File No. S360-36
Form C28-6812-1

44PS

IBM Systems Reference Library

IBM System/360 Model 44

Programming System

Guide to System Use

This publication contains detailed information for preparing programs to be executed under the IBM System/360 Model 44 Programming System. It discusses and illustrates the system's requirements, its capabilities, and the control statements and supervisor calls that are used with it.

A prerequisite publication is:

IBM System/360 Model 44 Programming System: Concepts and Facilities, Form C28-6810

PREFACE

The emphasis in this publication is not on programming but on executing programs. It is possible to write a program without knowing any of this information, with the possible exception of the material on supervisor calls, but such a program could not be executed under the programming system's control.

This publications's main purpose is to describe the control statements and the considerations that are necessary to use the programming system. It does not try to tell how the system functions except when a programmer needs such information to be able to make decisions.

Most of the contents of publication is directed to assembler language programmers. Portions of it, however, apply to FORTRAN programmers as well, and this is noted in the text.

The first section contains a general discussion of the Model 44 Programming System's capabilities and requirements. It also is intended to serve as a guide to the rest of the publication.

The second section covers data set creation and maintenance, symbolic units, and the phase and module libraries. The sections on job control, linkage editing, and utilities discuss these facilities and the control statements required for their use.

Input/output is described in the first of two sections about the system's supervisor calls. The section about system output illustrates the various types of output produced by the system routines, the dump program, messages, and the program history. Messages are listed and described individually in the section about error messages.

The stand-alone loader, which can be used to load and begin executing programs to be run without Model 44 programming system control, is discussed in the next section. The final section is about labels.

In addition to the publication cited on the cover, a programmer should be familiar with:

IBM System/360: System Summary, Form A22-6810

IBM System/360: Principles of Operation, Form A22-6821

IBM System/360 Model 44: Functional Characteristics, Form A22-6875

Assembler language programmers should be familiar with:

IBM System/360 Model 44 Programming System: Assembler Language, Form C28-6811

FORTRAN programmers should be familiar with:

IBM System/360 Model 44 Programming System, Guide to System Use for FORTRAN Programmers, Form C28-6813.

Publications about related topics, such as input/output devices, are listed in IBM System/360 Bibliography, Form A22-6822.

Second Edition

This publication is a major revision of, and makes obsolete, the publication IBM System/360 Model 44 Programming System: Supervisor Call (SVC) Functions, Form C28-6812-0. Major portions of that publication are included in this publication in the sections entitled "Input/Output" and "Other Supervisor Calls."

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

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

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

©International Business Machines Corporation 1966, 1967

INTRODUCTION AND GENERAL INFORMATION . . 7
Language Processors 7
Linkage Editor. 7
Utilities 8
Absolute Loader 8
System Conventions. 8
Other Features. 8
Supervisor Calls. 8

SYSTEM CONVENTIONS 9

Data Set Creation and Maintenance. . . . 9
Data Sets. 9
Volumes. 9
System Catalog 11
System Data Sets. 11

Symbolic Units 12
Standard System Units 13
Special System Data Sets 15
Replacing System Data Sets 15
Early Unit Assignments 15
Phase Library 15
Module Library. 16
Register Conventions. 17

JOB CONTROL. 18
Job Control Statements. 18
Control Statement Format. 19
Continuation Statements 19
Comments in Control Statements. . . . 20
Location. 20
Format Descriptions 21

Statement Formats. 21
Job Definition Statements 21
JOB Statement. 21
EXEC Statement 22
End-of-Job Statement 24
STOP Statement 25
End of Data Statement. 25
Data Set Definition Statements. . . . 25
ALLOC Statement. 25
Creating Data Sets With /* 31
ACCESS Statement 32
LABEL Statement. 35
Symbolic Unit Maintenance Statements 39
RESET Statement. 39
LISTIO Statement 39
Data Set Maintenance Statements . . . 40
DELETE Statement 40
CONDENSE Statement 41
RENAME Statement 41
CATLG Statement. 41
UNCATLG Statement. 42
Miscellaneous Statements. 42
PAUSE Statement. 42
Comments Statement 42
REWIND Statement 43
UNLOAD Statement 43

LANGUAGE PROCESSORS. 45

Assembler Program. 45
Assembler EXEC Parameters 45
Output Option Parameters 45
Update Parameters. 46
Update Listing 47
Magnetic Tape Operations 47
Options. 47
Space Allocation Parameters. . . . 48
Control Statement Example 49
Supervisor Calls. 49

FORTRAN Compiler 50

LINKAGE EDITOR 52
Sources of Input. 52
Linkage Editor Output 53
Control Sections. 54
Unnamed Control Sections 54
Entry Points. 55
Common. 55
External References 56
EXEC Statement Parameters 57

Control Statements 57
MODULE Statement 58
PHASE Statement. 58
INCLUDE Statement. 60
ENTRY Statement. 61
Linkage Editor Examples. 61

INPUT/OUTPUT 64

Read/Write Operations. 65
Control Characters. 67
Blocking. 67
Attention Interruptions 68

Label Processing Supervisor Calls. . . . 69
OPEN - SVC 2 69
CLOSE - SVC 3. 71

Read/Write Level Supervisor Calls. . . . 74
READ - SVC 4 74
WRITE - SVC 5. 75
CHECK - SVC 6. 76
NOTE - SVC 7 77
POINT - SVC 8. 77
Extended Tape Volume Supervisor Calls. 78
WEF - SVC 9. 78
REWIND - SVC 10. 78
UNLOAD - SVC 11. 78

OTHER SUPERVISOR CALLS 79

Flow Control Supervisor Calls. 79
FETCH - SVC 12 79
LOAD - SVC 13. 81

Termination Supervisor Calls 82

EOJS - SVC 14. 82
CANCEL - SVC 15. 82

Communication Region Supervisor Calls. . 82

User Communication Region. 82
INSERT - SVC 17. 84
EXTRACT - SVC 18 84
UPSAND - SVC 19 and UPSOR - SVC 20. 84

Conditional interruption Supervisor Calls 85
Save Areas. 85
Program Check 86
STXIPC - SVC 21. 86
RTXIPC - SVC 24. 86
Timer Services. 87
GETIME - SVC 16. 87
SETIME - SVC 23. 87
STXITC - SVC 22. 87
RTXITC - SVC 25. 87
Time of Day 87
Interval Timer. 87

UTILITIES. 89

DATA SET UTILITIES 89
Sequential Data Set Statements. . . . 89
Directoried Data Set Statements . . . 90
General 90
Utilities Options 91
Control Characters 91
Control Options. 91
Open and Close Options 93
Delimiter Options. 94
Size Options 95
Record Manipulation Options. . . . 95
Output Format Options. 98
Member Selection Options 100

VOLUME UTILITIES 101
INITIAL Statement. 102
SQUEEZE Statement. 103
MAP Statement. 104

STAND-ALONE DISK INITIALIZATION. 105
Initialization. 105
Alternate Track Assignment. 105
Control Statements. 105
Statement Format 106

SYSTEM OUTPUT. 107

Dump Facilities. 107
Dump Routine. 107
Assembler Language Coding. 107
Dump Format Codes. 108

Sample Listings. 111
Assembler Listing 111
Linkage Editor Listing. 117

MESSAGES 121
Assembler Error Indications 121
Assembler Diagnostic Process . . .121
Error Flags and Severity 122
Assembler Flags Summary Table. . .122
Flags and Their Meanings 123
Supervisor Messages 132
Job Control Messages. 132
Linkage Editor Messages 136
Warning Messages, Severity Code 4 137
Severe Errors, Severity Code 12. .137
Termination Messages, Severity Code 16 139
Text Messages. 140
Assembler System Errors 141
Utilities Messages. 141

LABELS 147

Direct Access Labels 147

Magnetic Tape Labels 148
Unlabeled Tapes. 148
Labeled Tapes. 148

Label Formats. 149
Volume Labels. 149
Direct Access Data Set Label, Format 1. 150
Direct Access Space Management Label, Format 5 154
Tape Data Set Labels 155

ABSOLUTE LOADER. 157

APPENDIX A. INPUT/OUTPUT CONTROL BLOCK FORMATS 158

Request Control Block. 158

File Control Blocks. 159

Unit Control Block 159

APPENDIX B. LANGUAGE PROCESSOR CARD FORMATS 161
ESD Card 161
TXT Card 161
RLD Card 161
END Card 161
REP Card 166

APPENDIX C. SYSTEM PROGRAM PHASE NAMES .167

INDEX. 169

Figure 1. Direct Access Volume. 10
Figure 2. JOB Statement 21
Figure 3. EXEC Statement. 22
Figure 4. ALLOC Statement 26
Figure 5. Volume Field Format and Parameters. 27
Figure 6. ACCESS Statement. 33
Figure 7. LABEL Statement 36
Figure 8. RENAME Statement. 41
Figure 9. SAMPLE Job Deck. 44
Figure 10. INITIAL Statement. 102
Figure 11. Stand-alone Disk Program INITIAL Statement 106
Figure 12. Sample Calling Sequence for Module Library Dump Routine 109
Figure 13. Sample Dump Listings 110
Figure 14. Assembler Listing. 112
Figure 15. Linkage Editor Listing . . .118
Figure 16. Volume Label Format. 149
Figure 17. Direct Access Data Set Format. 150
Figure 18. VTOC Definition Label Format. 152
Figure 19. Direct Access Space Management Label Format 154
Figure 20. Standard Tape Data Set Header and Trailer Label Format 155
Figure 21. ESD Card Format. 162
Figure 22. TXT Card Format. 163
Figure 23. RLD Card Format. 164
Figure 24. END Card Format. 165
Figure 25. Alternate END Card Format. .165
Figure 26. REP Card Format. 166

TABLES

Table 1. Standard Symbolic Unit and Data Set Assignments. 12
Table 2. Job Control Statements 18
Table 3. Volume Options 28
Table 4. Type Field Entries 29
Table 5. Assembler EXEC Parameters. . . 45
Table 6. Update Parameters. 48
Table 7. Input/Output Supervisor Calls 64
Table 8. Symbolic Unit (SYSUNI) Index Values. 65
Table 9. Input/Output Return Codes. . . 66
Table 10. ASA Codes. 67
Table 11. Incorrect Length Conditions . 74
Table 12. Non Input/Output Supervisor Calls 80
Table 13. Input/Output Units. 90
Table 14. Utilities Options 92
Table 15. Assembler Error Summary . . .128
Table 16. UPDATE Errors 130

The Model 44 Programming System has two main purposes:

1. To make it easier for a programmer to use the computing system.

2. To help the computing system operate at its most efficient level.

Easing the job of the programmer is best illustrated in the area of input/output operations. Instead of having to write literally hundreds of assembler language instructions, a programmer indicates what type of operation is to be done. He supplies a few items of variable information, and turns the rest of the job over to the system. The system executes the request and returns a code that tells the programmer how things went.

Efficiency is provided in many ways. One easily visible example is the way the system terminates one job and starts the other without delay. No manual operator intervention is required.

The heart of the system is a supervisor program that runs the entire system at all times. In a sense, all other programs are subroutines to this master program.

The supervisory programs prepare the system for each job, load the program to be executed, and stand by during execution to supply special services. The program being executed may be a user's problem program or one of the system's processing programs.

Loading and execution of any program is accomplished in a job step. There are one or more job steps in single job.

The system has five major processor programs. These are an assembler, a FORTRAN IV compiler, a linkage editor, a utilities program, and a job control processor. The job control processor is unique in that a programmer doesn't call it directly and its execution doesn't represent a job step. It is used solely by the system.

A problem program may be assembled in one job step, linkage edited in the second, and executed in the third. Actually, three separate programs are executed in this typical job: the system's assembler and linkage editor programs and the user's problem program. To the control program, there is no difference among them.

LANGUAGE PROCESSORS

The language processors are the assembler program and the FORTRAN IV compiler. These programs convert a programmer's coded input into a form suitable for processing by the linkage editor.

This publication discusses the assembler and FORTRAN compiler programs as programs to be executed. Details of assembler language coding are covered in the publication IBM System/360 Model 44 Programming System: Assembler Language, Form C28-6811.

FORTRAN IV programming for the Model 44 is explained in the publication IBM System/360 FORTRAN IV Language, Form C28-6515. Another publication, IBM System/360 Model 44 Programming System: Guide to System Use for FORTRAN IV Programmers, Form C28-6813, discusses the compiling, linkage editing, and execution of FORTRAN jobs.

An assembly or compilation job step, like any other job step, requires certain job control statements. These statements apply to all programs and are discussed in detail in the Job Control section of this publication. These statements provide for specification of certain parameters that apply only to the language processors.

For example, when the parameter LINK is specified in the EXEC job control statement, it informs the system that the language processor output is to be linkage edited later in the same job. The system then places this output where it will be readily available when execution of the linkage editor program starts.

These parameters and other information about the language processor programs are discussed in the Language Processors section of this publication.

LINKAGE EDITOR

The linkage editor program converts assembler and compiler output into a form suitable for loading and execution. All programs to be executed under control of the programming system must first be processed by the linkage editor.

Job control statements are used to initiate the linkage editor job step. Then there are four linkage editor control statements that are used to outline special requirements. In many cases, programmers will not need to use these statements since some programs require no more than a certain amount of standard processing.

The capabilities and requirements of the linkage editor are discussed in the Linkage Editor section of this publication.

UTILITIES

The system's utility routines handle many of the routine tasks that are common to all computer installations. These include initialization of disks and magnetic tapes, listing the contents of storage volumes, and transferring data from one type of storage to another, such as from cards to tape.

Job control statements initiate a utilities job step, and utilities control statements specify the job to be done. These statements can be used, for example, to indicate certain alterations that can be made to data as it is being copied from one place to another.

The Utilities section of this publication contains detailed information on the capabilities and requirements of the utilities program.

ABSOLUTE LOADER

An absolute loader program is supplied with the system, but it does not operate under system control. This program is used to load and initiate execution of installation programs that are not meant to operate under the programming system's control. It is described in the Absolute Loader section of this publication.

SYSTEM CONVENTIONS

As a member of the IBM System/360 series, the Model 44 follows many of the conventions and standards that apply to other IBM System/360 models and programming systems. This provides a considerable degree of compatibility between the Model 44 and other other IBM System/360 models.

The key basic elements of the programming system are discussed in the next section, System Conventions. This section covers data set creation and maintenance, symbolic units, the phase and module libraries, and register conventions.

It is assumed that many programmers are unfamiliar with this IBM System/360 terminology. The section attempts, therefore, to explain each concept as well as provide detailed information on how it applies to the Model 44 system. This is basic information for anyone using the programming system.

OTHER FEATURES

The Error Messages section of this publication lists the messages that the system writes to a programmer or operator. These messages indicate the status of a job at various points during its execution. They also attempt to explain the reason for premature termination of a job, when this is necessary, or indicate why a job cannot be run as requested by a programmer.

The system includes dump facilities that provide listings of the general and floating point registers and the contents of main storage. The System Output section indicates how a programmer requests these listings.

SUPERVISOR CALLS

The system provides many services for a program during execution. An assembler language programmer requests these services with supervisor calls. The compiler provides them for FORTRAN programmers. All input/output, for example, can be requested with supervisor calls. These services are discussed in two sections of this publication, one about input/output and the other about non input/output supervisor calls.

This section discusses data set creation and maintenance, use of symbolic units for input/output operations, the phase and module libraries, and register conventions.

DATA SET CREATION AND MAINTENANCE

All input and output under the programming system is to and from data sets. A data set is defined as a named collection of data.

A data set on a direct access device may be thought of as a box. It is created by an ALLOC job control statement, and, once created, its size is fixed. At any time, it may be empty, full, or partially full. The information in it can be deleted, changed, or augmented at any time, but its size cannot be changed. If it is not big enough for a program's purpose, a new data set must be created with another ALLOC statement.

A magnetic tape or unit record data set differs in that its size may be changed. It is manipulated in essentially the same ways as a direct access data set.

Data Sets

There are two types of data sets, sequential and directoried.

A sequential data set is the familiar structure in which blocks of records are written, one after the other, from the beginning of a storage area to its end. One record is read or written, and then the next sequential record is read or written until the program runs out of records or reaches the end of the data set.

Despite this definition, it is also possible to read and write direct access data sets non-sequentially. When a new data set is being created, the programming system can go through it and mark the starting point of each data block. Then the programmer can write a block of data anywhere in the data set and skip backward or forward to write the next block. This non-sequential processing of sequential data sets is discussed in more detail in the description of the ALLOC control statement later in this publication.

Each data set is named at the time that it is created. When a programmer wants to use a data set, he cites it by name in a control statement and assigns a symbolic unit name to it. The system ensures that all input/output operations referring to this unit are executed on the data set.

A directoried data set is a single data set that contains one or more independent members and a directory that indicates the name and location of each member.

If any data set can be thought of as a box, then a directoried data set is a box containing several compartments. The compartments differ in content and size. Old compartments can be removed at any time and new compartments can be added as long as there is room for them. The first compartment has a special purpose; it contains the name and location of each of the other compartments.

To process a directoried data set member, the data set name and the member name are specified in an ACCESS job control statement. The program then processes the data in the same manner as it does with a sequential data set.

Directoried data sets can be written only on direct access storage. Like sequential data sets, they can be preformatted for non-sequential reading and writing.

A directoried data set member can have more than one name. Each name is listed in the directory. This facility can be used, for example, for a subroutine that performs both sine and cosine evaluation. If it is listed in the directory under both SIN and COS, a programmer can call for it by a name that is meaningful in the context of his program.

Volumes

Data sets reside in volumes. A volume may be a reel of magnetic tape, a disk, or a stack of cards in the card reader.

A tape volume cannot contain more than one data set. A disk may hold many, both sequential and directoried. The card reader, in a sense, contains an endless number. A printer that is used regularly for messages between the system and the operator

may be considered as holding an endless output data set.

Every disk volume must have a volume label. This label contains a volume serial number that the system uses in determining the location of a data set.

A magnetic tape may also contain a volume label and data set header and trailer labels. This depends upon the standard at a given installation.

A disk volume contains a volume table of contents that identifies the name and location of each data set on it. It serves the same function as a directory does for directoried data sets.

Figure 1 illustrates the possible contents of a direct access volume.

The volume table of contents consists of the labels of the data sets in the volume. Every direct access data set must be labeled. A programmer supplies label information in a LABEL job control statement following the ALLOC statement that creates the data set.

All record blocks in a direct access data set must be the same length, although the lengths may differ among data sets. For a directoried data set, the block size must be the same for all members.

Data sets and directoried data set members can be eliminated at any time. The DELETE job control statement removes data sets or members without disturbing the rest of the volume.

When one or more members have been deleted from a directoried data set, other members can be shifted to fill the gaps. This is done with the CONDENSE job control statement. This moves all remaining members toward the beginning of the data set. All vacant space is gathered at the end and may be used for new members. The SQUEEZE utilities program performs the same function for data sets on a volume. All remaining data sets are shifted toward the beginning of the volume, and new data sets may be added at the end.

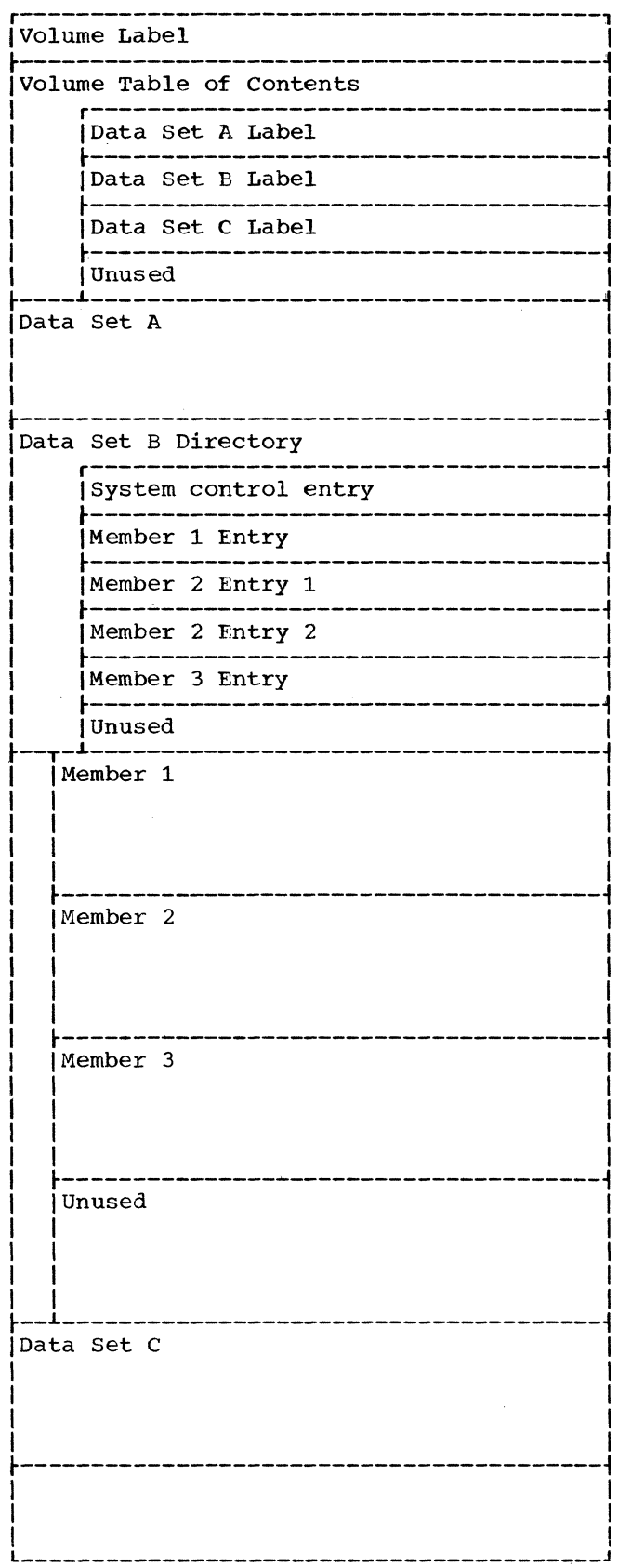


Figure 1. Direct Access Volume

System Catalog

The system catalog contains a list of data sets by name and volume. Its function is similar to that of the volume table of contents and a directoried data set directory.

A programmer creating a data set with an ALLOC statement can use an option in that statement to have its name entered in the system catalog. If it is not done at this time, a CATLG statement may be used for the same purpose. (There is also an UNCATLG statement to remove names from the catalog.)

A cataloged data set can be referred to subsequently solely by name. The programmer need not specify its location.

For example, a programmer uses the following job control statement:

```
//SYS002 ACCESS TESTDATA
```

This statement tells the system that the program needs a data set named TESTDATA and that symbolic unit SYS002 is to be assigned to it. This statement does not identify the location of the data set, either by volume serial number, type of device, or device address. The system gets as much of this information as it needs from the catalog.

No further action is necessary if the desired data set is on the system residence volume or if it was used earlier in the same job. Otherwise, the system writes a message telling the operator what volume is needed and where to mount it.

The system examines the volume table of contents to find out where the TESTDATA data set is on the disk. It also gets the addresses of the first and last data blocks that have been written in the data set. When the program begins its input/output operations, the unit is positioned to read or rewrite the first record in the data set. Had the optional parameter EXT been specified in the ACCESS statement, the unit would be positioned to write a new record after the last block written in the data set.

SYSTEM DATA SETS

The programming system uses several data sets for its routine operations. These are listed in Table 1.

Some of these data sets exist in fact; others are little more than names that are required for administrative purposes.

SDSABS, for example, contains the phase library. This library consists of programs that are ready for execution. When an EXEC job control statement names a program to be executed, the system expects to find it in this library. The program cannot be loaded for execution if it is not in this data set or if the system cannot find this data set.

SDSLOG is an example of a data set that may exist solely for administrative reasons. When system programs write messages to the operator they use symbolic unit SYSLOG for the output device. But a symbolic unit cannot be assigned directly to an output device -- in this case, a console printer. It must be assigned to a data set. So it is assigned to data set SDSLOG, and the address of the printer is given as this data set's location. Unlike SDSABS, any other data set name could be used. The key elements are the symbolic unit name and the printer address.

Table 1 lists the symbolic units, the data sets that serve as their standard assignments, the type of device that may be used, and the function of the data set.

The permissible devices column indicates what type of device can be used for each system data set. Some data sets must use the device indicated. When the system residence disk is mentioned, it is the only acceptable device. For most other system data sets, the devices listed are recommended but are not necessarily the only ones that can be used. For example, the column lists a printer and magnetic tape unit as devices for SDSOPT, the system output data set. In practice, disk also could be used but only with serious loss of efficiency.

Reassignment of symbolic units is discussed in more detail later in this section.

Table 1. Standard Symbolic Unit and Data Set Assignments

Symbolic Unit	Data Set	Permissible Devices	Function
SYSAB1	SDSABS	System Residence Disk	Phase Library, a directoried data set
SYSAB2	SDSABS	System Residence Disk	Required by the system's linkage editor program
SYSREL	SDSREL	Disk	Module library, a directoried data set
SYSLOG	SDSLOG	Console Printer-Keyboard	Communication with the operator
SYSRDR	SDSRDR	Card Reader Magnetic Tape	Job control input
SYSIPT	SDSIPT	Card reader Magnetic Tape	System input unit; may be same device as SYSRDR
SYSLST	SDSLST	Printer Magnetic Tape	System and diagnostic print output; communication with the programmer
SYSOPT	SDSOPT	Printer Magnetic Tape	Main output unit; may be assigned to same device as SYSLST
SYSPCH	SDSPCH	Punch Magnetic Tape	Punch output
SYSUAS	SDSUAS	System Residence Disk	Job Control unit assignment spill
SYSPSD	SDSPSD	Disk	Provides directory for assembler and compiler output on SYS000 which is input for linkage editor
	SDSCAT	System Residence Disk	System catalog
	SDSIPL	System Residence Disk	Initial program load routine
SYS000	SDS000	Disk Magnetic Tape	Assembler and compiler output; input for linkage editor
SYS001	SDS001	Disk	General system work
SYS002	SDS002	Magnetic Tape	
SYS003	SDS003	Any device	
SYS004	None		
SYS005	None		

SYMBOLIC UNITS

Assembler language programmers use symbolic unit names to refer to data sets on external storage. Use of these names permits greater efficiency and flexibility in programming.

FORTRAN programmers use data set reference numbers that the compiler associates with symbolic units.

The first column of Table 1 contains a list of the symbolic names that are used by this programming system. This list ends at SYS005. Additional names in ascending numerical sequence are available at some installations. The highest possible is SYS240.

An assembler language programmer selects a symbolic unit and uses it in coding his input/output operations. When the program is ready for execution, he uses an ALLOC or ACCESS statement to assign the name to a data set.

For example, his program may have to read data from a data set named TESTDATA. The programmer used SYS002 in coding input operations for this data set. For the execution job step, the programmer includes the following control statement:

```
//SYS002 ACCESS TESTDATA
```

Other parameters in this statement can indicate the physical location of the data set. It does not matter whether it is on disk, magnetic tape, or unit record. The system treats all operations for SYS002 as referring to the physical unit containing the TESTDATA data set.

A data set can be transferred from one storage medium to another, such as from cards to disk, without changing the program. Only the ACCESS statement has to be altered to reflect the new location of the data set. The program still refers to it by using the name SYS002. Similarly, one data set can be substituted for another without changing the program.

