAMMING WITH A VARIABLE NUMBER OF TASKS (MVT)

REGION Parameter

COMPILATION: If the compiler is being executed under the MVT option of OS, the REGION parameter, specified as 80K bytes in the COBUC and COBUCLG cataloged procedures, becomes significant (see the section "Using the Cataloged Procedures"). If the programmer wishes to override this value, he can specify a region size in either the JOB statement or in the EXEC statement of the compiler. The size specified should not be less than the value of SIZE in the PARM field of the EXEC statement.

The following examples illustrate both the default and the override cases:

Example 1

```
//JOB1    JOB   1234,J.SMITH
//STEP1   EXEC  COBUC
          .
          .
          .
```

In this example, the programmer accepts the REGION default value of 80K specified in the COBUC cataloged procedure.

Example 2

```
//JOB2    JOB   1234,J.SMITH
//STEP1   EXEC  COBUCLG,REGION=128K,   X
//              PARM.COB='SIZE=130000'
          .
          .
          .
```

In this example, the REGION default value is overridden. Rounding 130000 to the next highest 2K multiple, it becomes 131072, or 128K.

EXECUTION: Priority schedulers require that the REGION parameter be specified for execution of object programs, unless the programmer is willing to accept default region size. The default value is established in the input reader procedure. The region size needed for the execution of the object program is the sum of the following values:

1. The size of the object module after it has been link-edited with all of the necessary object time subroutines.
2. The size of the input/output buffers being used, multiplied by the blocking factor (standard sequential files are double buffered if no blocking factor is specified).
3. The size of the data management routines and control blocks that are used (see the publication IBM OS Storage Estimates).
4. Any GETMAIN macro instruction executed for USE LABELS, etc.
5. An additional 4K bytes.
6. If the Sort feature is used, 15,360 bytes plus any additional core storage assigned via the SORT-CORE-SIZE special register.

Intermediate Data Sets under MVT

Except when the Direct Sysout Writer is used, SYSIN and SYSOUT data resides in intermediate direct-access data sets. These data sets are used by the system to temporarily hold all of the job's input and output data.

SYSIN-SYSOUT CHARACTERISTICS: The input and output data set characteristics are determined by the system, but can be altered by the programmer if necessary. The procedure used to alter the default values depends on whether the data set is for input or output, as follows:

- For SYSIN data -- the programmer must request, at the time the job is submitted, that the operator use one of the several reader procedures available. Reader procedures are cataloged procedures that control the reader and vary according to the blocking factor specified.
- For SYSOUT data -- the programmer must use override statements as described in "Using the Cataloged Procedures."

Output is placed in the SYSOUT intermediate data set, except when the Direct SYSOUT Writer is used, in which case output goes directly to the printer, punch, or tape as in systems with the primary control program. Since nothing is written out until the completion of the job, the programmer must make sure that the SYSOUT data set is large enough to hold all of the possible output data of his program. The SPACE parameter of the DD statement is specified for SYSOUT with a specified default value. If the programmer determines that his output will exceed the default value, he can do either or both of two things:

1. Specify blocking of his data set with the DCB parameter of an override DD statement

2. Override the compilation step of a compiled procedure by specifying the SPACE parameter. An example of a statement that can be used is:

```
//COB.SYSPRINT  DD  SPACE=(121,(500,50))
```

Note: If the TRK or CYL subparameters of the SPACE parameter are used, the programmer should be aware that requests will differ depending upon the mass storage device used (2301, 2303, 2311, ..., etc.). To avoid this consideration, the average record-length subparameter can be used.

MULTIPLE OPEN AND CLOSE STATEMENTS: Under the MVT control program, input data following the DD * or DD DATA card becomes a single data set. Once a CLOSE statement is encountered. The data set is repositioned to the beginning of the data set. To avoid errors, the programmer should keep this in mind when using more than one OPEN and CLOSE statement for a data set assigned to SYSIN.

Note: Under MVT, a file must be closed before the STOP RUN or EXIT PROGRAM statement is executed. Failure to do this results in an abnormal termination.

EXECUTION TIME CONSIDERATIONS

The amount of main storage must be sufficient to accommodate at least:

- The control program
- Data management support
- The load module to be executed

When the OPTIMIZE option is specified, the number of procedure blocks in the program cannot exceed 255. A procedure block is approximately 4096 bytes of Procedure Division code.

COBOL programs compiled with any of the symbolic debugging options (STATE, FLOW, SYMDMP) have execution time requirements that differ from those of similar programs compiled without these options. If the SYMDMP option is in effect, the data set it required at compile time (SYSUT5) must be present at execution time.

The total space required for object-time debugging should be calculated as follows:

$$S_{TS} = S_{DBG} + [S_{FLW}] + \begin{matrix} S_{STN} \\ S_{SYMDMP} \end{matrix}$$

where:

S_{TS} = the total space

S_{DBG} = the space allocated once and only once for a run containing any object-time debugging options

S_{FLW} = the space required for the FLOW option

S_{STN} = the space required for the STATE option

S_{SYMDMP} = the space required for the SYMDMP option

- S_{DBG} = 3700 bytes

- S_{FLW} =(1672 + 4*nn + 10*P) bytes

  where

  nn = the number specified in the FLOW=nn parameter of the EXEC job control statement

  P = the total number of paragraph- names in a COBOL program

- S_{STN} =(1090 + 5*V) bytes

  where

  V = the number of verbs in the COBOL program (a number that is approximately equal to the number of statements in the program)

- S_{SYMDMP} =(11250 + S_{TABLES} +S_{DM})bytes

where
S_{TABLES} = the size of tables for SYMDMP
S_{DM} = the size of data management required for SYMDMP

$S_{TABLES} = (72*PC+[19*LC+[8*ON]+7*id]+[S_{ODOTAB}])$ bytes

where

PC = the number of program control cards

LC = the number of line control cards

ON = the number of line control cards with ON options

id = the number of identifiers requested on line-control cards

S_{ODOTAB} = the size of ODOTAB on the debug file (approximately 27 times the number of unique objects of OCCURS DEPENDING ON statements).

$S_{DM} = (818+S_{BSAM} + [S_{QSAM}])$ bytes

where

S_{BSAM} = 800 bytes = the space required for BSAM modules (when not in the LPA)

S_{QSAM} = 1424 bytes = the space required for QSAM modules (when not on the LPA) and no QSAM files are used in the program

The input/output device requirements for execution of the problem program are determined from specifications made in the Environment Division of the source program

Note: An IBM System/370 is required for execution if IBM-370 is specified as the computer-name with OBJECT-COMPUTER paragraph of the Configuration Section.

SORT FEATURE CONSIDERATIONS

The basic requirements for use of the Sort feature are:

- A System/360 model or System/370 with sufficient main storage to accommodate the load module to be executed plus a minimum of 15,360 bytes for execution of the sort program and any additional core storage assigned to the sort program via the SORT-CORE-SIZE special register.
- At least one mass storage device (which may be the system residence device) for residence of SYS1.SORTLIB.
- At least three tape units or one mass storage device for intermediate storage.

The following is a sample COBOL program and the output listing resulting from its compilation, linkage editing, and execution. The program creates a blocked, unlabeled, standard sequential file, writes it out on tape, and then reads it back in. It also does a check on the field called NO-OF-DEPENDENTS. All data records in the file are displayed. Those with a zero in the NO-OF-DEPENDENTS field are displayed with the special character Z. The records of the file are not altered from the time of creation, despite the fact that the NO-OF-DEPENDENTS field is changed for display purposes. The individual records of the file are created using the subscripting technique. TRACE is used as a debugging aid during program execution.

The output formats illustrated in the listing are described in "Output." Individual parts of the listing are numbered in accordance with the numbers used in the chapter "Output."

```
00001     100010 IDENTIFICATION DIVISION.                                        00900008
00002     100020 PROGRAM-ID.  TESTRUN.                                           00900010
00003     100030   AUTHOR. PROGRAMMER NAME.                                      00900012
00004     100040   INSTALLATION. NEW YORK PROGRAMMING CENTER.                    00900014
00005     100050   DATE-WRITTEN. JULY 12, 1968.                                  00900016
00006     100060 DATE-COMPILED. FEB 19,1972                                      00900018
00007     100070   REMARKS. THIS PROGRAM HAS BEEN WRITTEN AS A SAMPLE PROGRAM FOR 00900020
00008     100080     COBOL USERS. IT CREATES AN OUTPUT FILE AND READS IT BACK AS  00900022
00009     100090     INPUT.                                                      00900024
00010     100100 ENVIRONMENT DIVISION.                                           00900026
00011     100110 CONFIGURATION SECTION.                                          00900028
00012     100120   SOURCE-COMPUTER. IBM-360-H50.                                 00900030
00013     100130   OBJECT-COMPUTER. IBM-360-H50.                                 00900032
00014     100140 INPUT-OUTPUT SECTION.                                           00900034
00015     100150 FILE-CONTROL.                                                   00900036
00016     100160     SELECT FILE-1 ASSIGN TO UT-2400-S-SAMPLE.                   00900038
00017     100170     SELECT FILE-2 ASSIGN TO UT-2400-S-SAMPLE.                   00900040
00018     100180 DATA DIVISION.                                                  00900042
00019     100190 FILE SECTION.                                                   00900044
00020     100200 FD  FILE-1                                                      00900046
00021     100210     LABEL RECORDS ARE OMITTED                                   00900048
00022     100220     BLOCK CONTAINS 100 CHARACTERS                               00900050
00023     100225     RECORD CONTAINS 20 CHARACTERS
00024     100230     RECORDING MODE IS F                                         00900052
00025     100240     DATA RECORD IS RECORD-1.                                    00900054
00026     100250   01 RECORD-1.                                                  00900056
00027     100260     02 FIELD-A PICTURE IS X(20).                                00900058
00028     100270 FD  FILE-2                                                      00900060
00029     100280     LABEL RECORDS ARE OMITTED                                   00900062
00030     100290     BLOCK CONTAINS 5 RECORDS                                    00900064
00031     100300     RECORD CONTAINS 20 CHARACTERS                               00900066
00032     100310     RECORDING MODE IS F                                         00900068
00033     100320     DATA RECORD IS RECORD-2.                                    00900070
00034     100330   01 RECORD-2.                                                  00900072
00035     100340     02 FIELD-A PICTURE IS X(20).                                00900074
00036     100350 WORKING-STORAGE SECTION.                                        00900076
00037     100360     77 KOUNT PICTURE S99 COMP SYNC.
00038     100370     77 NOMBER  PICTURE S99 COMP SYNC.                           00900080
00039     100375   01 FILLER.                                                    00900082
00040 **  00380      02 ALPHABET PICTURE X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".00900084
00041     100395     02 ALPHA REDEFINES ALPHABET PICTURE X OCCURS 26 TIMES.      00900086
00042     100405     02 DEPENDENTS PICTURE X(26)  VALUE "01234012340123401234012340090088
00043     100410-    "0".                                                        00900090
00044     100420     02 DEPEND REDEFINES DEPENDENTS PICTURE X OCCURS 26 TIMES.   00900092
00045     100440   01 WORK-RECORD.                                               00900094
00046     100450     02 NAME-FIELD  PICTURE X.                                   00900096
00047     100460     02 FILLER PICTURE X VALUE SPACE.                            00900098
00048     100470     02 RECORD-NO PICTURE 9999.                                  00900100
00049     100480     02 FILLER PICTURE X VALUE SPACE.                            00900102
00050     100490     02 LOCATION PICTURE AAA VALUE "NYC".                        00900104
00051     100500     02 FILLER PICTURE X VALUE SPACE.                            00900106
00052     100510     02 NO-OF-DEPENDENTS PICTURE XX.                             00900108
00053     100520     02 FILLER PICTURE X(7) VALUE SPACES.                        00900110
00054     100522 01  RECORDA.
```

```
100524      02  A PICTURE S9(4) VALUE 1234.
100526      02  B REDEFINES A PICTURE S9(7) COMPUTATIONAL-3.
100530 PROCEDURE DIVISION.                                            00900112
100540 BEGIN. READY TRACE.                                            00900114
100550     NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED 00900116
100560     AND INITIALIZES COUNTERS.                                  00900118
100570 STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO KOUNT NOMBER.
100580     NOTE THAT THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE 00900122
100590     CONTAINED IN THE FILE, WRITES THEM ON TAPE, AND DISPLAYS   00900124
100600     THEM ON THE CONSOLE.                                       00900126
100610 STEP-2. ADD 1 TO KOUNT, ADD 1 TO NOMBER, MOVE ALPHA (KOUNT) TO
100620     NAME-FIELD.                                                00900130
100625         COMPUTE B = B + 1.
100630     MOVE DEPEND (KOUNT) TO NO-OF-DEPENDENTS.
100640     MOVE NOMBER TO RECORD-NO.                                  00900134
100650 STEP-3. DISPLAY WORK-RECORD UPON CONSOLE. WRITE RECORD-1 FROM  00900136
100660     WORK-RECORD.                                               00900138
100670 STEP-4. PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL TO 26.
100680     NOTE THAT THE FOLLOWING CLOSES OUTPUT AND REOPENS IT AS    00900142
100690     INPUT.                                                     00900144
100700 STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.                       00900146
100710     NOTE THAT THE FOLLOWING READS BACK THE FILE AND SINGLES OUT 00900148
100720     EMPLOYEES WITH NO DEPENDENTS.                              00900150
100730 STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8. 00900152
100740 STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO        00900154
100750     NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO         00900156
100760     STEP-6.                                                    00900158
100770 STEP-8. CLOSE FILE-2.                                          00900160
100780     STOP RUN.                                                  00900162
**XX                                                                  00900164
```

INTRNL NAME	LVL	SOURCE NAME	BASE	DISPL	INTRNL NAME	DEFINITION	USAGE	R	O	Q	M
DNM=1-148	FD	FILE-1	DCB=01		DNM=1-148		QSAM				F
DNM=1-167	01	RECORD-1	BL=1	000	DNM=1-167	DS 0CL20	GROUP				
DNM=1-188	02	FIELD-A	BL=1	000	DNM=1-188	DS 20C	DISP				
DNM=1-205	FD	FILE-2	DCB=02		DNM=1-205		QSAM				F
DNM=1-224	01	RECORD-2	BL=2	000	DNM=1-224	DS 0CL20	GROUP				
DNM=1-245	02	FIELD-A	BL=2	000	DNM=1-245	DS 20C	DISP				
DNM=1-265	77	KOUNT	BL=3	000	DNM=1-265	DS 1H	COMP				
DNM=1-280	77	NOMBER	BL=3	002	DNM=1-280	DS 1H	COMP				
DNM=1-296	01	FILLER	BL=3	008	DNM=1-296	DS 0CL52	GROUP				
DNM=1-310	02	ALPHABET	BL=3	008	DNM=1-310	DS 26C	DISP				
DNM=1-328	02	ALPHA	BL=3	008	DNM=1-328	DS 1C	DISP	R	O		
DNM=1-346	02	DEPENDENTS	BL=3	022	DNM=1-346	DS 26C	DISP				
DNM=1-366	02	DEPEND	BL=3	022	DNM=1-366	DS 1C	DISP	R	O		
DNM=1-382	01	WORK-RECORD	BL=3	040	DNM=1-382	DS 0CL20	GROUP				
DNM=1-406	02	NAME-FIELD	BL=3	040	DNM=1-406	DS 1C	DISP				
DNM=1-426	02	FILLER	BL=3	041	DNM=1-426	DS 1C	DISP				
DNM=1-440	02	RECORD-NO	BL=3	042	DNM=1-440	DS 4C	DISP-NM				
DNM=1-459	02	FILLER	BL=3	046	DNM=1-459	DS 1C	DISP				
DNM=1-473	02	LOCATION	BL=3	047	DNM=1-473	DS 3C	DISP				
DNM=1-491	02	FILLER	BL=3	04A	DNM=1-491	DS 1C	DISP				
DNM=2-000	02	NO-OF-DEPENDENTS	BL=3	04B	DNM=2-000	DS 2C	DISP				
DNM=2-026	02	FILLER	BL=3	04D	DNM=2-026	DS 7C	DISP				
DNM=2-040	01	RECORDA	BL=3	058	DNM=2-040	DS 0CL4	GROUP				
DNM=2-060	02	A	BL=3	058	DNM=2-060	DS 4C	DISP-NM				
DNM=2-071	02	B	BL=3	058	DNM=2-071	DS 4P	COMP-3	R			