Some symbolic units must be assigned to particular data sets for the programming system to operate properly. The system data set SDSREL, for example, contains the module library. Whenever the programming system needs a routine from this library, it has a read operation for symbolic unit SYSREL. If unit SYSREL is not assigned to data set SDSREL, the system is unable to find the routine it needs. Therefore, the standard assignment for SYSREL is to SDSREL. Symbolic units with such standard assignments are called system units.

Assignments of system units to standard system data sets are made by the installation operators. These assignments remain in effect for all programs. A problem program may use these symbolic names to refer to the system data sets without assigning them with an ACCESS or ALLOC statement.

The assignment of any symbolic unit can be changed by an ACCESS or ALLOC statement. If SYS002 is assigned to data set TESTDATA for one job step, it can be reassigned to data set FORMA with an ACCESS statement for the next job step. System units can be reassigned in this way, but such action must be done with care. Some system units should not be reassigned, and certain others should be reassigned only during certain job steps. Units are discussed individually later in this section.

All system units revert to their standard assignments at the end of each job. A unit can be returned to its standard assignment during a job by execution of a RESET job control statement.

For example, the standard assignment of SYSPCH may be to a card punch data set. A programmer may be using a utility program to copy a data set that should be written in punch format but is not to be punched into cards at this time. He invokes a utility function that writes its output in punch format on SYSPCH, but he reassigns SYSPCH to a magnetic tape data set. The utility program writes its output on SYSPCH, but it goes onto tape instead of punched cards. The programmer can change this assignment himself for the next job step, or the system will return SYSPCH to its card punch assignment at the end of the job.

Any number of symbolic units can be assigned to a single data set, but no symbolic unit can be assigned to more than one data set at a time.

The assignment of a symbolic unit to a direct access data set or data set member is not affected by the presence of other data sets on the same volume. When the program starts reading or writing, it begins with the first block in the data set or member. An exception to this is when the programmer indicates on an ACCESS statement that the unit is to be positioned after the last block written in the data set so that new material can be added to it.

STANDARD SYSTEM UNITS

The system units and the standard system data sets associated with them are as follows:

SYSAB1: The standard assignment for this unit is to system data set SDSABS on the system residence disk volume. SDSABS is the phase library, a directoried data set, that contains all programs that are to be executed under system control. This unit and data set are used by all system programs. If the assignment of SYSAB1 is

altered, the system is unable to function properly.

SYSAB2: This unit is also assigned to data set SDSABS. The unit and data set are used by the system's linkage editor program, and this standard assignment must be in effect during any linkage editing job step if the edited program is to be executed in the same job. Otherwise, the linkage editor attempts to write its output phases in whatever data set SYSAB2 is assigned to. This is successful only when the data set is a formatted directoried data set, as discussed later with block lengths of 720 bytes.

SYSREL: The standard assignment of SYSREL is to system data set SDSREL, the module library. This directoried data set usually is in the system residence volume. The module library contains the FORTRAN mathematical and service routines, the dump routine, and may also contain relocatable installation routines. This unit and data set are used by the linkage editor.

SYSLOG: SYSLOG's standard assignment is to the SDSLOG console printer-keyboard data set. All system programs use this unit for communication with the operator. It is expected to be assigned to a console printer-keyboard data set since messages calling for operator response may be written at any time.

SYSRDR: The standard assignment for this unit is to system data set SDSRDR. This is expected to be either a card reader or a magnetic tape containing card images. The system uses this unit to read job control statements. SDSRDR is, in a sense, an endless data set that may contain any number of cards. This unit may be used to read any card input in addition to control statements, or its assignment may be altered by an ALLOC or ACCESS control statement if subsequent control statements for the job are in another data set.

SYSIPT: The standard SYSIPT assignment is to system data set SDSIPT. System programs using SYSIPT expect input to be in the form of cards from the card reader or card images from magnetic tape. This assignment may be altered, however, to another data set on any type of device. All system programs require, at times indicated in this publication, that SYSIPT be assigned to data sets containing certain categories of input.

SYSLST: This unit's standard assignment is to the SDSLST data set, which usually is on a printer or magnetic tape. This unit is used for communication with the programmer. It may be assigned to any other data set that will be used for the same purpose. Output on SYSLST is written in printer format. The system may use this unit to write a message at any time during execution of a job. A problem program may use it similarly.

SYSOPT: The standard assignment of SYSOPT is to system data set SDSOPT. Output on SYSOPT is written in printer format. It is expected that SDSOPT is a printer data set or a magnetic tape that will be printed later. The unit, however, may be assigned to any type of data set. The language processors and utility programs use this unit for certain types of output, and reassignment to a non-printer data set may be desired.

SYSPCH: This unit's standard assignment is to the SDSPCH system data set. System output on SYSPCH is in punched card format. It is expected that SDSPCH will be a card punch data set or on a magnetic tape that may be processed through a card punch at some time. The language processors and utility programs use this unit for punched card output, and it may be used similarly by problem programs. It also may be assigned to any other type of data set when there is no punched card output involved.

SYSUAS: The standard assignment of SYSUAS is to system data set SDSUAS. This unit and data set may be required by the system at any time, so the assignment should not be altered. SDSUAS is used by the job control processor for storing input/output unit assignment information.

SYSPSD: System unit SYSPSD's standard assignment is to system data set SDSPSD. This unit and data set are used by the language processors and the linkage editor program. The standard assignment must be in effect during any linkage editing job step and during the assembly or compilation of any program that is to be linkage edited later in the same job.

SYS000: The standard assignment of SYS000 is to system data set SDS000. The language processors write their output modules in this data set when assembling or compiling a program that is to be linkage edited later in the same job. This standard assignment must be in effect during such language processing job steps and during any linkage editor job step.

SYS001, SYS002, and SYS003: These are general assignment units for use by system and problem programs. They have nominal standard assignments to system data sets SDS001, SDS002, and SDS003. These assignments may be altered freely. The language processors and utility programs use these units for certain operations. Linkage editor uses SYS001.

SYS004 and SYS005: These are general assignment units available for unrestricted use. They are not used by the system programs.

SYS006 through SYS240: These symbolic unit names can be used only by installations that modify the system programs distributed by IBM. For further information about this subject, see IBM System/360 Model 44 Programming System: Systems Programmer's Guide, Form C28-6814.

Unit name SYSDMY is used by certain system programs but is not available for use by problem programs.

Special System Data Sets

The system data sets SDSCAT and SDSIPL are also resident in the system residence volume. They are used only by systems programs, and their use does not affect the assignments of symbolic units. SDSCAT contains the system catalog, and SDSIPL contains the initial program loading routines.

Replacing System Data Sets

An installation can replace any system data set for a single run or permanently. It must ensure, however, that required material remains available to the system.

If, for example, an installation wants to add new routines to the module library, SDSREL, but this data set is full, another can be created. The installation can allocate a larger data set, copy the old contents of SDSREL into it, add new routines, and assign SYSREL to it.

Similarly, an individual programmer can create a data set that will function as SDSREL for a single job only. He creates his data set and assigns system unit SYSREL to it. Whenever his program or a system program wants a routine from the module library, it reads from system unit SYSREL and, therefore, from the programmer's special data set. It does not matter what name has been assigned to this special data set.

In a situation such as this, the programmer risks the possibility that the system will require a module library routine that he has not included in his special data set. In any case, at the end of the job, the system reassigns SYSREL to the SDSREL system data set.

Early Unit Assignments

Changes in symbolic unit assignments are indicated in ACCESS and ALLOC statements, but the system usually does not actually set up these new assignments until it processes an EXEC statement. (Set up of pending assignments also takes place when a DELETE, RENAME, or CONDENSE statement is processed, but these are special cases.)

Because of this, unit assignments are not actually altered until all control statements for the first job step of a job have been read. The /* end of data job control statement can be used to force an earlier set up.

For example, if SYSLST were normally assigned to a printer, and a programmer wanted to switch it to a tape, the following statements could be used:

```
// JOB

//SYSLST ALLOC TAPE,2400=FRESH

/*
```

(The ALLOC statement parameters are explained later in this publication.)

The JOB, ALLOC, and /* statements would be written on the printer, but all subsequent job control statements and messages would be written on the tape.

SYSLST would revert to the printer at the start of the next job when the job control processor resets all system units to their standard assignments. Special action by the operator is necessary if an assignment must be altered before the JOB statement is processed.

PHASE LIBRARY

The phase library is the source of all programs and program segments executed under system control.

Processing of a new problem program requires three job steps, although they need not all be in the same job. These are assembling or compiling, linkage editing, and execution.

Output from the linkage editor job step is placed in the phase library automatically by the system. It can be kept here temporarily or permanently. When execution is to start, the system searches the phase library for the program named in the EXEC job control statement that initiates the

job step. This program, or the first portion of it, is loaded into main storage, and execution of it commences.

Program segments in the phase library are called phases. Each phase represents a segment of code that is to be loaded into main storage at one time. It may be an entire problem program or part of a multiphase program. Each phase has been edited into absolute form by the linkage editor. All preliminary work has been completed, and it is ready for execution.

The IBM-supplied processors and transient supervisor functions reside permanently in this library. User-written programs may reside in it permanently or only for the duration of a single job. All additions to the library, permanent or temporary, are made through the linkage editor.

The library is a directoried data set. The directory contains an entry for each resident phase. Each entry contains an eight-character name for the phase it identifies. (Names of fewer than eight characters are padded on the right with blanks.) Directory entries also include information about where the phase is located in the library, its main storage loading address, and its main entry point. The system uses this information to find, load, and enter a phase unless the programmer provides an alternate load address. (An alternate load address can be specified with the LOAD supervisor call.)

A program is initially loaded for execution by specifying its name in an EXEC statement. If the programmer has arranged his program in several phases, all phases after the first are loaded by execution of FETCH or LOAD supervisor calls.

The one or more phases constituting a problem program can be kept in the phase library permanently by specifying the KEEP parameter in the EXEC statement initiating the linkage editor job step. Otherwise, the phases are eliminated at the end of the next job step or at the end of the job, whichever occurs first. Normally, therefore, the execution job step should immediately follow the linkage editing job step when KEEP is not specified.

One or more phases from the phase library can also be copied onto magnetic tape or punched into cards. This is done with the PCHABS utilities statement for directoried data sets.

The control statements, supervisor calls, and job step functions are discussed in detail in appropriate sections later in this publication.

The name of the phase library directoried data set is SDSABS. System units SYSAB1 and SYSAB2 usually are assigned to it. This library must reside on the system residence disk volume.

MODULE LIBRARY

The module library contains relocatable program modules, which have been assembled or compiled, and are available for incorporation into any program.

Such modules as specialized mathematical subroutines are designed for residence in this library. Considerable programming time can often be saved by using this library for permanent storage of various routines and subroutines that are used frequently by one or more installation programs.

These modules are incorporated into a program by the linkage editor. This can be done automatically or by specifically indicating in a linkage editor control statement that a certain module is to be included.

The module library is a directoried data set, named SDSREL. System unit SYSREL is assigned to it. New modules can be added to this library and old modules deleted in the same way that members are added to and deleted from any directoried data set.

In practice, any data set can be used as a module library by assigning SYSREL to it. When a substitute module library is used, however, it must contain all the routines the system will require during execution of the job. The system's module library uses a block length of 360 bytes with five logical records per block.

The directory of the module library contains the name of each routine in the library. For the IBM supplied routines, the names of all entry points in a routine are also listed in the directory. This enables the system to find the proper routine when only its entry point is defined as an external symbol in the using program.

The following is an example of the assembler language coding that can be used to incorporate a module library routine into a program and enter it.

```
            EXTRN     ROUTINE
            .
            .
            .
            L         12,ARTNE

            BALR      14,12
            .
            .
            .
ARTNE       DC        A(ROUTINE)
```

During linkage editing, the module library routine would be incorporated into the program. The linkage editor would supply the address of the entry point ROUTINE as the value of the address constant ARTNE. The program is then able during execution to load that address into a register and branch to it.

The compiler supplies the coding to obtain these routines for FORTRAN programs. In both cases, however, additional coding may be necessary to supply parameters containing information for the routine to process.

For information about routines in the module library and their required parameters, see the publication IBM System/360 Operating System, FORTRAN IV Library Subprograms, Form C28-6596.

REGISTER CONVENTIONS

Certain register conventions should be observed when preparing programs for execution under the Model 44 Programming System.

This publication specifically states when a system routine expects to find certain information in a particular general register and when it uses a register to pass information to a program. These situations occur most often when supervisor calls are executed, but other instances occur frequently.

The system adheres to the following conventions:

Registers 0 and 1 are used to pass information between the system routines and a problem program.

Registers 2 through 12 can be used freely by a problem program. Any system routine that uses them always saves and restores their contents before returning control to a problem program.

Register 13 is used, where specified, to inform the system of the address of an area where the contents of the general registers are to be saved while a system routine is being executed.

Register 14 is used to indicate a return address. When a program branches to a subroutine, for example, it can store the return address in this register and branch to this address when the subroutine is completed.

Register 15 is used for return codes. Its contents are destroyed by execution of any supervisor call. When control returns to a problem program after execution of a supervisor call, this register usually contains a code indicating whether the system was able to execute the requested operation properly.

The system's job control processor is loaded before each job and after each job step. It reads and interprets the problem program's job control statements and provides requested services and facilities.

The statements, in general, describe the machine and system resources that will be required to execute the program. They indicate which optional system features will be desired, and they specify administrative functions that should be performed.

When the job control processor finishes processing control statements, it causes loading of the required program into main storage from the phase library and initiates execution of the job step.

At the end of the job, the job control processor provides requested end-of-job services and restores the system to its status as of the beginning of the job. It then begins processing control statements for the next job.

JOB CONTROL STATEMENTS

Some control statements, such as JOB, are required for all jobs executed under the Model 44 Programming System. Most are optional, with their use depending upon which system services, above the minimum, the program wishes to use.

The statements are divided into groups, as follows:

1. JOB, EXEC, end of job, STOP, and end of data are the job definition statements.

2. The ALLOC, ACCESS, and LABEL statements are used to create new data sets and use old ones.

3. LISTIO and RESET are used with system unit assignments.

4. DELETE, CONDENSE, RENAME, CATLG, and UNCATLG are the data set maintenance statements.

5. There are four miscellaneous statements: PAUSE, REWIND, UNLOAD, and the comments statement.

The statements are shown in Table 2, and each is discussed in detail in this section.

Table 2. Job Control Statements

Statement	Function
Job Definition	
// JOB	Defines the start of a job.
// EXEC	Defines the start of a job step execution and the program to be executed.
/&	Delimits the end of a job.
// STOP	Delimits the end of a series of jobs.
/*	Delimits the end of data in the input stream.
Symbolic Unit Assignment	
// ALLOC	Allocates space for a new data set.
// ACCESS	Permits access to an existing data set.
// LABEL	Defines characteristics of a data set.
Symbolic Unit Maintenance	
// RESET	Restores unit assignments to status at start of job.
// LISTIO	Lists data set and device assignments on system log.
Data Set Maintenance	
// DELETE	Deletes a data set from a volume or a member from a directoried data set.
// CONDENSE	Condenses a directoried data set.
// RENAME	Renames a data set or a member of a directoried data set.
// CATLG	Enters a data set name into the catalog.
// UNCATLG	Removes a data set name from the catalog.
Miscellaneous	
// PAUSE	Allows pause for operator action.
* (comments)	Allows logging of comments to system log.
// REWIND	Rewinds a tape volume; repositions a data set on a direct access volume to beginning.
// UNLOAD	Rewinds and unloads a tape.

CONTROL STATEMENT FORMAT

Job control statements are designed for an 80-column punched card format. Although certain restrictions must be observed, the statements have several free form characteristics. Information must start in column 1 and cannot extend past column 71, and most fields must be entered in a specific order. But there is no requirement, for example, that an operation or operand field must start in a particular column.

If the length of a statement exceeds 71 characters, it may be continued on additional cards, as described later in this section.

Each statement contains from one to five fields.

1. The first field consists of two characters, such as //, that identify a job control statement.

2. A NAME field is permitted on JOB, EXEC, ALLOC, and ACCESS statements. This field may be used to assign a name to a program or program segment or to assign a symbolic unit to a data set.

3. An OPERATION field, such as JOB or CONDENSE, identifies the type of statement and the action required. It is required in all except the end of job, end of data, and comments statements where the initial characters perform these functions.

4. An OPERAND field contains the statement's required and optional parameters. A programmer uses these parameters to provide detailed information and, in many cases, to indicate which of several possible actions he wants to take.

5. A COMMENTS field. This field is described in detail later in this section.

The first two columns of any job control statement must contain the identifying characters. These are /& for the end of job statement, /* for the end of data statement, * followed by a blank for the comments statement, and // for all other job control statements.

If a name field is used in those statements which are permitted to have one, it must start in column 3. If the name field is not used, the third column must be blank.

In all other statements, except comments, the third column must be blank.

The operation field may start in any column after column 3, but it must be preceded and followed by at least one blank.

Similarly, the operand field may start in any column, but it must be preceded by at least one blank, and it must be followed by at least one blank.

Parameters within the operand field are separated by commas and parentheses. Blanks may be used only where this publication specifically states that they are permitted. In most cases, when the system detects a blank in the operand field, it assumes it has reached the end of the field. Any characters following the blank are treated as comments unless the statement is continued on another card, as described below. Column 72 is used to indicate whether the statement is continued.

Columns 73 through 80 are ignored by the system and may be used for any purpose.

A JOB statement could take any of the following forms. The words PARAM1 and PARAM2 simulate the actual parameters entered by a programmer.

```
//SAMPLE  JOB  PARAM1,PARAM2

// JOB PARAM1,PARAM2

// JOB
```

The /& end-of-job, /* end-of-data, and * comments statements are special purpose statements that do not have name, operation, and operand fields. They require only the classification characters in columns 1 and 2.

CONTINUATION STATEMENTS

Any statement can be continued on another card if the information does not all fit on one card. Any non-blank character in column 72 indicates that the following card is a continuation of the statement. This character is not considered part of the statement.

The name and operation fields of a statement cannot be continued. They must be specified entirely in the first card of a statement. The operand field must start in the first card. After its first parameter is entered, it may be continued on as many additional cards as necessary.

A statement can be interrupted only after a comma. For instance, a parameter, such as PARAM6, cannot be split so that PAR appears on one card and AM6, appears on the continuation card.

The last operand field parameter in the card must be followed by the comma that would normally fall in that position. The rest of the card may then contain comments, with the operand field resuming in column 16 of the next card.

It is not necessary to fill a card before continuing it. The text may stop in any column, and be followed by a comma, a blank, and additional blanks or comments through column 71. A non-blank character is entered in column 72.

A parameter field that consists of several entries enclosed in parentheses, such as (PARAM1,PARAM2,PARAM3),PARAMX, may be interrupted after any of the commas.

The continuation card must be the next card. The first two columns must contain the characters // and the continued operand field must start in column 16.

There is no limit on the number of continuation cards that may be used for a single statement. There must be a non-blank character in column 72 of each, except the last. The first two columns must contain the // entries, and the text must start in column 16.

If column 16 of any continuation card is blank, the text on it and any subsequent continuation card for the same statement is treated as comments.

The /& end of job, /* end of data, and * comments statements cannot be continued. There can, however, be any number of consecutive * comments statements.

COMMENTS IN CONTROL STATEMENTS

Comments may be included in job control statements. They are entered following the operand field and must be separated from the last parameter by at least one blank. Blanks may be included within the comments. Comments are written on SYSLST.

Comments on continuation cards may start after the end of the operand field and be continued on additional continuation cards, or each card may contain a portion of the operand field and comments. When the latter technique is used, a comma and at least one blank must separate the last entry for the operand field on each card from the comments on that card. The operand field resumes in column 16 of the next card.

Comments are permitted in statements that do not have an operand field, such as the end-of-job and end-of-data statements, as long as the comments are preceded by at least one blank. Comments on the /* end of data statement, however, are not printed. For statements in which an operand field is permitted but is not being used, the absence of the field must be indicated by a comma and at least one blank before the start of any comments.

Following are several sample control statements containing comments:

```
                1                               7
1               6                               2
//NAME JOB PARAM1,PARAM2 COMMENTS

//NAME JOB , COMMENTS

//STEPNAME EXEC PARAM1,PARAM2(PARAM3,     X
//              PARAM4),PARAM5 COMMENTS   X
//              COMMENTS

//STEPNAME EXEC PARAM1,PARAM2, COMMENTS  X
//              (PARAM3,PARAM4), COMMENTS X
//              PARAM5 COMMENTS

* COMMENTS

/& COMMENTS
```

Character Set

Statements may contain any of the 39 alphameric characters recognized by the system. The term "alphameric characters" refers to both alphabetic and numeric characters.

Alphabetic characters are defined for the system as the 26 regular alphabetic characters, A-Z, plus three special characters $ # @.

The numeric characters are 0-9.

The following special characters also may be used in job control statements:

```
*  asterisk
,  comma
=  equals sign
() parentheses
'  single quote
/  slash
```

LOCATION

The job control processor reads its job control statements from symbolic unit

SYSRDR. The standard assignment for this unit is to system data set SDSRDR on a card reader.

SYSRDR may be assigned to the same data set as SYSIPT. This may be a standard assignment or one for a single job. In such cases, job control statements and data may be combined in a single data set. A /* end of data statement must follow any data that is followed by control statements in the same data set.

FORMAT DESCRIPTIONS

The following conventions are used in this publication when describing the formats of control statements.

1. Upper case letters and punctuation marks, except as indicated below, represent information that must be coded in a statement exactly as shown.

2. Lower case letters and words are generic terms that represent fields for which information must or may be supplied. The programmer is expected to determine the exact characters that should be entered.

3. Brackets [ ] are used to represent an option that may be included or excluded, depending upon program requirements.

4. Braces {} are used to enclose fields where one of two or more alternatives must be chosen.

5. An ellipsis ... indicates that a field may be repeated a variable number of times. For example, (name,...) indicates that the program may specify any number of names in the field, subject to any limitations mentioned in the text, and that they are separated by commas.

When a comma or another punctuation mark appears within parentheses or brackets with a parameter, it is an integral part of that parameter. For example, the format of a control statement might be illustrated in this publication as follows:

```
// LABEL  [PARAM1][,PARAM2]
```

Use of this format means that both parameters are optional. They may be included in the statement, but they are not required. The statement could be punched in any of the following formats:

```
// LABEL  PARAM1,PARAM2
// LABEL  PARAM1
// LABEL ,PARAM2
```

It should not be punched as follows:

```
// LABEL  PARAM2
```

In this example, if PARAM2 is used, it must include the comma, regardless of whether PARAM1 also is used. If a parameter is not used, neither is any punctuation mark associated with it.

Except where stated otherwise, parameters should be punched in the same order as they are shown in the statement format.

STATEMENT FORMATS

Following are the descriptions and formats of the job control statements. A sample deck of job control statements is shown in Figure 9 at the end of this section.

JOB DEFINITION STATEMENTS

JOB Statement

The JOB statement defines the start of a job.

One JOB statement is required as the first statement in each job. It may be used to assign a name to the job, request a dump of main storage and the general registers if the job terminates abnormally, and to pass information to an installation accounting routine.

The JOB statement's format is shown in Figure 2.

```
//[jobname]  JOB  [DUMP  ] [,accounting information]
                  [NODUMP]
```

Figure 2. JOB Statement

The parameters are as follows:

jobname
: This field specifies the name of the job. If used, the name must consist of one to eight alphameric characters, the first of which must be alphabetic.

  This name identifies the job. It is stored in bytes 24 through 31 of the user communication region for the duration of the job.

DUMP
NODUMP
: This field is used to indicate whether a dump is wanted if the job terminates abnormally.

  The DUMP keyword causes a dump of the general registers and all of main storage onto SYSLST if the program terminates abnormally. The dump is in hexadecimal notation.

  The dump program and the available listing formats are discussed in the System Output section of this publication.

  The NODUMP keyword suppresses all dumping for the job.

  If neither keyword is specified, NODUMP is assumed.

accounting information
: This field may consist of up to 16 alphameric characters to be passed to an installation accounting routine. The first character cannot be a left parenthesis or a blank. Otherwise, blanks may be included among the 16 characters.

  The contents and format of the field are determined by the installation.

  If the installation has an accounting routine, these characters are stored in bytes 128 through 143 of the user communication region. They replace anything placed in these bytes by a previous JOB or EXEC statement.

Example:

```
//STATRUN JOB DUMP,137596
```

where

// are the required job control statement identification characters.

STATRUN is the name the programmer is assigning to the job.

JOB identifies the operation.

DUMP requests a listing of the general registers and the problem program portion of main storage if the program is cancelled before its planned end.

137596 is an illustration of a code the installation might require for its accounting routine.

EXEC Statement

The EXEC statement defines the end of job control information for a job step. It causes the system to load a program or program segment from the phase library and to commence executing it.

One EXEC statement is required for each job step. It must be the last statement processed before execution of the job step is to begin.

The EXEC statement names the program to be executed. This may be a system program, such as the linkage editor, or it may be the user's problem program.

The program to be executed must be resident in the phase library when the EXEC statement is processed. Any system program may be considered permanently resident in this library.

If the system is unable to find the program in the phase library and load it properly, the job is cancelled.

An EXEC statement may contain parameters to be passed to the problem program. It also may assign a name to the job step, specify settings for the user program and variable precision switches, and pass information to an installation accounting routine.

The format of the EXEC statement is shown in Figure 3.

```
//[stepname]  EXEC  [progname][(parameter list)][,(switch options)][,accounting
                                                                    information]
```

Figure 3. EXEC Statement

The parameters are as follows:

stepname

This field specifies the name of the job step. If used, the name must be one to eight alphameric characters, the first of which is alphabetic.

A job step name is stored in bytes 32 through 39 of the user communication region while the job step is being executed. If the job terminates abnormally, this information can be used to help determine the point of termination.

When the EXEC statement is used to initiate an assembly or compilation job step, the name in this field is assigned to the output module. This name is then used when referring to the module in the linkage editing job step.

progname

This field specifies the name of the program to be executed. The name must be one to eight alphameric characters, the first of which must be alphabetic.

When a system program is to be executed, its name is given in this field. For example, EXEC LNKEDT causes the system to load and begin executing the linkage editor program. The names of other system programs are FORTRAN for the FORTRAN IV language processor, ASSEMBLE for the assembler language processor, and UTILS for the system's utilities program.

The name of a problem program is specified in this field to initiate an execution job step. It is not necessary to specify a name, however, if the problem program was linkage edited in the immediately preceding job step and execution is to begin with its first phase.

Otherwise, this field must specify a phase name that was assigned during linkage editing. If the programmer did not supply a phase name at that time, the linkage editor gave the phase the same name as the first language processor output module that it processed. (The module received its name from the stepname field of the EXEC statement that initiated the assembly or compilation job step.)

For a multi-phase program, this field specifies the name of the phase that is to be executed first.

The program named must reside in the phase library. If the system is unable to find the program and load it properly, the job is cancelled. System programs are permanently resident in this library.

Appendix C contains a list of phase names used by system programs. Duplication is not permitted. A problem programmer must not use any of these names for his program or as phase names.

parameter list

This field may specify up to six parameters to be used by the program that is about to be executed. If, for example, the linkage editor is to be executed, this field could be used to pass the parameter KEEP, which indicates that a program is to be kept in the phase library until specifically deleted.

Each parameter is discussed in more detail in the section concerning the program to which it applies. An installation may define additional parameters. During execution of the job step, these parameters are saved in bytes 56 through 103 of the user communication region. A programmer can use them to establish his own execution-time options.

Parameters in the field are separated by commas, and the entire field is enclosed in parentheses. Each parameter may consist of up to eight alphameric characters.

switch options

This field is used to specify settings for the variable precision switch on the operator's console and for the user program switch in the user communication region.

Either or both switch options may be specified. If both are given, they are separated by a comma. The entire field must be enclosed in parentheses. No comma is needed if only one option is specified.

The variable precision switch specifies the precision required for floating point operations. Precision may be 8, 10, 12, or 14 bits.[1] In general, the highest precision provides greatest accuracy and the lowest precision provides greatest speed.

---

[1]Performance statistics given for mathematical subroutines in the module library are based on a VPS setting of 14. See IBM System/360 Operating System, FORTRAN IV Library Subprograms, Form C28-6596.

The switch must be set manually by the operator. This field is used to ensure that his setting is correct. If it is not correct, a message is written giving the proper setting, as indicated by this field.

The field is written as VPSxx, where xx may be 08, 10, 12, or 14.

The system tests the operator's setting. If it is correct, the system proceeds with normal processing. If it is not, the setting specifed in this statement is written, and the system pauses until the operator indicates that he has reset the switch. The system tests it again and proceeds if it is correct or pauses until the operator resets it again. This process continues until the setting is correct. The default setting is 14. This value is set each time a JOB card is read.

The user program switch is byte 40 in the user communication region. It may be used by the problem program for communication between job steps.

The EXEC statement may specify an initial setting for the switch. Assembler language coding and supervisor calls are required to reset or read the switch. The switch is reset to zeros at the beginning of each job.

This field in the EXEC statement contains up to eight characters. Each character corresponds with a bit in the switch byte. Each character in the statement must be either 0, 1, or X. A 0 in the statement causes its corresponding bit to be set to 0. A 1 results in a 1 bit, and an X indicates that the corresponding bit is to be left as it is. If fewer than eight characters are given, the rightmost positions are assumed to be X's.

accounting information

This field may consist of up to 16 alphameric characters to be passed to an installation accounting routine. The first character cannot be a left parenthesis or a blank. Otherwise, blanks may be included among the 16 characters.

These characters are stored in bytes 128 through 143 of the user communication region. They replace anything placed in these bytes by a previous JOB or EXEC statement.

The format and contents of this field are specified by the installation.

Example:

```
//RUN1 EXEC PHASE1(201,ON),(VPS14,00001111)
```

where

// are the required job control statement identification characters.

RUN1 is the name the programmer is assigning to the job step.

EXEC identifies the type of operation.

PHASE1 is the name of the program or program phase in the phase library that is to be loaded and executed.

201 and ON are illustrations of parameters to be stored in the user communication region for use by the problem program.

VPS14 specifies a variable precision switch setting of 14 for floating point operations.

00001111 sets the first four bits of the user program switch to zeros and the last four bits to ones.

This example does not include an installation accounting routine code. One could follow the switch settings, preceded by a comma. It would replace any accounting routine code specified by a previous EXEC or JOB statement.

Example:

```
// EXEC PHASE1
```

This statement causes the system to load and enter the same program phase as in the previous example. It is a valid statement that makes use of none of the optional parameters. Even the PHASE1 parameter could be omitted if this is the name of the first phase of a program that was linkage edited in the immediately preceding job step.

End-of-Job Statement

The /& end-of-job statement indicates the end of a job. It must be the last statement in each program deck.

(A job step is terminated by execution of an EOJS supervisor call. This includes the final step of the job.)

This statement causes the system's job control processor to terminate the job and to prepare for the next one. This includes returning system units to their standard

assignments, performing any cataloging operations that were requested during the job, and reinitializing system records.

If a job is canceled before it reaches its normal end, the system skips all remaining control and program statements until it detects a /& statement.

The statement's format is:

```
/&
```

The rest of the statement is blank, unless comments are desired and are entered according to the previously specified rules.

STOP Statement

The STOP statement is used to denote the end of a job input stream. It is for use more by the system operator than by a problem programmer.

This statement may be used after a series of one or more jobs. It causes the system to enter the installation accounting routine, if there is one, and then close all system files.

The system waits for input. The operator may initiate another job or stop the system. If the system is stopped, an initial program load procedure is required to resume system operations.

The statement's format is:

```
// STOP
```

The rest of the statement is blank, unless comments are desired.

End of Data Statement

The /* end-of-data statement denotes the end of input data in the job input stream.

This statement is used to separate program data and instructions from subsequent control statements. It is required after any data set on the system input unit, SYSIPT, when that unit is the same as the system reader, SYSRDR.

The statement also indicates the end of input to the linkage editor and language processor programs. When input and control statements are in separate data sets, it goes in the input data set.

All program must recognize the /* statement or an equivalent. The requirement is to ensure that no problem program inadvertantly reads a control statement that is required for the next job step or job or for the proper termination of the current job.

The statement also has specialized uses that are discussed with other topics in this publication.

The statement's format is:

```
/*
```

The rest of the statement is blank, unless comments are wanted.

DATA SET DEFINITION STATEMENTS

The data set definition statements are ALLOC, ACCESS, and LABEL. ALLOC is used to create and name a data set. ACCESS is used to identify an existing data set that will be used by a program and to assign a symbolic unit to it. LABEL provides additional information about the data set for use by the system in preparing optional and required labels and control blocks for input/output operations.

ALLOC Statement

The ALLOC statement is used to create and name a data set.

An ALLOC statement is required to create a new data set in a direct access volume. Its use is optional but recommended for creating other types of data sets. It also is used to assign symbolic units to data sets or ensure that they are properly assigned.

Although ALLOC is used to create directoried data sets, ACCESS statements are required to create members.

For direct access and labeled magnetic tape data sets, an ALLOC statement must be followed immediately by a LABEL statement which provides additional information about the data set. Once a direct access data set is created, its size cannot be changed. It is not necessary, however, to make any

use of a data set in the job or job step in which it is created.

The ALLOC statement also may be used to assign a symbolic unit to a data set, enter the data set's name and location in the system catalog, and format the area allocated for a direct access data set so that it may be used for random access processing.

The possible formats of the ALLOC statement and brief descriptions of its fields are shown in Figure 4. The parameters of the ALLOC statement are as follows:

SYSxxx
This field assigns a symbolic unit to the data set.

This field is not required if no input/output operations will be performed on the data set during the job. It should be left blank if a symbolic unit will be assigned by an ACCESS statement later in the job.

This field assigns a symbolic unit to an entire data set, beginning at its first block. Since the first block of a directoried data set is the start of its directory, an ACCESS statement is used to make an assignment to a member. This applies whether the member already exists or will be created by the job.

Formats	
For direct access data sets: //[SYSxxx] ALLOC dsname[,volume],data length[,directory length][,ROUND][,FMT][,CATLG]	
For magnetic tape and unit record data sets: //[SYSxxx] ALLOC dsname[,volume][,CATLG]	
Fields in brief	
//	Required. Identifies job control statement.
SYSxxx	Assigns symbolic unit to the data set. Not required if no data will be read from or written into the data set during the job step.
ALLOC	Required. Identifies type of statement. Must be preceded and followed by at least one blank.
dsname	Required. Assigns name to the data set being created.
volume	Identifies type of data set (direct access, magnetic tape, or unit record) and its location. Required except when the data set is to be created on the system residence disk volume.
data length	Required to indicate how much disk space should be reserved for a direct access data set.
directory length	Required for directoried data sets only. Indicates number of entries in directory so system can reserve enough space within the data set for a directory.
ROUND	Indicates that allocation of space for a direct access data set is to begin and end on a cylinder boundary.
FMT	Formats entire area of disk data set to permit immediate nonsequential reading and writing of data blocks.
CATLG	Enters name of data set in system catalog.

Figure 4. ALLOC Statement

dsname

This field specifies the name of the data set. The name may be one to eight alphameric characters, the first of which must be alphabetic.

volume

This field identifies the volume and device on which the data set is to be created. Figure 5 shows the field's format and parameters.

When this field is omitted, the data set is created on the system residence disk volume.

The field consists of one of the following sets of entries:

```
SAME[(options)]={dsname}
                {SYSxxx}

{type  }[(options)]=['volid']
{devadr}           [FRESH ]
```

The right side of the field need not be specified for unit record and printer data sets.

The series beginning with SAME is used when the new data set is to be created on the same volume as a known existing data set.

The series beginning with type is used when the chief consideration is the type of device that is to hold the data set.

The devadr series tells the system to create the data set on whatever device is at a specific address.

If both the type and devadr methods are used within the same job step, the devadr request should be made first if both refer to the same type of device. Otherwise, the system, in processing the type request, might select the device intended for use in the devadr request.

The devadr method also should be used when more than one device of a given type exists on the system and a user wants to mount a volume before starting the job. When type is specified, the system selects a device and instructs the operator to mount the volume on it. It can be difficult to predict which of several possible devices will be selected.

```
SAME[(options)]={dsname}
                {SYSxxx}

{type  }[(options)]=['volid']
{devadr}           [FRESH ]
```

Fields in brief	
SAME	Indicates that the volume is the same as one used previously in the same job.
options	Used to request special features for input/output. See Table 3.
dsname	Name of a data set on the same volume that was used earlier in the job.
SYSxxx	Name of symbolic unit assigned to data set used earlier in the job.
type	Type of device to be used. See Table 4.
devadr	Address of device to be used.
'volid'	Identification number of volume to be used.
FRESH	Requests volume containing no active data sets.

Figure 5. Volume Field Format and Parameters

SAME

The SAME keyword indicates that the new data set is to be created on the same volume as an existing data set. Since the system can determine the location of the exiting data set, it needs no further information about where the new one should go. If necessary, the console operator is instructed to mount the volume containing the indicated data set.

The other data set must be one of the system data sets for which an ALLOC or ACCESS statement was processed previously in the job or a cataloged data set. It may be identified by name or by the name of a symbolic unit that is currently assigned to it.

Table 3. Volume Options

Device	Option	Entry in Option Field[1]	Explanation
7 track tape (with or without Convert Feature)	density[2]	200	200 bytes/inch
		556	556 bytes/inch
		800	800 bytes/inch
	parity	O	Odd
		E	Even
	Translate Feature	T	Translate ON
		NT	Translate OFF
7 track tape (with Convert Feature)	Convert Feature	C	Convert ON
		NC	Convert OFF
9 track tape (with dual density feature)	density[2]	800	800 bytes/inch
		1600	1600 bytes/inch
Disk	Write Check	WRCHK	Write Check ON
		NOWRCHK	No Write Check
			Default condition is as specified in label.

[1]Underlined entries are default options.

[2]On dual density tapes, the density specified for the volume label, if any, overrides any subsequent density specificaton. When another density is specified for reading or writing data sets or other labels on the volume, it is ignored. For other types of tape, a data set must be read in the same density as it was written, or an error code is returned when the data set is opened.

dsname
: This field specifies the name of an existing data set on the same volume that is to contain the new data set. This field is used with the SAME keyword to indicate the location of the new data set. The name specified may be that of a data set for which an ALLOC or ACCESS statement was processed earlier in the job, a system data set, or of a cataloged data set.

SYSxxx
: This field identifies a symbolic unit that is currently assigned to a data set on the same volume that is to contain the new data set.

options
: The possible entries for this field are shown in Table 3. The choice of those to be used varies according to whether the device involved is unit record, direct access storage, or magnetic tape. If nothing is specified, the default options are assumed where applicable.

For an IBM 2400 series magnetic tape unit with a 7-track read/write head, up to four options may be specified. These specify tape density and parity and indicate whether the translate or convert features are to be used.

The translate feature is used to switch data between 7-track and 9-track tapes. The convert feature is used to read and write binary data on 7-track tapes. If the convert option is ON, the system assumes odd parity and the translate feature OFF.

For a 9-track tape with the dual density feature, density may be specified.

Table 4. Type Field Entries

Entry	Explanation
SDSD	Single Disk Storage Drive (2315 Disk Cartridge)
1316	IBM 1316 Disk Pack mounted on a 2311 Disk Storage Drive
2400	IBM 2400 Magnetic Tape Unit with 9-track read/write head; 800 bpi only
2400H	IBM 2400 Magnetic Tape Unit with 9-track read/write head; 1600 bpi only
2400D	IBM 2400 Magnetic Tape Unit with 9-track read/write head; dual density
2400T7	IBM 2400 Magnetic Tape Unit with 7-track read/write head
2400T7C	IBM 2400 Magnetic Tape Unit with 7-track read/write head and the Convert Feature
1052	IBM 1052 Console Printer-keyboard
1442	IBM 1442 Model N1 Card Read-Punch
1442P	IBM 1442 Model N2 Card Punch
2520	IBM 2520 Card Read-Punch
2520P	IBM 2520 Model B2 or B3 Card Punch
2501	IBM 2501 Card Reader
2540	IBM 2540 Card Read-Punch (Reader side)
2540P	IBM 2540 Card Read-Punch (Punch side)
1403	IBM 1403 Printer, Model 2, 3, or N1 (132 characters)
1403M7	IBM 1403 Printer, Model 7 (120 characters)
1443	IBM 1443 Printer, Model N1 (120 characters)
1443S	IBM 1443 Printer, Model N1 (144 characters) (Special Feature)

For a disk volume, only one option, write validity checking, may be specified. When this is specified, the system reads back each data block that is written. No data is transferred in this read-back procedure, but the system checks for an error indication. Standard recovery procedures are initiated if any error is detected. This operation requires an additional disk revolution for each data block that is written. This field overrides the WRCHK field of the data set's LABEL statement for the job. If write checking is not requested here or in the LABEL statement, the system assumes it is not wanted.

Options may be specified in any order. The entire field is enclosed in parentheses, and if there is more than one entry, they are separated by commas.

type

This field specifies the type of device that the data set is to be created on. Possible types are shown in Table 4.

The system selects a device that meets the type specification. It writes a message telling the operator to mount the volume on the selected device.

'volid'

The 'volid' field specifies the one-to-six character serial number of a disk or magnetic tape that is to contain the new data set. For a unit record device or unlabeled tape, this field may be omitted.

The entry must be enclosed in single quotes, as indicated.

The volume serial number is created

when an installation initializes a volume using the system utility programs. This number can be obtained from the volume label, a listing of volume contents, and, in some cases, from a listing of symbolic unit assignments. In most installations, a volume is labeled externally with its identification number.

When this field is not specified, the system assumes the volume serial number consists of six blanks.

The LISTIO statement, which lists the current assignments of all system units, also provides a listing of volume serial numbers.

FRESH

The FRESH keyword indicates that an empty direct access or magnetic tape volume is to be used. The volume is empty in the sense that it contains no other data sets. It must have been initialized by the system volume initialization utility routines.

When a FRESH volume is requested, the system writes a message instructing the operator to mount an empty volume and to suspend processing until he indicates this has been done. If the volume is labeled, the system checks to ensure that it does not contain an active data set. The message to the operator indicates which unit the FRESH volume should be mounted on.

The system does not recognize the volume identification code of a FRESH volume until the end of the job step. Therefore, if it is necessary to refer to such a volume in a subsequent job control statement, the reference must be to a data set on the volume, as in the following example:

```
// ALLOC DATAONE,0C1=FRESH,25

// LABEL 720

// ALLOC DATATWO,SAME=DATAONE,50

// LABEL 360
```

The first ALLOC statement creates a data set on a FRESH direct access volume. To create another data set on the same volume, the programmer identifies the volume by citing the name of the first data set.

The same technique would be used to create a directoried data set on a FRESH volume and then write a member in it in the same job step. The data set is created by an ALLOC statement, and then a member is created by using an ACCESS statement that cites the data set name in its volume identification field.

devadr

This field specifies the address of a device that is to contain the new data set.

The address is stated in hexadecimal as cuu, where c is the channel address and uu is the address of a device attached to that channel.

c = 0 for the standard multiplex channel
c = 1 or 2 for one of the optional high speed multiplex channels
uu = 00 to FE (0 to 254 in hexadecimal).

data length

This field specifies the total number of blocks to be allocated for a direct access data set.

The maximum amount of space depends, of course, upon the amount of available space remaining in the volume. In no case, however, can more than 65,535 blocks be allocated for a single data set.

The total specified in this field does not include the number of blocks required for the directory of a directoried data set. (The block length is specified in the LABEL statement that accompanies an ALLOC statement when a direct access data set is created.)

directory length

This field specifies the number of entries that will be made for a directoried data set. The total includes entries for additional names, if any.

The system determines the amount of space that will be required for the directory based on the number of entries stated. It uses one additional entry for control purposes. The total amount of space is rounded up to an integral number of blocks. Each entry requires 24 bytes.

The maximum number of entries that can be requested is 65,534.

ROUND

The ROUND keyword indicates that allocation of space for a direct access data set is to begin and end on a cylinder boundary. The amount of space allocated will be at least equal to the amount requested in this statement.

FMT

The FMT format keyword causes the system to write sequential blocks containing zeros throughout the area reserved for a direct access data set.

The blocks are the size specified in the LABEL statement for the data set. If the data set is directoried, both the directory and the data area are formatted.

This facility enables a program to write or read any data block within the data set at any time; thereby making non-sequential processing possible.

Assembler language programmers may use the POINT supervisor call to go directly to the proper position for writing or reading any block.

CATLG

The CATLG keyword specifies that an entry for the new data set is to be made in the system catalog at the end of the job. The system will make a catalog entry containing all the information needed for it to find and use the data set when it is referred to in future jobs. The data set may be located thereafter simply by citing its name in an ACCESS statement.

A data set is not cataloged if the job is canceled.

Example:

```
                1                                7
1               6                                2
//SYS002  ALLOC  CHARLEY,                         X
//              1316(NOWRCHK)='PON573',           X
//              240,FMT
```

// are the required job control statement identification characters.

SYS002 assigns symbolic unit SYS002 to the data set that is being created.

ALLOC identifies the operation.

CHARLEY is the name the programmer is assigning to the data set.

1316 indicates that the data set is to be created on a 1316 Disk Pack mounted on a 2311 Disk Storage Drive.

NOWRCHK means that write validity checking is not wanted. (In practice, it is not necessary to specify this unless the default options in the IBM distributed system have been changed. The field could have been written 1316='PON573' with identical effect.)

'PON573' is the identification number of the 1316 disk volume that is to contain the new data set.

240 is the number of blocks to be reserved for the data set. (The size of the blocks is given in a LABEL statement.)

FMT instructs the system to format the disk for non-sequential processing.

Example:

```
//SYSPCH ALLOC OUTPUT,2400=FRESH
```

// are the required job control statement identification characters.

SYSPCH is the name of the symbolic unit to be assigned to the new data set. (In this example, a system unit is being reassigned from its standard card punch assignment to a magnetic tape.)

ALLOC identifies the type of operation.

OUTPUT is the name the programmer is assigning to the new data set.

2400=FRESH indicates that the system is to write the data set on any 2400 magnetic tape volume that does not contain an active data set.

Example:

```
//SYSPCH ALLOC OUTPUT,1442=
```

where all fields are the same as in the above example except that the 1442= parameter assigns SYSPCH to a card read-punch.

Creating Data Sets With /*

The /* statement can be used to create one or more data sets on disk without executing a full program. This is illustrated in the following example:

```
// JOB
// ALLOC DS,SDSD='VOL1',10
// LABEL 726
/*
/&
```

These statements cause the system to create a data set, named DS, on disk volume VOL1. Enough space is reserved for a ten block data set with 726 bytes per block.

ACCESS Statement

An ACCESS statement is used for the following:

1. To identify an existing data set for use in a program.

2. To add new members to an existing directoried data set.

3. To assign a symbolic unit to a data set or directoried data set member.

4. To effect a dummy symbolic unit assignment for program testing.

No ACCESS statement is needed to use a standard system data set unless reassignment of a symbolic unit is required. Also, none is needed to use a data set that was used previously in the same job if a symbolic unit is still assigned to it.

When the IGN parameter is specified in the operand field, the symbolic unit is not physically assigned to an actual data set. The unit's index number may be used, however, in a program's input/output coding to determine, for example, whether the requests are properly coded.

It is not necessary to use an ACCESS statement to refer to a system data set unless the standard assignment of the symbolic unit assigned to it has been or must be altered. For example, symbolic unit SYSIPT is usually assigned to the system data set SDSIPT. This can be altered by an ALLOC or ACCESS statement that assigns SYSIPT to another data set. The unit can be reassigned by another ALLOC or ACCESS statement or returned to its standard assignment by a RESET statement. Or a programmer can use another ALLOC or ACCESS statement to assign another unit to the SDSIPT data set temporarily. In any case, all system units revert to their standard assignments at the end of the job.

The possible formats of the ACCESS statement and brief descriptions of its fields are shown in Figure 6.

The parameters are as follows:

SYSxxx
Use of this field assigns a symbolic unit to the data set or member whose name is given later in the statement. A symbolic unit name, such as SYSOPT or SYS002, is indicated. Input/output operations on the data set or member are initiated by referring to this unit in the coding that requests the operations.

This field is not required if the data set will be referred to only in the Job Control maintenance statements, DELETE, CONDENSE, and RENAME.

More than one symbolic unit may be assigned to a single data set through the use of additional ACCESS statements. This might be done if the program refers to more than one member of a single directoried data set.

System units that are specially assigned in this way revert to their standard system assignments at the end of the job or when they are cited in a RESET statement, whichever comes earlier. They also can be reassigned by subsequent ACCESS and ALLOC statements.

dsname
This field specifies the name of the desired data set.

No further identification is needed for system data sets, such as SDSPCH; cataloged data sets; or those for which an ALLOC or ACCESS statement was processed previously in the program. For others, the location must be specified in the volume field.

The ACCESS statement accepts names of up to 44 characters for data sets created under other System/360 programming systems.

member name
This field specifies:

1. A name of an existing member of the directoried data set whose name was cited in the dsname field, or

2. The name or names that are to be assigned to a new member that will be written during this job step.

When the name of an existing member is cited, all input/output operations for the symbolic unit assigned to this data set are treated as references to the named member. The member cannot be enlarged, but data within it can be manipulated freely or replaced.

Formats

For direct access data sets:

```
//[SYSxxx] ACCESS dsname[(member name[,member name,...])][,volume][,EXT]
                                                                  [,NEW]
```

For magnetic tape or unit record data sets:

```
//SYSxxx ACCESS dsname[,volume][,EXT]
```

For program testing:

```
//SYSxxx ACCESS IGN
```

Fields in brief

//	Required. Identifies job control statement.
SYSxxx	Assigns symbolic unit to the data set or member. Not required for certain maintenance functions.
ACCESS	Required. Identifies type of statement. Must be preceded and followed by at least one blank.
dsname	Identifies a data set that is to be used in the program.
member name	Identifies an existing directoried data set member that is to be used in a program. Alternatively, this field is used to create and name a new member in an existing directoried data set.
volume	Indicates location of the desired data set. Not required for system data sets (listed in Table 1) or for data sets with entries in the system catalog. Also not required if location was specified in previous ALLOC or ACCESS statement in the same job.
EXT	To position an input data set after its last block.
NEW	For directoried data sets only. Indicates that a new member is being created.
IGN	Indicates that no physical input/output operations are to be performed by the unit specified in the SYSxxx field. Permits testing programs without actually reading or writing data.

Figure 6. ACCESS Statement

If a new member is being added to the directoried data set, the NEW parameter must be specified elsewhere in the ACCESS statement. Only one new member may be added to any given directoried data set during a single job step. The new member cannot be added if it would increase the allocated size of the data set or if the directory is full.

The system assumes that anything written during the job step on the symbolic unit assigned to this data set is part of the new member. Once written, the size of a member cannot be changed; that is, its size is fixed at the end of the job step.

An entry is made in the directory for each name. Multiple entries for a member are identical except for the name. The member may then be referred to by any of the names with identical effect. Any of the names may be changed or deleted later without affecting the member as long as at least one entry for the member remains in the directory.

Additional names, if any, must be specified at the time the member is

created. They cannot be added later. A name cannot duplicate any other name in the directory.

The only limit to the number of names a member may have is that the total number of entries in the directory may not exceed the number specified in the ALLOC statement that created the directoried data set.

volume

This field specifies the location of the volume or physical input/output device containing the data set.

This field is not required if the data set is entered in the system catalog, if it has been referred to in a previous ALLOC or ACCESS statement in the job, or if it is a system data set.

The format of this field is:

```
┌──────────────────────────────────┐
│                                  │
│SAME[(options)]=(dsname           │
│                (SYSxxx           │
│                                  │
│devadr)[(options)]=['volid']      │
│type  (                           │
└──────────────────────────────────┘
```

The parameters have the same meaning and are specified in the same way as for the ALLOC statement. An exception is that the FRESH option does not apply to ACCESS.

Volume information retrieved from the system catalog is treated as a type request. The system selects a device and tells the operator to mount the required volume on it.

EXT

The EXT parameter positions an input data set immediately after its last block of data.

When this parameter is not specified, a data set is positioned just before its first data block when the system passes control to the problem program. EXT may be used to save time repositioning a data set for updating.

EXT cannot be used with a directoried data set.

The action of EXT is equivalent to positioning at the end of a data set when it is used with a magnetic tape or with a direct access data set that is full or has been formatted with the FMT parameter.

When used with a 2315 disk cartridge data set, positioning is immediately before an end of file mark.

No action is taken for an EXT parameter until a data set is opened. If EXT has been specified, the data set is then positioned accordingly.

EXT should not be used with an empty data set. If specified for an empty tape, for example, the system is unable to find a block of data and the result is a runaway tape.

NEW

The NEW keyword indicates that a new member is being added to a directoried data set.

This keyword must be used whenever a new member is being created.

IGN

The IGN keyword indicates that no physical input/output operations are to be performed on the symbolic unit.

This facility is designed for use in testing new programs.

Write requests referring to the symbolic unit cited in the name field of this ACCESS statement are ignored. Read requests generate an end-of-file condition.

Example:

The purpose of this example is to add data to an existing direct access data set.

```
//SYS002 ACCESS SET1,0A2='TRW749',EXT
```

SYS002 is the symbolic unit being assigned to the data set.

SET1 is the name of the data set.

0A2 is the physical address of the direct access device on which the volume containing the data set is to be mounted.

TRW749 is the identification number of the volume containing the data set.

EXT instructs the system to position the data set after the last block of data currently in it.

Example:

The purpose of this example is to add a new member to a directoried data set.

```
                1                               7
1               6                               2
//SYS001  ACCESS  SET4(MEMBERA,MEMBERB),   X
//               0A2='TRW749',NEW
```

SYS001 is the symbolic unit to be assigned to the data set.

SET4 is the name of the directoried data set in which the new member is to be written.

MEMBERA and MEMBERB are two names that are to be assigned to the new member.

0A2=TRW749 identifies the volume and location, as in the previous example.

NEW indicates that a new member is being created.

Example:

The purpose is to assign a symbolic unit to a magnetic tape data set.

```
//SYS002 ACCESS SET6,2400=
```

SYS002 is the symbolic unit to be assigned to the data set.

SET6 is the name of the data set.