MEMORY MAP

TGT	00248
SAVE AREA	00248
SWITCH	00290
TALLY	00294
SORT SAVE	00298
ENTRY-SAVE	0029C
SORT CORE SIZE	002A0
RET CODE	002A4
SORT RET	002A6
WORKING CELLS	002A8
SORT FILE SIZE	003D8
SORT MODE SIZE	003DC
PGT-VN TBL	003E0
TGT-VN TBL	003E4
VCONPTR	003E8
LENGTH OF VN TBL	003EC
LABEL RET	003EE
CURRENT PRIORITY	003EF
DBG R14SAVE	003F0
COBOL INDICATOR	003F4
A(INIT1)	003F8
DEBUG TABLE PTR	003FC
SUBCOM PTR	00400
SORT DDNAME	00404
UNUSED	0040C
DBG R11SAVE	00420
UNUSED	00424
PRBL1 CELL PTR	00428
GENCBTBL PTR	0042C
UNUSED	00430
TA LENGTH	00431
UNUSED	00434
OVERFLOW CELLS	0043C
BL CELLS	0043C
DECBADR CELLS	00448
TEMP STORAGE	00450
TEMP STORAGE-2	00458
TEMP STORAGE-3	00458
TEMP STORAGE-4	00458
BLL CELLS	00458
VLC CELLS	00460
SBL CELLS	00460
INDEX CELLS	00460
SUBADR CELLS	00460
ONCTL CELLS	00468
PFMCTL CELLS	00468
PFMSAV CELLS	00468
VN CELLS	0046C
SAVE AREA =2	00470
SAVE AREA =3	00470
XSASW CELLS	00478
XSA CELLS	00478

```
LITERAL POOL (HEX)

004F0 (LIT+0)      00000001  1C00001A  004805EF  48000000  C0000000

     DISPLAY LITERALS (BCD)

00504 (LTL+20)   'WORK-RECORD'


                    PGT                         00498

              OVERFLOW CELLS                    00498
              VIRTUAL CELLS                     00498
              PROCEDURE NAME CELLS              004B8
              GENERATED NAME CELLS              004CC
              DCB ADDRESS CELLS                 004E4
              VNI CELLS                         004EC
              LITERALS                          004F0
              DISPLAY LITERALS                  00504


 REGISTER ASSIGNMENT

  REG 6    BL =3
  REG 7    BL =1
  REG 8    BL =2

WORKING-STORAGE STARTS AT LOCATION 00088 FOR A LENGTH OF 00060.J

58       VERB   1
                   000510                       START    EQU   *
                   000510   07 00                        BCR   0,0
                   000512   58 F0 C 00C                  L     15,00C(0,12)          V(ILBODBG4)
                   000516   05 EF                        BALR  14,15
                   000518   58 F0 C 010                  L     15,010(0,12)          V(ILBOFLW1)
                   00051C   05 1F                        BALR  1,15
                   00051E   0000003A                     DC    X'0000003A'
                   000522   58 F0 C 014                  L     15,014(0,12)          V(ILBODSP0)
                   000526   05 1F                        BALR  1,15
                   000528   000140                       DC    X'000140'
                   00052B   05F5F840404040               DC    X'05F5F840404040'
58       VERB   2
                   000532   96 40 D 048                  OI    048(13),X'40'         SWT+0
61       VERB   3
                   000536   58 F0 C 00C                  L     15,00C(0,12)          V(ILBODBG4)
                   00053A   05 EF                        BALR  14,15
                   00053C   58 F0 C 010                  L     15,010(0,12)          V(ILBOFLW1)
                   000540   05 1F                        BALR  1,15
                   000542   0000003D                     DC    X'0000003D'
                   000546   58 F0 C 014                  L     15,014(0,12)          V(ILBODSP0)
                   00054A   05 1F                        BALR  1,15
                   00054C   000140                       DC    X'000140'
                   00054F   05F6F140404040               DC    X'05F6F140404040'
61       VERB   4
                   000556   58 F0 C 00C                  L     15,00C(0,12)          V(ILBODBG4)
                   00055A   05 EF                        BALR  14,15
```

```
                00055C  58 10 C 04C                 L     1,04C(0,12)        DCB=1
                000560  D2 03 D 060 C 018           MVC   060(4,13),018(12)  WC=01         V(ILBOERR1)
                000566  D2 02 1 039 D 061           MVC   039(3,1),061(13)                 WC=02
                00056C  58 10 C 04C                 L     1,04C(0,12)        DCB=1
                000570  D2 01 1 032 C 060           MVC   032(2,1),060(12)                 LIT+8
                000576  D2 01 1 060 C 062           MVC   060(2,1),062(12)                 LIT+10
                00057C  50 10 D 228                 ST    1,228(0,13)        SAV3
                000580  92 8F D 228                 MVI   228(13),X'8F'      SAV3
                000584  41 10 D 228                 LA    1,228(0,13)        SAV3
                000588  0A 13                       SVC   19
                00058A  58 10 C 04C                 L     1,04C(0,12)        DCB=1
                00058E  18 21                       LR    2,1
                000590  58 F0 1 030                 L     15,030(0,1)
                000594  05 EF                       BALR  14,15
                000596  50 10 D 1F4                 ST    1,1F4(0,13)        BL =1
                00059A  58 70 D 1F4                 L     7,1F4(0,13)        BL =1
61     VERB    5
                00059E  D2 01 6 000 C 058           MVC   000(2,6),058(12)   DNM=1-265     LIT+0
                0005A4  D2 01 6 002 C 058           MVC   002(2,6),058(12)   DNM=1-280     LIT+0
65     VERB    6
                0005AA                      PN=01   EQU   *
                0005AA  58 F0 C 00C                 L     15,00C(0,12)       V(ILBODBG4)
                0005AE  05 EF                       BALR  14,15
                0005B0  58 F0 C 010                 L     15,010(0,12)       V(ILBOFLW1)
                0005B4  05 1F                       BALR  1,15
                0005B6  00000041                    DC    X'00000041'
                0005BA  58 F0 C 014                 L     15,014(0,12)       V(ILBODSP0)
                0005BE  05 1F                       BALR  1,15
                0005C0  000140                      DC    X'000140'
                0005C3  05F6F540404040              DC    X'05F6F540404040'
65     VERB    7
                0005CA  48 30 C 05A                 LH    3,05A(0,12)        LIT+2
                0005CE  4A 30 6 000                 AH    3,000(0,6)         DNM=1-265
                0005D2  40 30 6 000                 STH   3,000(0,6)         DNM=1-265
65     VERB    8
                0005D6  48 30 C 05A                 LH    3,05A(0,12)        LIT+2
                0005DA  4A 30 6 002                 AH    3,002(0,6)         DNM=1-280
                0005DE  40 30 6 002                 STH   3,002(0,6)         DNM=1-280
65     VERB    9
                0005E2  41 40 6 008                 LA    4,008(0,6)         DNM=1-328
                0005E6  48 20 6 000                 LH    2,000(0,6)         DNM=1-265
                0005EA  4C 20 C 05A                 MH    2,05A(0,12)        LIT+2
                0005EE  1A 42                       AR    4,2
                0005F0  5B 40 C 058                 S     4,058(0,12)        LIT+0
                0005F4  50 40 D 218                 ST    4,218(0,13)        SBS=1
                0005F8  58 E0 D 218                 L     14,218(0,13)       SBS=1
                0005FC  D2 00 6 040 E 000           MVC   040(1,6),000(14)   DNM=1-406     DNM=1-328
67     VERB   10
                000602  FA 30 6 058 C 05C           AP    058(4,6),05C(1,12) DNM=2-71      LIT+4
68     VERB   11
                000608  41 40 6 022                 LA    4,022(0,6)         DNM=1-366
                00060C  48 20 6 000                 LH    2,000(0,6)         DNM=1-265
                000610  4C 20 C 05A                 MH    2,05A(0,12)        LIT+2
                000614  1A 42                       AR    4,2
                000616  5B 40 C 058                 S     4,058(0,12)        LIT+0
                00061A  50 40 D 21C                 ST    4,21C(0,13)        SBS=2
                00061E  58 E0 D 21C                 L     14,21C(0,13)       SBS=2
                000622  D2 00 6 04B E 000           MVC   04B(1,6),000(14)   DNM=2-0       DNM=1-366
                000628  92 40 6 04C                 MVI   04C(6),X'40'       DNM=2-0+1
69     VERB   12
                00062C  48 30 6 002                 LH    3,002(0,6)         DNM=1-280
                000630  4E 30 D 208                 CVD   3,208(0,13)        TS=01
                000634  F3 31 6 042 D 20E           UNPK  042(4,6),20E(2,13) DNM=1-440     TS=07
                00063A  96 F0 6 045                 OI    045(6),X'F0'       DNM=1-440+3
70     VERB   13
                00063E  58 F0 C 00C                 L     15,00C(0,12)       V(ILBODBG4)
                000642  05 EF                       BALR  14,15
                000644  58 F0 C 010                 L     15,010(0,12)       V(ILBOFLW1)
                000648  05 1F                       BALR  1,15
                00064A  00000046                    DC    X'00000046'
                00064E  58 F0 C 014                 L     15,014(0,12)       V(ILBODSP0)
                000652  05 1F                       BALR  1,15
                000654  000140                      DC    X'000140'
                000657  05F7F040404040              DC    X'05F7F040404040'
```

```
70         VERB  14
                 00065E   58 F0 C 00C                    L     15,00C(0,12)          V(ILBODBG4)
                 000662   05 EF                          BALR  14,15
                 000664   58 F0 C 014                    L     15,014(0,12)          V(ILBODSP0)
                 000668   05 1F                          BALR  1,15
                 00066A   0002                           DC    X'0002'
                 00066C   00                             DC    X'00'
                 00066D   000014                         DC    X'000014'
                 000670   0D0001FC                       DC    X'0D0001FC'           BL =3
                 000674   0040                           DC    X'0040'
                 000676   FFFF                           DC    X'FFFF'
70         VERB  15
                 000678   58 F0 C 00C                    L     15,00C(0,12)          V(ILBODBG4)
                 00067C   05 EF                          BALR  14,15
                 00067E   D2 13 7 000 6 040              MVC   000(20,7),040(6)      DNM=1-167        DNM=1-382
                 000684   58 10 C 04C                    L     1,04C(0,12)           DCB=1
                 000688   18 21                          LR    2,1
                 00068A   58 10 C 04C                    L     1,04C(0,12)           DCB=1
                 00068E   58 00 1 04C                    L     0,04C(0,1)
                 000692   58 F0 1 030                    L     15,030(0,1)
                 000696   44 00 1 060                    EX    0,060(0,1)
                 00069A   50 10 D 1F4                    ST    1,1F4(0,13)           BL =1
                 00069E   58 70 D 1F4                    L     7,1F4(0,13)           BL =1
                 0006A2                         GN=01    EQU   *
                 0006A2   58 10 D 224                    L     1,224(0,13)           VN=01
                 0006A6   07 F1                          BCR   15,1
72         VERB  16
                 0006A8                         PN=02    EQU   *
                 0006A8   07 00                          BCR   0,0
                 0006AA   58 F0 C 00C                    L     15,00C(0,12)          V(ILBODBG4)
                 0006AE   05 EF                          BALR  14,15
                 0006B0   58 F0 C 010                    L     15,010(0,12)          V(ILBOFLW1)
                 0006B4   05 1F                          BALR  1,15
                 0006B6   00000048                       DC    X'00000048'
                 0006BA   58 F0 C 014                    L     15,014(0,12)          V(ILBODSP0)
                 0006BE   05 1F                          BALR  1,15
                 0006C0   000140                         DC    X'000140'
                 0006C3   05F7F240404040                 DC    X'05F7F240404040'
72         VERB  17
                 0006CA   58 00 D 224                    L     0,224(0,13)           VN=01
                 0006CE   50 00 D 220                    ST    0,220(0,13)           PSV=1
                 0006D2   58 00 C 038                    L     0,038(0,12)           GN=02
                 0006D6   50 00 D 224                    ST    0,224(0,13)           VN=01
                 0006DA                         GN=02    EQU   *
                 0006DA   48 30 6 000                    LH    3,000(0,6)            DNM=1-265
                 0006DE   49 30 C 05E                    CH    3,05E(0,12)           LIT+6
                 0006E2   58 F0 C 03C                    L     15,03C(0,12)          GN=03
                 0006E6   07 8F                          BCR   8,15
                 0006E8   58 10 C 020                    L     1,020(0,12)           PN=01
                 0006EC   07 F1                          BCR   15,1
                 0006EE                         GN=03    EQU   *
                 0006EE   58 00 D 220                    L     0,220(0,13)           PSV=1
                 0006F2   50 00 D 224                    ST    0,224(0,13)           VN=01
75         VERB  18
                 0006F6   58 F0 C 00C                    L     15,00C(0,12)          V(ILBODBG4)
                 0006FA   05 EF                          BALR  14,15
                 0006FC   58 F0 C 010                    L     15,010(0,12)          V(ILBOFLW1)
                 000700   05 1F                          BALR  1,15
                 000702   0000004B                       DC    X'0000004B'
                 000706   58 F0 C 014                    L     15,014(0,12)          V(ILBODSP0)
                 00070A   05 1F                          BALR  1,15
                 00070C   000140                         DC    X'000140'
                 00070F   05F7F540404040                 DC    X'05F7F540404040'
75         VERB  19
                 000716   58 F0 C 00C                    L     15,00C(0,12)          V(ILBODBG4)
                 00071A   05 EF                          BALR  14,15
                 00071C   58 10 C 04C                    L     1,04C(0,12)           DCB=1
                 000720   58 30 1 02C                    L     3,02C(0,1)
                 000724   91 0F 3 00C                    TM    00C(3),X'0F'
                 000728   05 50                          BALR  5,0
```

```
                00072A  47 E0 5 010                 BC    14,010(0,5)
                00072E  58 20 1 04C                 L     2,04C(0,1)
                000732  4B 20 1 052                 SH    2,052(0,1)
                000736  50 20 1 04C                 ST    2,04C(0,1)
                00073A  58 10 C 04C                 L     1,04C(0,12)       DCB=1
                00073E  50 10 D 228                 ST    1,228(0,13)       SAV3
                000742  92 90 D 228                 MVI   228(13),X'90'     SAV3
                000746  41 10 D 228                 LA    1,228(0,13)       SAV3
                00074A  0A 14                       SVC   20
                00074C  58 20 C 04C                 L     2,04C(0,12)       DCB=1
                000750  58 10 2 014                 L     1,014(0,2)
                000754  96 01 2 017                 OI    017(2),X'01'
                000758  1B 44                       SR    4,4
                00075A  43 40 1 005                 IC    4,005(0,1)
                00075E  4C 40 1 006                 MH    4,006(0,1)
                000762  41 00 4 008                 LA    0,008(0,4)
                000766  41 10 1 000                 LA    1,000(0,1)
                00076A  0A 0A                       SVC   10
75    VERB  20
                00076C  58 F0 C 00C                 L     15,00C(0,12)      V(ILBODBG4)
                000770  05 EF                       BALR  14,15
                000772  58 10 C 050                 L     1,050(0,12)       DCB=2
                000776  D2 03 D 060 C 018           MVC   060(4,13),018(12) WC=01          V(ILBOERR1)
                00077C  D2 02 1 039 D 061           MVC   039(3,1),061(13)                 WC=02
                000782  58 10 C 050                 L     1,050(0,12)       DCB=2
                000786  D2 01 1 032 C 064           MVC   032(2,1),064(12)                 LIT+12
                00078C  50 10 D 228                 ST    1,228(0,13)       SAV3
                000790  92 80 D 228                 MVI   228(13),X'80'     SAV3
                000794  41 10 D 228                 LA    1,228(0,13)       SAV3
                000798  0A 13                       SVC   19
78    VERB  21
                00079A                      PN=03   EQU   *
                00079A  58 F0 C 00C                 L     15,00C(0,12)      V(ILBODBG4)
                00079E  05 EF                       BALR  14,15
                0007A0  58 F0 C 010                 L     15,010(0,12)      V(ILBOFLW1)
                0007A4  05 1F                       BALR  1,15
                0007A6  0000004E                    DC    X'0000004E'
                0007AA  58 F0 C 014                 L     15,014(0,12)      V(ILBODSP0)
                0007AE  05 1F                       BALR  1,15
                0007B0  000140                      DC    X'000140'
                0007B3  05F7F840404040              DC    X'05F7F840404040'
78    VERB  22
                0007BA  58 F0 C 00C                 L     15,00C(0,12)      V(ILBODBG4)
                0007BE  05 EF                       BALR  14,15
                0007C0  58 10 C 050                 L     1,050(0,12)       DCB=2
                0007C4  18 21                       LR    2,1
                0007C6  D2 02 2 021 C 041           MVC   021(3,2),041(12)                 GN=04+1
                0007CC  58 F0 1 030                 L     15,030(0,1)
                0007D0  05 EF                       BALR  14,15
                0007D2  50 10 D 1F8                 ST    1,1F8(0,13)       BL =2
                0007D6  58 80 D 1F8                 L     8,1F8(0,13)       BL =2
                0007DA  D2 13 6 040 8 000           MVC   040(20,6),000(8)  DNM=1-382      DNM=1-224
                0007E0  58 50 C 02C                 L     5,02C(0,12)       PN=04
                0007E4  07 F5                       BCR   15,5
78    VERB  23
                0007E6                      GN=04   EQU   *
                0007E6  58 10 C 030                 L     1,030(0,12)       PN=05
                0007EA  07 F1                       BCR   15,1
79    VERB  24
                0007EC                      PN=04   EQU   *
                0007EC  07 00                       BCR   0,0
                0007EE  58 F0 C 00C                 L     15,00C(0,12)      V(ILBODBG4)
                0007F2  05 EF                       BALR  14,15
                0007F4  58 F0 C 010                 L     15,010(0,12)      V(ILBOFLW1)
                0007F8  05 1F                       BALR  1,15
                0007FA  0000004F                    DC    X'0000004F'
                0007FE  58 F0 C 014                 L     15,014(0,12)      V(ILBODSP0)
                000802  05 1F                       BALR  1,15
                000804  000140                      DC    X'000140'
                000807  05F7F940404040              DC    X'05F7F940404040'
```

```
79         VERB  25
                  00080E   58 20 C 044                       L      2,044(0,12)          GN=05
                  000812   95 F0 6 04B                       CLI    04B(6),X'F0'         DNM=2-0
                  000816   07 72                             BCR    7,2
                  000818   95 40 6 04C                       CLI    04C(6),X'40'         DNM=2-0+1
                  00081C   07 72                             BCR    7,2
79         VERB  26
                  00081E   92 E9 6 04B                       MVI    04B(6),X'E9'         DNM=2-0
                  000822   92 40 6 04C                       MVI    04C(6),X'40'         DNM=2-0+1
80         VERB  27
                  000826                          GN=05      EQU    *
                  000826   58 10 C 068                       L      1,068(0,12)          LIT+16
                  00082A   50 10 D 230                       ST     1,230(0,13)          PRM=1
                  00082E   41 20 D 230                       LA     2,230(0,13)          PRM=1
                  000832   58 F0 C 00C                       L      15,00C(0,12)         V(ILBODBG4)
                  000836   05 EF                             BALR   14,15
                  000838   58 F0 C 014                       L      15,014(0,12)         V(ILBODSP0)
                  00083C   05 1F                             BALR   1,15
                  00083E   8001                              DC     X'8001'
                  000840   10                                DC     X'10'
                  000841   00000B                            DC     X'00000B'
                  000844   0C00006C                          DC     X'0C00006C'          LIT+20
                  000848   0000                              DC     X'0000'
                  00084A   00                                DC     X'00'
                  00084B   000014                            DC     X'000014'
                  00084E   0D0001FC                          DC     X'0D0001FC'          BL =3
                  000852   0040                              DC     X'0040'
                  000854   FFFF                              DC     X'FFFF'
80         VERB  28
                  000856   58 10 C 028                       L      1,028(0,12)          PN=03
                  00085A   07 F1                             BCR    15,1
82         VERB  29
                  00085C                          PN=05      EQU    *
                  00085C   07 00                             BCR    0,0
                  00085E   58 F0 C 00C                       L      15,00C(0,12)         V(ILBODBG4)
                  000862   05 EF                             BALR   14,15
                  000864   58 F0 C 010                       L      15,010(0,12)         V(ILBOFLW1)
                  000868   05 1F                             BALR   1,15
                  00086A   00000052                          DC     X'00000052'
                  00086E   58 F0 C 014                       L      15,014(0,12)         V(ILBODSP0)
                  000872   05 1F                             BALR   1,15
                  000874   000140                            DC     X'000140'
                  000877   05F8F240404040                    DC     X'05F8F240404040'
82         VERB  30
                  00087E   58 F0 C 00C                       L      15,00C(0,12)         V(ILBODBG4)
                  000882   05 EF                             BALR   14,15
                  000884   58 10 C 050                       L      1,050(0,12)          DCB=2
                  000888   58 30 1 02C                       L      3,02C(0,1)
                  00088C   91 0F 3 00C                       TM     00C(3),X'0F'
                  000890   05 50                             BALR   5,0
                  000892   47 E0 5 010                       BC     14,010(0,5)
                  000896   58 20 1 04C                       L      2,04C(0,1)
                  00089A   4B 20 1 052                       SH     2,052(0,1)
                  00089E   50 20 1 04C                       ST     2,04C(0,1)
                  0008A2   58 10 C 050                       L      1,050(0,12)          DCB=2
                  0008A6   50 10 D 228                       ST     1,228(0,13)          SAV3
                  0008AA   92 90 D 228                       MVI    228(13),X'90'        SAV3
                  0008AE   41 10 D 228                       LA     1,228(0,13)          SAV3
                  0008B2   0A 14                             SVC    20
                  0008B4   58 20 C 050                       L      2,050(0,12)          DCB=2
                  0008B8   58 10 2 014                       L      1,014(0,2)
                  0008BC   96 01 2 017                       OI     017(2),X'01'
                  0008C0   1B 44                             SR     4,4
                  0008C2   43 40 1 005                       IC     4,005(0,1)
                  0008C6   4C 40 1 006                       MH     4,006(0,1)
                  0008CA   41 00 4 008                       LA     0,008(0,4)
                  0008CE   41 10 1 000                       LA     1,000(0,1)
                  0008D2   0A 0A                             SVC    10
83         VERB  31
                  0008D4   58 F0 C 00C                       L      15,00C(0,12)         V(ILBODBG4)
                  0008D8   05 EF                             BALR   14,15
```

```
0008DA                          GN=06    EQU    *
0008DA  58 F0 C 01C                      L      15,01C(0,12)         V(ILBOSRV1)
0008DE  07 FF                            BCR    15,15
0008E0  50 D0 5 008             INIT2    ST     13,008(0,5)
0008E4  50 50 D 004                      ST     5,004(0,13)
0008E8  50 E0 D 054                      ST     14,054(0,13)
0008EC  91 20 D 048                      TM     048(13),X'20'        SWT+0
0008F0  47 E0 F 02E                      BC     14,02E(0,15)
0008F4  58 20 D 1B8                      L      2,1B8(0,13)
0008F8  91 40 D 049                      TM     049(13),X'40'        SWT+1
0008FC  47 E0 9 000                      BC     14,000(0,9)
000900  96 04 2 000                      OI     000(2),X'04'
000904  58 F0 2 038                      L      15,038(0,2)
000908  41 F0 F 004                      LA     15,004(0,15)
00090C  07 FF                            BCR    15,15
00090E  94 EF D 048                      NI     048(13),X'EF'        SWT+0
000912  58 F0 C 000                      L      15,000(0,12)         VIR=1
000916  05 EF                            BALR   14,15
000918  50 10 D 1B8                      ST     1,1B8(0,13)
00091C  12 00                            LTR    0,0
00091E  07 89                            BCR    8,9
000920  96 10 D 048                      OI     048(13),X'10'        SWT+0
000924  58 F0 C 004             INIT3    L      15,004(0,12)         VIR=2
000928  05 EF                            BALR   14,15
00092A  05 F0                            BALR   15,0
00092C  91 20 D 048                      TM     048(13),X'20'        SWT+0
000930  47 E0 F 016                      BC     14,016(0,15)
000934  58 00 B 048                      L      0,048(0,11)
000938  98 2D B 050                      LM     2,13,050(11)
00093C  58 E0 D 054                      L      14,054(0,13)
000940  07 FE                            BCR    15,14
000942  96 20 D 048                      OI     048(13),X'20'        SWT+0
000946  41 60 0 004                      LA     6,004(0,0)
00094A  41 10 C 020                      LA     1,020(0,12)          PN=01
00094E  41 70 C 058                      LA     7,058(0,12)          LIT+0
000952  06 70                            BCTR   7,0
000954  05 50                            BALR   5,0
000956  58 40 1 000                      L      4,000(0,1)
00095A  1E 4B                            ALR    4,11
00095C  50 40 1 000                      ST     4,000(0,1)
000960  87 16 5 000                      BXLE   1,6,000(5)
000964  41 80 D 1F4                      LA     8,1F4(0,13)          OVF=1
000968  41 70 D 207                      LA     7,207(0,13)          TS=01-1
00096C  05 10                            BALR   1,0
00096E  58 00 8 000                      L      0,000(0,8)
000972  1E 0B                            ALR    0,11
000974  50 00 8 000                      ST     0,000(0,8)
000978  87 86 1 000                      BXLE   8,6,000(1)
00097C  D2 03 D 224 C 054                MVC    224(4,13),054(12)    VN=01          VNI=1
000982  58 60 D 1FC                      L      6,1FC(0,13)          BL =3
000986  58 70 D 1F4                      L      7,1F4(0,13)          BL =1
00098A  58 80 D 1F8                      L      8,1F8(0,13)          BL =2
00098E  58 E0 D 054                      L      14,054(0,13)
000992  07 FE                            BCR    15,14
000000  90 EC D 00C             INIT1    STM    14,12,00C(13)
000004  18 5D                            LR     5,13
000006  05 F0                            BALR   15,0
```

```
000008  45 80 F 010                 BAL   9,010(0,15)
00000C  E3C5E2E3D9E4D540            DC    X'E3C5E2E3D9E4D540'
000014  C1D5E2F4                    DC    X'C1D5E2F4'
000018  07 00                       BCR   0,0
00001A  98 9F F 024                 LM    9,15,024(15)
00001E  07 FF                       BCR   15,15
000020  96 02 1 034                 OI    034(1),X'02'
000024  07 FE                       BCR   15,14
000026  41 F0 0 001                 LA    15,001(0,0)
00002A  07 FE                       BCR   15,14
00002C  00000924                    ADCON L4(INIT3)
000030  00000000                    ADCON L4(INIT1)
000034  00000000                    ADCON L4(INIT1)
000038  00000498                    ADCON L4(PGT)
00003C  00000248                    ADCON L4(TGT)
000040  00000510                    ADCON L4(START)
000044  000008E0                    ADCON L4(INIT2)
000048                              DS    15F
000084  00000000                    DC    X'00000000'
```

```
*STATISTICS*       SOURCE RECORDS =    84      DATA DIVISION STATEMENTS =    25      PROCEDURE DIVISION STATEMENTS =
*OPTIONS IN EFFECT*      SIZE =   81920  BUF =     2768  LINECNT = 57  SPACE1, FLAGW,   SEQ,   SOURCE
*OPTIONS IN EFFECT*        DMAP,    PMAP, NOCLIST, NOSUPMAP, NOXREF,   SXREF,   LOAD, NODECK, QUOTE, NOTRUNC,   FLOW=
*OPTIONS IN EFFECT*      NOTERM, NONUM, NOBATCH, NONAME, COMPILE=01,   STATE, NORESIDENT, NODYNAM, NOLIB, NOSYNTAX
*OPTIONS IN EFFECT*      NOOPT, NOSYMDMP
```