2400 indicates a nine track tape with density of 800 bits per inch.

It is assumed that the tape contains no labels. Since no location is specified, the system will select a tape unit from its list of those available and instruct the operator where to mount the tape containing data set SET6.

LABEL Statement

The LABEL statement is used with the ALLOC and ACCESS statements to provide additional information about a data set.

A LABEL statement is required:

--Immediately after an ALLOC statement for a direct access data set.

--Immediately after an ALLOC or an ACCESS statement for a magnetic tape data set when the tape contains volume and data set labels and the installation standard is to label tape data sets.

--To indicate that control characters are being used in unit record operations.

A LABEL statement must not be used:

--After an ACCESS statement for a direct access data set.

--For an unlabeled magnetic tape data set in an installation where the standard is to label tape data sets.

Use of a LABEL statement after an ALLOC or ACCESS statement is optional at other times. Its effect in such cases is to provide information that is inserted in a data set's file control block. The system uses file control blocks in handling its input/output operations. The system fills in those portions of the blocks that it requires for these operations. A problem programmer can ignore the existence of file control blocks, but he can refer to the information in them if he wishes. The formats for these blocks are shown in Appendix A.

Certain fields of the LABEL statement are not used by the Model 44 Programming System. They are included to make data sets compatible with the operating systems for other System/360 models. Such fields are identified in the text.

The possible formats of the LABEL statement and brief descriptions of its fields are shown in Figure 7.

The fields of the LABEL statement are as follows:

block length
: This field specifies the block length in bytes.

  The maximum value depends upon the device being used. In no case can it exceed 65,535 bytes.

  The maximum block size for FORTRAN programs is 360 bytes except for direct access data sets. For these, the block length should be the same as the record length specified for the data set in the program's DEFINE FILE statement.

  All blocks within a direct access data set must be the same size. Block lengths may differ between different data sets. The maximum block length for a directoried data set is 726 bytes. The minimum is 24 bytes.

Formats

For direct access data sets:

```
// LABEL block length[,expiration date][,'ds type'][,CTLCHR ][,WRCHK  ] [,RECLEN=n]
                                                   [,CTLASA] [,NOWRCHK]

                                                                [,KEYLEN=n][,KEYLOC=n]
```

For magnetic tape and unit record data sets:

```
// LABEL [block length][,expiration date] [,CTLCHR ][,RECLEN=n]
                                          [,CTLASA]
```

When information is the same as that for another data set in same job:

```
// LABEL SAME={dsname }
              {SYSxxx}
```

Fields in brief

//	Required. Identifies job control statement.
LABEL	Required. Identifies type of statement. Must be preceded and followed by at least one blank.
block length	Specifies number of bytes in each data block. For variable length blocks on magnetic tape, specifies number of bytes in largest block.
expiration date	Indicates when data set may be deleted. If not specified, current date is assumed.
ds type	Indicates organization of data set. If not specified, sequential organization is assumed. See text for code numbers.
CTLCHR	Indicates first byte of each record block contains special channel command code character.
CTLASA	Indicates first byte of each record block contains ASA character for manipulating printer or unit record device.
WRCHK NOWRCHK	Indicates whether write validity checking is wanted. If neither is specified, NOWRCHK is assumed.
RECLEN=n	Specifies size of logical records. For variable length records, specifies size of largest logical record.
KEYLEN=n	Specifies number of bytes in key field of direct access data blocks.
KEYLOC=n	Specifies location of first byte of key field within each block.
SAME	Indicates that label information for this data set is the same as that for another data set for which an ALLOC or ACCESS statement was processed previously in the same job.
dsname	Specifies name of data set that has label information that may be copied for this data set.
SYSxxx	Specifies a symbolic unit that has been assigned to a data set that has label information that may be copied for this data set.

Figure 7. LABEL Statement

This field must be specified for disk data sets. If specified for any data set, the information is entered in the data set's file control block.

For variable length blocks on magnetic tape, the field should indicate the size of the largest block.

For compatibility with other System/360 operating systems, the block length must be an integral multiple of the logical record length for fixed-length records. For variable length records, which produce variable length blocks, the maximum block size should be specified.

expiration date
This field indicates the year and day when the data set may be deleted.

This date is specified as yyddd, where yy is the year (00 through 99) and ddd is the day (001 through 366).

If this field is omitted, the current date is used.

The system's utility programs can be used to obtain a listing of all expired data sets in a volume. These data sets can then be eliminated from the volume, or their contents can be written into new data sets with new expiration dates.

The system examines expiration dates when a direct access volume is processed by the SQUEEZE utility routine. The space occupied by expired data sets is treated as vacant.

ds type
This field specifies the data set type for inclusion in the format 1 direct access label.

This field is included for compatibility with other System/360 programming systems only and is not used by the Model 44 Programming System, which supports only sequential organization.

One of the following entries may be used:

Hexadecimal Code	Description
'2000'	Direct access organization
'4000'	Sequential organization
'8000'	Indexed sequential organization
'0200'	Library organization
'0000'	Organization not defined

If this field is omitted, '4000' for sequential organization is assumed.

CTLCHR
This parameter indicates that the first byte of each block of records is an input/output control character.

If this parameter is specified, the system uses the first byte of each block when writing unit record or printer data sets. The control character is treated as data when written in disk or tape data sets and when this parameter is not specified. If specified when writing in disk or tape, the presence of a control character is noted in the data set label.

CTLASA
The CTLASA parameter indicates that the first byte of each block of records contains an ASA control character for physical manipulation of printers and unit record devices.

The control character must be one of those listed in Table 10 in the input/output section of this publication.

When a data set is being written on a printer or unit record device, the control character is used to control the printer or for stacker selection. When the data set is being written on disk, the presence of a control character is noted in its label.

The control character is treated as data except when the data set is being written on a printer or unit record device and the CTLASA parameter has been specified.

WRCHK
NOWRCHK
The WRCHK keyword specifies that write validity checking is to be performed on the data set as it is written.

When write validity checking is requested, the system reads back each block that is written in a data set. No data is transferred, but the system checks for errors. Standard error recovery procedures are executed when necessary. Use of this facility requires an extra disk revolution for each block that is written.

This field is used only with data sets that are to be written on a disk volume.

NOWRCHK indicates that no write validity checking is desired.

When either WRCHK or NOWRCHK is specified in a LABEL statement, it governs all subsequent writing operations for the data set. A programmer can override this LABEL option for a single run by using the WRCHK-NOWRCHK option in the ALLOC or ACCESS statement for the data set.

If this option is not specified in a LABEL, ACCESS, or ALLOC statement, the system default option, NOWRCHK, is assumed.

RECLEN=n

This field specifies the size of the data set's logical records.

The maximum record length is 65,535 bytes.

For data sets with fixed-length records in fixed-length blocks, n states the number of bytes in each logical record.

For variable-length tape records, n states the number of bytes in the largest logical record.

This information is entered into the format 1 direct access label and into the file control block used for input/output operations. A program may obtain the n value from the file control block for use in blocking and unblocking.

If this field is not specified, the system assumes record length equals block length.

KEYLEN=n

This field specifies the size of the key field for blocks written on direct access volumes. The size is given as n bytes, where n cannot exceed 255.

A programmer may select any portion of a block for use as a key. This field and the KEYLOC=n field indicate the location and length of this key information.

The specified bytes are written as the key field in the direct access record block.

If this field is used, the key length is entered in the format 1 direct access label. The Model 44 Programming System uses this field only when a data set is being written on a 1316 disk pack. For keys to be written, this field must be followed by a KEYLOC field with n equal to 0.

KEYLOC=n

This field specifies the location of the key field within each block.

The value of n is the index of the first byte of the key field. That is, if the key field starts with the 25th byte, KEYLOC=25 is specified. The value of n cannot exceed 65,535.

If n equals 0 and the KEYLEN field specifies a non-zero value, keys are written for data sets on a 1316 Disk Pack. In any case, the KEYLOC value is entered in the format 1 direct access label.

SAME={dsname / SYSxxx}

Use of this field indicates that label information for this data set is the same as for another data set. This instructs the system to fill the LABEL statement fields for this data set with information from the other data set's label.

This can be done only if an ALLOC or ACCESS statement for the other data set was processed previously in the same job or the other data set is a system data set. In the case of tape, the referenced ALLOC or ACCESS statement must have been followed by a LABEL statement.

The other data set may be identified either by name or by the name of the symbolic unit, such as SYS002, that is currently assigned to it.

Example 1:

The purpose is to provide maximum label information for a new direct access data set.

```
                1                                7
1               6                                2
//  LABEL  240,67334,'4000',CTLASA,              X
//             NOWRCHK,RECLEN=80,                X
//             KEYLEN=10,KEYLOC=0
```

240 is the number of bytes in each record block.

67334 is the expiration date, the 334th day of 1967.

'4000' indicates a sequential data set.

CTLASA signifies that the first character in each record block is an ASA control character for manipulation of printers or

unit record devices when the data set is written.

NOWRCHK means that write validity checking of data written in the data set is not wanted.

RECLEN=80 specifies a logical record length of 80 bytes. There are three logical records in each data block.

KEYLEN=10 instructs the system to write a 10-byte key field when writing each data block.

KEYLOC=0 indicates that the key field starts at the beginning of each data block.

Example:

The purpose is to provide minimum label information for a direct access data set.

```
// LABEL 240
```

where 240 is the number of bytes in each block of data.

Example:

The purpose is to indicate that the first byte of each block in a data set that is to be written on a unit record device or printer contains a control character.

```
// LABEL ,CTLASA
```

SYMBOLIC UNIT MAINTENANCE STATEMENTS

The symbolic unit maintenance statements are LISTIO and RESET. They are used in conjunction with the data set definition statements which may alter the assignments of system units. LISTIO provides a listing of current assignments. RESET restores system units to their standard assignments if their assignments have been changed by ALLOC or ACCESS statements.

RESET Statement

The RESET statement is used to restore one or more symbolic units to their standard assignments.

The statement is used when an assignment has been altered by an ALLOC or ACCESS statement in a previous job step. This statement applies only to those units listed in Table 1.

One RESET statement may be used to restore all system units or just one. If more than one is to be restored, but not all, a separate RESET statement is required for each unit.

The standard assignment is the one that was specified when the system was generated or one made by the operator. A listing of these assignments can be obtained with the LISTIO statement.

Regardless of whether RESET statements are used, all system units are restored to their standard assignments at the end of the job.

The RESET statement's format is:

```
// RESET [SYSxxx]
```

SYSxxx
: This field specifies the system unit that is to be restored to its standard assignment.

  If this field is left blank, all system units are restored to their standard assignments.

  The statement is ignored if this field specifies a system unit currently performing its standard assignment.

LISTIO Statement

The LISTIO statement is used to obtain a listing of current symbolic unit assignments.

Three types of listings may be obtained. The first reports the current assignment of a single unit. The second lists all assignments made or altered in the current job. The third provides a listing for all system units that have assignments.

The listing is written on SYSLOG and SYSLST. It includes the symbolic unit's name, its current channel and unit address, the volume identification serial number of the volume it is assigned to, and the name of the data set to which the symbolic unit is assigned.

The statement's format is:

```
// LISTIO [PROG  ]
          [SYSxxx]
```

If neither PROG nor SYSxxx is specified, the current assignments of all system units are listed.

PROG
: The PROG keyword causes a listing of all symbolic unit assignments made through execution of ALLOC and ACCESS statements in the current job.

  This listing does not include those units that have been restored to their standard assignments by RESET statements or those whose status has not been altered.

SYSxxx
: This field specifies a system unit name. It requests a listing for that particular unit.

DATA SET MAINTENANCE STATEMENTS

The five data set maintenance statements, DELETE, CONDENSE, RENAME, CATLG, and UNCATLG, specify actions to be taken on or for a data set.

DELETE and CONDENSE are used to eliminate data sets and directoried data set members that are no longer needed. RENAME changes the name of a data set or member. CATLG and UNCATLG are used to make and remove entries in the system catalog.

Any data set referred to in DELETE, CONDENSE, or RENAME statements must have been cited in ALLOC or ACCESS statements previously in the job.

DELETE Statement

A DELETE statement is used to eliminate a data set or a directoried data set member.

When a directoried data set member has more than one entry in the directory, this statement can be used to remove one or more of the entries. The member remains active as long as it is represented by at least one entry in the directory.

When an entire data set is deleted, the system removes its entry from the volume table of contents, updates the volume's format 5 space management label to reflect the removal, and, if applicable, removes the data set's entry from the system catalog.

The data set is not physically altered at this point. It cannot be referred to, however, and the system treats the space it occupies as vacant. The same applies to a directoried data set member when all its entries have been removed from the directory.

The space occupied by a deleted data set can be assigned to a new data set. Eventually, however, the free space tends to become fragmented into many small areas. When this happens, the volume should be processed by the SQUEEZE utility routine if it is a 2315 disk. This routine shifts all active data sets toward the beginning of the volume, filling any gaps left by deleted data sets. The space at the end of the volume is then available for reassignment.

The space occupied by a deleted member within a directoried data set cannot be reassigned. The CONDENSE job control statement is used to shift active members toward the beginning of a directoried data set. New members can then be added at the end.

An ALLOC or ACCESS statement must have been processed previously in the job for any data set cited in a DELETE statement. An exception is that entire system data sets, but not individual members, can be deleted at any time.

A separate DELETE statement is required for each data set to be deleted. Any number of directoried data set members can be deleted with a single DELETE statement as long as all are members of the same directoried data set.

The statement's format is:

```
// DELETE dsname[(member name,...)]
```

dsname
: This field specifies the name of a data set that is to be deleted or of a directoried data set that contains one or more members that are to be deleted.

member name
: This field specifies a name or alternate name of a directoried data set member. The name is deleted from the directory. Other entries for the same member are not disturbed unless they also are listed in this statement.

Any number of names may be listed, but all must apply to members of the same data set. If the list includes a name that is not in the directory, a warning message is written on SYSLST and the name is otherwise ignored.

CONDENSE Statement

The CONDENSE statement is used to shift the contents of a directoried data set to fill space occupied by members and directory entries that have been deleted.

The space occupied by deleted members and entries is treated as though it were empty. Other members and entries are shifted toward the beginning of the data set to fill this space. The total size of the data set or directory is not changed. Also, there is no change in the order in which the remaining members and entries appear.

After the condensing, all empty space is at the end of the directory and at the end of the data set. New members may be added, and new entries may be made in the directory.

Any data set cited in a CONDENSE statement must have been referred to in an ALLOC or ACCESS statement previously in the program.

The statement's format is:

```
// CONDENSE dsname
```

dsname
: This field specifies the name of a directoried data set that is to be condensed.

RENAME Statement

The RENAME statement is used to change the name of a data set or directoried data set member.

When a data set is renamed, the name is changed in the volume table of contents and, if applicable, in the system catalog.

The name of a directoried data set member is changed in the directory. Other names, if any, are not affected.

If the new name duplicates an existing name in the system catalog, volume table of contents, or data set directory, the operation is not performed, and an error message is written on SYSLST.

An ACCESS or ALLOC statement for this data set must have been executed previously in the job.

The statement's format is shown in Figure 8.

The parameters are:

old dsname
: This field specifies a data set name that is to be changed.

  This field may contain up to 44 characters. It can be used to change the names of data sets created under other System/360 programming systems to names of eight characters or less for use with the Model 44 system.

new dsname
: This field specifies the new name for the data set.

dsname
: This field specifies the name of a directoried data set containing a member whose name is to be changed.

old member name
: This field specifies the directoried data set member name that is to be changed.

new member name
: This field specifies the new name for the directoried data set member.

CATLG Statement

The CATLG statement is used to make an entry for a data set in the system catalog.

```
// RENAME   {old dsname,new dsname                          }
            {dsname(old member name,new member name)}
```

Figure 8. RENAME Statement

A cataloged data set may be referred to in most cases just by name, without any need for stating its location. Catalog entries are retained until specifically deleted by an UNCATLG statement or until the data set is deleted.

Catalog entries also may be made through use of the CATLG field of the ALLOC statement that creates a data set.

If the data set name specified in a CATLG statement duplicates a name already in the catalog, the operation is rejected and a message is written on SYSLST.

The statement's format is:

```
// CATLG dsname[,type[(options)]='volid']
```

dsname
: This field specifies the data set name that is to be entered in the system catalog.

type(options)='volid'
: This field identifies the volume on which the data set is located.

  This field may be omitted if an ALLOC or ACCESS statement for the data set has been processed previously in the job.

  Entries in this field are specified in the same way as for the type [(options)]='volid' subfield of the volume field of the ALLOC and ACCESS statements.

UNCATLG Statement

The UNCATLG statement is used to delete a data set entry from the system catalog.

Removal of the catalog entry does not change the data set itself or the volume containing it. Its entry in the volume table of contents also is not affected.

The statement's format is:

```
// UNCATLG dsname
```

dsname
: This field specifies the name of the data set whose entry is to be removed from the catalog.

MISCELLANEOUS STATEMENTS

There are four miscellaneous statements: PAUSE, comments, REWIND, and UNLOAD.

PAUSE Statement

The PAUSE statement is used between job steps to suspend processing of control statements until the operator takes some action.

The statement has an optional comments field that may be used to write a message to the operator. This message is printed on SYSLOG, which usually is the Console Printer-Keyboard, and processing is suspended. When the operator is ready to resume processing, he types a code on the Console Printer-Keyboard.

The operator may use this time for various manual actions, such as mounting or switching volumes, or, possibly, cancelling a job.

The statement's format is:

```
// PAUSE [comments]
```

comments
: Any entry in this field is printed as a message to the operator on SYSLOG just before the system suspends processing.

Comments Statement

The comments statement is used for communication with the operator.

The statement starts with an asterisk * in its first position. The second position must be blank. The remainder of the statement, up to column 72, may contain any characters, including blanks.

When such statements are included among control statements, they are written on SYSLOG as they are encountered. They have no effect on the processing of control statements or execution of the program.

The statement's format is:

```
* [any comments]
```

REWIND Statement

The REWIND statement is used to rewind a magnetic tape to its load point or to reset the block count of a direct access or unit record data set to zero.

The direct access block count indicates the current position of the data set. For a unit record data set, this record indicates the number of blocks that have been read or written for that data set up to that point in the program.

If REWIND is requested for a symbolic unit assigned to a data set that has not been closed, job control attempts to close the data set before executing the REWIND.

The statement's format is:

```
// REWIND SYSxxx
```

SYSxxx
: This field specifies the symbolic unit on which the REWIND function is to be performed.

UNLOAD Statement

The UNLOAD statement is used to rewind and unload a magnetic tape volume. It also can be used with direct access and unit record data sets to reset their block counts to zero and to logically disconnect the symbolic units assigned to them.

Direct access and unit record data sets are disconnected in the sense that the system refuses to accept input/output operations for the symbolic units assigned to them for the rest of the job.

If UNLOAD is requested for a symbolic unit assigned to a data set that has not been closed, job control attempts to close the data set before executing the UNLOAD.

The statement's format is:

```
// UNLOAD SYSxxx
```

SYSxxx
: This field specifies the symbolic unit for which the UNLOAD function is to be performed.

```
1               1                                                          7
                6                                                          2

//JOB1 JOB , DEFINE START OF JOB

//MODULE  EXEC  ASSEMBLE(LINK)     ASSEMBLE MODULE AND ASSIGN THE       X
//              NAME "MODULE" TO IT

assembler input statements

/*

//        EXEC  LNKEDT             LINKAGE EDIT THE MODULE THAT WAS     X
//              JUST ASSEMBLED

linkage editor control statements, if any

/*

//SYS001  ALLOC  DATAONE,1316='VOL835',100     CREATE A NEW DATA SET

//        LABEL  360               DEFINE BLOCK LENGTH OF NEW DATA SET

//SYS002  ACCESS  DATATWO(MEMBER),  ACCESS OLD DATA SET SO NEW MEMBER   X
//                        SAME=SYS001,NEW     CAN BE ADDED

//        EXEC                       EXECUTE THE PROBLEM PROGRAM

/&
```

Figure 9. Sample Job Deck

The Model 44 Programming System includes two language processors: an assembler program for assembler language coding and a compiler for FORTRAN IV coding. Detailed information about writing programs in these languages is provided in the previously cited language publications. This section deals with additional information required to assemble or compile programs that have already been written.

The assembler language portion of this section discusses the job control statements required to initiate an assembly job step, the assembler's updating capabilities, and the EXEC statement parameters that may be invoked for an assembly job step.

The FORTRAN portion covers EXEC statement parameters for a compilation job step. Detailed information about the FORTRAN compiler is available in the Guide to System Use for FORTRAN Programmers, cited in the Preface. It covers source program diagnostics, compiler limitations, special considerations for constructing multiphase programs, linking to assembler language routines, etc.

ASSEMBLER PROGRAM

The system's assembler program is loaded and entered with an EXEC ASSEMBLE statement, as follows:

```
//NAME1  EXEC  ASSEMBLE[(parameter list)]
                        [accounting information]
```

// identifies it as a job control statement.

NAME1 is the name to be assigned to the assembled module.

ASSEMBLE is the name of the program phase to be loaded and entered.

Parameter list represents a series of parameters that may be specified. The parameters that can be used are discussed individually later in this section.

The module name, NAME1 in this example, is used to refer to the assembled module, if necessary, after it is assembled. This name is entered in the SDSPSD data set, which serves as a directory to SDS000 during linkage editing. This name is used to refer to this assembled module in a linkage editor control statement.

ASSEMBLER EXEC PARAMETERS

There are three groups of parameters that can be specified in the parameter list field of an EXEC statement initiating an assembly job step. They are listed in Table 5 and discussed in the following text. Up to six may be used.

These parameters indicate options that the programmer wants to invoke for the assembly. Some apply only when the update feature is being used. Others can apply to any assembly.

In this list, default options are underlined. These are options that the system assumes when no other option in the same category is specified.

Table 5. Assembler EXEC Parameters

Output Option Parameters		Update Parameters	Space Allocation Parameters
LINK	NOLINK	ASSEMBLE	SYMBMAX
DECK	NODECK	UPDASMB1	SYMBMIN
LIST	NOLIST LISTC	UPDASMB2	SYMBn
XREF	NOXREF	UPDASMB3	
LNCTn		UPDATE1	
		UPDATE2	

Output Option Parameters

The following parameters apply to any assembly, regardless of whether updating is involved. Generally, they indicate which types of assembler output are wanted.

LINK
NOLINK

Use of this parameter indicates whether the assembler output is to be linkage edited later in the same job.

LINK causes the system to write assembler output on SYS000 for linkage editing later in the same job. When NOLINK is specified, no linkage data set is created.

If neither LINK nor NOLINK is specified, NOLINK is assumed.

In any case, NOLINK is assumed if the assembler program detects errors that would prohibit successful execution. If LINK is specified and the name field of the EXEC ASSEMBLE statement is blank, the linkage editor rejects the assembled module.

All programs must be processed by the linkage editor before they can be executed.

DECK
NODECK

This parameter specifies whether assembler output should be written on SYSPCH for punching into cards.

DECK causes the system to write a copy of the assembled output module on SYSPCH, which may be assigned to either a magnetic tape or card punch data set. The deck can be used subsequently as input for the linkage editor.

If neither DECK nor NODECK is specified, DECK is assumed.

LIST
NOLIST
LISTC

This parameter indicates whether a listing of the assembly is wanted.

LIST causes the system to write a listing of the assembled output module on SYSOPT. Error flags, if any, are printed to the left of the statements causing them.

LISTC provides the same type of listing as LIST, but source input statements that are conditionally not assembled are listed as comments.

LIST is assumed when neither LIST, NOLIST, nor LISTC is specified.

XREF
NOXREF

This parameter specifies whether a listing of symbolic cross references is wanted.

This listing contains all normal symbolic names used in a source program and identifies all points in the program where the symbols appear. The listing excludes variable symbols and sequence symbols.

If XREF is specified, this cross reference listing is written on SYSOPT. If NOXREF is specified, no cross reference listing is provided.

If neither parameter is specified, XREF is assumed.

LNCTn

This option specifies the maximum number of text lines to be printed on each page of an assembly listing. The range is from 1 to 99. If nothing is specified as n, 61 is assumed.

As an example of using the output option parameters, consider a job in which a program is to be assembled, linkage edited, and executed in three successive job steps. The EXEC statement for the assembly job step could read as follows:

```
//NAME1 EXEC ASSEMBLE(LINK,NODECK)
```

This statement causes the system to load and to begin executing the assembler program. This program reads the programmer's assembler language input statements from SYSIPT and produces an assembled output module. The module is named NAME1, and it may be referred to by this name in linkage editor control statements, if any are needed.

Since the LINK parameter is specified, the assembled output module is written on SYS000 where it will be readily available for the linkage editor in the next job step. The LIST and XREF parameters are not specified but are assumed, so a listing and a symbolic cross reference directory are written on SYSOPT. No deck is punched.

Update Parameters

The assembler's update feature enables it to modify a source program and to assemble it during the same job step. It can accept a source language data set and an edit data set as input and produce an assembled module and a copy of the updated source data set.

The assembler language publication contains information about the update statements and how they are used to modify a program.

The update parameters instruct the assembler about updating required and what is to be done with the updated program. These may be specified in addition to the output option parameters, described previously, which instruct the system about how to treat the assembler output.

When a source program data set is to be updated and assembled, the assembler program first performs the required updating. It then assembles the updated program.

Update Listing

During an update operation, the assembler produces a listing on SYSLST showing the results of its processing. All control statements are listed as they are processed. Any errors that are detected are indicated. Finally, all symbolic statements that are inserted into or deleted from the source program deck are listed with a notation that they were inserted, replaced, or deleted.

Magnetic Tape Operations

When magnetic tape data sets are being used in an updating operation, the old source program data set on SYS002 and the updated source program data set on SYS003 are not rewound. At the end of each updating job step, a tape mark is written on the SYS003 output data set and the data set is repositioned to a point just preceding this tape mark. A subsequent updating writes over this mark. In this way, an updated program data set created in multiple job steps contains only one tape mark.

When updating is completed, these units can be rewound with the job control statements REWIND and/or UNLOAD.

Options

One of six update parameters can be specified in an EXEC statement. The parameters are ASSEMBLE for an assembly with no updating; UPDASMB1, UPDASMB2, and UPDASMB3 for updating and assembling; and UPDATE1 and UPDATE2 for updating with no assembling. If none of these is specified, ASSEMBLE is assumed.

The update parameters may be used to reserialize a program deck even when no other editing is wanted.

Table 6 illustrates the symbolic units that are used for each parameter.

Individually, the parameters are as follows:

ASSEMBLE
: This parameter is used for a regular assembly. No updating is performed. The source program data set must be on SYSIPT. This parameter is assumed when none of the other update and assemble parameters is specified.

UPDASMB1
: An updating operation is performed with an old source program data set on SYS002 and an edit data set on SYSIPT. The updated program is assembled. An updated source program data set is written on SYS003.

UPDASMB2
: This operation has only one input, a source program data set on SYSIPT. The program is assembled. A copy of the source program data set, serialized, if requested, is written on SYS003.

UPDASMB3
: An updating operation is performed with an old source program data set on SYS002 and an edit data set on SYSIPT. The updated program is assembled. The only difference between this and the UPDASMB1 operation is that the assembler does not provide a copy of the updated source program.

UPDATE1
: An updating operation is performed with an old source program data set on SYS002 and an edit data set on SYSIPT. An updated source program data set is written on SYS003. The program is not assembled.

UPDATE2
: When this parameter is specified, an input source program data set on SYSIPT is written on SYS003. The program is not assembled. This parameter may be used to copy a program data set and to serialize it.

Table 6. Update Parameters

Option	SYSIPT Contains	SYS002 Contains	Assembly	SYS003 Contains
ASSEMBLE	Source program	(Not used)	Yes	(Not used)
UPDASMB1	Edit data set	Old data set	Yes	New data set
UPDASMB2	Old data set	(Not used)	Yes	New data set
UPDASMB3	Edit data set	Old data set	Yes	(Not used)
UPDATE1	Edit data set	Old data set	No	New data set
UPDATE2	Old data set	(Not used)	No	New data set

For example, to update, duplicate, and assemble a program that is to be linkage edited later in the same job, the following EXEC statement could be used:

```
//NAME1 EXEC ASSEMBLE(LINK,NODECK,UPDASMB1)
```

With this statement, the original program deck would be read and updated by the edit statements. An updated source program deck would be written on SYS003, and an assembled program module would be written on SYS000. An assembly listing and cross reference listing would be written on SYSOPT.

Space Allocation Parameters

The space allocation parameters enable a programmer to specify priorities for allocation of main storage space during an assembly.

In a sense, there are three contenders for available main storage space during an assembly. They are:

1. The update program.
2. The assembler's symbol table.
3. Input/output buffers.

Each of these requires a certain minimum amount of main storage space. If the requirement for any one of them can be reduced or eliminated, the others can use the additional space advantageously. If, for example, no updating is required during the assembly, space used by the update program can be used to increase the size of the symbol table or to establish additional input/output buffers.

UPDATE PROGRAM: The update program is loaded into main storage as part of the assembler program when one of the update parameters is specified in an EXEC statement.

Main storage space is required for the update program itself and for the additional data sets that are used in an updating operation. In addition to an input/output buffer, the program requires an 88 byte parameter list for each data set.

When no updating is requested, this space is made available for other uses.

INPUT/OUTPUT BUFFERS: At least one input/output buffer is established for each data set involved in an updating and/or assembly process. The size of a buffer varies according to the block sizes in the data set.

When additional space is available, it is used to establish extra buffers for the data sets. This generally reduces the time required for an assembly.

SYMBOL TABLE: The symbol table contains entries for the symbols and literals in the source program.

One entry is required for each unique normal symbol, each unique variable symbol that appears as the name of a SETA instruction, and each unique literal of eight characters or less. For longer literals,

an additional entry is required for each additional eight characters or portion thereof. (The length of a literal includes the = character at its beginning and the closing quote mark or right parenthesis.)

In addition, the assembler uses two entries for its location counter, one for a dummy variable symbol, and one for each unnamed START, CSECT, or COM in the program.

The amount of space available for the symbol table varies according to the requirements for updating, parameter lists, and input/output buffers. In some cases, the assembler also takes space from the cross reference table, so that this table may appear truncated in listings for programs with an exceptionally large number of symbols and literals. If the assembler runs out of space, it continues, but the output module is unacceptable for linkage editing and execution.

PARAMETERS: The following three EXEC statement parameters enable a programmer to indicate his program's symbol table requirements. If none of these is specified, SYMBMIN is assumed.

SYMBMIN
: When this parameter is specified or assumed, the assembler assigns double buffers to all data sets to obtain optimum assembly speed. Storage space is allocated for buffers and updating, if necessary, and the remaining available space is used for the symbol table. This parameter is assumed when none of these three is specified.

SYMBMAX
: This parameter is used for programs with a large number of symbols and literals. Only one input/output buffer is established for each data set. If no updating parameters are specified, this provides the largest possible amount of space for the symbol table.

SYMBn
: In this parameter, n is the number of entries in the symbol table. It cannot exceed 9999. The programmer estimates the requirements, and the system reserves enough space for a symbol table with the specified number of entries. As much of the remaining space as possible is used for input/output buffers.

  For internal efficiency, n should be five to ten per cent larger than the number of symbol table entries whenever possible. If n is so large that there is not enough room for essential buffers, it is disregarded and SYMBMAX is assumed.

CONTROL STATEMENT EXAMPLE

The task in this example is to update a source program and assemble it. The assembled module is to be linkage edited in the next job step.

The old source program is on magnetic tape volume WES132. The edit statements are in punched cards.

```
//PROG1 JOB

//SYS002 ACCESS ORIGINAL,2400='WES132'

//SYS003 ALLOC NEWSORCE,2400=FRESH

//MOD1 EXEC ASSEMBLE(LINK,NODECK,UPDASMB1)
```

(When SYSRDR and SYSIPT are assigned to the same data set, the edit statements immediately follow the EXEC statements. The edit statements are followed by a /* end of data statement and the control statements for the next job step.)

PROG1 is the name of the job.

ORIGINAL is the name of the old source program data set.

2400='WES132' identifies a magnetic tape with the identification number WES132.

NEWSORCE is the name being assigned to the updated source data set.

2400=FRESH indicates that the new source data set should be written on any empty magnetic tape volume.

This example assumes that the standard assignment of SYSIPT is to the card reader containing the edit statements. It further assumes that the installation does not use data set labels on magnetic tapes. Otherwise, a LABEL statement might be required for the NEWSORCE data set.

It produces an updated program data set on SYS003, an assembled module on SYS000, and a listing and cross reference directory on SYSOPT.

SUPERVISOR CALLS

A supervisor call is a request by a problem program for the system to provide a specific service. The Model 44 Programming

System provides several different types of services that are available for use by a problem program in this manner.

Assembler language programmers call these functions directly. They are supplied by the compiler for FORTRAN programmers.

The system's routines will, for example, perform all input/output services for a program. They may be used to load other programs or program segments from the phase library, store information in the user communication region, or provide special program routines that are entered only when and if certain conditions develop.

The program requests one of these services by executing a supervisor call (SVC) instruction. The assembler instruction SVC 4 causes the system to read a block of data from an input data set. For program clarity, this instruction may be written

```
SVC READ
```

if, elsewhere in the program, there is the assembler instruction

```
READ EQU 4
```

The examples in this publication follow this format.

Execution of an SVC instruction causes an interruption. The system stops executing the problem program and branches to a supervisor routine that examines the SVC operand to determine which service is being requested. It then branches again to one of its specialized routines designed to provide the requested service.

The system retains control until the request is satisfied, except for input/output operations. In the latter case, control returns to the problem program, unless the program has specified otherwise, so that it can continue processing while the input/output operation is in progress. The system automatically regains control when an input/output interruption needs to be processed.

The program, in a sense, never knows that it was interrupted. It picks up exactly where it left off when the interruption occurred, except where specified otherwise in the discussions of the individual supervisor calls elsewhere in this publication. These discussions specify the information the program must provide to enable the system to satisfy the request and they note the changes that are made in the contents of the general registers and in main storage as a result.

The names of the supervisor calls, such as READ, UPSAND, STXITC, are used for reference only. They have no special meaning to the assembler or the system. They have meaning only when equated to a numeric value and used in a supervisor call, as indicated previously.

FORTRAN COMPILER

The following parameters may be specified in the parameter list field of an EXEC job control statement initiating a FORTRAN compilation job step. They are similar to the output option parameters that can be specified by an assembler language programmer, but in some cases the default options differ.

In the following list, the default options are underlined. The system assumes these default options apply when no parameter for the function is specified.

Parameters may be written in any order. They are separated by commas.

DECK
NODECK
: This option is used to have a copy of the compiler output punched into cards.

  When DECK is specified, compiler output is written in card format on symbolic unit SYSPCH. This unit may be assigned to a card punch or magnetic tape data set. This data set may be used subsequently as input for the linkage editor.

  If neither DECK nor NODECK is specified, NODECK is assumed.

SOURCE
NOSOURCE
: This option may be used to obtain or suppress a listing of the source program.

  If SOURCE is specified, the compiler writes a listing of the source program on SYSOPT.

  If NOSOURCE is given, no source language listing is provided.

  If neither is specified, SOURCE is assumed.

LINK
NOLINK
: This option indicates whether the program is to be linkage edited later in the same job.

All programs must be linkage edited before they can be executed. When LINK is specified, the compiler writes its output module in system data set SDS000, where it is available as input for the linkage editor program in a subsequent job step.

If neither LINK nor NOLINK is used, LINK is assumed.

If both LINK and SOURCE are specified, explicitly or by default, the compiler's output module is written on SYS000 and a listing of the source program is written on SYSOPT.

If LINK is specified and the name field of the EXEC FORTRAN statement is blank, the linkage editor rejects the assembled module.

BCD
EBCDIC
This option identifies the card code used in preparing the source program. program.

The BCD parameter must be specified when the BCDIC code has been used.

If neither is specified, EBCDIC is assumed.

MAP
NOMAP
This option indicates whether a compiler storage map is wanted.

The map lists all variables defined in the source program. This includes local and COMMON variables, their associated attributes, and the relative locations assigned to referenced source labels, stored alphameric text, and the start of constant and temporary pools.

If MAP is specified, a storage map is written on SYSOPT. This listing is suppressed when NOMAP is given.

If neither is specified, NOMAP is assumed.

For example, the following EXEC statement could be used to initiate a compilation job step for a program that is to be linkage edited and executed later in the same job.

```
//NAME1 EXEC FORTRAN(NODECK)
```

This statement causes the compiler to write the compiled output module on SYS000 for input to the linkage editor.

In addition, the compiler writes a listing of the source program, any error indications, and a storage map on SYSOPT.

The compiler assigns the name NAME1 to the output module. It may be referred to by this name in linkage editor control statements.

If the program contains severe errors, the linkage editor will not accept it.

The system's linkage editor program converts assembler and compiler output modules into a form suitable for loading and execution. All programs to be executed under system control must first be processed by the linkage editor.

A problem program may consist of one language processor output module. Or it may be made up of several modules, including some from previously executed jobs, that must be combined with new modules. A module may be independent and self-contained, or there may be numerous cross-references between modules and between control sections within modules.

In processing a problem program, the linkage editor converts input modules into one or more phases. A phase is a segment of machine-language code that is loaded into main storage by a single FETCH or LOAD supervisor call. It may contain one or more of the modules that were input to the linkage editor. Its size and composition are determined by the programmer.

More than one phase may be resident in main storage at a time, and a program may be designed so that one phase is not loaded until execution of another phase in the same program has been completed.

To produce phases, the linkage editor:

1. Supplies main storage addresses. It determines where each module will be loaded and it replaces the relative addresses supplied by the language processors with actual main storage addresses.

2. Completes linkage. Cross-references between modules and control sections are resolved. When, for example, module A contains an address constant for an entry point in module B, the linkage editor determines the main storage address for the entry point in B. It then supplies this address as the value of the address constant in A.

3. Incorporates module library routines. An assembler-language programmer can use the linkage editor to incorporate a routine from the module library into his program simply by defining the name of the routine as an EXTRN symbol in one of his modules. Or he can specifically instruct the linkage editor to include such a routine by naming it in the linkage editor INCLUDE control statement. For FORTRAN programmers, the compiler recognizes the external requirement and produces the necessary coding.

4. Places output phases in the phase library. From there, they can be loaded for execution or copied onto magnetic tape or punched cards. They can be kept indefinitely in this library or deleted after the next job step.

Linkage editing is performed as a separate job step. It might come, for example, as the second step in an assemble linkage edit-execute procedure.

The linkage editor program is called and entered by an EXEC LNKEDT job control statement. This statement causes the system to load and begin executing the linkage editor program, which starts by reading the linkage editor control statements discussed later in this section.

For problem programs requiring only a minimum of the linkage editor's services, a /* end-of-data statement immediately after the EXEC statement is all that is required.

The rest of this section discusses the functions of the linkage editor and the facilities it offers. It contains detailed information about how to use these facilities to gain greater flexibility in programming.

SOURCES OF INPUT

The linkage editor has four sources of input:

SDSIPT: The system input data set that contains the linkage editor control statements. This may also contain modules that were assembled or compiled in previous jobs and are not resident in the module library but must be incorporated into the current program.

SDS000: This system data set contains modules from two sources. The language processors' output modules are stored here if the EXEC statement that initiates the assembly or compilation job step contains the LINK parameter. This parameter specifies that the language processor output is

to be placed in SDS000 for linkage editing later in the same job.

The SDS000 data set also contains modules that are being incorporated into the problem program through use of linkage editor MODULE statements. These are modules that were assembled or compiled in previous jobs. They are placed in SDS000 after being read in from the SDSIPT data set.

SDSPSD: This data set contains a directory for the modules in the SDS000 data set. In effect, the contents of SDSPSD and SDS000 constitute a single directoried data set. All member names are deleted from this directory at the end of a job.

SDSREL: The SDSREL data set is the module library. This library contains the FORTRAN input/output, mathematical and service routines, dump routines, and any other relocatable modules that an installation wishes to include. These routines are available for incorporation into any problem program.

This listing of input sources assumes that standard system assignments are in effect. When, for example, the linkage editor needs a module library routine, it reads from system unit SYSREL. The linkage editor has no way of determining whether SYSREL actually is assigned to data set SDSREL. Maintaining such assignments or making substitute assignments is a function of the installation operator, as described in the second section of this publication.

A programmer can assign SYSREL to another data set that would serve as the module library data set for a single job step or job, but anyone doing this must ensure that the substitute data set contains all the routines that will be required by system programs as well as the problem program, and that blocking factors are the same.

LINKAGE EDITOR OUTPUT

Input to the linkage editor is in units of modules. Output is in units of phases.

A phase is that portion of a program that can be loaded into main storage by a single FETCH or LOAD supervisor call. The program specifies in linkage editor control statements which modules should be included in a phase. A program may require only one phase, or it may use several. In practice, the maximum size of a phase is limited only by the size of the problem program portion of main storage. The maximum size that the linkage editor can produce is 92,160 words (368,640 bytes.)

The linkage editor provides an overlay facility for multiphase programs. Basically, this permits loading and execution of one phase, followed by loading of the next phase into the same area and execution of it.

More than one phase of a multiphase program may be present in main storage at one time. Usually, one phase is designated as a ROOT phase that will be resident throughout execution of the entire program. It can be used to control the loading and execution of other phases.

Output phases from the linkage editor are stored in the phase library. They are deleted from this library at the end of the next job step unless the KEEP parameter was specified in the EXEC LNKEDT job control statement.

While in the phase library, phases can be loaded for execution or, with the system's utilities routines, copied into another data set. If the PCHABS utility statement is used for such copying, phases are copied in a format suitable for loading with the absolute loader program or for return to the phase library by the linkage editor. (Note that if a programmer changes the origin of a phase, address constants in it and in other phases referring to it must be adjusted for the relocation.)

The phase library is system data set SDSABS. The linkage editor uses system unit SYSAB2 to enter phases into this library. If SYSAB2 is assigned to another data set, the linkage editor attempts to use it, but it is successful only if the other data set is directoried, was formatted by use of the ALLOC statement's FMT parameter, and is set up for a block length of 720 bytes.

At execution time, any phase or group of phases in the phase library can be loaded and executed as part of a single program. That is, phases that were linkage edited in different jobs can be used together as long as there are no cross references between them.

For example, a ROOT phase can use the FETCH supervisor call to load and enter a second phase that was linkage edited in another job. This second phase can use another FETCH supervisor call to load and enter a third phase. If the third phase was linkage edited at the same time as the ROOT phase, it and the ROOT phase may refer to each other, but neither can refer directly to anything within the second phase.

The first phase of a multiphase program is loaded and entered through use of an

EXEC job control statement that specifies the name of the phase when initiating the execution job step. Thereafter, the problem program controls loading and execution of phases through its use of FETCH and LOAD supervisor calls.

CONTROL SECTIONS

A control section is a block of coding that can be relocated without affecting the operating logic of the rest of a program. An assembler language programmer defines a control section with a CSECT instruction.

Any language processor output module may contain one or more control sections. Each of these may contain references to points in other control sections in the same language processor output module and to control sections in other modules. The language processors supply addresses for referenced points for all control sections within each module. The linkage editor must resolve references between different modules.

The following rules governing the linkage editor's processing of control sections apply principally to duplication of control sections. This duplication can happen with large programs that combine several modules, especially when some of the input modules are from the module library. A basic assumption is that control sections with the same name are identical. The linkage editor attempts to conserve main storage space by eliminating unnecessary duplication. At the same time, it attempts to permit duplication when this is the programmer's intention.

1. If the name of a control section in a non-ROOT phase matches the name of a ROOT phase control section, the control section in the non-ROOT phase is ignored. All external references to the name are treated as referring to the ROOT phase control section.

2. If two or more control sections in the same phase have the same name, only the first is actually incorporated into the phase. All references to the name are treated as referring to this control section.

3. If a control section name in a non-ROOT phase matches the name of a control section in another non-ROOT phase, a warning message is written, but the linkage editor accepts the duplication. All subsequent references to the name, however, are treated as referring to the one that was most recently processed.

   No warning message is written if the duplication was caused by the automatic incorporation of a module library routine. Some of these modules may be incorporated into several different phases to satisfy a programmer's requirements. Any reference to such a control section is resolved to the one in the current phase. References are resolved to control sections in previous non-root phases only when the NOAUTO option has been specified for the current phase or for the job step. If searching the module library does not resolve a reference, it may be resolved in a subsequent phase.

   Control section duplication from the module library can be avoided by specifically including frequently used routines in a ROOT phase. This can be done with a linkage editor INCLUDE statement, as discussed later in this section.

4. A control section should not have the same name as an entry point. An exception to this rule is when both names are in the same module and have the same address.

5. If a control section has the same name as a COMMON control section, it is incorporated into the current phase. All references to the name are treated as referring to the regular control section, and no space is reserved for the COMMON control section.

   If the two control sections differ in length, the amount of space reserved depends upon which is processed first. If the regular control section appears first, the amount of space is determined solely by its length. If the COMMON control section appears first, enough space is reserved for the larger of the two sections, and a warning message is written.

Unnamed Control Sections

An unnamed control section is not treated in the same way as a named control section. When a module contains one or more unnamed control sections, they all are incorporated into the phase. There is no search for matching names. A blank COMMON is never resolved to this type of entry, and, since entry points and external references are always named, they also are never resolved to this type of entry.

ENTRY POINTS

An entry point is a named point within a program module that will be referred to during execution by another module.

For example, a programmer writing module A wants to branch to an instruction named ALPHA in module B. Module A defines ALPHA as an external symbol and includes an address constant for it. The other module defines ALPHA as an entry point. The two modules are assembled or compiled separately, so the language processors are unable to resolve the cross reference. This must be done by the linkage editor when the modules are combined into a phase. In practice, it supplies the address of ALPHA as the value of the address constant in module A. The program is then able to load this address into a register and branch to it.

The following rules apply to the definition and naming of entry points.

1. An entry point name must not duplicate another entry point name in the same program. This does not apply, however, when the duplication is caused by using a module in more than one phase.

2. An entry point name must not duplicate a control section name unless both are in the same module and are assigned to the same main storage address.

3. An entry point must not have the same name as a COMMON control section under any circumstances.

In addition, it is advantageous to have all entry points in a module library routine listed as additional names for the routine in the module library directory. When this is done, the program needs to define the entry point as an external symbol. It is not necessary to list both the entry point name and the routine name as external symbols in order to have the routine incorporated into the phase automatically. The module library routines supplied by IBM follow this practice.

COMMON

A programmer may use one or more COMMON control sections to contain material that must be present in main storage throughout the execution of his entire program. This material can then be inserted, referred to, or retrieved by any phase.

A COMMON control section can be named or unnamed. Both assembler language and FORTRAN programmers can specify the length of a COMMON control section and the names and lengths of fields within it.

Assembler language programmers can insert data into a COMMON control section only during execution. This restriction does not apply to FORTRAN programmers. They can provide the data for a COMMON control section with a BLOCK DATA statement. In this case, the block data is compiled as a separate module. When a control section within it has the same name as a COMMON section in another module, the linkage editor treats references to the COMMON section as applying to the block data module.

The same effect can be obtained by assembler language programmers by defining a regular control section with the required data in it. The programmer must ensure that such a control section is in a phase that is resident in main storage when references are made to it.

COMMON control sections are assigned addresses at the beginning of the problem program area of main storage. The linkage editor determines, near the end of the job step, how much space is required for COMMON and reserves this amount. Then, in assigning addresses for phase origins, it guards against overlaying any part of a COMMON control section. The linkage editor cannot ensure this protection, however, when the programmer provides a specific main storage address or displacement for a phase origin. (See the discussion of the PHASE statement later in this section.)

To refer to a COMMON control section, an assembler language programmer must include an address constant in his main program for the name of the COMMON control section. If COMMON is unnamed, there must be an address constant for its first field. The address that is supplied may then be used as a base register value for references to COMMON. The register that is used must be defined as a base register, with a USING statement, for the COMMON control section.

This procedure must be followed for each COMMON control section in an assembler language program.

The linkage editor observes the following rules in processing COMMON control sections.

1. If a program defines two or more COMMON control sections with the same name, space is reserved for only one. If different lengths are indicated for each, the longest length is used.

2. If a COMMON control section and a regular control section have the same name, they are treated as a single regular control section.

   The resulting control section is incorporated into the phase in which the regular control section appears. If their lengths differ, the linkage editor's action depends upon which is processed first.

   If the regular control section appears first in the input stream, its length is used. If the COMMON section appears first, enough space is reserved for whichever is longer, and a warning message is written if their lengths differ. This applies even when the regular control section is brought in from the module library.

3. A COMMON control section must not have the same name as an entry point in the same program.

4. Any external references in the program that match names of COMMON entries are resolved to the COMMON entry.

EXTERNAL REFERENCES

Cross references between input modules are said to be external. Any module may contain a reference to an entry point or control section name in another module. The symbols referred to must be defined as external in the module making the reference and as entry points in the module containing the named items. It is not necessary to define a control section name as an entry point.

In assembler language, an external reference might appear as follows:

```
         EXTRN  THETA
         .
         .
         .
         L      12,ATHETA
         .
         .
         .
ATHETA   DC     A(THETA)
```

where THETA is an entry point in another module, and ATHETA is an address constant. The linkage editor determines the main storage address of THETA and supplies it as the value of the address constant. During execution, the program can load this address into a register and branch to it or use it as a base address for references to other points within the other module.

(If, for any reason, the programmer subsequently changes the loading address of the phase containing THETA, he must also change the value of any address constants referring to points within that phase.)

The linkage editor resolves external references as they appear in the input stream. In doing so, it considers the phases in which the reference and the matching name appear.

In general, if the linkage editor encounters a reference to a control section or entry point in the current phase or a ROOT phase, it can be resolved immediately. For references to other parts of the program, it must determine whether the NOAUTO option has been specified.

The NOAUTO option suppresses the linkage editor's automatic search of the module library. This search is done at the end of any phase in which there are external references that haven't been resolved. If the module library directory contains a name that matches the reference, the module library routine is incorporated into the phase and the reference is resolved to it. This facility enables a programmer to bring module library routines into his program without having to provide INCLUDE statements for each one that is wanted.

The linkage editor also assumes that a programmer may want to use the same module library routine in more than one phase. Therefore, unless the NOAUTO option has been specified, the linkage editor never resolves an external reference to a name in a previous non-ROOT phase.

The NOAUTO option may be specified in an EXEC LNKEDT statement for the entire job step or in a PHASE statement for the processing of a single phase.

In summary, the linkage editor goes through the following process when it encounters an external reference:

1. It examines its list of control section and entry point names. If it finds a matching name in the current or ROOT phase, the reference is resolved. If it finds a matching name in a previous non-ROOT phase, the reference is resolved only if the NOAUTO option is in effect for the current phase.

   If it finds two or more matching names, the reference is resolved to the one most recently processed. (This could happen when a module is included in more than one phase.)

If no acceptable matching name is found, it continues processing the current phase.

2. If a matching name is found subsequently in the current phase, the external reference is resolved to it.

3. If the reference is not resolved by the end of the phase and the NOAUTO option has not been specified, the linkage editor examines the module library directory. If the reference matches the name or alternate name of any module library routine, this routine is incorporated into the phase and the reference is resolved to it.

   This procedure continues until all unresolved references have been checked against the module library directory.

4. Finally, a control section or entry point in a subsequent phase may resolve the reference.

   If any address constants remain unresolved at the end of the job step, the linkage editor issues an error severity message such that the program cannot be executed. A list of unresolved symbols is written if the MAP option has been specified in the EXEC LNKEDT statement. A message is written indicating the number of unresolved address constants regardless of whether MAP is specified.

EXEC STATEMENT PARAMETERS

Certain linkage editor parameters may be specified in the optional parameter list field of the EXEC LNKEDT job control statement. These are KEEP, NOKEEP, MAP, NOMAP, and NOAUTO.

KEEP
NOKEEP
: The KEEP keyword is used to indicate that the output of the linkage editor is to be kept permanently in the phase library. The NOKEEP keyword indicates that the linkage editor output may be eliminated from the phase library at the end of the next job step. If neither keyword is specified, NOKEEP is assumed.

MAP
NOMAP
: The MAP keyword indicates that a storage map and certain warning messages are to be produced at the end of the job step. NOMAP indicates that the map and these warning messages are not needed. If neither is specified, MAP is assumed.

  For a phase, the map provides the address of its initial entry point, the addresses of the first and last bytes it occupies, and the block number in the phase library where the text of the phase starts. The map provides the name, loading address, and relocation factor for each control section and the name and loading address of each entry point. It also indicates any entry points that are not referred to by the program by placing an * next to the entry name and lists any unresolved external references. It prints the name, load address, and length of any COMMON control sections, regardless of which option is in effect.

NOAUTO
: The NOAUTO keyword suppresses automatic searching of the module library for names matching unresolved external references for the entire linkage editing job step. When this search is not suppressed, module names that match unresolved references are incorporated into the program. This search can be suppressed for a single phase by use of the NOAUTO option in the PHASE statement.

CONTROL STATEMENTS

There are four linkage editor control statements; MODULE, PHASE, INCLUDE, and ENTRY.

These statements provide required information for the linkage editor and enable a program to invoke certain optional facilities. Other options may be exercised by use of parameters in the EXEC job control statement that initiates the linkage editor job step.

A program requiring only the minimum linkage editing need not use any of these statements. A /* end of data job control statement immediately following the EXEC LNKEDT statement indicates that the program contains no linkage editor control statements, and only the minimum services are required. A /* statement can also be placed after any module statements and their associated modules with the same effect.

The linkage editor control statements are also required if the program is to be divided into phases.

The MODULE statement places language processor output modules from previous jobs onto SYS000 so they can be incorporated into the current program.

The PHASE statement defines the beginning of a phase, assigns it a name, and indicates where it is to be loaded.

The INCLUDE statement indicates which modules or parts of modules are to be included in the phase defined by a PHASE statement.

The ENTRY statement defines the end of the linkage editor input and may indicate the initial entry point for execution.

Each of these statements contains an operation field and an operand field. The operation field, identifying the statement type, must be preceded and followed by at least one blank. No blanks are permitted in the operand field.

Comments may be written in linkage editor control statements in the same manner as in job control statements. They must be separated from the last entry of the operand field by at least one blank, and they cannot extend beyond column 71. No continuation cards are permitted. The only job control statement that is permitted is the /* end-of-data statement. No comments are allowed in an ENTRY statement that does not contain an operand field.