CROSS-REFERENCE DICTIONARY

DATA NAMES	DEFN	REFERENCE			
A	000055				
ALPHA	000041	000065			
ALPHABET	000040				
B	000056	000067			
DEPEND	000044	000068			
DEPENDENTS	000042				
FIELD-A	000027				
FIELD-A	000035				
FILE-1	000016	000061	000070	000075	
FILE-2	000017	000075	000078	000082	
KOUNT	000037	000061	000065	000068	000072
LOCATION	000050				
NAME-FIELD	000046	000065			
NO-OF-DEPENDENTS	000052	000068	000079		
NOMBER	000038	000061	000065	000069	
RECORD-NO	000048	000069			
RECORD-1	000026	000070			
RECORD-2	000034	000078			
RECORDA	000054				
WORK-RECORD	000045	000070	000078	000080	

PROCEDURE NAMES	DEFN	REFERENCE
BEGIN	000058	
STEP-1	000061	
STEP-2	000065	000072
STEP-3	000070	000072
STEP-4	000072	
STEP-5	000075	
STEP-6	000078	000080
STEP-7	000079	
STEP-8	000082	000078

CARD	ERROR MESSAGE	
	IKF1100I-W	2 SEQUENCE ERRORS IN SOURCE PROGRAM.
55	IKF2190I-W	PICTURE CLAUSE IS SIGNED, VALUE CLAUSE UNSIGNED. ASSUMED POSITIVE.

61
65

COBOL library subroutines perform operations that require such extensive coding that it would be inefficient to place the coding in the object module each time it is needed.

COBOL library subroutines are stored in the COBOL library (SYS1.COBLIB). The required subroutines are inserted in load modules by the linkage editor.

There are several major categories of COBOL library subroutines, namely: subprogram linkage, object-time program operations (i.e., data conversions, arithmetic operations, test conditions, data manipulation, data management, and special features), and object-time debugging. The categories are described in this order.

Table 35 later in this chapter includes a list of COBOL library subroutines, their storage requirements, and the associated calling information.

In addition, Q routines, which are not classified as COBOL library subroutines, are used to calculate the length of variable-length fields and the location of variably located fields resulting from an OCCURS clause with a DEPENDING ON option.

SUBROUTINES FOR SUBPROGRAM LINKAGE

The subroutines that control the loading of library subroutines or subprograms and the exiting from programs or subprograms are described here.

ENTER Subroutine (ILBONTR0)

The ILBONTR0 subroutine is used (1) when the RESIDENT option is an effect, to load one copy of each subroutine called by the main program or any of its subprograms into any region/partition; and (2) when the DYNAM option is in effect, to call any subprogram specified in a CALL literal or CALL identifier statement, first loading it if it has not already been loaded into that region/partition.

When a program finishes execution, this routine deletes all the subroutines called by the program except those subroutines that are being used by another program in the region/partition. It also deletes any subprogram in the CANCEL literal or CANCEL identifier statement.

STOP RUN Version 4 Subroutine (ILBOSRV0)

The ILBOSRV subroutine is called by all programs compiled by the Version 4 compiler. This routine returns control to the system, if the calling program is the main program, or to the caller, if it is not.

STOP RUN Subroutine (ILBOSTP0)

The ILBOSTP subroutine acts as a non-reenterable interface between a program compiled by the IBM Full American National Standard COBOL Version 3 Compiler, or a non-COBOL program and the Version 4 subroutine library. It may be entered from COBOL programs or subprograms.

OBJECT-TIME PROGRAM OPERATIONS

COBOL LIBRARY CONVERSION SUBROUTINES

Eight numeric data formats are permitted in COBOL -- five external (for input and output) and three internal (for internal processing).

The five external formats are these: (1) external or zoned decimal, (2) external floating-point, (3) sterling display, (4) numeric edited, and (5) sterling report. The three internal formats are these: (1) internal or packed decimal, (2) binary, and (3) internal floating-point.

The conversions from internal decimal to external decimal, from external decimal to internal decimal, and from internal decimal to numeric edited are done in-line. The other conversions are performed by the COBOL library subroutines shown in Table 33, and by the separate sign subroutine.

Separate Sign Subroutine (ILBOSSN0)

The ILBOSSN0 subroutine converts separately signed data-names to internal decimal format and then checks for a valid sign. If the sign is valid, this subroutine generates the corresponding overpunch in the receiving field. If not, it causes an object time message to be issued and the job to be terminated.

Table 33. Functions of COBOL Library Conversion Subroutine (Part 1 of 2)

Subroutine Name and Entry Points	Conversion From	Conversion To
ILBOEFL2	External Floating-point	Internal Decimal
ILBOEFL1	External Floating-point	Binary
ILBOEFL0	External Floating-point	Internal Floating-point
ILBOBID0[1]	Binary	Internal Decimal
ILBOBID1[1]		
ILBOBID2[1]		
ILBOBIE0[1]	Binary	External Decimal
ILBOBIE1[1]		
ILBOBIE2[1]		
ILBOBII0[2]	Binary	Internal Floating-point
ILBOBII1[2]		
ILBOTEF0[2]	Binary	External Floating-point
ILBOTEF1[2]		
ILBOTEF2	Internal Decimal	External Floating-point
IFBOTEF3	Internal Floating-point	External Floating-point
ILBOIBD0	Internal Decimal	Binary
ILBOIDB1	External Decimal	Binary
ILBODCI1	Internal Decimal	Internal Floating-point
ILBODCI0	External Decimal	Internal Floating-point
ILBOIFD0	Internal Floating-point	Internal Decimal
ILBOIFD1	Internal Floating-point	External Decimal

[1]The entry points used depend on whether the double-precision number is in registers 0 and 1, or 2 and 3, or 4 and 5, respectively.
[2]The entry points are for single-precision binary and double-precision binary, respectively.
[3]This entry point is used for calls from other COBOL library subroutines.

Table 33. Functions of COBOL Library Conversion Subroutines (Part 2 of 2)

Subroutine Name and Entry Points	Conversion	
	From	To
ILBOIFB1	Internal Floating-point	Binary integer and a power of 10 exponent
ILBOIFB2[3] ILBOIFB0[3]	Internal Floating-point	Binary
ILBOIDR0	Internal Decimal	Sterling Report
ILBOIDT0	Internal Decimal	Sterling Non-Report
ILBOSTI0	Sterling Non-Report	Internal Decimal
ILBOCVB0	External decimal	Binary
ILBOCVB1	External decimal	Binary

[1]The entry points used depend on whether the double-precision number is in registers 0 and 1, or 2 and 3, or 4 and 5, respectively.
[2]The entry points are for single-precision binary and double-precision binary, respectively.
[3]This entry point is used for calls from other COBOL library subroutines.

Table 34. Function of COBOL Library Arithmetic Subroutines

Subroutine Name	Function
ILBOXMU0	Internal Decimal Multiplication (30 digits * 30 digits = 60 digits)
ILBOXDI0	Internal Decimal Division (60 digits/30 digits = 60 digits)
ILBOXPR0	Exponentiation of an Internal Decimal Base by a Binary Exponent
ILBOFPW0	Floating-point Exponentiation
ILBOGPW0[1]	Floating-point Exponentiation

[1]The ILBOGPW0 entry point is used if the exponent has a picture specifying an integer. The ILBOFBW0 entry point is used in all other cases.

COBOL LIBRARY ARITHMETIC SUBROUTINES

Most arithmetic operations are performed in-line. However, involved calculations, such as exponentiation, and calculations with very large numbers, such as decimal multiplication of two 30-digit numbers, are performed by COBOL library subroutines. These subroutine names and their functions are given in Table 34.

COBOL LIBRARY SUBROUTINES FOR TESTING CONDITIONS AT OBJECT TIME

Several subroutines are used to test conditions that determine the path of control the object program selects. Such subroutines are described below.

Class Test Subroutine (ILBOCLS0)

The ILBOCLS0 subroutine is used to perform class tests for variable-length items and those fixed-length items over 256 bytes long, to determine whether a field is alphanumeric.

Note: The following tables are placed in the library for use by the in-line coding generated and the subroutines called for by both class test and TRANSFORM:

- ILBOATB0 -- alphabetic class test
- ILBOETB0 -- external decimal class test
- ILBOITB0 -- internal decimal class test
- ILBOTRN0 -- transformation
- ILBOUTB0 -- unsigned internal decimal class test
- ILBOWTB0 -- unsigned external decimal class test

COMPARE Subroutine (ILBOVCO0)

The ILBOVCO0 subroutine compares two operands, one or both of which are of variable lengths. They may exceed 256 bytes.

Compare with Figurative Constant Subroutine (ILBOIVL0)

The ILBOIVL0 subroutine compares the identifier to a figurative constant. The figurative constant must always be the second operand. If it is first in the source program, the operands are reversed and the condition code to be passed on is inverted before this subroutine is called.

COBOL LIBRARY DATA MANIPULATION SUBROUTINES

Subroutines are used to manipulate data in main storage in response to the MOVE, TRANSFORM, STRING, and UNSTRING statements. (Data manipulation in response to the EXAMINE statement is performed in-line by the object program.)

MOVE Subroutine (ILBOVMO0 and ILBOVMO1)

The MOVE subroutine is used when one or both operands is variable in length. They may exceed 256 bytes. The MOVE subroutine is also used for READ and WRITE statements processed in conjunction with the SAME RECORD AREA clause. The subroutine has two entry points, depending on the type of move: ILBOVMO0 (left-justified) and ILBOVMO1 (right-justified).

MOVE Subroutine for System/370 (ILBOSMV0)

This special MOVE subroutine is used when the length of the receiving field is either greater than 512 bytes or variable. The subroutine transfers characters to a right-justified receiving field.

MOVE to Alphanumeric-Edited Field Subroutine (ILBOANE0)

The ILBOANE0 subroutine moves a data-name, literal, or figurative constant into a right- or left-justified alphanumeric edited field.

MOVE to Numeric-Edited Field Subroutine (ILBONED0)

The ILBONED0 subroutine is called by the UNSTRING subroutine to move characters from a packed decimal field into a numeric-edited receiving field.

TRANSFORM Subroutine (ILBOVTR0)

The ILBOVTR0 subroutine translates variable-length items.

STRING Subroutine (ILBOSTG0)

The ILBOSTG0 routine combines the partial or complete contents of two or more subfield(s) into a single field. This routine transfers characters from the sending item(s) to the receiving item in the same way that moves from alphanumeric item(s) to alphanumeric item(s) are effected.

UNSTRING Subroutine (ILBOUST0)

The ILBOUST0 routine separates continuous data in a sending field, placing it in multiple receiving fields.

COBOL LIBRARY DATA MANAGEMENT SUBROUTINES

COBOL library subroutines are called to process the following verbs: DISPLAY, TRACE, EXHIBIT, ACCEPT, START (when generic key is specified), READ (BSAM), WRITE (BSAM), CLOSE (BSAM), OPEN (BSAM), RECEIVE (TCAM), and SEND (TCAM); library subroutines are also called for I/O errors, printer spacing, and printer overflow.

DISPLAY, TRACE, and EXHIBIT Subroutine (ILBODSP0)

The ILBODSP0 subroutine is used to print, punch, or type data, usually in limited amounts, on an output unit. TRACE and EXHIBIT are kinds of DISPLAY.

The acceptable forms of data for this subroutine are:

1. Display
2. External decimal
3. Internal decimal (converted by the subroutine to external decimal)
4. Binary (converted by the subroutine to external decimal)
5. External floating-point

Internal floating-point numbers must be converted to external floating-point numbers before the subroutine is called.

Note: If the contents of a data-name are such that when converted they will exceed 18 decimal digits, the ILBODSP0 subroutine cannot process them and the results are unpredictable.

DISPLAY Subroutine (ILBODSS0)

The ILBODSS0 subroutine prints or types data of a certain kind on SYSPRINT or at the console. This subroutine is used instead of ILBODSP0 when there are no requests by the program for TRACE or EXHIBIT, and no variable-length or floating-point items; when there are no requests for display upon SYSPUNCH; and when neither the RESIDENT nor the DYNAM option is in effect.

ACCEPT Subroutine (ILBOACP0)

The ILBOACP0 subroutine is called to read from SYSIN or from the operator's console at execution time. For SYSIN, a logical record size of 80 is assumed. If the size of the data item being accepted is less than 80 characters, the data must appear as the first set of characters within the input record. If the size of the data item is greater than 80 characters, as many records as necessary are read until the storage area allocated to the data item is filled. If the data item is greater than 80 characters, but is not an exact multiple of 80, the remainder of the last logical record is not accessible. For the console, a maximum of 114 characters are accepted and either 114 characters or the length of the item, whichever is smaller, is moved to the operand named in the ACCEPT statement.

Generic Key START Subroutine (ILBOSTR0)

The ILBOSTR0 subroutine is called when a USING KEY clause is coded with the START verb for ISAM files. The subroutine formats the search argument so that data management can get control to search for the generic key.

Checkpoint Subroutine (ILBOCKP0)

The ILBOCKP0 subroutine generates a checkpoint record, continuing the status of a program when a checkpoint is taken. This record is written on a checkpoint data set.

Error Intercept Subroutine (ILBOERR0)

The ILBOERR0 subroutine is used to test for various error conditions, and passes control to the interpretive-statement specified in the INVALID KEY option phrase or to the USE FOR ERROR declarative section depending on the type of error and error handling options specified. The entry points used for error processing by ILBOERR0 are:

ILBOERR1 Standard Sequential Files

ILBOERR2 Direct and Relative Files Accessed Sequentially
ILBOERR3 Indexed Files Accessed Sequentially
ILBOEER4 Direct and Relative Files Accessed Randomly
ILBOERR5 Indexed Files Accessed Randomly

Printer Overflow Subroutine (ILBOPTV0)

The ILBOPTV0 subroutine is used to control printer overflow testing and page ejection.

Printer Spacing Subroutine (ILBOSPA0)

The ILBOSPA0 subroutine is used to control printer spacing.

BSAM WRITE/CLOSE and BDAM OPEN Subroutine (ILBOSAM0)

The ILBOSAM0 routine processes input/output statements for direct or relative files accessed sequentially. It also handles OPEN statements and CLOSE statements with the REEL option for directly organized output files accessed randomly.

BSAM READ Subroutine (ILBOSPN0)

The BSAM read routine reads segments of a logical record and assembles them into a complete logical record. The routine is called by a compiler-generated READ code for a spanned record direct BSAM file.

RECEIVE Subroutine (ILBOREC0)

The ILBOREC0 subroutine transfers a message, a message segment, or part of a message or message segment from the message control program to the COBOL application program. This routine always updates the input communication description (CD) entry as well as processes the IF MESSAGE clause(s), if any.

RECEIVE Initialization Subroutine (ILBORNT0)

The ILBORNT0 subroutine builds the control block that communicates with the input queue associated with the cdname specified in the RECEIVE statement.

Queue Analyzer Object-Time Subroutine (ILBOSQA0)

The ILBOSQA0 subroutine is called by the ILBOREC0 routine if the COBTPQD data set is present. This routine searches the COBTPQD data set for a member that corresponds to the name in the SYMBOLIC QUEUE field (defined in the COBOL source statements). If a match is found, the analyzer reads the member into main storage, using it to validate the SYMBOLIC SUB-QUEUE name(s) in the input CD of the COBOL source program. The analyzer also identifies the first valid DD name for the queue structure and gives this name to the ILBOREC0 routine.

Queue Structure Description Subroutine (ILBOQSU0)

The ILBOQSU0 subroutine creates a partitioned data set with one member for each queue structure defined in the COBOL-like source statements. This routine also generates a printed listing of the structure element, as well as of error messages, if any.

SEND Subroutine (ILBOSND0)

The ILBOSND0 subroutine transfers a message, a message segment, or part of a message or message segment from the COBOL application program to the message control program. This routine always updates the output CD entry.

SEND Initialization Subroutine (ILBOSNT0)

The ILBOSNT0 subroutine (ILBOSNT0) subroutine builds the control block that communicates with the output queue associated with the cdname specified in the SEND statement.

COBOL LIBRARY SUBROUTINES FOR SPECIAL FEATURES

Subroutines are used for three of the special features of COBOL:

- Sort feature
- Table handling feature (SEARCH statement)
- Segmentation feature (GO TO statement)

Also, a subroutine is called in response to the use of the following special registers: CURRENT-DATE, DATE, DAY, TIME, and TIME-OF-DAY.

Sort Feature Subroutine (ILBOSRT0)

The ILBOSRT0 subroutine acts as an interface between the COBOL calling program and the Sort/Merge program via the entry point name SORT.

SEARCH Subroutine (ILBOSCH0)

The ILBOSCH0 subroutine performs a binary search on a specified level of a table. It is used for the SEARCH ALL statement.

Segmentation Subroutine (ILBOSGM0)

The ILBOSGM0 subroutine is used to load segments of a program that are not in core storage and to pass control from one segment to the other.

GO TO DEPENDING ON Subroutine (ILBOGDO0)

The ILBOGDO0 subroutine uses the value of a particular data-name as an index into a list of constants for each PN specified and then transfers control to the proper PN. If the value of the data-name is greater than the number of PN's specified, control returns to the next instruction after the calling sequence.

Date-and-Time Subroutine (ILBODTE0)

This group of subroutines performs five functions in response to the use of the special registers CURRENT-DATE, DATE, DAY, TIME, and TIME-OF-DAY. The list below indicates the function of each of the entry points, and the format of each result in the receiving field of the specified MOVE or ACCEPT statement.

ILBODTE0 -- day/month/year

ILBODTE1 -- hour minute second

ILBODTE2 -- year month day

ILBODTE3 -- year day

ILBODTE4 -- hour minute second hundredth of a second

OBJECT-TIME DEBUGGING

Three options are available for object-time debugging: the statement number option (STATE), the flow trace option (FLOW), and the symbolic debugging option (SYMDMP). The subroutines for the first two options provide debugging information at abnormal termination of a program; the subroutines for the third option provide debugging information either at abnormal termination or dynamically during the execution of a program. All of these subroutines are under the control of and are supervised by the debug control subroutine ILBODBG0. The debug control subroutine is described first, followed by the subroutines that are called in response to the specification of the STATE, FLOW, and SYMDMP options.

Debug Control Subroutine (ILBODBG0)

The ILBODBG0 subroutine is called once at entry point ILBODBG0 for each COBOL program for which any of the debugging options have been specified. This subroutine handles linkage and input/output for the STATE, FLOW, and SYMDMP options. It also produces the program name, the condition code, and the last PSW message at the time of the abnormal termination.

Flow Trace Subroutine (ILBOFLW0)

The ILBOFLW0 subroutine produces a formatted trace of the last "n" of COBOL procedures executed prior to an ABEND. It initializes, builds, and writes out the flow trace table.

Statement Number Subroutine (ILBOSTN0)

The ILBOSTN0 subroutine processes the STATE option and determines both the card number and the verb number for the last statement executed before the ABEND, and then generates a message containing this information.

Symbolic Dump Subroutine (ILBOD10 and ILBOD20)

The ILBOD10 subroutine is called when the SYMDMP option is in effect; this routine calls other modules as necessary for SYMDMP initialization. The ILBOD20 subroutine services SYMDMP output requests from DBG0. SYMDMP generates the following information as output on the SYSDBOUT data set: a copy of all SYMDMP control statements; diagnostic messages; dynamic dumps of user-selected data areas at strategic points during program execution; an abnormal termination statement number message; and the complete abnormal termination dump. In addition, modifications are made to the COBOL program in main storage if dynamic dumping is requested for the program.

Note: When SYMDMP services are requested for a job step, the sequence of events is, in general, as follows: (1) initialization -- for the first COBOL program in a job step, then for all other COBOL programs in that job step, and finally for independent program segments; (2) processing -- first for dynamic dump requests, and then for abnormal termination dumps.

SYMDMP Error Message Subroutine (ILBODBE0)

"ERRTN" of ILBODBG1 entry point

The ILBODBE0 subroutine is called by the PRINT routine of the debug control subroutine to format the appropriate error message in the SYSDBOUT output buffer.

For additional information on the FLOW, STATE, and SYMDMP options and their relationship to other COBOL options, see the chapter entitled "Symbolic Debugging Features" and the section "Options for the Compiler" in the chapter entitled "Job Control Procedures."

Table 35 includes a list of COBOL library subroutines, their storage requirements, and the associated calling information. The subroutines are arranged alphabetically by the characters following 'ILBO'. The list includes subroutines that are called directly by the object program -- primary subroutines -- and the subroutines they call -- secondary subroutines. Some subroutines (for example, ILBOANE) function as both primary and secondary subroutines.

The superscripts that accompany several of the entries refer to footnotes at the end of the table. Footnotes that appear with the names of subroutines indicate routines that are conditionally obtained, that are secondary subroutines only, or that may never reside in the MVT link pack area (LPA) or the MFT resident reusable routine area (RRR). The footnotes that appear with some of the numeric values indicate whether the information represents a maximum value, a minimum value, or an estimated value. In all cases, the numeric values represent decimal bytes rounded off to the nearest 50.

For descriptions of the primary subroutines and of the major secondary subroutines, see the sections of this appendix entitled "Subroutines for Subprogram Linkage," "Object-Time Program Operations," and "Object-Time Debugging."