MODULE Statement

The MODULE statement identifies a module that was assembled or compiled in a previous job and is to be incorporated into the current program.

The statement instructs the linkage editor to place the module that follows it on SYS000 and to make a directory entry for it on SYSPSD. Once entered, these modules are treated in control statements and by the linkage editor in the same manner as modules created by the language processors earlier in the job.

MODULE statements and their associated modules must precede any PHASE, INCLUDE, and ENTRY statements on the system input unit, SYSIPT.

The module following a MODULE statement should contain all the ESD, TXT, RLD, and END cards (or card images) produced by the assembler, as well as any REP patch cards added by the programmer. The formats of these cards are shown in Appendix B.

The cards should be in the following order: ESD, TXT, RLD, REP, and END. A module must contain at least one ESD card and an END card.

The format of the MODULE statement is:

```
| MODULE module name |
```

module name
: This field assigns a name to the module. The name must be one to eight alphameric characters, the first of which is alphabetic.

  This name is used to refer to the module in INCLUDE statements.

PHASE Statement

The PHASE statement defines the start of a phase, assigns a name to it, and indicates where it should be loaded. It also may be used to designate a ROOT phase.

A PHASE statement should be the first linkage editor control statement when a program consists entirely of modules assembled earlier in the same job and modules from the module library. If the program also includes modules assembled in previous jobs, a PHASE statement follows these modules and their associated MODULE statements.

When the first statement after an EXEC LNKEDT job control statement is a /* end-of-data statement, the linkage editor creates a dummy phase. The dummy phase contains all modules currently resident on SYS000 and any module library routines that are referred to by the program. Its name is the same as that of the first module in SDS000. Loading of such a phase starts at the first double word-boundary location in the problem program portion of main storage, or, if the program contains one or more COMMON control sections, at the first double-word boundary location after the end of COMMON.

The format of the PHASE statement is:

```
| PHASE name,origin[,NOAUTO] |
```

name
: This field specifies the name of the phase.

The name must be one to eight alphameric characters, the first of which is alphabetic.

This name identifies the phase in the phase library. It is the name used in an EXEC job control statement or in a FETCH or LOAD supervisor call to load the phase into main storage for execution.

origin

This field indicates where the phase should be loaded.

A programmer selects one of five possible entries for this field.

No matter which entry is chosen, loading of a phase always starts at a location aligned on a double-word boundary. If the program specifies an origin that is not so aligned, the linkage editor adjusts the origin to the next location that is aligned.

The most common origin for the first phase of a program is at the first double-word boundary location in the problem program area. If, however, the program includes a COMMON control section, the origin is adjusted to the first double-word boundary location after the end of the COMMON control section.

When specifying an origin for any phase, the programmer should ensure that it is within the problem program area.

The formats for the possible origin entries are:

```
symbol[(phase)][±relocation]
*[±relocation]
S[±relocation]
+displacement
ROOT
```

symbol[(phase)][±relocation]

The symbol portion of this field specifies a symbolic address within the program. It must be the name of a phase, control section or entry point that has already been processed by the linkage editor.

If the symbol is the name of a phase and there is a control section or entry point with the same name, the address of the last entered name is used as the origin address.

The (phase) portion of this field is used when the symbol is the name of a control section or entry point that appears in more than one phase. In such cases, this field contains the name of the proper phase. The phase name must be enclosed in parentheses, as indicated.

The ±relocation portion of this field is used to indicate a displacement from the address of the symbol. The address is either increased or decreased by the number of bytes specified in this field. If, for example, the name cited in the symbol subfield has been assigned to location 10,000 and a relocation factor of 100 is given, loading of the phase will start at location 10,100 or the first double-word boundary location thereafter. The relocation factor may be expressed as a decimal value of one to eight digits or as a hexadecimal value of one to six digits. If hexadecimal, it must take the form X'hhhhhh' where hhhhhh represents the field of digits.

When specifying a relocation factor, the programmer should ensure that the adjusted address is within the problem program area.

*[±relocation]

The * notation indicates that loading is to start at the next double word location counter available to the linkage editor location counter.

This notation might be used for a phase that is to be resident in main storage at the same time as a previously processed phase.

If this is the first phase in a program, loading starts at the first double word boundary location in the problem program area (assuming there is no COMMON control section.)

Relocation, if used, is expressed in the same way as for the symbol[(phase)][±relocation] field.

S[±relocation]

The S notation indicates that loading of the phase is to start at the first double-word boundary location in the problem program area.

The effect is the same as using the * notation for the first phase.

Relocation, if used, is expressed in the same way as for the symbol[(phase)][±relocation] field.

+Displacement

This field is used to specify an absolute main storage address where loading is to start.

This address is given relative to location 0. It may be expressed as a hexadecimal value of one to six digits or as a decimal value of one to eight digits. If hexadecimal, the field is written as X'hhhhhh' where hhhhhh represents the hexadecimal digits.

If the address specified is not on a double-word boundary, loading starts at the next following address that is properly aligned. This address may be further adjusted by the linkage editor if the program contains one or more COMMON control sections. In this case, it is increased by the length of the COMMON control sections.

ROOT

Use of this keyword identifies a phase as a ROOT phase.

The linkage editor assumes that any phase identified as ROOT is to be present in main storage throughout execution of the entire program. Only the first phase may be designated as a ROOT phase.

Loading of a ROOT phase starts at the first double word boundary location in the problem program area (assuming there are no COMMON control sections.)

If a ROOT phase contains a control section or entry point name that also appears in other phases, all references to that name are treated as referring to the control section or entry point in the root phase.

Unless the NOMAP option is specified in the EXEC LNKEDT statement, a warning message is written on SYSLST at the end of the job step if any phase has specified an origin that would overlay any part of a ROOT phase during execution.

NOAUTO

The NOAUTO keyword suppresses automatic searching of the module library for names that match unresolved external references appearing in the phase.

Suppression of the automatic module library search applies only to the current phase. It can be suppressed for the entire job step by specifying the NOAUTO option in the EXEC LNKEDT job control statement.

Use of this option does not affect the incorporation of any library modules specifically cited in an INCLUDE statement.

INCLUDE Statement

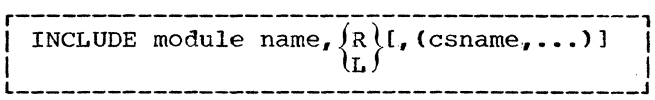
The INCLUDE statement identifies a module that is to be included in a particular phase.

This statement also may be used to indicate that only certain control sections from a module are to be included in the phase.

This statement follows a PHASE statement. It names the modules from the module library and SYS000 that are to be combined in the phase. Modules that are put on SYS000 by MODULE statements are treated in the same way as those put on SYS000 by the assembler or compiler earlier in the job.

The statement has three operands, the first two of which are required. The first gives the name of the module, and the second identifies its location. The third may be used to indicate which control sections should be included.

The format of the INCLUDE statement is:



module name

This field specifies the name of the module. For modules assembled earlier in the job, this name is the same as the name specified in the stepname field of the EXEC ASSEMBLE(LINK) job control statement. For modules resident in the module library, this name is the same as appears in the library directory. For modules placed on SYS000 with MODULE statements, this name is the name cited in the operand field of the MODULE statement.

R

The R notation indicates that the module is located in the module library. With it, a module can be incorporated regardless of whether the NOAUTO option is in effect.

L

The L notation indicates that the module is on SYS000.

csname

This field specifies the names of up to five control sections within the module that are to be included in the phase.

Control sections are incorporated into a phase in the same order as they

appear in a module. The order in which they are listed in this statement is not significant. If control sections must be incorporated in a specific order, they should be cited in separate INCLUDE statements.

Portions of the module that are not included in this list may be cited in INCLUDE statements later in the same phase, in other phases, or disregarded. Usually, a control section that is used by more than one phase would be included in a ROOT phase.

If more than one name is specified, they are separated by commas. The entire field is enclosed in parentheses.

ENTRY Statement

The ENTRY statement identifies the end of a program and may be used to specify the program's initial entry point.

This should be the last control statement processed by the linkage editor. The linkage editor, at this point, should have all the control information and input modules that it needs to create all the program's output phases.

This statement may be omitted if there is a /* end-of-data statement after the last linkage editor control statement. The system assumes the initial entry point is the first location in the first phase or the first transfer address, if any, on an END card in the first phase.

The format of the ENTRY statement is:

```
┌──────────────────────────────────────┐
| ENTRY [name]                         |
└──────────────────────────────────────┘
```

name

This field identifies the program's initial entry point. When the system is ready to pass control to the problem program for execution in a subsequent job step, it enters the program at this point.

Any symbol in this field must have been defined in the program as a control section name or entry point. If the name appears in more than one phase, it is assumed to refer to the most recently processed phase.

If this field is omitted, the initial entry point is assumed to be the first transfer point specified in an END statement of the modules in the first phase. If control sections within the first module have been split between two or more phases, and the END statement refers to an entry point in another phase, the system assigns the transfer on the END card to the phase in which the control section containing the entry point is placed. Other phases are assigned their origin as the transfer address.

Otherwise, the initial entry point is assumed to be the first location in the first phase.

Linkage Editor Examples

Example 1: Single module, single phase.

The purpose is to assemble, linkage edit, and execute a single, self-contained module.

```
//SAMPLE1 JOB DUMP,137596

//PROG1 EXEC ASSEMBLE(LINK)

(Source program statements go here if SYSIPT is assigned to same data set as SYSRDR.)

/*

// EXEC LNKEDT

/*

(Additional job control statements, such as ACCESS, ALLOC, and LABEL, may go here if required for the problem program.)

// EXEC

/&
```

Example 2: Single module, multiple phase.

The purpose is to assemble and linkage edit a single module that contains three control sections, named CSECTA, CSECTB, and CSECTC. The module is to be divided into three phases for execution. The first phase is present throughout execution. The third phase overlays the second.

```
//SAMPLE2 JOB DUMP,137596
//PROG2 EXEC ASSEMBLE(LINK)
(Source program statements if SYSIPT assigned to same data set as SYSRDR.)
/*
// EXEC LNKEDT
   PHASE PH1,S
   INCLUDE PROG2,L,(CSECTA)
   PHASE PH2,*
   INCLUDE PROG2,L,(CSECTB)
   PHASE PH3,PH2
   INCLUDE PROG2,L,(CSECTC)
   ENTRY
/*
(Additional job control statements, if needed.)
// EXEC
/&
```

Example 3: Multiple module, single phase.

The purpose is to combine two previously assembled modules for execution as a single phase. Execution is to start with a control section named CSECTD.

```
//SAMPLE3 JOB DUMP,137596
// EXEC LNKEDT
   MODULE PROG3
   (Assembled module statements)
   MODULE PROG4
   (Assembled module statements)
   PHASE PROGX,S,NOAUTO
   INCLUDE PROG3,L
   INCLUDE PROG4,L
   ENTRY CSECTD
/*
(Additional job control statements, if needed.)
// EXEC
/&
```

Example 4: Multiple module, multiple phase.

The purpose is to combine four modules containing several control sections into a three phase program. Two modules have not been assembled. One was assembled previously, and the other is in the module library. They are as follows:

Module	Control Sections	Status
LIBMOD	LM1, LM2	Module library
NEWA	A, A1, A2	To be assembled
NEWB	B1, B2	To be assembled
OLDMOD	OLM1, OLM2	Previously assembled

These modules and control sections are to be arranged in phases, as follows:

Phase	Contents	Origin
PH1	A, A1, LIBMOD	After supervisor
PH2	OLM2, A2, B1	After PH1
PH3	OLM1, B2	After PH1

Sample statements are as follows:

```
//SAMPLE4 JOB DUMP,137596
//NEWA EXEC ASSEMBLE(LINK)
(Source statements for module NEWA)
/*
//NEWB EXEC ASSEMBLE(LINK)
(Source statements for module NEWB)
/*
// EXEC LNKEDT
   MODULE OLDMOD
(Assembled statements for module OLDMOD)
   PHASE PH1,ROOT
   INCLUDE NEWA,L,(A,A1)
   INCLUDE LIBMOD,R
   PHASE PH2,*
   INCLUDE OLDMOD,L,(OLM2)
   INCLUDE NEWA,L,(A2)
   INCLUDE NEWB,L,(B1)
   PHASE PH3,PH2
   INCLUDE OLDMOD,L,(OLM1)
   INCLUDE NEWB,L,(B2)
   ENTRY
/*
// EXEC
/&
```

There are two levels of input/output operations. The first is the read/write level in which a problem program invokes system routines to perform all input/output operations. The system sets up the needed instructions, executes the operation, and applies standard interruption analysis and error recovery techniques. This level provides device independence in that the same problem program coding is used regardless of whether the device is a magnetic tape unit, direct access storage device, card read/punch, or printer.

The second level is the execute channel program (EXCP) level in which the problem program provides its own input/output routines. The EXCP level can be used with devices not supported by the system or to manipulate devices in a manner not provided by the read/write level routines. Device routines written and tested at the EXCP level may subsequently be incorporated into the read/write level through reassembly of the supervisor.

Both levels may be used within a single program.

This publication deals in detail with the read/write functions. The EXCP level is discussed in the publication IBM System/360, Model 44 Programming System, Systems Programmers Guide, Form C28-6814.

A summary of the input/output supervisor calls appears in Table 7.

Table 7. Input/Output Supervisor Calls

Mnemonic	Code	Description
EXCP	SVC 0	EXCP (Execute Channel Program) is used by programs that provide their own input/output routines.
WAIT	SVC 1	WAIT is used to delay further execution of a program until an EXCP operation has been completed.
OPEN	SVC 2	OPEN is used for tape label processing and repositioning of data sets before the data sets are used in a program.
CLOSE	SVC 3	CLOSE is used for label processing of data sets after they have been used in a program.
READ	SVC 4	READ is used to transmit a block of data from an input data set to an area of main storage.
WRITE	SVC 5	WRITE is used to transmit a block of data from an area of main storage to an output data set.
CHECK	SVC 6	CHECK is used to delay further execution of a program until an input/output operation, other than one initiated by EXCP, has been completed.
NOTE	SVC 7	NOTE is used to determine the current position of a data set.
POINT	SVC 8	POINT is used to reposition a data set.
WEF	SVC 9	WEF (Write End-of-File mark) is used to write an end-of-file mark in an output data set on magnetic tape.
REWIND	SVC 10	REWIND is used to reposition a magnetic tape to its load point.
UNLOAD	SVC 11	UNLOAD is used to rewind and unload a magnetic tape volume.

READ/WRITE OPERATIONS

The read/write level supervisor calls are READ, WRITE, NOTE, POINT, CHECK, WEF, REWIND, and UNLOAD. A program requiring an input/output operation issues one of these supervisor calls and provides certain information in parameters and control blocks so that system routines can execute the operation.

The system uses three types of control blocks. They are:

- Request control blocks, which contain general reference information relevant to an individual input/output request.
- File control blocks, which contain information pertaining to a particular data set and the volume in which it resides.
- Unit control blocks, which contain information about a specific input/output device.

The formats for these blocks are shown in Appendix A. A problem program at the read/write level may refer to the information in any of them, but the request control block is the only one that it must define. The file and unit control blocks are in the supervisor portion of main storage.

The problem program defines a 40-byte request control block for each data set referred to by its input/output requests. The program also must provide one item of information within the block, the SYSUNI index value for the system unit being used in the operation. This value can be obtained from Table 8. The program fills the rest of the block with hexadecimal zeros. The system obtains all other information required for the block from various sources and handles the block's remaining construction and maintenance.

One of the parameters accompanying each input/output request in a program is the address of a request control block. The system then uses the information in this and other blocks to prepare channel commands, schedule and execute the requested operation, analyze the results for errors or abnormal conditions, such as end of file, and post a return code so the program can determine how the operation turned out.

Table 8. Symbolic Unit (SYSUNI) Index Values

Symbolic Unit	Hexadecimal Code	Decimal Code
SYSAB1	01	1
SYSAB2	02	2
SYSREL	03	3
SYSLOG	04	4
SYSRDR	05	5
SYSIPT	06	6
SYSLST	07	7
SYSOPT	08	8
SYSPCH	09	9
SYSPSD	0A	10
SYSDMY	0B	11
SYSUAS	0C	12
	0D	13
reserved	0E	14
	0F	15
SYS000	10	16
SYS001	11	17
SYS002	12	18
.		
.		
.		
SYS009	19	25
SYS010	1A	26
.		
.		
.		
SYS015	1F	31
SYS016	20	32
SYS017	21	33
.		
.		
.		
SYS200	D8	216

A return code is saved in the request control block until the next time the program requests an input/output operation using the same block. The system examines the code to see how the previous operation terminated. If the termination was abnormal, the system places the code in register

15, resets the code area of the request control block to 00, and returns control to the problem program. The new request is ignored, but since the control block's return code area has been cleared, the request may be reissued.

It is up to the problem program to decide what to do about the abnormal termination. Such conditions are noted when the system detects a permanent transmission error, an end-of-file or end-of-extent indication, or an apparently valid request that could not be executed, such as READ for a printer. Table 9 contains a list of the possible return codes and their meanings.

Table 9. Input/Output Return Codes

Hexadecimal Code	Description
00	Normal return
04	Operation dependent: 1. An end-of-file mark was detected during the last READ or POINT operation. 2. An end-of-extent condition was detected during the last WRITE operation. The data set is positioned just prior to the block that causes the overwrite. The WRITE request has not been fulfilled.
08	A permanent transmission error occurred during the last operation. The data set is positioned just past the block containing the error.
0C	The last operation was terminated without transmitting all the data. The position of the data set is not known.
10	The last operation was terminated because of an invalid request, such as a READ request for a printer.
14	The last operation was terminated with an incorrect length condition. This applies only to a READ or WRITE that does not invoke incorrect length suppression.

A program should not attempt to determine how an operation turned out by looking into the control block to examine the code. The operation may still be in progress, and the code would not have been set. A program that must know how an operation turned out before it resumes processing can invoke the CHECK function.

When CHECK is invoked, control does not return to the problem program until the operation has been completed and a return code is posted in register 15. The control block return code area is cleared, and the next request is accepted regardless of whether the previous operation terminated abnormally.

The following text explains which considerations are necessary when a program makes an input/output request.

When the supervisor call is issued, the system examines register 1 for the address of a parameter list. This list contains the address of a request control block and it may also contain, depending on the type of operation, the address of an input/output buffer and the address of a count of the number of bytes to be transmitted.

The system examines the request control block to determine how the last operation associated with it terminated.

- If the last operation terminated normally (or was followed by a CHECK), the system proceeds with processing the new request.
- If the last operation still is in progress, the system waits until it terminates and has been analyzed before proceeding with the new request.
- If the last operation terminated abnormally (and this condition was not detected by a CHECK), the new request is ignored. The system places the abnormal termination code in register 15, resets the control block return code area to 00, and returns control to the problem program. It is up to the problem program to act on the abnormal termination condition and determine whether the new request should be reissued.

An abnormal condition code in one request control block does not affect operations using other request control blocks. The system treats each block independently, even if two or more blocks refer to the same data set.

If the last operation terminated normally or the request is reissued, the system

determines whether the required channel and device facilities are available. It also examines the new request and its parameters for errors.

If the required facilities are free, the system continues processing the request. If they are in use, the system queues the new request and returns control to the problem program. The system will automatically resume processing the request when the facilities are available for it.

If the system discovers errors in the parameters of the new request, it posts an error code in the request control block and returns to the problem program. This code is detected by the CHECK or by the next request using the same block.

When the required facilities are free and the request has reached the top of the queue, the system constructs the required channel program and starts the operation. This channel queue method is designed to permit maximum use of all available channel facilities while providing the greatest amount of overlapped processing.

Termination of an operation causes an input/output interruption that is analyzed by the system. Standard error recovery methods are applied, when necessary, and a return code is posted in the request control block.

CONTROL CHARACTERS

A programmer may specify in the LABEL statement for a data set that the first byte of each record block is an output control character.

The LABEL statement parameter CTLASA indicates that the control byte is for physical manipulation of printers or unit record devices. This permits a programmer to specify his own carriage spacing or stacker selection requirements. The control byte must contain the bit configuration for the American Standards Association code for the desired operation. The characters that produce the proper bit configuration are shown in Table 10.

The LABEL statement parameter CTLCHR indicates that the control character is not necessarily an ASA character but is one recognized by System/360. The programming system can use this character when writing unit record and printer data sets.

A control character is treated as data when blocks containing it are written on a device to which it does not apply. At such times, specifying the CTLASA or CTLCHR parameter causes the system to note the presence of control characters in the data set's label. Even though this is noted in a label, the parameter must be specified again if the data set is subsequently written on a printer or unit record device and the control character is to be used.

Table 10. ASA Codes

Code	Definition
Printers	
b (blank)	Space one line before printing
0	Space two lines before printing
-	Space three lines before printing
+	Suppress space before printing
1	Skip to channel 1
2	Skip to channel 2
3	Skip to channel 3
4	Skip to channel 4
5	Skip to channel 5
6	Skip to channel 6
7	Skip to channel 7
8	Skip to channel 8
9	Skip to channel 9
A	Skip to channel 10
B	Skip to channel 11
C	Skip to channel 12
Unit Record	
V	Select card pocket 1
W	Select card pocket 2

BLOCKING

A program must handle its own buffering, blocking, and unblocking. All blocks within a data set must be the same length. The system's read/write routines transmit one block at a time, although a program, when

reading, may elect to have only a partial block put into its buffers. These capabilities are covered in more detail in the discussions of each individual supervisor call.

Programs using a 2315 Disk Cartridge obtain greatest efficiency by writing blocks that are either one, two, four, or eight sectors in length. Calculations of block sizes also should note that the system uses the first five bytes of each sector, when blocks are one sector or less in length, for the addresses of the track and any alternate track. When blocks are more than one sector long, these five-byte address fields are written only in the first sector for the block.

ATTENTION INTERRUPTIONS

An attention interruption occurs when the operator presses the request key on the Console Printer Keyboard. This attention condition is recognized immediately, and the system executes any input/output operation that it requires. The interruption is processed without destroying any problem program information.

When an attention interruption is recognized, the system examines word 2 of the appropriate unit control block for the address of a special attention input/output block permanently resident in main storage. This address is moved to word 3 of the unit control block, and the system branches to the device routine address given in the attention input/output block to read the message typed on the keyboard.

LABEL PROCESSING SUPERVISOR CALLS

The OPEN and CLOSE supervisor calls are used to prepare data sets for processing and to ensure their proper disposition when processing is completed.

All direct access data sets must be labeled. Unit record data sets do not have labels, but OPEN and CLOSE can be used with them for purposes other than label processing.

The programming system, as distributed by IBM, assumes that magnetic tapes will be unlabeled. They will not contain volume or data set header and trailer labels. An installation can modify the system supervisor, however, to get full label processing of tapes. The actions of OPEN and CLOSE on tapes depends on which standard the installation has adopted. This is discussed in detail in this section.

It is not necessary to open or close system data sets, such as SDSIPT or SDSREL, as long as standard system unit assignments are in effect. This function is handled by the system.

The system's language processors, linkage editor, and utilities programs open and close any data sets they use on symbolic units SYS000 and up. System unit data sets are opened and closed by job control.

The system also attempts to close any problem program output data sets on symbolic units SYS000 and higher that have not been closed at the end of a job. This is done before the system resets all system units to their standard assignments, if any have been altered by the program.

Open and close must be used when adding new members to directoried data sets and when increasing the amount of data in any disk data set if the new material is to be preserved after the end of the job. This is because close causes the system to update its block count information and, when applicable, make directory entries for new members. If this record keeping is not done, a programmer is unable to refer to the new material in future jobs. Directoried data set members are opened and closed in the same manner as data sets.

The system routines for executing OPEN and CLOSE are present in main storage only when needed. They are loaded each time an OPEN or CLOSE supervisor call is issued. Processing time can be saved by using the system's ability to open or close more than one data set with a single supervisor call.

The applicable codes are:

OPEN - SVC 2

CLOSE - SVC 3

OPEN - SVC 2

The OPEN supervisor call is used to reposition direct access, unit record, and magnetic tape data sets and, when applicable, verify or create magnetic tape labels.

Use of OPEN also ensures that a data set is closed at the end of the job. If the program does not close it, the closing is handled by the job control processor.

OPEN's actions and use vary according to the type of data set, as follows:

Direct access data sets. The system creates or verifies direct access volume and data set labels when it processes ALLOC and ACCESS statements. An OPEN supervisor call for a direct access data set does not involve any label processing, but it can be used to repostion a data set.

A data set that is to be preserved after the end of the job must be opened and closed if new material is added to it. Closing causes the job control processor to update its block count information for the data set. If this count is not updated, a programmer cannot refer to the new blocks in future jobs.

A direct access data set can be in one of two positions when the system first turns control over to the problem program. It can be positioned just prior to the first data block of the data set or data set member, or it can be positioned just after its last block of data. This second position is obtained by specifying the EXT parameter in an ACCESS statement.

A directoried data set is positioned just prior to the first block of the desired member. The EXT parameter does not apply.

If, however, the data set was opened for input, used, and closed without repositioning earlier in the job, it may be positioned anywhere. The repositioning request is made by a code in the supervisor call's parameters.

If a direct access data set is being opened for the first time in a job, the EXT parameter in an ACCESS statement overrides any request for repositioning. The data set remains positioned after its last data block.

If repositioning is not requested, the data set is not disturbed during execution of the supervisor call.

Unit record data sets. Again, no label processing is involved, but a unit record or printer data set can be opened for repositioning.

A request for repositioning causes the block count field of the data set's file control block to be reset to zero. The block count is a record of the number of blocks that have been read or written by the program.

Magnetic tape data sets. The first consideration here is whether the installation standard is to use volume and data set labels on tapes.

In an unlabeled shop, the system treats all tapes as unlabeled, regardless of whether they actually do contain labels. A LABEL statement can be used to provide information for a data set's file control block, but there is no label processing.

Use of OPEN ensures that a required volume is mounted and enables a programmer to request repositioning.

If repositioning is requested, the system rewinds the volume and positions it just before its first data record. If the tape is empty, it is positioned just after its load point. Specifying the EXT parameter on an ACCESS statement overrides any repositioning request the first time the data set is opened in a job, and it is positioned after its last data block. (EXT should be specified only for input data sets. If specified for an output data set, the results are unpredictable.)

If repositioning is not requested, the data set is not disturbed unless it was used earlier in the job and is positioned after an end-of-file mark. In such a case, it is backed up to a point between the data set's last record and the end of file mark.

In a labeled tape shop, OPEN can be used with labeled and unlabeled volumes. If the volume is not labeled, it and the data set are treated as unlabeled.

1. Labeled volume with labeled data set. Any ALLOC or ACCESS statement referring to a labeled data set on a labeled volume must be followed by a LABEL statement.

   OPEN reads the volume label to ensure that the correct volume has been mounted. It then reads the data set header label. For an input data set, it ensures that the data set name in the header label is the correct one. For an output data set, it examines the expiration date to make certain the tape does not contain an active data set. If the date has expired, a new header label is written.

   An output data set is positioned for the writing of the first data block.

   Repositioning of an input tape data set and the effect of the EXT parameter are the same as for tapes in an unlabeled shop.