Table 35. Calling and Storage Information for COBOL Library Subroutines (Part 1 of 5)

Primary Subroutine	Calling Information	Size	Dynamic Work Area	Secondary Subroutines	Size	Dynamic Work Area	Total Amount
ILBOACP (ACCEPT)	Called by compiled code	500	100	None			600 3002 23003
ILBOANE (MOVE alphanumeric-edited field)	Called by compiled code and by ILBOUST	350	0	None			350
ILBOANF (MOVE figurative constant)	Called by compiled code	150	0	None			150
ILBOATB (Alphabetic table for class test)	Used for ILBOCLS	300	0	None			300
ILBOBID (Binary to internal decimal)	Called by compiled code	150	0	None			150
ILBOBIE (Binary to external decimal)	Called by compiled code	150	0	None			150
ILBOBII (Binary to internal floating-point)	Called by compiled code and by ILBODCI, ILBOEFL	500	0	None			500
ILBOCKP (Checkpoint)	Called by compiled code	100	0	None			100
ILBOCLS (Class test)	Called by compiled code	150	0				150
ILBOCOM[4] [9] (Subroutine communications)	Link-edited or loaded by compiled code and by ILBOSRV; used by most COBOL library subroutines	150	0	None			150
ILBOCVB (Decimal to binary/binary to decimal)	Called by compiled code and by ILBOUST and ILBOSTG	1050	300[5]	None			1350
ILBODBG (Debug control)	Called by compiled code if FLOW, STATE, or SYMDMP is specified	2725,	950 [7]	ILBODBG1			
				ILBODBG2			
				ILBODBG3			
				ILBODBG4			
				ILBODBG5			
				ILBODBG7			
				ILBODBE[7] [8]	1200	0	
				ILBOSTN[7]	1109	110	
				ILBOFLW[7]	1600	600[3]	
				ILBOD01[7] [8]	750		

Table 35. Calling and Storage Information for COBOL Library Subroutines (Part 2 of 5)

Primary Subroutine	Calling Information	Size	Dynamic Work Area	Secondary Subroutines	Size	Dynamic Work Area	Total Amount
ILBOD010	Called when SYMDMP is in effect			ILBOD10	2550[1]	3100	
				ILBOD11[7 8]	750	0	
				ILBOD12[7 8]	1750[1]	0	
				ILBOD13[7 8]	1550	0	
				ILBOD14[7 8]	1500	0	
				ILBOD20[7 8]	950	0	
				ILBOD21[7 8]	1500	25/ODO	
				ILBOD22[7 8]	2500	0	
				ILBOD23[7 8]	3800	0	
				ILBOD24[7 8]	4050[1]	0	
				ILBOD25[7 8]	1200[1]	0	
ILBODCI (Decimal to internal floating-point)	Called by compiled code			ILBOIDB	150	0	822
ILBODSP (DISPLAY, TRACE, EXHIBIT)	Called by compiled code	1938	104	None			2100
ILBODSS[4] (DISPLAY)	Called by compiled	350[1]	0	None			350[1]
ILBODTE (Date, day, and time)	Called by compiled code	500	0	None			350
ILBOEFL (Conversion from external floating-point)	Called by compiled code	600	0	ILBOIOB ILBOBII	150 450	0 0	1200
ILBOERR (Error intercept)	Called by the system	500	0	None			500
ILBOETB (External decimal table for class test)	Used by ILBOCLS	300	0	None			300
ILBOFLW (Flow trace option)	Called by compiled code and by ILBODBG	1600	600[3]	None			2300[3]
ILBOFPW (Floating-point exponentiation)	Called by compiled code	800	0	None			800
ILBOGDO (GO TO DEPENDING ON)	Called by compiled code	100	0	None			100
ILBOGPW (Floating-point exponentiation to a binary exponent)	Called by compiled code	100	0	None			100
ILBOIDB (Decimal to binary)	Called by compiled code or by ILBODCI	150	0	None			150

Table 35. Calling and Storage Information for COBOL Library Subroutines (Part 3 of 5)

Primary Subroutine	Calling Information	Size	Dynamic Work Area	Secondary Subroutines	Size	Dynamic Work Area	Total Amount
ILBOIDR (Internal decimal to sterling report)	Called by compiled code	1700	0	None			1700
ILBOIDT (Internal decimal to sterling non-report)	Called by compiled code	600	0	None			600
ILBOIFB (Internal floating-point to decimal or binary)	Called by compiled code or by ILBOIFD or ILBOTEF	300	0	None			300
ILBOIFD (Internal floating to decimal or binary)	Called by compiled code	200	0	ILBOIFB	300	0	
ILBOITB (Internal decimal table for class test)	Called by compiled code	300	0	None			300
ILBOIVL (Comparison with figurative constant)	Called by compiled code	100	0	None			100
ILBOPTV (Printer overflow)	Called by compiled code	150	0	None			150
ILBOQSU[4] [10] (Queue structure utility program)	Called by JCL	6500	4000	None			10,500
ILBOREC (RECEIVE)	Called by compiled code	2400	0	ILBORNT[7] ILBOSOA[10]	900 2000	255/queue blocks 200/buffer units PDS member SIZE	3800[2]
ILBOSAM (BSAM WRITE and CLOSE/BDAM OPEN)	Called by compiled code	1104	0	None			1104
ILBOSCH (SEARCH)	Called by compiled code	700	0	None			700
ILBOSGM (Segmentation)	Called by compiled code	400	0	ILBODBG	2000[1]	600[2]	3000[1] [2]

Table 35. Calling and Storage Information for COBOL Library Subroutines (Part 4 of 5)

Primary Subroutine	Calling Information	Size	Dynamic Work Area	Secondary Subroutines	Size	Dynamic Work Area	Total Amount
ILBOSMV (MOVE to right-justified field for System/370)	Called by compiled code	50	0	None			50
ILBOSND (SEND)	Called by compiled code	1450	0	ILBOSNT[1]	600	255/queue blocks 200/buffer 200/buffer	
ILBOSPA (Printer spacing)	Called by compiled code		0	None			1000
ILBOSRT (Sort)	Called by compiled code	900	200	None			1100
ILBOSRV (STOP RUN for Version 4)	Called by a program compiled by the Version 4 compiler	300	0	None			300
ILBOSSN (Separately signed numeric)	Called by compiled code	200[1]	0	ILBOSRV ILBODBG	300 2000[1]	0 600[2]	3100[1 2]
ILBOSTG (STRING)	Called by compiled code	600	0	ILBOCVB	1050	300	1950
ILBOSTI (Sterling non-report to internal decimal)	Called by compiled code	600	0	None			600
ILBOSTN (Statement number option)		1100	110	ILBODBG	2000[1]	600[2]	3950[1 2]
ILBOSTP (STOP RUN)	Called by a non-COBOL program	100[1]	0	ILBOSRV	300	0	400[1]
ILBOSTR (START with generic key)	Called by compiled code	100	0	None			100
ILBOTEF (Conversion to external floating-point)	Called by compiled code or by ILBOD23	700	0	ILBOBIE	150	0	
ILBOTRN (TRANSFORM table)	Used by ILBOVTR	300	0	None			300
ILBOUST (UNSTRING)	Called by compiled code	2000	250[5]	ILBONED[7 8] ILBOANE[7 8] ILBOCVB[7]	1400 350 1050	 0	5050[3]

Table 35. Calling and Storage Information for COBOL Library Subroutines (Part 5 of 5)

Primary Subroutine	Calling Information	Size	Dynamic Work Area	Secondary Subroutines	Size	Dynamic Work Area	Total Amount
ILBOVTB (Unsigned internal decimal table for class test)	Called by compiled code	300	0	None			300
ILBOVCO (Variable-length comparison)	Called by compiled code	550	0	None			550
ILBOVMO (Variable-length name)	Called by compiled code	600	0	ILBOSRV	300	0	
				ILBOADR[7]	300	0	
				ILBODBG[7]	2000[1]	600	3200[1,2]
ILBOVTR (TRANSFORM)	Called by compiled code	150	0	None			150
ILBOWTB (Unsigned external decimal table for class test)	Used by ILBOCLS	300	0	None			300
ILBOXDI (Decimal division)	Called by compiled code and by ILBOXPR	300	0	None			300
ILBOXMU (Decimal multiplication)	Called by compiled code and by ILBOXPR	200	0	None			200
ILBOXPR (Decimal fixed-point exponentiation)	Called by compiled code	700	0	ILBOXDI	300	0	

Notes:

1. The size given is an estimate.
2. The size given is a minimum.
3. The size given is a maximum.
4. The subroutine indicated may never reside in the MVT link pack area (LPA) or the MFT resident reusable routine area (RRR).
5. The 256-byte storage area obtained by subroutine ILBOCVB is used by subroutine ILBOUST.
6. Because the ILBODBG subroutine dynamically loads and deletes subroutines as they are needed, depending on the options specified, it is possible only to estimate a minimum and/or a maximum amount of storage used by any one of the debugging options. For each storage estimate given below, the effect of possible core fragmentation is not considered.
   a. Basic debug package -- 3100 bytes
   b. Debug with the STATE option -- 3950 bytes
   c. Debug with the FLOW option -- 892 bytes
   d. Debug with the SYMDMP option -- 11,372 bytes minimum and 13,348 bytes maximum
   e. Debug with dynamic dumps -- 15,112 bytes minimum and 18,198 bytes maximum; (whenever a primary routine calls subroutine ILBODBG, the storage requirement is that for the basic package size).
7. The subroutine or dynamic work area indicated is obtained conditionally.
8. The subroutine indicated is never called as a primary subroutine.
9. The subroutine indicated must be on-line at execution time.

In this appendix, each field of the data control block is listed by the name of the operand of the assembler-language macro instruction that can specify a value for that field. Tables 36 through 40 illustrate the data control blocks for sequential, direct, relative, and indexed files. Some of the data control block fields can be referred to with the DCB parameter of the DD statement. However, any field filled in by the COBOL compiler cannot be overridden except for the indexed file OPTCD field in which the L-subparameter is set by the compiler using DCB exit.

Values for fields for which no entry appears in the column headed "COBOL Source" may be supplied by the DD statement or by the data set label.

For information concerning the specification of values for data control block fields, see the DCB macro instruction for the different file processing techniques in the publication _IBM OS Supervisor and Data Management Macro Instructions_.

Note: The DCB subparameters are discussed under "User Defined Files" in the chapter "User File Processing."

Table 36. Data Control Block Fields for Standard Sequential Files

Data Control Block Field	Explanation of Field	COBOL Source	Applicable DD Statement DCB Subparameters
BFALN	Alignment	(COBOL specifies doubleword boundary)	
BFTEK	Buffering technique (S or E)	(COBOL specifies S)	
BLKSIZE	Maximum length of block	BLOCK CONTAINS Data record description	BLKSIZE
BUFCB	Address of buffer pool	SAME AREA	
BUFL	Length of each buffer		
BUFNO	Number of buffers assigned to DCB	RESERVE	BUFNO=N(default=2)
BUFOFF			(BUFOFF=[n|1])
DDNAME	Name of DD statement	ASSIGN clause	
DSORG	Access method	ASSIGN clause ACCESS clause	
EODAD	Address of user's end-of-data-set exit routine for input data set	READ...AT END	
EROPT	Error option		(EROPT=[ACC|SKP|ABE])
EXLST	Address of exit list	Used by the compiler for USE...LABEL, etc.	
LRECL	Logical record length	FD entry	LRECL
MACRF	Type of macro instruction	OPEN INPUT, READ OPEN OUTPUT, WRITE OPEN I-O, READ, WRITE REWRITE	
OPTCD	Optional service provided by control program		(OPTCD=[W|C|WC|I|Q])
RECFM	Characteristics of records in data set	RECORDING MODE Record description ADVANCING POSITIONING BLOCK CONTAINS APPLY RECORD-OVERFLOW	(RECFM=D)
SYNAD	Address of error exit routine	Used by compiler for INVALID KEY and USE AFTER ERROR	RECFM={S|T}

Table 37. Data Control Block Fields for Direct and Relative Files Accessed Sequentially

Data Control Block Field	Explanation of Field	COBOL Source	Applicable DD Statement DCB Subparameters
BLKSIZE	Maximum length of block	Data record description	
DDNAME	Name of DD statement	ASSIGN clause	
DSORG	Access method	ASSIGN clause ACCESS clause	
EODAD	Address of end-of-data-set exit (input)	READ...AT END	
EXLST	Address of exit list	USE...LABEL PROCEDURE	
KEYLEN	Length of key	ACTUAL KEY[1] (length of ACTUAL KEY - 4)	
LRECL	Logical record length	FD entry	LRECL
MACRF	Type of macro instruction	OPEN INPUT, READ OPEN OUTPUT, WRITE (DIRECT ONLY)	
OPTCD	Optional service to be provided by control program		[OPTCD=W|T]
RECFM	Characteristics of records in data set	RECORDING MODE Record description APPLY RECORD-OVERFLOW	
SYNAD	Address of error exit routine	USE AFTER ERROR INVALID KEY	

[1]Direct files only; for relative files, the field is 0.

Table 38. Data Control Block Fields for Direct and Relative Files Accessed Randomly

Data Control Block Field	Explanation of Field	COBOL Source	Applicable DD Statement DCB Subparameters
BLKSIZE	Maximum length of block	Data record description	
DDNAME	Name of DD statement	ASSIGN clause	
DSORG	Access method	ASSIGN clause ACCESS clause	
EXLST	Address of exit list	USE...LABEL, etc.	
KEYLEN	Length of key for each physical record	ACTUAL KEY[1] (length of ACTUAL KEY - 4)	
LIMCT	Search limits		LIMCT=n (OPTCD=E must be specified)
MACRF	Type of macro instruction	OPEN INPUT, READ OPEN OUTPUT, WRITE (DIRECT ONLY) OPEN I-O, READ, WRITE (DIRECT ONLY), REWRITE	
OPTCD	Option service to be provided by the control program		OPTCD=E/W
RECFM	Characteristics of records of data set	RECORDING MODE APPLY RECORD-OVERFLOW Record description	
SYNAD	Address of error exit routine	Used by compiler for INVALID KEY and USE AFTER ERROR	

[1]Direct files only, for relative files this field is 0.

Table 39. Data Control Block Fields for Indexed Files Accessed Sequentially

Data Control Block Field	Explanation of Field	COBOL Source	Applicable DD Statement DCB Subparameters
BFALN	Buffer alignment (F or D)	(COBOL specifies D)	
BKLSIZE	Maximum length of block	BLOCK CONTAINS	BLKSIZE
BUFCB	Address of buffer pool	SAME AREA	
BUFNO	Number of buffers assigned to DCB	RESERVE	BUFNO=N(default=2)
CYLOFL	Number of overflow tracks for each cylinder		CYLOFL=XX
DDNAME	Name of DD statement	ASSIGN clause	
DSORG	Access method	ACCESS clause ASSIGN clause	
EODAD	Address of user's end-of-data-set exit routine for input data set	READ...AT END	
EXLST	Address of exit list	Used by the compiler	
KEYLEN	Length of key for each logical record	RECORD KEY	
LRECL	Logical record length	FD entry	LRECL
MACRF	Type of macro instruction	OPEN INPUT, READ, START OPEN OUTPUT, WRITE OPEN I-O, READ, START, REWRITE	
NTM	Maximum number of cylinder index tracks		NTM=XX
OPTCD	Optional services		OPTCD=I|R|W|Y|M|U|L (must also have NTM=M)
RECFM	Characteristics of records in data set	RECORDING MODE RECORD DESCRIPTION BLOCK CONTAINS	
RKP	Relative position of record key in logical record	RECORD KEY	
SYNAD	Address of error exit routine	Used by the compiler for INVALID KEY, USE AFTER ERROR	

Table 40. Data Control Block Fields for Indexed Files Accessed Randomly

Data Control Block Field	Explanation of Field	COBOL Source	Applicable DD Statement DCB Subparameters
BFALN	Buffer alignment (F or D).	(COBOL specifies D)	
DDNAME	Name of DD statement.	ASSIGN clause	
DSORG	Access method.	ACCESS clause ASSIGN clause	
EXLST	Address of exit list.	Used by the compiler	
KEYLEN	Key length.	NOMINAL KEY	
LRECL	Logical record length.	FD entry	
MACRF	Type of macro instruction.	OPEN INPUT, READ OPEN I-O, READ, WRITE, REWRITE,	
MSHI	Address of area for highest level index of data set.	APPLY CORE-INDEX	
MSWA	Address of area reserved for control program. Required for variable length records.	TRACK-AREA	
SMSI	Size for area provided for highest level index of the data set.	APPLY CORE-INDEX	
SMSW	Number of bytes reserved for main storage work area.	TRACK-AREA	

In general, compilation is faster when:

1. Options in the EXEC statement are specified to:

   a. Make more main storage available (the SIZE option)

   b. Optimize the space available for buffers (the BUF option)

   c. Suppress output (the NOSOURCE, NODECK, NOLOAD, and the SUPMAP options, among others)

   d. Suppress object code if one or more E-level messages are generated.

2. The maximum block size for a compiler data set is specified.

3. A disk configuration and separate channels for utility data sets are used.

4. Separate devices (i.e., not the same mass storage unit) on the same channel are used.

Compilation time is also affected by the speed of the devices allocated to the data sets. For example, a tape device is faster than a printer for printed output. The blocking information that follows applies to MFT or MVT.

PERFORMANCE CONSIDERATIONS

The OS Full American National Standard COBOL Compiler, Version 4, provides additional opportunities for saving either main storage or time. For example, specification of the Optimized Code Feature, the COBOL Library Management Feature, the Dynamic Subprogram Feature, or all three of these features, can result in a considerable saving in main storage. The notes given below provide additional performance information on programs run with these and other new features.

- When the Optimized Code Feature is requested, via the OPTIMIZE compiler option, execution time is reduced for non-I/O bound programs; however, compilation time is increased.

- Specification of the COBOL Library Management Facility, via the RESIDENT compiler option, results in a saving of both main storage and secondary storage, as well as of time at the link-edit step and the initial program load for the program.

- Dynamic invocation and release of COBOL subprograms, specified by the DYNAM compiler option, also results in savings in main storage.

- A syntax checking compilation, specified by the SYNTAX or SYNTAX compiler option, saves machine time. Depending on which compiler options are chosen, as well as the various source program statements, compile time can be reduced by as much as 20% to 70%.

The symbolic dump feature, specified by the SYMDMP option, can save much debugging time. However, use of this option can decrease performance expectations for programs run with it. That is, such programs require additional time for the compile, link-edit, and execute job steps. They also require more main storage than programs run without this feature.

For information about requesting any of these options,see the section "Options for the Compiler" in the chapter on "Job Control Procedures".

BLOCK SIZE FOR COMPILER DATA SETS

The blocking factor specified for compiler data sets other than utility data sets must be permissible for the device the data set is on. In addition, for the SYSLIN data set, it must be permissible for the linkage editor used. (Any block size specified for a utility data set in a DD statement is overridden by the compiler.) If a block size other than the default option is needed, it can be requested by specifying the BLKSIZE subparameter of the DCB parameter in the DD statement for the data sets. The format of the subparameter is:

```
DCB=(,BLKSIZE=nnn)
```

where nnn is equal to N times the logical record size in bytes, and $1 \le N \le M$. M is equal to the blocking factor permissible for the device, and, in the case of SYSLIN, to the blocking factor permissible for the linkage editor used.

If blocking is desired, the record format for SYSPRINT [DCB=(,RECFM=nnn)] should be specified as FBA. The record format for SYSIN, SYSLIN, SYSPUNCH, and SYSLIB should be specified as FB.

The logical record size for SYSPRINT is 121 bytes. The logical record size for SYSIN, SYSLIN, SYSPUNCH, and SYSLIB is 80 bytes.

Note: For compile, link-edit, and execute cases when labeled volumes are used, RECFM and BLKSIZE must be given for SYSLIN in the compile step only. If BLKSIZE is specified for SYSPUNCH, LRECL must also be specified. The 44K version of the linkage editor supports input data sets with a blocking factor of up to 40 specified.

HOW BUFFER SPACE IS ALLOCATED TO BUFFERS

Once the amount of space available for a compilation is determined, the compiler subtracts the amount required for itself. From the space remaining, it then computes the space available for utility and input/output data set buffers. If space still remains, the compiler makes use of it for internal processing.

Once the amount of space available for buffers is determined, the compiler calculates how this space is to be divided. First, it computes the amount of space required for the buffers of the input/output data sets. From the space remaining, it determines the maximum buffer size, and hence block size, possible for a utility data set. The utility data sets all have the same block size. Thus, the block size of a utility data set is dependent on the amount of space available for buffers. If a block size has been specified in a DD statement for a utility data set, it is overridden.

A larger buffer size for a utility data set allows for faster processing. However, if the program being compiled takes up a large amount of the available storage, a smaller space for buffers enables the compiler to use more main storage for internal processing.

The following describes how the space available for buffers is determined and how it is allocated to buffers.

Let A represent the space that can be allocated to these buffers. It is determined as follows:

1. If neither the BUF nor the SIZE option of the PARM parameter of the EXEC statement is specified, A equals the default value for buffer space. This value is specified at system generation time. The minimum value is 2768 bytes except when BATCH is specified, when it is 2928 bytes.

2. If the SIZE option is specified, but BUF is not, A equals (SIZE - 80K) / 4 plus the default value for buffer space.

3. If BUF is specified (whether or not SIZE is specified), A equals the value specified for BUF.

   Note: The minimum difference between SIZE and BUF must always be equal to or greater than the difference between the minimum SIZE value and the minimum BUF value (81920 bytes - 2768 bytes; or, when BATCH is specified, 81920 bytes - 2928 bytes).

4. If BUF is smaller than 2768, or for BATCH 2928, bytes (the minimum value), a warning message is printed and the minimum value is assumed. If BUF is too large to allow minimum table space for compilation, a warning message is printed and the default value (or the minimum value, if the default value is also too large) is assumed.

The programmer must make sure that the amount of buffer space allocated by the system is sufficient, taking into consideration the block sizes specified for the compiler data sets. The allocated buffer space is divided as follows:

1. Let B represent the amount of buffer space to be allocated for input/output data sets. B is computed as either equal to:

   2 times the block size of SYSPRINT + SYSIN + SYSLIB

   or

   2 times the block size of SYSPRINT + SYSPUNCH + SYSLIN

   whichever is larger. The maximum allowable value of B is A - 1280 bytes. If the computed value is greater than the maximum allowable value, a diagnostic message is printed and compilation is abandoned.

   If the block sizes are not specified in the DD statements, the following default values are assumed:

Data Set	Default Value (bytes)
SYSIN	80
SYSLIN	80
SYSPUNCH	80
SYSLIB	80
SYSPRINT	121
SYSTERM	121

Let C represent the amount of buffer space to be allocated for each utility data set. Therefore, C equals the block size of data sets, SYSUT1, SYSUT2, SYSUT3, and SYSUT4, respectively.

$$\text{If } A \le 6B, \text{ then } C = \frac{A - B}{5}$$

$$\text{If } A > 6B, \text{ then } C = \frac{A}{6}$$

If C > maximum block size permitted for any device a utility data set is on, then the maximum block size is the value chosen for C. The minimum block size for a utility data set is 255 bytes.

The COBOL compiler can be invoked by a problem program at execution time through the use of the ATTACH or the LINK macro instruction, i.e., dynamic invocation. Dynamic invocation of COBOL compiled programs can be accomplished through the use of the LINK or the LOAD macro instruction.

INVOKING THE COBOL COMPILER

The problem program must supply the following information to the COBOL compiler:

- The options to be specified for the compilation
- The ddnames of the data sets to be used during processing by the COBOL compiler
- The header to appear on each page of the listing

Name	Operation	Operand
[symbol]	LINK ATTACH	EP=IKFCBL00, PARAM=(optionlist [,ddnamelist], [,headerlist]),VL=1

where:

EP
: specifies the symbolic name of the COBOL compiler. The entry point at which execution is to begin is determined by the control program (from the library directory entry).

PARAM
: specifies, as a sublist, address parameters to be passed from the problem program to the COBOL compiler. The first fullword in the address parameter list contains the address of the COBOL option list. The second fullword contains the address of ddname list. If standard ddnames are to be used and no header list is specified, this list may be omitted. If standard ddnames are to be used and a header list is specified, this entry should contain the address of a word of binary zeros, aligned on a halfword. The last fullword contains the address of the header list. This list may be omitted.

option list
: specifies the address of a variable length list containing the COBOL options specified for compilation. For additional details, see the description of the EXEC statement in the chapter "Job Control Procedures." This address must be written even though no list is provided.

The option list must begin on a halfword boundary. The two high-order bytes contain a count of the number of bytes in the remainder of the list. If no options are specified, the count must be zero. The option list is free form with each field separated from the next by a comma. No blanks or zeros should appear in the list.

ddname list
: specifies the address of a variable length list containing alternative ddnames for the data sets used during COBOL compiler processing. If standard ddnames are used, this operand may be omitted.

The ddname list must begin on a halfword boundary. The two high-order bytes contain a count of the number of bytes in the remainder of the list. Each name of less than eight bytes must be left justified and padded with blanks. If an alternate ddname is omitted from the list, the standard name will be assumed. If the name is omitted within the list, the 8-byte entry must contain binary zeros. Names can be omitted from the end merely by shortening the list.

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

ddname 8-byte Entry	Name for Which Substituted
1	SYSLIN
2	not applicable
3	not applicable
4	SYSLIB
5	SYSIN
6	SYSPRINT
7	SYSPUNCH
8	SYSUT1
9	SYSUT2
10	SYSUT3
11	SYSUT4
12	SYSTERM
13	SYSUT5

header list
: specifies the address of a variable-length list containing information to be included in the heading on each page of the listing. The list must begin on a halfword boundary. The two high-order bytes should contain a count of the number of bytes in the new heading information; the next four bytes of the list should contain the page number at which the heading is to start, in EBCDIC format.

VL
: specifies that the sign bit is to be set to 1 in the last fullword of the address parameter list.

When the COBOL compiler completes processing, a return code is placed in register 15. For additional details, see the discussion of the COND parameter in the chapter "Job Control Procedures."

INVOKING COBOL COMPILED PROGRAMS

Linkage editor control cards should be specified as follows:

1. For the PROGRAM-ID program-name, a NAME card.
2. For each ENTRY literal-1, an ALIAS card should be specified in a COBOL program that is to be dynamically invoked.

Limitations on the size of a COBOL source program should be considered in relation to the capacities of both the COBOL compiler and the various linkage editors. This appendix contains information to aid the programmer in determining how his source program affects usage of space at compilation time and linkage editing time.

COMPILER CAPACITY

The capacity of the COBOL compiler is limited by two general conditions: (1) the total contiguous space available must be sufficient for compilation and (2) an individual table may not have a length greater than 32,767 bytes, with the exception of the ADCON and cross-reference tables. If either of these conditions is not met during compilation, one of the following error messages will be issued:

IKF0001I-D SIZE PARAMETER TOO SMALL FOR THIS PROGRAM.

IKF0010I-D A TABLE HAS EXCEEDED THE MAXIMUM PERMISSIBLE SIZE.

In either case, compilation is terminated. However, in the first case, the program may be recompiled with a larger SIZE parameter. The size of the ADCON and cross-reference tables is not limited to 32,767 bytes.

If a table overflows, the following error message will be generated, and the user will need to rerun the program in a larger region.

IKF6007I-D TABLE OVERFLOW. PMAP LOAD MODULE OR DECK WILL BE INCOMPLETE. INCREASE SIZE PARAMETER.

Minimum Configuration SOURCE PROGRAM Size

The compiler will accept and compile a 1500 card program in the minimum machine configuration (80K). Within an 80K byte environment, the user should not specify buffer size for the compiler files. Of course, the various reader procedures may affect the value required for SIZE and BUF parameters. The compiler will allocate the minimum required amounts that are 256 bytes for each of the 4 intermediate files, 80 bytes for each system file with the exception of SYSOUT for which 120 bytes are allocated. Double buffering will be assumed.

Within this configuration, assuming no REPORT SECTION, the compiler will accept:

- Three hundred procedure references assuming an average procedure-name length of 12 characters
- Twenty-five OCCURS clauses with the DEPENDING ON option
- Ten files assuming an average of three subordinate record entries
- Four hundred literals assuming an average of eight bytes

EFFECTIVE STORAGE CONSIDERATIONS

The amount of core storage within the compiler's partition and the limitation on the size of an individual internal table are two factors that limit the capacity of the compiler. The limitation on the size of internal tables can, in some instances, be overcome by the spilling over of some tables onto external devices. However, spilling over may cause a severe degradation of performance. The core storage limitation should not be reached by any reasonable use of the language. However, within a limited storage capacity excessive use of certain features and combination of features in the language could make compilation impossible. Some of the features that significantly affect storage usage are the following:

1. ADCON Table

   Each entry occupies 8 bytes. This table is not limited to the maximum size of 32,767 bytes. Entries are based on the:

   - Number of 4096-byte segments in the Working-Storage Section
   - Number of 4096-byte segments in a file buffer area
   - Number of referenced procedure-names
   - Number of implicit procedure-name references such as those generated by IF, SEARCH, and GENERATE

statements, ON SIZE ERROR, INVALID KEY, and AT END options, the OCCURS clause with the DEPENDING ON option, USE sentences, and the Segmentation feature.

- Number of files

2. Procedure-name Table

   This table contains the number of definitions written in a section and unresolved procedure references. Procedure references are resolved at the end of a section if the definition of the procedure-name is in that section or a preceding section. Therefore, forward references beyond a section impact space. Approximately 900 unqualified entries are possible. A maximum number of 16,255 entries may be specified.

3. OCCURS DEPENDING ON Table

   This table contains an entry for each unique object of an OCCURS clause with the DEPENDING ON option. The size of an entry is 2 + length of name + length of each qualifier bytes.

4. Index Table

   An entry is made for each INDEXED BY clause consisting of 11 bytes for each index.

5. File Table

   An entry is made for each file specified in the program. Each entry occupies 60 bytes of storage.

6. Report Writer Tables

   A considerable amount of information is maintained for each RD such as controls, sums, headings, footings, routines to be generated, and so on. The contents of the table are increased by qualification and subscripting in the Report Section. Approximately 30 reports can be processed without exceeding the limit of the table.

7. Dictionary Table

   An entry is made for each procedure-name and each data-name in the program. A procedure entry consists of (7 or 9 + length of name) bytes. A data entry consists of (length of name + n) bytes, where n is determined by the attributes of the data item. Some of the features that contribute to the value n are:

   - One byte for each character in a numeric edited or alphanumeric edited item picture
   - Five bytes for an elementary item with a Sterling Report picture clause
   - Three bytes for an item subordinate to an OCCURS clause

8. Literal Tables

   The total length of all literals may not exceed 32511 bytes. No more than 16255 literals may be specified.

9. Miscellaneous Tables

   The presence of the following items causes entries to be made into tables that affect the total space required for compilation.

   - SAME [RECORD] AREA clause
   - Subscripting
   - Intermediate Arithmetic Results
   - Complex Arithmetic Expressions
   - Complex Logical Expressions
   - APPLY clauses
   - Special-Names
   - RERUN clauses
   - Error messages
   - XREF
   - Segmentation feature

LINKAGE EDITOR CAPACITY

Some COBOL program and linkage editor considerations are listed below as a further guide in preparing a source program. Consult the publication IBM OS Linkage Editor and Loader, for additional information on linkage editor capacities and processing.

1. All COBOL object programs, with the exception of segmented programs, consist of a single CSECT (control section). The size of the object module may be determined by looking at the location of the last instruction in INIT3 in the object code listing (see the section entitled "Output") or from the END card.

2. The size of the object module is greatly increased by any of the following:

   a. The blocking factor and alternate area reservation of randomly accessed files

b. The specification of the SAME AREA clause for sequentially accessed files

3. RLD (Relocation List Dictionary) cards are part of the load module, and are used by the linkage editor to compute the address constants for the load module. The number of RLDs produced by the compiler can be determined by the following formula:

   number of RLDs = number of unique subprograms called + number of COBOL library routines called + number of nonresident segments

4. The output text of the compiler is written out in a sequence that differs from the order indicated by the location counters contained in each output item. This sequence difference may result in a strain on the facilities of the linkage editor.

5. VALUE clauses in the Working-Storage Section may result in many discontinuous text records.

6. The object module produced by the COBOL compiler may not be sorted prior to the linkage editor step.

This appendix contains a brief summary of input/output (I/O) error conditions for each of the file processing techniques. More detailed information on error conditions can be found in the following publications:

IBM OS Supervisor and Data Management Macro Instructions

IBM OS System Control Blocks

STANDARD SEQUENTIAL, DIRECT, AND RELATIVE FILE PROCESSING TECHNIQUE (SEQUENTIAL ACCESS)

Register 1 contains error bits indicating the exact cause of an error. Conditions causing input/output errors and suggested user responses are as follows:

- *I/O Error Conditions*:

1. Input Error
2. Output Error
3. Invalid Request (BSAM only)

Suggested User Response:

For BSAM, display the error message. Processing of the file is limited to CLOSE. For QSAM, display the error message and then execute the EROPT option in the DD statement. Note that the EROPT option gives the user three choices:

- ACC - Accept the error block and continue processing
- SKP - Skip to the next block.
- ABE - Terminate the job.

DIRECT AND RELATIVE FILE PROCESSING TECHNIQUE (RANDOM ACCESS)

The DECB contains two error condition bytes at location DECB + 4. Conditions causing input/output errors and suggested user responses are as follows:

- *I/O Error Conditions*:

1. Record Not Found
2. Invalid Request
   a. Requested block outside data set.
   b. Attempt to add fixed-length record with key beginning with hexadecimal FF.

Suggested User Response:

Condition caused by invalid key. Processing of the file may be continued.

- *I/O Error Condition*

Space Not Found

Suggested User Response:

Processing of the file may be continued. CLOSE, READ, or REWRITE statements may be executed for the file.

- *I/O Error Conditions*:

1. Uncorrectable I/O Error
2. Uncorrectable Error, Not I/O

Suggested User Response:

Processing of the file is limited to CLOSE.

INDEXED FILE PROCESSING TECHNIQUE (SEQUENTIAL ACCESS)

The DCB contains two error condition bytes named EXCD1 and EXCD2, at location DCB + 80. Conditions causing I/O errors and suggested user responses are as follows:

- *I/O Error Conditions*:

1. Sequence Check
2. Duplicate Record

Suggested User Response:

Condition caused by INVALID KEY. Processing of the file may be continued.

• I/O Error Conditions:

1. Space Not Found

2. Uncorrectable Output Error

3. Unreachable Block (Input)

4. Unreachable Block (Update)

Suggested User Response:

Processing of the file is limited to CLOSE.

• I/O Error Conditions:

Uncorrectable Input Error

Suggested User Response:

The user may attempt to bypass the block containing the error. If, in reading the next block, the error does not recur, he may continue processing without closing the file. If the error persists, processing of the file is limited to CLOSE.

INDEXED FILE PROCESSING TECHNIQUE (RANDOM ACCESS)

The DECB contains an error condition byte at location DECB + 24. Conditions causing I/O errors and suggested user responses are as follows:

• I/O Error Condition:

1. Record Not Found

2. Duplicate Record

Suggested User Response:

Condition caused by INVALID KEY. Processing of the file may be continued.

• I/O Error Condition:

Space Not Found

Suggested User Response:

Processing of the file may be continued. The record may be written after changing the keys and executing a WRITE statement if a cylinder overflow area is available for the new value of the keys. CLOSE or READ may be executed for the file.

• I/O Error Condition:

Invalid Request

Suggested User Response:

Processing of the file is limited to CLOSE.

• I/O Error Conditions:

1. Uncorrectable I/O Error.

2. Unreachable Block--Index Cannot Be Read.

Suggested User Response:

Processing of the file is limited to CLOSE. The user can try to execute the instruction again. If the error persists, he can close the file or perform file recovery procedures.

Indexed data sets are created and retrieved using special subsets of DD statement parameters and subparameters. They can occupy up to three different areas of space:

- Prime Area -- This area contains data records and related track indexes. It exists for all indexed data sets.
- Overflow Area -- This area contains overflow from the prime area when new data records are added. It is optional.
- Index Area -- This area contains master and cylinder indexes associated with the data set. It exists for any indexed data set that has a prime area occupying more than one cylinder.