2. Unlabeled tape volume with unlabeled data sets. No LABEL statement can be used with an ALLOC or ACCESS statement for an unlabeled data set on an unlabeled volume.

   OPEN checks for a volume label. If it finds one on an output OPEN, it checks for a header label expiration date to ensure that the volume does not contain an active data set. The volume label is erased when the first record of the data set is written.

   If no volume label is found, the tape is treated in the same manner as an unlabeled tape in an unlabeled shop.

When OPEN is executed, register 1 must contain the address of a parameter list. This list consists of one full-word entry for each data set to be opened. Each entry is aligned on a full-word boundary and contains the address of a 4-byte control information area. The first byte of the last entry in the parameter list must contain the hexadecimal code 80, indicating the end of the list.

The four bytes of control information are specified in hexadecimal as follows:

```
CONTRL   DC   XL4'uurrpp cc'
```

uu is the SYSUNI index value of the symbolic unit containing the data set. This value can be obtained from Table 8.

rr = 00 if the data set is not to be repositioned.
= 01 if repositioning is wanted.

pp = 00 if an input data set is being opened.
= 01 if an output data set is being opened.

cc is a byte reserved for use by the

system if a return code is needed to notify the program of errors.

After OPEN has been executed, control returns to the instruction following the SVC. The low-order byte of register 15 contains the hexadecimal code 00 if there were no errors. If register 15 contains 04, indicating errors, the cc field for the applicable system unit contains one of the following codes:

Hexadecimal Code	Description
01	The system cannot find the data set. The problem may be that the data set is not properly defined, an incorrect SYSUNI code was used, the volume is mounted on a device that is disconnected or not operating, an error occurred in positioning the data set, or, in a labeled shop, the operator instructed the system to treat the data set as a dummy file. If the problem is a positioning error, the volume is unloaded and treated as a dummy file. The data set is not opened.
02	This code applies only to labeled tape shops. For input, it indicates that either a volume or header label is missing, but a LABEL statement was used. The absence of a label is noted, so no attempt will be made to verify a trailer label when CLOSE is given. For an output data set, this code signifies the absence of both a volume label and header label. In either case, the data set is opened.

After returning the error code, the system takes no further notice of these conditions.

If a more serious error occurs in a labeled shop, the system writes a message on SYSLOG requesting operator intervention. These conditions include input/output errors while processing a supervisor call, mounting a wrong volume, detection of the wrong data set name in a header label, or detection of an active data set on a tape that is supposed to have expired.

The operator has three alternatives. He can instruct the system to resume processing and use the data set anyway, to resume processing and treat the data set as a dummy, or to cancel the job. If he elects to treat it as a dummy, the cc field of the return code parameter is set to 01, the volume is rewound and unloaded, and all references to the symbolic unit assigned to it are treated as though it had been cited in an ACCESS IGN statement.

Following is an example of the use of the OPEN supervisor call.

This example assumes three data sets are to be opened, two input data sets on SYSIPT and SYS001 and an output data set on SYSOPT. The names of the data sets or data set members needed for the program have been specified in control statements. Repositioning of SYS001 and SYSOPT is desired.

```
OPEN     EQU  2
         .
         .
         .
         LA   1,PARAM

         SVC  OPEN

         BAL  14,ANALYS
         .
         .
         .
         DS   0F

PARAM    DC   A(CTLIPT)

         DC   A(CTL001)

         DC   X'80'

         DC   AL3(CTLOPT)

CTLIPT   DC   XL4'06000000'

CTL001   DC   XL4'11010000'

CTLOPT   DC   XL4'08010100'
```

In this example, the Branch and Link (BAL) instruction is given to branch to an analysis routine as soon as the system has finished executing OPEN. ANALYS is not a system routine. It represents a routine that should be included in a problem program. It examines the return codes, takes any necessary action, and returns to the main program by branching to the address in register 14.

CLOSE - SVC 3

The CLOSE supervisor call is used to ensure proper disposition of data sets.

With CLOSE, a program can:

- Reposition a tape volume, unit record, or direct access data set.

- Disconnect a symbolic unit to protect the contents of its data set.
- Verify or create standard trailer labels in magnetic tape data sets.
- Cause the system to create a directory entry for a new directoried data set member.

As with OPEN, the actions and use of CLOSE vary according to the type of data set and whether the installation standard is to use labels on magnetic tapes.

Direct Access Data Sets. For an output direct access data set, CLOSE writes an end-of-file mark.

A data set that is to be preserved after the end of the job must be opened and closed if new material is added to it. Closing causes the job control processor to update its block count information for the data set at the end of a job step. If this count is not updated, a programmer cannot refer to the new blocks in future jobs.

A new directoried data set member must be closed for output. This causes the system to create a directory entry for the new member at the end of the job step. If this is not done, the system is unable to find the new member in the future, and it is erased the next time another new member is added to the data set.

CLOSE for an input data set or for a data set that is to be left "as is" does not disturb the data set's position.

A repositioning code of 01 for a direct access data set causes the block count in the data set's file control block to be set to zero. The block count is the record of the number of blocks read or written by the program. A repositioning code of 02 zeros the block count and disconnects the data sets. It is disconnected in the sense that no further input/output operations will be accepted for the symbolic unit assigned to the data set until it is reassigned or reset.

A new directoried data set member must be closed for output. This causes the system to create a directory entry for the new member at the end of the job step.

The system does not write end-of-file marks after members in formatted directoried data sets. These are data sets in which the ALLOC statement's FMT option was specified when they were created. The system uses other means to determine the end of a formatted directoried data set member. The effect is the same as though the end-of-file mark had been written.

Unit record data sets. CLOSE can be used for unit record data sets to zero the data set's block count and disconnect the symbolic unit assigned to it.

A reposition code of 01 zeros the block count in the data set's file control block. A code of 02 zeros the block count and disconnects the symbolic unit.

Magnetic tape data sets. The action of CLOSE depends on whether the installation standard is to use labels.

In an unlabeled tape shop, CLOSE can be used to write end-of-file marks and to rewind or rewind and unload a volume.

For an output data set, CLOSE writes two end-of-file marks and backspaces over the last one.

For an input data set, nothing is done to the data set unless repositioning is requested.

Repositioning codes can be used to indicate that the tape should be rewound to its load point or rewound and unloaded.

In a labeled tape shop, CLOSE can be used to write and verify trailer labels, write end-of-file marks, reposition a data set, or rewind and unload a volume.

For an output data set, CLOSE writes an end-of-file mark and a standard trailer label. It then writes two end of file marks and backspaces over the last one.

For an input tape data set, CLOSE reads the trailer label and verifies it against the header label and control statement information. If there is any discrepancy between the data set name in the trailer label and the name in its file control block, it is indicated in a return code. The data set is positioned after the end-of-file mark following the trailer label.

A labeled tape also can be closed "as is" with no label processing and without disturbing the data set unless repositioning is requested. If requested, the volume can be rewound or rewound and unloaded.

When CLOSE is executed, register 1 must contain the address of a parameter list. This list consists of one full-word entry, aligned on a full-word boundary, for each system unit to be closed. Each entry contains the address of a four-byte control information area. The first byte of the last entry contains hexadecimal 80, signifying the end of the parameter list.

The control information for each system unit is specified in hexadecimal as follows:

```
CONTRL   DC   XL4'uurrppcc'
```

uu is the SYSUNI index of the system unit to be closed.

rr
- = 00 if the data set is not to be repositioned.
- = 01 if the block counts of direct access and unit record data sets are to be reset to 0 or a magnetic tape is to be rewound to load point.
- = 02 if the system unit is to be disconnected. A magnetic tape also is rewound and unloaded.

pp
- = 00 to close an input data set and to verify an input trailer label on magnetic tape.
- = 01 to close an output data set and update a direct access label, write an end-of-file mark, or create a magnetic tape trailer label.
- = 02 if no end-of-file mark is to be written, and no label processing is desired.

cc is a byte reserved for use by the system if a code is needed to identify errors.

To facilitate device independence, the system ignores control codes that do not apply to the device being used.

After CLOSE has been executed, control returns to the instruction following the SVC. The low-order byte of register 15 contains the hexadecimal code 00 if there were no errors. This code is 04 if any of the designated system units could not be closed, and the cc field of its control word contains one of the following codes:

Hexadecimal Code	Description
01	The system cannot find the data set. The problem may be that an incorrect SYSUNI index was used, the symbolic unit is disconnected, the device is down, or the operator elected to treat the data set as a dummy because of a serious error condition. The position of the data set is unpredictable. It is flagged closed, if possible.
02	Reserved.
04	This code applies only to labeled tape shops. An input tape contained a standard header label but does not contain a standard trailer label. The data set is closed.
08	This code applies only to labeled tape shops. The data set name in an input tape differs from the data set name in its file control block. The data set is closed.

If a serious input/output error occurs during execution of a CLOSE supervisor call, a message requesting operator intervention is written on SYSLOG. The operator can instruct the system to use the data set anyway, treat it as a dummy, or cancel the job. If desired, the data set can be disconnected.

READ/WRITE LEVEL SUPERVISOR CALLS

The five primary input/output supervisor calls at the read/write level are READ, WRITE, CHECK, NOTE, and POINT. These supervisor calls enable a program to read and write data, to reposition a data set, and to ensure that a data transmission terminated satisfactorily. A general discussion of their functioning was given at the beginning of this input/output section.

The applicable SVC codes are:

READ - SVC 4

WRITE - SVC 5

CHECK - SVC 6

NOTE - SVC 7

POINT - SVC 8

The parameters for each of these supervisor calls must include the address of a request control block. Each request control block referred to in a program must be defined by the program as a 40-byte area. The program also must provide the SYSUNI index for the system unit involved. The SYSUNI index must be the first byte of the block. The rest of the block must be filled with zeros. This is illustrated in the examples accompanying the discussion of the individual supervisor calls.

READ - SVC 4

The READ supervisor call is used to transmit one block of data from an input data set to an area of main storage.

When READ is executed, register 1 must contain the address of a parameter list. This list consists of three full words, each aligned on a full-word boundary. The first word contains the address of a request control block. The second word contains the address of the area in main storage where the input data is to go. The third word contains the address of a four-byte count field, which also must be aligned on a full-word boundary.

The count field has the format xx00yyyy, where xx is an incorrect length suppression code and yyyy is a count of the number of bytes to be transmitted. When the read request is executed, yyyy bytes are read from the input data set into the area specified by the second word in the parameter list.

The incorrect length condition occurs when the count field specifies a byte count that differs from the block size of the input records. The system reads one block at a time. If the program specifies a count smaller than the block size, only the number of bytes specified by the program are transmitted. If the count is larger than the block size, only one block is transmitted. In either case, an incorrect length condition occurs. This condition is ignored if the xx byte of the count field is hexadecimal 20. If the first byte is 00, an incorrect length code is posted in the request block. This causes an abnormal condition return the next time an input/output request uses this block.

Specifically, the conditions that cause an incorrect length return code are shown in Table 11.

Table 11. Incorrect Length Conditions

Type of Device	Condition
Tape	For reading, the requested byte count does not equal the number of bytes in a record. For writing, this code is never set.
2315 Disk	For reading or writing, the requested byte count does not equal the block size stated when the data set was created.
2311 Disk	For reading, the requested byte count does not equal the block size of the records. For writing, the requested byte count does not equal the block size stated when the data set was created.
Printer	The requested byte count does not equal the size of the printer's buffer.
Cards	The requested byte count does not equal the size of the card image.

The following sample coding could be used to read a 360-byte block from an input data set on system unit SYS004. Any incorrect length condition is suppressed.

```
READ     EQU  4

SYS004   EQU  20

         DS   0F

RCB01    DC   AL1(SYS004)

         DC   39X'00'

IOBUFF   DS   90F
         .
         .
         .
         LA   1,PARAM

         SVC  READ

         BAL  14,ANALYS
         .
         .
         .
ANALYS   BC   15,CKCOD(15)

CKCOD    BC   15,0(14)

         BC   15,EOF

         BC   15,ERROR

         BC   15,NODTMT

         BC   15,IRQST
         .
         .
         .
         DS   0F

PARAM    DC   A(RCB01)

         DC   A(IOBUFF)

         DC   A(COUNT)

COUNT    DC   X'2000'

         DC   H'360'
```

At the opening of this example, SYS004 is equated with 20, its SYSUNI index value that goes into the first byte of the request control block, and the word READ is equated with its proper SVC code 4. This is done solely for ease in programming. In either case, the actual numeric value could be used at the required places in the subsequent coding.

The following steps set up a request control block and define an input buffer. The addresses of these two areas are the first two entries in the parameter list PARAM at the bottom of the example. The count field is set up to read 360 bytes and a 20 prefix to suppress any incorrect length condition.

The first of the executable instructions loads the address of the parameter list into register 1. The READ supervisor call follows. When the system returns control to the problem program, it will branch to the ANALYS routine to analyze the return code in register 15 to determine how the previous input/output operation on the same request control block turned out. The Branch and Link instruction puts a return address in register 14 to facilitate a return to this point when the ANALYS routine is finished.

ANALYS is not a system routine. It represents the sort of error-checking a problem program should do.

The ANALYS routine uses the return code in register 15 as an index for another branch. If the code is 00, the branch is to the next instruction which returns control to the main program. Otherwise, the branch is to one of the specialized routines provided elsewhere in the program to handle each of the four abnormal conditions.

When the abnormal condition routines are entered, register 1 still contains the address of the parameter list which contains the address of the request control block involved in the abnormally terminated operation. The routine may examine this and the file control block to determine what should be done.

WRITE - SVC 5

The WRITE supervisor call is used to transmit a block of data from an area of main storage to an output data set.

When WRITE is executed, register 1 must contain the address of a parameter list. This list consists of three words aligned on full-word boundaries. The first word contains the address of a request control block. The second word contains the address in main storage of the block of data to be written. The third word contains the address of a 4-byte count field which must be aligned on a full-word boundary.

The count field has the format xx00yyyy, where xx represents an incorrect length suppression code, as for READ, and yyyy represents, in hexadecimal, the number of bytes to be written.

The incorrect length code applies when the program is working with a device that has defined physical limitations. An incorrect length condition is created, for example, by an attempt to write a block larger or smaller than 80 bytes on a card punch. This condition also can be caused by an attempt to write a block larger or smaller than other blocks in the data set.

A program may not write outside the previously defined limits of a direct access data set or data set member. An attempt to do this causes an end-of-extent abnormal return condition.

In other respects, WRITE functions and is programmed in the same general manner as a READ supervisor call.

CHECK - SVC 6

The CHECK supervisor call is used to delay further execution of a problem program until a requested read/write level operation has been completed.

CHECK may be issued at any time following any other read/write level supervisor call. It has no effect, however, after OPEN or CLOSE. Its parameters include the address of the request control block used in the requested operation. Control does not return to the user until all pending operations involving the block have been completed. Before returning, the system posts the return code for the last completed operation in register 15 and resets the return code area of the request control block to 00 so that another operation request can be accepted immediately.

When CHECK is executed, register 1 must contain the address of a parameter list. This list consists of one word, aligned on a full-word boundary, containing the address of a request control block. The parameter list may be the same list used for the operation to which the CHECK applies.

Control returns to the instruction following the CHECK SVC. Register 15 contains 00 if no abnormal conditions were detected in the operation. A 00 return code also appears if the request control block has not been used or if it was examined previously by CHECK and there has been no intervening operation. If the current operation terminates abnormally, register 15 contains one of the non-zero return codes shown in Table 6.

CHECK may be used as follows:

```
SYS003   EQU  19

WRITE    EQU  5

CHECK    EQU  6

         DS   0F

RCB01    DC   AL1(SYS003)

         DC   39X'00'

IOBUFF   DS   90F
         .
         .
         .
         LA   1,PARAM

         SVC  WRITE

         SVC  CHECK

         BAL  14,ANALYS
         .
         .
         .
ANALYS   BC   15,CKCOD(15)
         .
         .
         .
         DS   0F

PARAM    DC   A(RCB01)

         DC   A(IOBUFF)

         DC   A(COUNT)

COUNT    DC   X'2000'

         DC   H'360'
```

This example calls for writing a block of 360 bytes in the data set on system unit SYS003. The incorrect length condition is suppressed. The WRITE supervisor call is followed immediately by a CHECK. This combination causes the system to delay further execution of the program until the write operation has been completed and analyzed and a return code has been posted in register 15. The abnormal return code section of the request control block has been reset to 00, and the block is ready to accept another request.

This example uses the branch to the ANALYS routine to examine the return code in register 15, as did the READ example.

A combination of operations that would result in an error code not being examined should be avoided. If, for example, the three operations, WRITE, WRITE, and CHECK, all use the same request control block, and the first WRITE terminates abnormally, the

second WRITE is rejected. An error code is placed in register 15, and the error section of the request control block is cleared. When CHECK is issued, the system finds no operation in progress and no error code in the request control block, so it resets register 15 to zero and returns to the main program with no indication of the abnormal termination. This condition could be avoided by using CHECK after each WRITE or by entering an analysis routine between the second WRITE and CHECK.

NOTE - SVC 7

The NOTE supervisor call is used to determine the current position of a data set.

NOTE generally is used with the POINT supervisor call, SVC 8, for nonsequential processing of data sets. NOTE causes the system to provide the number of the next block within a data set. The number given is based on the position of the block within the data set or data set member and not on the number of blocks that have been read or written by the program.

When NOTE is executed, register 1 must contain the address of a parameter list. This list consists of two full words aligned on full-word boundaries. The first word contains the address of a request control block. The second word contains the address of a four-byte area where the system is to place the block count. This area also must be aligned on a full-word boundary. The count is stored in it as a hexadecimal value. Control returns to the instruction following the SVC.

The system determines the block count after all pending input/output operations for the data set have been completed. Control returns to the instruction following the SVC.

Register 15 contains the hexadecimal code 00 if no abnormal conditions were detected in the request control block. If the previous operation terminated abnormally and was not checked, the NOTE request is ignored and register 15 contains a non-zero return code.

POINT - SVC 8

The POINT supervisor call is used to reposition a data set to a specified block.

POINT generally is used with the NOTE supervisor call, SVC 7, for nonsequential processing of data sets. POINT causes the system to reposition a data set to a place immediately following a specified block. If, for example, a block count of 3 is given, the data set is positioned so that the next block that would be read by a READ request is the fourth block.

When POINT is executed, register 1 must contain the address of a parameter list. This list consists of two full words aligned on full-word boundaries. The first word contains the address of a request control block. The second word contains the address of a 4-byte area which also is aligned on a full-word boundary. This area contains the desired block count, expressed in hexadecimal.

To reposition a data set so that the next block to be read is the fourth block within the data set (or data set member), the following sample coding might be used:

```
SYS003   EQU  19

POINT    EQU  8

         DS   0F

RCB02    DC   AL1(SYS003)

         DC   39X'00'
         .
         .
         .
         LA   1,PARAM

         SVC  POINT

         BAL  14,ANALYS
         .
         .
         .
PARAM    DC   A(RCB02)

         DC   A(BLKCT)

BLKCT    DC   F'3'
```

The data set is repositioned to a point between its third and fourth blocks. Control returns to the instruction following the SVC. Register 15 contains the hexadecimal code 00 if no abnormal conditions were noted in the request control block. If the previous operation terminated abnormally and was not checked, the POINT request is ignored and register 15 contains a non-zero return code.

As an example of using NOTE and POINT together, a program may issue a NOTE supervisor call to determine the current position of a data set. The program then

determines the number of blocks to be skipped or backspaced, modifies the block count parameter (BLKCT in the previous example) accordingly, and issues a POINT supervisor call that uses the same parameter list as the NOTE.

EXTENDED TAPE VOLUME SUPERVISOR CALLS

The WEF, UNLOAD, and REWIND supervisor calls are used by the system's OPEN and CLOSE routines. All three are available for general use, but problem programs should use the OPEN and CLOSE supervisor calls to obtain the same results. Use of these functions instead of OPEN or CLOSE may cause accidental loss of data or malfunctioning of system routines in later operations on the same data sets. They should not be used for direct access volumes.

The applicable codes are:

WEF - SVC 9

REWIND - SVC 10

UNLOAD - SVC 11

WEF - SVC 9

The WEF supervisor call is used to write an end-of-file mark on magnetic tape volumes.

An end-of-file mark indicates the end of a data set. When a program subsequently reads this data set and detects this mark, an abnormal return condition is recorded in the request control block.

When WEF is executed, register 1 must contain the address of a parameter list. This list consists of one word, aligned on a full word boundary, containing the address of a request control block.

Control returns to the instruction following the SVC. The low-order byte of register 15 contains the hexadecimal code 00 if no abnormal conditions were noted in the request control block. If the previous operation terminated abnormally and was not checked, register 15 contains a non-zero return code and the WEF request is ignored.

REWIND - SVC 10

The REWIND supervisor call is used to reposition a magnetic tape volume to its load point.

Parameter specifications and return codes are the same as for WEF.

UNLOAD - SVC 11

The UNLOAD supervisor call is used to rewind and unload a magnetic tape volume.

Parameter specifications and return codes are the same as for WEF.

The system includes 14 supervisor calls for services not related to input/output. They are shown in Table 12.

These supervisor calls are used to fetch or load program phases from the phase library, terminate a job step or job, manipulate information in the user communication region, and provide for special program check and timer interruption routines.

FLOW CONTROL SUPERVISOR CALLS

The flow control supervisor calls are FETCH and LOAD. They are used to load programs and program segments from the phase library.

The applicable codes are:

FETCH - SVC 12

LOAD - SVC 13

FETCH - SVC 12

The FETCH supervisor call is used to load a program or program segment from the phase library into main storage and transfer control to it.

The system uses FETCH to load and enter a user program. This user program may then invoke FETCH to load and enter phases of a program. FETCH is designed so that the calling routine can pass one full word of information such as the address of a parameter list, to the called routine.

A phase called by FETCH must be loaded and entered at the addresses specified in its phase library directory entry. If another load point is desired, the LOAD supervisor call should be used.

When the FETCH supervisor call is executed, register 1 must contain the address of a parameter list. This list consists of either one or two full words, aligned on full-word boundaries. If there are two words, the system assumes the second word is a parameter to be passed to the new phase. Before the system transfers control to this new phase, it places the contents of this second word into register 1.

The first byte of the first word in the parameter list is a hexadecimal code, either 00 or 80. The 00 code tells the system the list consists of two words, and the second word is to be put into register 1 for use by the new phase. The 80 code indicates that the list consists of only one word and there is no information to be passed.

The remaining three bytes of this word contain the address of an 8-byte location aligned on a full-word boundary elsewhere in main storage. This location must contain the name of the desired phase in EBCDIC characters. If the name is fewer than eight characters, it must be padded on the right with blanks.

To load and enter a phase named PROGNAME, the following coding could be used.

```
FETCH    EQU  12
              .
              .
              .
         LA   1,PARAM

         SVC  FETCH
              .
              .
              .
         DS   0F

PARAM    DC   X'80'

         DC   AL3(LOC)

LOC      DC   C'PROGNAME'
              .
              .
              .
```

The system searches the phase library for an entry named PROGNAME, loads it at the address specified in its directory entry, and enters it at the address specified in its directory entry. On entry to the phase, register 1 contains binary zeros since there was no parameter to be passed. Register 15 contains the address of the entry point. Other registers are unchanged.

If the system is unable to find and load the phase correctly, the job is cancelled.

Table 12. Non-Input/Output Supervisor Calls

Mnemonic	Code	Description
FETCH	SVC 12	FETCH is used to load a program or program segment into main storage from the phase library and transfer control to it.
LOAD	SVC 13	LOAD is used to load a program or program segment into main storage from the phase library.
EOJS	SVC 14	EOJS (End-of-Job-Step) is used to terminate a job step.
CANCEL	SVC 15	CANCEL is used to terminate a job.
GETIME	SVC 16	GETIME (Get Time) is used to obtain the time of day from the system timer.
INSERT	SVC 17	INSERT is used to store information in the system's user communication region.
EXTRACT	SVC 18	EXTRACT is used to obtain the location of the user communication region.
UPSAND	SVC 19	UPSAND (User Program Switch AND) is used to obtain the logical product (AND) of the user program switch byte and the low-order byte of register 1.
UPSOR	SVC 20	UPSOR (User Program Switch OR) is used to obtain the logical sum (OR) of the user program switch byte and the low-order byte of register 1.
STXIPC	SVC 21	STXIPC (Set Exit for Program Check Interruption) is used to notify the system that the program contains a special routine to be entered when certain types of program check interruptions occur.
STXITC	SVC 22	STXITC (Set Exit for Timer Interruption) is used to notify the system that the program contains a special routine to be entered when a timer interruption occurs.
SETIME	SVC 23	SETIME (Set Interval Timer) is used to set a value in the system's interval timer.
RTXIPC	SVC 24	RTXIPC (Program Check Interruption Return) is used at the end of a program check interruption routine to return control to the interrupted main program.
RTXITC	SVC 25	RTXITC (Timer Interruption Return) is used at the end of a timer interruption routine to return control to the interrupted main program.

To load and enter the same phase as in the previous example and pass to it the address of a parameter list, the following coding could be used:

```
FETCH    EQU12
           .
           .
           .
         LA 1,PARAM

         SVCFETCH
           .
           .
           .
         DS 0F

PARAM    DC A(LOC)

         DC A(LIST)

LOC      DC C'PROGNAME'

LIST     (list of parameters or other
           information)
```

Since the first byte of the first word in this parameter list is 00, the system loads the phase as before, but changes the contents of register 1 before entering it. On entry, register 1 contains the address of LIST, since this was the second word in the parameter list. The values in register 15 are the same as in the previous example. Other registers are unchanged.

LOAD - SVC 13

The LOAD supervisor call is used to load a program segment from the phase library into main storage.

LOAD causes the system to search the phase library directory for an entry for the desired phase. Upon finding it, the system loads the phase into main storage, places the address of its main entry point in register 1, and returns control to the calling routine. The phase is loaded either at the address specified in its directory entry or at an alternate address specified in the parameters accompanying the supervisor call. The entry point address that is placed in register 1 is obtained from the directory entry and is modified, if necessary, to reflect any change in loading address.

When the LOAD supervisor call is executed, register 1 must contain the address of a parameter list. This list consists of either one or two full words, aligned on full-word boundaries. If there is a second word, the system assumes that the phase is to be loaded at an address other than the one specified in the phase's directory entry. The alternate address must be within the problem program area.

The first byte of the first word in the parameter list is a hexadecimal code, either 00 or 80. The 00 code tells the system the list consists of two words; the second word contains the address of another full-word, aligned on a full word boundary, that contains the loading address. The 80 code indicates that the list consists of only one word and the phase is to be loaded at the load address in its directory entry.

The remaining three bytes of the first word contain the address of an 8-byte location elsewhere in main storage. This location must be aligned on a full-word boundary and must contain the name of the desired phase in EBCDIC characters. If the name is less than eight characters, it must be padded on the right with blanks.

For example, to load a phase named SBRUTINE, the following coding could be used.

```
LOAD     EQU  13
               .
               .
               .
         LA   1,PARAM

         SVC  LOAD
               .
               .
               .
         DS   0F

PARAM    DC   X'80'

         DC   AL3(LOC)

LOC      DC   C'SBRUTINE'
               .
               .
               .
```

The system finds SBRUTINE, loads it at the address specified in its directory entry, and returns control to the instruction following the SVC. On return, register 1 contains the address of the phase's main entry point. The low-order byte of register 15 contains the hexadecimal code 00, indicating no errors. If the system is unable to find the name of the desired phase in the phase library directory, register 15, on return, contains the code 04. Other registers are unchanged.