Indexed data sets must reside on mass storage volumes. Because an Indexed data set can be associated with more than one type of unit, it is not usually cataloged.

Creating an Indexed Data Set

Indexed data sets are created with from one to three DD statements. One of the statements must define the prime area. If additional areas are to be defined, the DD statements must appear in the following sequence:

1. Index area
2. Prime area
3. Overflow area

This order must be maintained even if one of the statements is absent. Only the first DD statement defining the data set can contain a name field. Other statements, if any, must have a blank name field.

The subset of DD statement parameters used to create an indexed data set excludes the asterisk, DATA, DUMMY, DDNAME, SYSOUT, SUBALLOC, and SPLIT parameters. The remaining DD statement parameters -- DSNAME, UNIT, VOLUME, LABEL, DCB, DISP, SPACE, SEP, and AFF -- are all valid. However, certain restrictions must be followed in using these parameters.

DSNAME: Required. In addition to giving the data set name, the DSNAME parameter identifies the area being defined, i.e., DSNAME=name(INDEX), DSNAME=name(PRIME), and DSNAME=name(OVERFLOW).

Notes:

- If the data set is temporary, name is replaced with &&name.
- If only one DD statement is used to define the entire data set, DSNAME=name(PRIME) or DSNAME=name should be used.

UNIT: Required, unless VOLUME=REF is used. The first subparameter identifies a mass storage unit. If separate statements for the prime and index areas are included, request the same number of units for the prime area as there are volumes. The DEFER subparameter cannot be specified on any of the statements. Another way of requesting units is by using the unit affinity subparameter, AFF.

Notes:

- DD statements for prime and overflow areas must indicate the same type of unit.
- The DD statement for the index area can indicate a unit type different than the others.

VOLUME: Optional. Can be used to request private volumes (PRIVATE), to retain private volumes (RETAIN), or to make specific volume references (SER or REF).

LABEL: Optional. Can be used to specify a retention period (EXPDT or RETPD) and/or password protection (PASSWORD).

DCB: Required. Can be used to complete the data control block if it has not been completed by the processing program. Either DSORG=IS or DSORG=ISU must be included in the list of attributes, even though this attribute was provided in the processing program. If more than one DD statement is used to define the data set, the DCB parameters in the statements must not contain conflicting attributes.

DISP: Optional. Must be coded to keep the data set (KEEP), to catalog it (CATALG), or to pass it to a later job step (PASS). An indexed data set can be cataloged using CATLG only if all three areas are defined by the same DD statement.

Note:

- Indexed data sets defined by more than one DD statement can be cataloged by using the system utility program IEHPROGM, provided all volumes reside on the same type of unit. The utility program IEHPROGM is described in the publication IBM OS Utilities.

SPACE: Required. Space must be requested using either the recommended nonspecific allocation technique or the more restricted absolute track (ABSTR) technique. All DD statements used to define the data set must request space using the same technique.

If the nonspecific space allocation technique is used, space must be requested in units of cylinders (CYL). The quantity of space requested is assigned to the area identified in the DSNAME parameter. If more than one unit is requested, this quantity of space is allocated to each volume used by the data set. Incremental space cannot be requested for indexed data sets. If one DD statement is used to define both the index and prime areas, the size of the index must be indicated in the SPACE parameter of the DD statement defining the prime area. The subparameters RLSE, MXIG, ALX, and ROUND cannot be used. Contiguous space can be requested on each of the volumes occupied by the data set with the subparameter CONTIG. If CONTIG is coded on one of the statements, it must be coded on all of them.

If the absolute track technique of allocating space is used, the number of tracks must be equivalent to an integral number of cylinders. The address of the beginning track must correspond with the first track of a cylinder other than the first cylinder on a volume. If more than one unit is requested, space is allocated beginning at the specified address and continuing through the volume and onto the next volume until the request has been satisfied. If one DD statement is used to define both the index and prime areas, indicate the size of the index (in tracks) in the SPACE parameter of the DD statement defining the prime area. This number must also be equivalent to an integral number of cylinders.

Notes:

- The first volume to be allocated for the prime area of an indexed data set cannot be the volume from which the system is loaded (the IPL volume).
- Space can be requested on more than one volume only on the DD statement that defines the prime area.

SEP AND AFF: Optional. Channel separation from earlier data sets can be requested on any of the DD statements in the group. In order to have areas of an indexed data set written using separate channels, units should be requested by their actual address (e.g., UNIT=190).

Figure 120 illustrates a valid set of DD statements for creating an indexed data set. Note that each area is defined by a separate DD statement.

```
//OUTPUT4  DD  DSNAME=MHB(INDEX),UNIT=2301,DCB=DSORG=IS,             X
//             SPACE=(CYL,10,,CONTIG),DISP=(,KEEP)

//         DD  DSNAME=MHB(PRIME),DCB=DSORG=IS,UNIT=(2311,2),         X
//             VOLUME=SER=(334,335),DISP=(,KEEP),                    X
//             SPACE=(CYL,25,,CONTIG)

//         DD  DSNAME=MHB(OVFLOW),DCB=DSORG=IS,UNIT=2311,            X
//             VOLUME=SER=336,SPACE=(CYL,25,,CONTIG),DISP=(,KEEP)
```

Figure 120. Creating an Indexed Data Set

Table 41. Area Arrangement for Indexed Data Sets

CRITERIA			Restrictions on Unit Types and Number of Units Requested	Resulting Arrangement of Areas
Number of DD Statements	Types of DD Statements	Index Size Coded?		
3	INDEX PRIME OVFLOW	-	PRIME and OVFLOW must specify the same unit type.	Separate index. prime, and overflow areas.
2	INDEX PRIME	-	None	Separate prime and overflow areas, with an index at the end of the prime area.
2	PRIME OVFLOW	No	Both statements must specify the same type of unit.	Prime area and overflow area with an index at its end.
2	PRIME OVFLOW	Yes	Both statements must specify the same unit type. The statement defining the prime area cannot request more than one unit.	Prime area with embedded index and overflow area.
2	PRIME	No	None	Prime area with index at its end. Unused index areas, if any, used for overflow.
1	PRIME	Yes	Cannot request more than one unit.	Prime area with embedded index area.

The manner in which the areas of an indexed data set are arranged is based primarily on two criteria:

- The number of DD statements used to define the data set.
- The types of DD statements used (as reflected in the DSNAME parameter).

An additional criterion arises when a DD statement is not included for the index area:

- The index size and whether or not it has been coded in the SPACE parameter of the DD statement defining the prime area.

Table 41 illustrates the arrangements resulting from various permutations of the foregoing criteria. In addition, it points out restrictions on the number and type of units that can be requested for each permutation.

Retrieving an Indexed Data Set

Indexed data sets are retrieved with the DD statement parameters DSNAME, UNIT, VOLUME, DCB, and DISP. Channel separation requests can be made using the SEP and AFF parameters. If all areas of the data set reside on the same type of unit, the entire data set can be retrieved with one DD statement. If the index resides on a different type of unit, two DD statements must be used.

DSNAME: Required. Identify the data set by its name. If it was passed from a previous step, identify it by a backward reference or its temporary name. Do not include the terms INDEX, PRIME, or OVFLOW.

UNIT: Required, unless the data set was passed on one volume. Identify the unit type. If the data set resides on more than one volume and all units are the same type, request the total number of units required by all areas. If the index area resides on a different type of unit, use two DD statements, each indicating the number of units of the specified type required.

VOLUME: Required, unless the data set was passed on one volume. Identify the volumes by their serial numbers (SER), listed in the same sequence as they were when the data set was created.

DCB: Required, unless the data set was passed. This parameter is used to complete the data control block if it was not completed in the program. Include either DSORG=IS or DSORG=ISU.

DISP: Required. Identify the data set as OLD or MOD and give its new disposition, to change its disposition.

Figure 121 shows how to retrieve the indexed data set created by the illustration in Figure 120.

```
//INPUT     DD  DSNAME=MHB,DCB=DSORG=IS,UNIT=2301,DISP=OLD
//          DD  DSNAME=MHB,DCB=DSORG=IS,UNIT=(2311,3),DISP=OLD,          X
//              VOLUME=SER=(334,335,336)
```

Figure 121. Retrieving an Indexed Data Set

This checklist illustrates general job control procedures for compiler, linkage editor, and execution processing. More than one example may be used for a job step. The checklist is intended as an aid to preparing procedures, not as an inclusive list of the options and parameters.

COMPILATION

Figure 122 shows a general job control procedure for a compilation job step. The following cases demonstrate how to add to or modify the general procedure to obtain various processing options.

Case 1: Compilation Only -- No Object Module Is to Be Produced

The general procedure should be used. A listing is produced. It will include the default or specified options of the PARM parameter that affect output. Any diagnostic messages are listed, unless listing of warning messages is suppressed by the FLAGE option of the PARM parameter and only warning messages are produced.

Case 2: Source Module from Card Reader

Modify the end of the procedure as follows:

```
//SYSIN  DD  *
   (source module)
/*
```

If the DD * convention is used, the SYSIN DD statement must be the last DD statement for the job step, and the source module must follow. If another job step follows the compilation, the EXEC statement for that step follows the /* statement.

Case 3: Object Module Is to Be Punched

Add the statement:

```
//SYSPUNCH  DD  SYSOUT=B
```

Note: If DECK is not the installation default condition, it must be specified in the PARM parameter of the EXEC statement.

Case 4: Object Module Is to Be Passed to Linkage Editor

Add the statement:

```
//SYSLIN     DD     DSNAME=(subparms),       X
//                  UNIT=SYSDA,              X
//                  SPACE=(subparms),        X
//                  DISP=(MOD,PASS)
```

Note: If LOAD is not the installation default condition, it must be specified in the PARM parameter of the EXEC statement.

```
//jobname  JOB  acctno,name,MSGLEVEL=1
//stepname EXEC PGM=IKFCBL00,PARM=(options)
//SYSUT1    DD   UNIT=SYSDA,SPACE=(subparms)
//SYSUT2    DD   UNIT=SYSDA,SPACE=(subparms)
//SYSUT3    DD   UNIT=SYSDA,SPACE=(subparms)
//SYSUT4    DD   UNIT=SYSDA,SPACE=(subparms)
//SYSPRINT DD   SYSOUT=A
//SYSIN     DD   DSNAME=dsname,UNIT=SYSSQ,VOLUME=(subparms),          X
//               DISP=(OLD,DELETE)
```

Figure 122. General Job Control Procedure for Compilation

Case 5: Object Module Is to Be Saved

The object module can be saved by cataloging it, by keeping it, or by adding it as a member of a library. Add the SYSLIN statement as shown in examples A, B, or C.

• A. Cataloging

```
//SYSLIN   DD   DSNAME=dsname,                X
                       NEW
//              DISP=(       ,CATLG),         X
                       MOD
//              VOLUME=(subparms),            X
//              LABEL=(subparms),             X
                      SYSDA
//              UNIT=          ,              X
                      SYSSQ

                SPACE
//              SPLIT      =(subparms)
                SUBALLOC
```

• B. Keeping

```
//SYSLIN   DD   DSNAME=dsname,                X
                       NEW
//              DISP=(       ,KEEP),          X
                       MOD
//              VOLUME=(subparms),            X
//              LABEL=(subparms),             X
                      SYSDA
//              UNIT=          ,              X
                      SYSSQ

                SPACE
//              SPLIT      =(subparms)
                SUBALLOC
```

• C. Adding a Member to an Existing Library

```
//SYSLIN   DD   DSNAME=dsname(member),        X
//              DISP=OLD
```

Case 6: COPY Statement in COBOL Source Module or a BASIS Card in the Input Stream

Add the SYSLIB DD card(s), as shown in examples A, B, or C.

A. COPY

```
//SYSLIB   DD   DSNAME=copylibname,DISP=SHR
```

B. BASIS Card

```
//SYSLIB   DD   DSNAME=basislibname,DISP=SHR
```

C. Both BASIS and COPY

```
//SYSLIB   DD   DSNAME=basislibname,DISP=SHR

//         DD   DSNAME=copylibname,DISP=SHR
```

(DD statements for additional copylibs may follow.)

LINKAGE EDITOR

Figure 123 shows a general job control procedure for a linkage editor job step. The following cases show how to add to or modify the procedure to obtain various processing options.

Case 1: Input from Previous Compilation in Same Job

Change the SYSLIN statement to

```
//SYSLIN   DD   DSNAME=*.stepname.SYSLIN,    X
//              DISP=(OLD,DELETE)
```

where stepname is the name of the previous compilation job step and ddname is SYSLIN. If the input is to be saved, specify KEEP rather than DELETE.

Case 2: Input from Card Reader

Change SYSLIN statement and the end of the procedure as follows:

```
//SYSLIN   DD *
    (object module(s))
/*
```

If the DD * convention is used, the SYSLIN DD statement must be the last DD statement in the job step. If another job step follows the link-edit step, the EXEC statement for that job step follows the /* statement.

Case 3: Input Not from Compilation in Same Job

Specify in the SYSLIN DD statement where the object modules to be used as input are stored. (Only one member of a library can be specified in the SYSLIN DD statement.)

```
//jobname    JOB    acctno,name,MSGLEVEL=1
                .
                .
                .
//stepname   EXEC   PGM=IEWL,PARM=(options)
//SYSPRINT   DD     SYSOUT=A
//SYSLMOD    DD     DSNAME=&&name(member),UNIT=SYSDA,DISP=(NEW,PASS),    X
//                  SPACE=(subparms)
//SYSLIB     DD     DSNAME=SYS1.COBLIB,DISP=OLD
//SYSUT1     DD     UNIT=SYSDA,SPACE=(subparms)
//SYSLIN     DD     DSNAME=dsname,DISP=OLD
```

Figure 123. General Job Control Procedure for a Linkage Editor Job Step

Case 4: Output to Be Placed in Link Library

Change the SYSLMOD statement as follows:

```
//SYSLMOD  DD DSNAME=SYS1.LINKLIB(member),X
//            DISP=OLD
```

where member is the name of the load module that is to be added to the link library. No other information is needed in the statement.

Case 5: Output to Be Placed in Private Library

Change the SYSLMOD statement as follows:

```
//SYSLMOD  DD  DSNAME=dsname(member),      X
//             DISP=OLD
```

where member is the name of the load module to be added, and dsname is the name of an existing library. If the library is not cataloged, UNIT and VOLUME parameters must be specified.

Note: See "Using the DD Statement" for an example of creating a new library and storing the load module as its first member.

Case 6: Output to Be Used Only in this Job

The general procedure should be used. The load module is stored in a temporary library.

EXECUTION TIME

Figure 124 shows a general job control procedure for an execution-time job step. The following cases show how to add to or modify the general procedure to obtain various processing options.

Case 1: Load Module to Be Executed Is in Link Library

Use the general procedure, where progname in the EXEC statement is the member name of the load module.

Case 2: Load Module to Be Executed Is a Member of Private Library

The JOBLIB statement must follow the JOB statement, as in the following statements:

```
//JOB1      JOB
//JOBLIB    DD     DSNAME=MYLIB,                X
//                   DISP=(OLD,PASS)
//STEP1     EXEC   PGM=PAYROLL
              .
              .
              .
//STEP2     EXEC   PGM=ACCOUNT
              .
              .
              .
```

```
//stepname EXEC   PGM=progname
//ddname   DD     (parameters for user-specified data sets)
                  .
                  .
                  .
```

Figure 124. General Job Control Procedure for an Execution-Time Job Step

The JOBLIB statement defines the private library MYLIB. No volume or unit parameters are given since the library is cataloged. Since JOBLIB has the disposition PASS, both steps can execute members of the library named in the JOBLIB statement. If only the first step executes a load module from the library, the disposition PASS on the JOBLIB statement need not be included.

Case 3: Load Module to Be Executed Is Created in Previous Linkage Editor Step in Same Job

Change the EXEC statement as follows:

```
//stepname   EXEC   PGM=*.stepname.SYSLMOD
```

where stepname following PGM is the name of the linkage editor job step that created the load module.

Case 4: Abnormal Termination Dump

Add the statement:

```
//SYSABEND   DD   SYSOUT=A
```

This statement requests a full dump if abnormal termination occurs during execution.

Case 5: DISPLAY Is Included in Source Module

Add the statement:

```
//SYSOUT   DD   SYSOUT=A
```

Case 6: DISPLAY UPON SYSPUNCH Is Included in Source Module

Add the statement:

```
//SYSPUNCH   DD   SYSOUT=B
```

Case 7: ACCEPT Is Included in Source Module

If the data is in the input stream, add the statement:

```
//SYSIN    DD   *

           (data)

/*
```

(See Case 2 under "General Job Control Procedures for a Compilation Job Step" for a discussion of the DD * convention.)

Case 8: Debug Statements EXHIBIT or TRACE Are Included in Source Module

Use the statement (unless it is already included):

```
//SYSOUT   DD   SYSOUT=A
```

Note: If the job step already includes a SYSOUT DD statement for some other use, another need not be inserted.

Case 9: Object Time Symbolic Debugging Options

```
//SYSDBOUT DD SYSOUT=A required for all
                         options
//SYSDBG   DD *          required for SYMDMP
                         option
   (control cards)
/*
   debug DDname card also needed
```

In this appendix, each field of the Task Global Table (Figure 125) and of the Program Global Table (Figure 126) is listed by its relative location in main storage. Each field is further described in the discussion associated with Figures 125 and 126.

TASK GLOBAL TABLE

The Task Global Table (TGT) is used to record and save information needed during execution of the object program. It begins with a series of fixed-length fields followed by a series of variable-length fields. These fields are illustrated in Figure 125 and are described in this section.

Relative Location	Field
0	SAVE AREA
72	SWITCH
76	TALLY
80	SORT SAVE
84	ENTRY SAVE
88	SORT CORE SIZE
92	RET CODE
94	SORT RET
96	WORKING CELLS
400	SORT FILE SIZE
404	SORT MODE SIZE
408	PGT-VN TABLE
412	TGT-VN TABLE
416	VCON PTR
420	LENGTH OF VN TBL
422	LABEL RET
423	CURRENT PRIORITY
424	DBG R14SAVE
428	ANSC
432	A(INIT1)
436	DEBUG TABLE PTR
440	SUBCOM PTR
444	SORT DDNAME
452	SYSTDD
453	Unused
472	DBG R11SAVE
476	Unused
480	PRB1 CELL PTR
484	Unused
489	TA LENGTH
492	Unused

Figure 125. Fields of the Task Global Table (Part 1 of 2)

Relative Location	Field
500 beginning of variable-length portion	OVERFLOW
	BL
	DECABDR
	TEMP STORAGE
	TEMP STORAGE-2
	TEMP STORAGE-3
	TEMP STORAGE-4
	BLL
	VLC
	SBL
	IND
	SUBADR
	ONCTL
	PFMCTL
	PFMSAV
	VN
	SAVE AREA-2
	SAVE AREA-3
	XSASW
	XSA
	PARAM
	RPTSAV AREA
	CHECKPT CTR
	VCON TBL
	DEBUG TABLE

Figure 125. Fields of the Task Global Table (Part 2 of 2)

The lengths of the variable-length fields are determined by the requirements of the program (if not required, a particular field may not exist in the object program).

SAVE AREA
the program's save area; used to provide standard subroutine linkage when this program is called (by the Operating System or by another program) and when this program calls other programs.

SWITCH
a fullword switch. Only the following bits are used:

Bit	Meaning
0	Indicates a size error in series addition or subtraction. If a SIZE ERROR clause was included in the source statement, and a size error occurs before all data items in the series have been added or subtracted, this bit is set to 1. It is tested after the entire addition or subtraction is complete. If the value is 1, the instructions generated for the ON SIZE ERROR clause are executed.
1	Used for TRACE. It is set to 1 by the execution of a READY statement, and reset to 0 by a RESET statement. If the program uses a TRACE statement, there are instructions to test this bit at the point of definition for every source program procedure-name (PN). If it is on, the DISPLAY subroutine (ILBODSP0) is called to print the card number of the procedure-name. (See "Appendix B: COBOL Library Subroutines" for a description of the DISPLAY subroutine.)
2	Indicates program initialization. Set to 1 by routine INIT3 to show that initialization has been performed. Tested by INIT3 so that if the module is re-entered, INIT3 can perform re-entry functions instead of initialization functions.
3	Main or subprogram switch. Set by INIT2 if this is a main program.
4	Used for SYMDMP. It is set to 1 if the symbolic debug option is in effect for the program. This bit is tested by the object-time COBOL library debugging control subroutine ILBODBG0.
5	Used for FLOW. It is set to 1 if the flow trace option is in effect for the program. This bit is tested by the object-time COBOL library debugging control subroutine ILBODBG0.
6	Used for STATE. It is set to 1 if the statement number option is in effect for the program. This bit is tested by the object-time COBOL library debugging control subroutine ILBODBG0.
7	Used for OPT. It is set to 1 if optimization has been requested for the program or if the SYMDMP or STATE and OPT, or FLOW and OPT options have been specified.
8	Used for IF MESSAGE or the OVERFLOW option of the STRING or UNSTRING verb. It is set to 1 when the MESSAGE condition being tested is true or if an overflow condition occurs in the execution of STRING or UNSTRING. It is tested by the generated code.
9	Used for CALL, CANCEL, or a recursive CALL. It is set to 1 by the generated code for the CALL or CANCEL verb. It is tested by INIT2 to determine whether a recursive CALL condition exists.
10-11	Unused
12	Used for QUOTE IS APOST. It is set to 1 if the apostrophe is to be used to delineate literals and to be used in the generation of figurative constants.
13	Used for SYMDMP. It is set to 1 if when SYMDMP is requested the program

contains a floating-point item.

14 Always set to 1.

15 Indicates maximum length for a variable-length field. Before the execution of a Q-Routine, this bit is set to 1 if the VLC and SBL for the field are to be set to their maximum possible values, rather than a value depending on the current value of a data item. The maximum value is the value of X in the clause "OCCURS X TIMES DEPENDING ON...".

16 SRVBIT set on if ILBOLOM is link-edited with program.

24-31 DECIMAL-POINT IS COMMA clause byte. If this clause was specified, the byte contains a comma in EBCDIC. If not, it contains a decimal point.

TALLY
a fullword used for source program references to the special register TALLY.

SORT SAVE
a fullword used during the execution of a SORT RETURN statement to contain the GN for the next sequential instruction following the RETURN.

ENTRY SAVE
a fullword used to save the entry point of the program during INIT2 and INIT3 execution.

SORT CORE SIZE
a fullword for the SORT-CORE-SIZE special register as used in the source program.

RET CODE
a halfword for the RETURN-CODE special register, which is used in the source program to provide a completion code on a STOP RUN, EXIT PROGRAM, or GOBACK statement, or to store the return code from a called program. It is the user's responsibility to set this code.

SORT RET
a halfword used to contain the return code from a SORT operation.

WORKING CELLS
variable-length cells used by COBOL library subroutines called by the program. The total length of the field is 304 bytes.

SORT FILE SIZE
a fullword for the SORT-FILE-SIZE special register as used in the source program.

SORT MODE SIZE
a fullword for the SORT-MODE-SIZE special register as used in the source program.

PGT-VN TBL
a fullword pointer to that part of the VN field of the PGT containing VN's for independent segments.

TGT-VN TBL
a fullword pointer to that part of the VN field of the TGT containing VN's for independent segments.

VCON PTR
pointer to the VCON TBL field of the TGT. This is required because the VCON TBL field is variably located, and the VCON PTR is fixed within the TGT.

LENGTH OF IND VN TBL
a halfword containing the length of that part of the VN field (the length is the same for both the TGT and PGT) containing VN's for the independent segments.

LABEL RET
the LABEL-RETURN special register for nonstandard labels. If an error occurs in such a label, it is the user's responsibility to place a nonzero value into this 1-byte cell.

CURRENT PRIORITY
if the STATE compiler option is specified for a segmented program, the segmentation subroutine ILBOSGM0 inserts the priority of the segment currently in the transient area. This field is initialized to zero.

DBG R14SAVE
indicates the contents of register 14. A routine of the debug control subroutine ILBODBG0 is called to save this information before the execution of any instruction that passes control outside the COBOL program.

ANSC
identifies the object program as an American National Standard COBOL program.

INIT1 ADCON
address of INIT1 used for GOBACK, STOP RUN, and EXIT PROGRAM instructions, and for segmentation coding.

TGTTAB PTR
: if the FLOW SYMDMP or STATE compiler options are specified, this field points to the TGTTAB.

SUBCOM PTR
: a pointer to the subroutine communications (SUBCOM) area in the COBOL subroutine library.

SORT DDNAME
: an eight-byte area for the SORT-MESSAGE special register, which is used in the source program to allow the user to specify to the Sort/Merge program where to place the messages it issues.

SYSTDD
DBG R11SAVE
: indicates the contents of register 11. When the dynamic dumping routine of the debug control subroutine ILBODBG0 receives control, it places the return address to the in-line code of the calling program in register 11. Therefore, the contents of register 11 must be saved.

PRBL1 CELL PTR
: a fullword cell containing the address of the first PROCEDURE BLOCK cell in the PGT.

TA LENGTH
: a halfword initialized to the length of the largest segment with a nonzero priority.

OVERFLOW
: if the TGT is longer than 4096 bytes, this field contains one fullword cell pointing to each 4096-byte area after the first. The cell is loaded into a register when a base is required for the overflow area.

BL
: base locators. Each BL cell is a fullword containing an address in the data area. There is one BL pointing to the beginning of the Working-Storage Section and one for each file in the File Section. More than one BL is assigned if an area is larger than 4096 bytes.

DECBADR
: DECB addresses. There is one fullword cell pointing to the address of the DECB for each basic file.

TEMP STORAGE
: temporary storage for arithmetic operations. TS space is allocated in doubleword blocks.

TEMP STORAGE-2
: temporary storage for nonarithmetic instructions. These cells are variable in length.

TEMP STORAGE-3
: temporary storage used to align fields of data described by the SYNCHRONIZED option. The field begins on a doubleword boundary.

TEMP STORAGE-4
: temporary storage cells used for the SEARCH ALL table-handling verb. The field starts on a doubleword boundary.

BLL
: base locators for the Linkage Section. Each BLL cell is a fullword containing the address of an area passed as a result of an ENTRY statement, a label record, a totaled area, a sort description entry, or a GIVING option in a USE...ERROR statement.

VLC
: variable-length cells. Each VLC is a halfword whose value is set by the execution of a Q-Routine. It contains the current length of a variable-length field. There is one VLC for each OCCURS...DEPENDING ON clause and all items to which it is subordinate.

SBL
: secondary base locators. Each SBL cell is a fullword set by the execution of a Q-Routine. It contains the current address of a field which is variably located because it follows a variable-length field.

IND
: fullword cells, each containing the current value of an INDEX-NAME. There is one IND cell for each INDEX-NAME defined in a file.

SUBADR
: subscript addresses. Each SUBADR cell is a fullword containing the address for a subscripted reference.

ONCTL
: control counters for ON statements. Each is a fullword initialized to zero.

PFMCTL
: PERFORM control counters and DEBUG saved location. Each PFMCTL cell is a fullword used for a PERFORM n TIMES statement to count the number of times the procedure has been performed. For DEBUG, a PFMCTL cell is used to save the contents of register 14 when the DEBUG packet is entered. DEBUG packets are called by BALR 14,15.

PFMSAV
PERFORM saved locations. Each is a fullword used to contain an address. For PERFORM, the cell is used to store the address of the next sequential instruction after the performed procedure, when that procedure is being executed because of a PERFORM. This is to enable the procedure to be executed in-line.

VN
variable procedure-names. Each VN cell is a fullword containing the current address of a branch point which may change during program execution because of an ALTER or PERFORM statement.

SAVE AREA-2
pointer to the save area provided for label- and error-processing declaratives.

SAVE AREA-3
variable number of fullwords used for OPEN parameters.

XSASW
1-byte EXHIBIT switches. These are used as first-time switches for the coding generated for the EXHIBIT CHANGED statement. They are also used in certain types of SORT statements and ON statements.

XSA
EXHIBIT saved area cells. These are variable in length and are referred to in the coding generated for an EXHIBIT CHANGED statement. There is one XSA for each operand to be exhibited with a CHANGED option. These cells are also used for SORT and RELEASE verbs.

PARAM
parameter area of fullwords, containing parameter lists for macro instruction expansions of certain source statements. The size of the parameter area equals the largest number of words required for any one expansion.

RPTSAV
six words used to save branch addresses during the execution of Report Writer routines, if the Report Writer is used.

CHECKPT CTR
fullword cells used to count the number of file records processed for a file for which checkpoints are to be taken.

VCON TBL
8-byte V-type address constants for nonresident segments. The format of each entry is:

Byte	Contents
0	Priority number
1-3	0
4-7	VCON to independent segment

DEBUG TABLE
table used by the flow trace and statement number and symbolic debug COBOL library subroutines. The format depends on the options specified.

- If the FLOW compiler option is specified:

Byte(s)	Contents
0	Number of traces requested
1-3	Unused

- If the STATE option is specified:

Byte(s)	Contents
0-3	Start of Q-Routines, or if none, start of INIT2.
4-7	Size of Declaratives (not including Report Writer) Section.
8-11	Starting address of PROCTAB in object module.
12-15	Starting address of SEGINDX in object module.
16-19	Ending address of SEGINDX in object module.

- If both the FLOW and STATE compiler options are specified:

Byte(s)	Contents
0	Number of traces requested
1-19	The same as shown above for the STATE option.

- If the SYMDMP option is specified:

Byte(s)	Contents
0-3	Start of Q-Routines, or if none, start of INIT2.
4-5	Hashed compilation indicator.

- If both the SYMDMP and FLOW options are specified:

Byte(s)	Contents
0	Number of traces requested.
1-5	The same as shown above for the SYMDMP option.

PROGRAM GLOBAL TABLE

The Program Global Table (PGT) contains data referenced by procedure instructions. All the fields in the PGT are variable in length. PGT data is never modified by procedure instructions; rather, it remains constant throughout program execution.

The fields in the PGT are illustrated in Figure 126 and described in the text below.

DEBUG LINKAGE AREA
OVERFLOW
VIRTUAL
VIRTUAL EBCDIC NAMES
PN
GN
DCBADR
VNI
LITERAL
DISPLAY LITERAL
PROCEDURE BLOCK

Figure 126. Fields of the Program Global Table

DEBUG LINKAGE AREA
: a 12-byte area that contains the linkage for dynamic dumps. If the SYMDMP option is not specified, this area does not exist.

OVERFLOW
: if the entire PGT exceeds 4096 bytes in length, there is one fullword OVERFLOW cell pointing to each 4096-byte section after the first. The cell is loaded into a register when a base is needed to refer to the section of the PGT.

VIRTUAL
: each virtual is a fullword containing the address of an external procedure (the result of an ESD and RLD in the object module) unless either the DYNAM or the RESIDENT option is in effect. If either of these options is in effect, the virtuals corresponding to library subroutines are written as EBCDIC'/ 00 00 00';/ in addition, if the DYNAM option is in effect, the virtuals corresponding to user subprograms contain the relative displacement of the subprogram name from the beginning of the PGT. It is required because of a CALL statement in the source program or a branch to a COBOL library object-time subroutine.

VIRTUAL EBCDIC NAMES
: indicates the EBCDIC names of library subroutines and user subprograms. If either the DYNAM or the RESIDENT option is in effect, the EBCDIC names of all library subroutines that are to be dynamically loaded are listed; in addition, if DYNAM is in effect, the EBCDIC names of all user subprograms that are to be dynamically called are listed. Each VIRTUAL EBCDIC NAME cell is a doubleword containing the name of the subroutine or subprogram, left justified and padded with blanks if necessary. If neither DYNAM nor RESIDENT is in effect, this field does not exist.

PN
: source program procedure-names. When the OPT option is in effect, only those PN's associated with ALTER and declaratives references receive PN cells. Each PN cell is a fullword containing the address of the first instruction in a block of coding. The addresses of the PN's are in the same order as their definition in the source program. The program branches by loading an address from the PGT and ~~then branching to it.~~

GN
: compiler-generated procedure-names. When the OPT is in effect, only those GN's associated with AT END and INVALID KEY references receive GN cells. Each GN is a fullword containing the address of the first instruction in a block of coding. GN's are used in the same way as PN's. They were generated to provide addresses for branches implied but not stated in the source program. They are stored in the PGT in the order in which they were generated.

DCBADR
: DCB addresses. Each DCBADR cell is a fullword containing the address of a data control block in the data area of the program. There is one DCBADR cell for each DCB generated.

VNI
: variable procedure-name initialization cells. There is one fullword VN cell for each variable procedure-name in the program. It contains the initial value of the VN, and is used to initialize the VN values in the TGT. VN's are generated to contain branch addresses which vary because of PERFORM or ALTER statements.

LITERAL
literals referred to by procedure instructions. The literals are variable in length. There is no duplication in storage, since duplicate literals were eliminated.

DISPLAY LITERAL
literals used by calling sequences rather than instructions. They are variable in length; duplication was eliminated. each cell is a fullword containing the address of a procedure block. The compiler assigns these cells only when the OPT option is in effect.

PROCEDURE BLOCK
each cell is a fullword containing the address of a procedure block. The compiler assigns these cells only when the OPTIMIZE option is in effect.

(Where more than one page reference is given, the major reference appears first.)

READER'S COMMENTS

TITLE: IBM OS Full American National Standard COBOL Compiler and Library, Version 4 Programmer's Guide

ORDER NO. SC28-6456-0

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. All comments and suggestions become the property of IBM.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or to the IBM Branch Office serving your locality.

Corrections or clarifications needed:

Page *Comment*

Please include your name and address in the space below if you wish a reply.

Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

cut along this line

fold fold

FIRST CLASS
PERMIT NO. 33504
NEW YORK, N.Y.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM CORPORATION
1271 Avenue of the Americas
New York, New York 10020

Attention: PUBLICATIONS

fold fold

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]