The same phase could be loaded at a different location by use of the following coding:

```
LOAD     EQU  13
              .
              .
              .
         LA   1,PARAM
         SVC  LOAD
              .
              .
              .
PARAM    DC   A(LOC)
         DC   A(ALP)
LOC      DC   C'SBRUTINE'
ALP      DC   A(loading address)
              .
              .
              .
```

This sample coding causes the system to find the desired phase and load it at the alternate address specified at location ALP. Control returns, as before, to the instruction following the SVC with the address of the phase's main entry point in register 1. The return codes in register 15 are the same as in the previous example. Other registers are unchanged.

Some precautions are necessary when changing the load point of a phase. If, for example, any other phase contains address constants referring to points in a relocated phase, the address constants must be adjusted to reflect the change in the loading address. The values of inter-phase address constants are determined during linkage editing.

TERMINATION SUPERVISOR CALLS

The termination supervisor calls are EOJS and CANCEL. They can be used for either normal or abnormal termination of programs and program segments.

The applicable codes are:

EOJS - SVC 14

CANCEL - SVC 15

EOJS - SVC 14

The EOJS supervisor call is used to terminate a job step.

EOJS instructs the system to load the job control processor to read control statements and commence execution of the next job step. The next job step may be part of the same program or the first job step in a new program.

A job step is entered by a standard linkage, so it also may be terminated by a standard return sequence.

A program should check before issuing EOJS to ensure that no input/output operations are pending. Completion of an input/output operation after EOJS has been executed could cause malfunctioning of the job control processor.

CANCEL - SVC 15

The CANCEL supervisor call is used to terminate a job.

When a CANCEL supervisor call is executed, the system terminates the current job immediately. A message for the operator is written, and a dump is taken if a dump was requested in the job's control statements. The system then loads the job control processor which reads the system input unit, ignoring all statements until a /& end-of-job control statement is detected. Normal end of job processing is not performed.

COMMUNICATION REGION SUPERVISOR CALLS

The communication region supervisor calls are INSERT, EXTRACT, UPSAND, and UPSOR. They are used for communication between a problem program and the system's user communication region.

The applicable codes are:

INSERT - SVC 17

EXTRACT - SVC 18

UPSAND - SVC 19

UPSOR - SVC 20

USER COMMUNICATION REGION

The user communication region is an area within the system supervisor that may be used by both system programs, such as the assembler, and by problem programs. Programs may read information from this area, but must use supervisor calls to insert or alter information to avoid accidental destruction of system data.

Its contents are as follows:

Word	Byte	Description
0,1	0-7	Date, set by the operator, in the form yyddd.
2	8-11	Address of the first byte of the problem program area.
3	12-15	Address of the last byte available for use by the problem program.
4	16-19	Address of the highest byte in the problem program area filled by a phase loaded by means of any FETCH or LOAD supervisor call.
5	20-23	Address of the last byte in the problem program area filled by the most recent FETCH or LOAD supervisor call.
6,7	24-31	Job name in EBCDIC characters.
8,9	32-39	Job step name in EBCDIC characters.
10	40	User program switch byte. This byte is set to zeros whenever the system reads a JOB control statement.
10	41	Highest assembly error severity. Reset to zero by JOB statement. 00 - Normal. No errors. 04 - Warning messages listed. Execution should be successful. 08 - Error messages listed. Execution may fail. 0C - Severe errors. Execution impossible. 10 - Terminal errors. Job has been cancelled.
10	42,43	Not used.
11	44-47	User interprogram communications area. This area may be used by one job step to preserve information for use by a later job step. This area is set to zeros whenever the system reads a JOB control statement.
12,13	48-55	User intraprogram communications area. This area may be used by one job step phase to preserve information for use by another phase within the same job step. This area is set to zeros whenever the system reads an EXEC control statement initiating a new job step.
14-31	56-127	Up to six 8-byte EBCDIC option parameters from the job step EXEC statement are stored here by the job control processor. If fewer than six parameters are stored, the area is padded on the right with blanks. The area is reset to blanks before the beginning of the next job step. The full 72 bytes of this area and the 8-byte intraprogram communication area also are used by the system and processor programs between job steps for temporary storage of certain control statements.
32-35	128-143	Up to 16 bytes of accounting information are stored here for use by installation routines. This information, in EBCDIC form, is obtained from JOB and/or EXEC statements.
36-xx	144-xxx	Data generated by installation accounting routines may be stored here. The upper limit of the area is determined by the installation when the resident supervisor is reassembled. This field is not included in the IBM distributed system. As distributed, the user communication region occupies 36 words, bytes 0 through 143.

INSERT - SVC 17

The INSERT supervisor call is used to store information in the user communication region.

The system does not require a problem program to provide any information in this area other than that contained in control statements. If a program does use the area, however, it should use the INSERT supervisor call to reduce the chances of accidental destruction of data needed by the system.

It is not necessary to know the location of the region to use INSERT. To refer to information already stored, the location can be determined by use of the EXTRACT supervisor call.

INSERT cannot be used to alter the contents of words 0 through 10, bytes 0 through 43 of the user communication region.

INSERT cannot be used to modify the user communication region permanently. The region is reinitialized when the initial program load procedure is executed.

When the INSERT supervisor call is executed, register 1 must contain the address of a parameter list. This list consists of two words aligned on full-word boundaries. The first word contains the address in the problem program area of the information to be stored in the user communication region. The second word contains the address of a 4-byte area containing control information.

The first byte of control information must be hexadecimal 00. The second byte gives the number of 4-byte words to be stored in the region. The last two bytes indicate where in the region the data is to be stored. This last location is expressed in terms of the word where the system is to start storing the information, the first word in the region being word 0.

For example, to store eight bytes of data in the intraprogram communications area, words 12 and 13, bytes 48 through 55, the following coding could be used:

```
INSERT   EQU  17
              .
              .
              .
         LA   1,PARAM
         SVC  INSERT
              .
              .
              .
PARAM    DC   A(LOC)
         DC   A(CONTRL)
              .
              .
              .
LOC      (data to be stored)
         DS   0F
CONTRL   DC   X'00'
         DC   AL1(2)
         DC   AL2(12)
```

The system stores the eight bytes of information at LOC in the intraprogram communications area and returns control to the instruction following the SVC. The low order byte of register 15 contains the hexadecimal code 00, indicating no errors.

If an attempt is made to use INSERT to store information outside the communications region or in words 0 through 10, nothing is stored, and register 15, on return, contains the hexadecimal code 04.

Other registers are unchanged.

EXTRACT - SVC 18

The EXTRACT supervisor call is used to obtain the location of the communication region.

EXTRACT causes the system to put the address of byte 0 of the communication region in register 1 and return control to the calling program. The address of any particular word or byte within the region is obtained by adding the byte count to this value.

UPSAND - SVC 19 and UPSOR - SVC 20

The UPSAND and UPSOR supervisor calls are used to set or alter the contents of the user program switch byte in the user communications region. This byte is used for communication between job steps and within a job step.

When UPSAND is used, the logical product (AND) of the user program switch byte and the low-order byte of register 1 is stored in the user program switch byte.

When UPSOR is used, the logical sum (OR) of the user program switch byte and the low-order byte of register 1 is stored in the user program switch byte.

When the two bytes are combined by either UPSAND or UPSOR, they are matched bit for bit.

With UPSAND, if each of the corresponding bits is a 1, the result is a 1. If either is 0, the result is 0.

With UPSOR, if either of the corresponding bits is a 1, the result is a 1. If both are 0, the result is 0.

These combinations are illustrated by the following:

A	B	UPSAND	UPSOR
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

When an UPSAND or UPSOR supervisor call is executed, register 1 must contain a mask byte in its low-order positions, bits 24 through 31. The system alters the user program switch byte accordingly and returns control to the instruction following the SVC. No error codes apply.

For example, to set the user program switch byte to all 1's, the following coding could be used:

```
UPSOR    EQU  20
         IC   1,MASK
         SVC  UPSOR
              .
              .
              .
MASK     DC   X'FF'
```

As a result of this UPSOR supervisor call, byte 40 of the communications region is set to 11111111. This byte is reset to all 0's when the system reads a JOB control statement initiating another job.

CONDITIONAL INTERRUPTION SUPERVISOR CALLS

The primary purpose of the conditional interruption supervisor calls is to notify the system that the problem program is providing special routines to be executed when certain conditions occur.

The problem program is able, for example, to inform the system that the program should be interrupted when a specified time interval elapses. A supervisor call gives the system the address of the routine that is to be entered. When the time runs out, the system interrupts the main program and enters the special routine. Execution of another supervisor call at the end of this routine returns control to the main program at the point where it was interrupted.

In addition to timer services, special routines can be provided for certain types of program check interruptions. Program check interruptions occur when the system detects a programming error.

These supervisor calls are in effect only during the job step in which they are issued.

The applicable codes are:

GETIME - SVC 16

STXIPC - SVC 21

STXITC - SVC 22

SETIME - SVC 23

RTXIPC - SVC 24

RTXITC - SVC 25

SAVE AREAS

A program that provides its own interruption routines must also provide save areas. A save area consists of 20 words (80 bytes) where the system can store the contents of the interrupted program's 16 general registers and program status word while the special routine is being executed. This information is restored when control returns to the interrupted program through execution of another supervisor call at the end of the special routine.

The system stores the contents of the interrupted program's registers immediately after the interruption occurs. When the program's interruption routine is entered, register 15 contains the address of the interruption routine, and register 13 con-

tains the address of the save area. Other registers are unpredictable. The interruption routine is entered in the problem program mode and is subject to the same interruptions as the main problem program.

Words 0 and 2 of the save area are used by the system for system information. The contents of register 13 are saved in word 1. The contents of registers 14, 15, and 0 through 12 are saved, in that order, in words 3 through 17. The last two words are for the old PSW.

It is possible for a timer interruption to occur, for example, while a program check interruption routine is being executed. The program, therefore, must provide separate save areas for each type of interruption that it handles. Otherwise, the occurrence of two interruptions at approximately the same time would destroy data needed to return to the interrupted main program.

Save areas and interruption routines must be located in the problem program area of main storage. Each save area must be aligned on a double-word boundary. If the problem program specifies a save area address or interruption routine entry point that is not in the problem program area, the system does not accept the operation.

PROGRAM CHECK

STXIPC - SVC 21

RTXIPC - SVC 24

The STXIPC supervisor call informs the system of the addresses of a program's program check save area and the entry point of its program check interruption routine.

The RTXIPC supervisor call is used at the end of a program check interruption routine to restore the registers and to return control to the interrupted main program.

A program check interruption occurs when the system detects a programming error, such as an incorrect operand, exceptional results, such as fixed-point overflow, or a violation of system or machine restrictions. When a program check interruption occurs, the system takes one of three possible actions:

1. Cancel. The job is terminated and a message to the operator describes the reason for termination.

2. Dump and Cancel. A message to the operator is written. A listing of the contents of all general and floating point registers and of main storage is written, and the job is terminated. This dump must have been requested in the job control statements.

3. Transfer to a User Routine. If an STXIPC supervisor call has been executed prior to the interruption, the system can, in many cases, transfer to a program check interruption routine in the problem program instead of terminating the job. This routine is able to analyze the cause of the program check condition, and it also has the ability to return to the interrupted program.

The system does not transfer to a user's program check routine if the interruption is caused by one of the following conditions:

- Illegal operation code
- Privileged operation in the problem program state
- Addressing

In these cases, the system always cancels the job. A dump is written if it was requested.

If the interruption was not due to one of the foregoing conditions and an STXIPC supervisor call has been executed successfully, the system stores the contents of the 16 general registers and PSW in the user's program check save area and enters the user's program check interruption routine. This routine may examine the registers and PSW and correct the program check condition, ignore it, or cancel the job. If this routine ends with an RTXIPC supervisor call, the system returns to the interrupted program at the point of interruption.

When an STXIPC supervisor call is executed, register 13 must contain the address of the user's program check save area. Register 1 must contain the address of the entry point of his program check interruption routine. These addresses can be changed only by execution of another STXIPC.

If the STXIPC specifies a save area address or an interruption routine entry point that is not within the problem program area, the system does not accept the operation. Control returns to the problem program with a hexadecimal 04 return code in register 15. If there are no errors and

the operation is accepted, the return code in register 15 is 00.

If a program check condition develops while a program check interruption routine is being executed, the job is cancelled.

STXIPC expires at the end of the job step. The address of the user routine or save area can be changed during the job step by executing another STXIPC supervisor call. The effect of an STXIPC can be terminated before the end of the job step by issuing another STXIPC in which the address of the user routine is given as 0. The system will then handle all program check interruptions instead of entering the user routine.

RTXIPC should be used only in those routines that are entered because program check interruption occurred after execution of STXIPC.

When RTXIPC is executed, the system loads the general registers with the contents of the user's program check save area. Control returns to the interrupted program at the point of interruption.

TIMER SERVICES

GETIME - SVC 16

SETIME - SVC 23

STXITC - SVC 22

RTXITC - SVC 25

The system maintains two timer services. The first is the time of day, set by the operator as part of the initial program loading procedure. The second is an interval timer that may be used by a problem program to cause interruptions when a desired time interval expires.

The time, in both cases, is expressed in units of 1/19200 second.

TIME OF DAY

The GETIME supervisor call is used to obtain the time of day. Execution of GETIME causes the system to place a value in register 0. Dividing this value by 19,200 converts it into the number of seconds since "midnight."

Control returns to the instruction following the GETIME supervisor call.

INTERVAL TIMER

The interval timer facility is used by the problem program to cause an interruption after a designated time period expires. This timer occupies a 32-bit word at location 80 in main storage.

A program inserts a value in the interval timer with the SETIME supervisor call. The value is reduced every 1/50 or 1/60 of a second, depending on conditions at the individual installation. When the system is equipped with the High Resolution Timer feature, the value is reduced approximately every 13 microseconds.

The interval timer requires 12 hours to go from its maximum value to 0. An external interruption occurs when the value changes from positive to negative. If the proper supervisor calls have been executed, the system enters the problem program's timer interruption routine. Otherwise the interruption is ignored.

A timer interruption routine is not entered immediately if the interruption occurs while the system is executing a routine in which interruptions are masked out. The system's input/output routines, for example, mask out all interruptions except machine check and program check. Other types of interruptions occuring during such periods are accepted and processed when the masked routine has been completed.

Other factors that may delay entry into a timer routine are pauses, when the system suspends processing pending some operator action, and execution of instructions that tie up the central processing unit for extended periods.

There are three supervisor calls for use with the interval timer.

SETIME is used to insert a value into the timer.

STXITC is used to notify the system that a timer interruption routine should be entered when the inserted value expires.

RTXITC is used at the end of a timer interruption routine to return to the interrupted program.

A SETIME supervisor call is ignored unless an STXITC supervisor call has been executed previously in the job step. The value inserted by SETIME must be the number of 1/19200 second intervals desired before a timer interruption occurs.

When the SETIME supervisor call is executed, register 1 must contain a timer value. This value is inserted in the interval timer, and control returns to the instruction following the SETIME supervisor call. This value can be changed at any time by means of another SETIME supervisor call. An interruption occurs when this value changes from positive to negative. If the interruption occurs after the end of the job step, the system ignores it.

STXITC informs the system of the addresses of a program's timer interruption save area and the entry point of its timer interruption routine. These addresses can be changed at any time during the job step by executing another STXITC. If 0 is specified as the address of the user routine, subsequent timer interruptions are processed by the system, and the user routine is not entered.

STXITC must be executed before SETIME, or SETIME is ignored.

When an STXITC supervisor call is executed, register 1 must contain the address of the entry point of the user's timer interruption routine. Register 13 must contain the address of the user's timer interruption save area. Control returns to the instruction following the STXITC supervisor call.

Register 15 contains the hexadecimal code 00 if there were no errors and the operation was accepted. A return code of 04 indicates that the program specified a save area address or timer routine entry point that was not within the problem program area, and the operation was not accepted.

RTXITC should be used only in timer interruption routines.

When RTXITC is executed, the system restores the general registers from the save area. Control returns to the point of interruption unless the instruction address portion of the old PSW has been changed by the timer interruption routine.

No attempt should be made to return to the interrupted program by means other than this supervisor call.

System utility programs are available for manipulation and maintenance of data sets and volumes.

With the data set routines, all or any part of a data set or directoried data set member may be copied onto disk or magnetic tape, punched into cards, printed, or both punched and printed. Several options are available to permit manipulation of the material being copied and to control the output format. These routines also facilitate the creation and maintenance of libraries and other directoried data sets.

The volume utilities routines initialize disk and magnetic tape volumes for use by the system. For direct access volumes, they also can be used to assign alternates for defective tracks, list the contents of a volume, and update and condense a volume to remove expired data sets and make room for new ones.

The utility routines may be used as an independent job or as a job step within a larger job. Any number of utilities functions can be performed during a single job step, depending upon the availability of required system units for input/output.

An EXEC UTILS job control statement is used to call and enter the system's utilities program. The EXEC statement should be followed on the system input unit, SYSIPT, by utilities control statements. There are no applicable EXEC statement parameters.

The utilities control statements supplement the job control statements. Job control statements, such as ACCESS, ALLOC, and LABEL, define and locate the data sets and volumes that are to be used, and utilities control statements indicate what work is to be done on them.

When, for example, a data set is being copied from cards onto magnetic tape, an ALLOC statement is needed to define the output data set. It is not necessary to have an ACCESS statement for the input if SYSIPT is used and is assigned to the card reader data set. An EXEC UTILS job control statement causes the system to load and enter the utilities program, and a utilities COPY statement initiates the actual copying.

Each utility statement contains an operation field, identifying the statement type, and an operand field, in which any desired options are listed. The operation field must be preceded and followed by at least one blank. No blanks are permitted in the operand field.

Comments and continuation cards may be used in the same way as for job control statements, except that continuation cards do not start with //.

When necessary, messages to the operator are written on SYSLOG. Messages to the programmer are written on SYSLST.

DATA SET UTILITIES

There are two sets of data set utilities statements. The first set is used mainly with sequential data sets. The second applies to directoried data sets.

The sequential data set utilities statements are COPY, PRINT, PUNCH, and PRTPCH. The directoried data set statements are CPYMEM, PRTMEM, PCHMEM, PPMEM, and PCHABS.

The sequential data set statements may be used to copy a single member of a directoried data set without its directory entries. The directoried data set statements must be used to copy more than one member or to copy from one directoried data set into another.

SEQUENTIAL DATA SET STATEMENTS

The COPY statement is used to copy a data set. The data set that is being copied may be on disk, magnetic tape, or punched cards. Input must be on SYSIPT or SYS002. Output goes on SYS003, which may be assigned to any type of device data set.

The PRINT statement is used to obtain a printout of the contents of a data set. Input must be on SYSIPT or SYS002. Output goes on SYSOPT.

The PUNCH statement is used to have the contents of a data set punched into cards. Input must be on SYSIPT or SYS002. Output goes on SYSPCH.

The PRTPCH statement is used to have the contents of a data set printed and punched into cards. Input must be on SYSIPT or SYS002. Output goes on SYSOPT and SYSPCH.

DIRECTORIED DATA SET STATEMENTS

The CPYMEM statement is used to copy one or more members of a directoried data set into another directoried data set. Input must be on SYSIPT or SYS002. Output goes on SYS003. A member is not copied if the output directoried data set already contains a member with the same name.

The PRTMEM statement is used to have one or more members of a directoried data set printed. Printing of each member starts on a new page, preceded by information from its directory entry or entries. Input must be on SYSIPT or SYS002. Output goes on SYSOPT.

The PCHMEM statement is used to have one or more members of a directoried data set punched into cards. Each member is preceded by one header card (or card image) for each of its directory entries. Each member is followed by a /* end-of-data card unless the ENDOUT option is used to specify a different delimiter. The output format is such that it can be used later on SYSIPT as input for another copying action, if desired. Input must be on SYSIPT or SYS002. Output goes on SYSPCH.

The PPMEM statement causes one or more members of a directoried data set to be written as though both PRTMEM and PCHMEM were specified. Input must be on SYSIPT or SYS002. Output goes on SYSOPT and SYSPCH.

The PCHABS statement is used to copy members of the phase library directoried data set (SDSABS). Each member of the phase library is an individual phase. The whole library may be copied, or the INCL and EXCL options can be used to designate specific phases to be copied. The output from PCHABS is suitable as input for system construction and for the absolute loader. It also may be used as input for the linkage editor. (To copy phases from one phase library to another, the statement CPYMEM SIZIN=(720,720) with an appropriate INCL or EXCL option should be used.) Input for PCHABS is from SYSAB2. Output goes on SYSPCH.

Allocation of input/output units is summarized in Table 13.

Table 13. Input/Output Units

Operation	Input	Output
COPY CPYMEM	SYS002	SYS003
COPY* CPYMEM*	SYSIPT	SYS003
PRINT PRTMEM	SYS002	SYSOPT
PRINT* PRTMEM*	SYSIPT	SYSOPT
PUNCH PCHMEM	SYS002	SYSPCH
PUNCH* PCHMEM*	SYSIPT	SYSPCH
PRTPCH PPMEM	SYS002	SYSOPT SYSPCH
PRTPCH* PPMEM*	SYSIPT	SYSOPT SYSPCH
PCHABS	SYSAB2	SYSPCH

GENERAL

ACCESS and ALLOC job control statements are used as elsewhere to define data sets and assign symbolic units.

No member names are cited in the ACCESS statement when the directoried data set utilities statements are being used. Only the name of the directoried data set is given. The INCL and EXCL utilities options specify which members should be copied.

The data set name and a member name are cited in an ACCESS statement only when a sequential data set utilities statement is being used to copy a single member without its directory entries. The member is then treated as though it were a separate data set.

The interpretation of a /* statement depends upon when and where it is detected, as follows:

1. When utilities control statements are being read from SYSIPT, a /* statement signifies the end of control statements for an operation.

2. For an operation using the * notation, such as COPY*, a /* statement on SYSIPT signifies the end of data unless the ENDIN option has been used to specify a different delimiter. If so, a /* statement is treated as data.

3. For a directoried data set operation using the asterisk notation, such as PRTMEM*, a /* statement on SYSIPT indicates the end of a member unless the ENDIN option specifies another delimiter. Therefore, the last input member

must be followed by two delimiter statements, one for end of member and the other for end of data. If this is not done, control statements following the last input member on SYSIPT may be treated as additional data.

UTILITIES OPTIONS

The options that may be invoked in utilities control statements are discussed in this section. Any number of options may be specified in a single statement. Table 14 shows the permissible options by statement.

Options may be specified in any order. If no options are specified, the indicated default options are assumed.

There are two formats for the utilities options. The first uses a keyword followed immediately by an equal sign followed immediately by a parameter selected by the programmer, such as SIZIN=(80,1). Other options require only a single word parameter. The parameter IGRED, for example, informs the system that the redundancy option is being exercised to indicate that copying should continue if an irretrievable read error is encountered.

Control Characters

Records that are being copied may contain special characters to control printers and card punches. These characters specify carriage spacing, page ejection, stacker selection, etc.

The American Standards Association control characters are shown in Table 10 in the "Input/Output Supervisor Calls" section of this publication. One of these characters may be included as the first byte of each logical record in a data set. When the proper utilities options are specified, this character is used for physical control of the output unit.

A programmer indicates the presence of a control character by specifying either the CTLASA or CTLCHR parameter in a LABEL statement. This LABEL statement must immediately follow the ACCESS or ALLOC statement for the output data set. If either of these parameters is specified, it is noted in the data set's file control block and the system uses the character in its input/output operations.

For printing, the presence of a forms control character is indicated by using the SPACE=P option. The first byte in each logical record is then used to control the printer. If SPACE=P is not specified, the character is copied as part of the data record.

For card punching, the option SIZOUT=81 is used when input records contain a stacker selection character that is not truncated during copying. The system then assumes the first character in each output record is a control character that is not to be punched.

Control Options

*

The * notation indicates that input is on SYSIPT. If this notation is not used, input must be on SYS002. This notation is included as part of the operation field of the utilities control statement. It is written, for example, as COPY* or PCHMEM*.

The asterisk notation generally is used when input is in the form of punched cards or card images. System unit SYSIPT can read both the utilities control statements and the data set to be copied.

For directoried data sets, input must be in the form produced by PCHMEM. This means each member must be preceded by one or more header cards, with each card containing the information for a directory entry. (The "System Output" section contains an illustration of a header card.) Each member must be followed by a /* end-of-data statement or a delimiter defined by the ENDIN option. If the * notation is used, neither the INCL option nor the EXCL option can be used to indicate which members should be copied; the entire data set is copied.

Job Control Statements Option

This option instructs the system what to do if it encounters a job control statement other than the /* end-of-data statement.

A job control statement, other than /*, has no meaning to the utility program. If a statement appears, the program assumes it either represents an error or it is part of the input data set that is to be copied into the output data set.

Table 14. Utilities Options

Option	COPY	PRINT	PUNCH	PRTPCH	CPYMEM	PRTMEM	PCHMEM	PPMEM	PCHABS
*	X	X	X	X	X	X	X	X	
Job Control	X	X	X	X	X	X	X	X	
Redundancy	X	X	X	X	X	X	X	X	X
OPENIP[1]	X	X	X	X	X	X	X	X	
OPENOP	X				X				
CLOSIP[1]	X	X	X	X	X	X	X	X	
CLOSOP	X				X				
ENDIN	X	X	X	X	X	X	X	X	
ENDOUT	X	X	X	X	X	X	X	X	
SIZIN	X	X	X	X	X	X	X	X	
SIZOUT	X	X	X	X	X	X	X	X	
TRUNC	X	X	X	X	X	X	X	X	
PAD	X	X	X	X	X	X	X	X	
START	X	X	X	X	X	X	X	X	
SEQIN	X	X	X	X	X	X	X	X	
SEQOUT		X	X	X	X	X	X	X	X
FILL	X	X	X	X	X	X	X	X	
MODE		X				X			
SPACE		X		X		X		X	
Numbering		X		X		X		X	
Type	X	X	X	X		X	X	X	
HEAD		X		X		X		X	
LINES		X		X		X		X	
INCL						X	X	X	X
EXCL					X	X	X	X	X

[1]This option does not apply when input is on SYSIPT.

One of the parameters IGJCL or ABJCL is specified.

IGJCL instructs the system to copy the job control statement from the input to the output data set. The ABJCL parameter instructs the system to terminate the operation if a job control statement is detected. If the statement was on SYSIPT, the job control processor is called to act on it.

If neither parameter is specified, IGJCL is assumed if the input data set is on SYS002. ABJCL is assumed if the input data set is SYSIPT.

Redundancy

This option indicates whether copying should proceed or terminate if an irretrievable read redundancy is encountered.

One of the two parameters, IGRED or ABRED, is specified.

If IGRED is given, the block containing the read error is copied as transmitted and copying of the rest of the data set proceeds. ABRED indicates that copying is to stop if an irretrievable read error is encountered.

If neither parameter is given, ABRED is assumed.

Example:

The purpose is to copy a deck of punched cards onto magnetic tape.

```
//SAMPLE1 JOB

//SYS003 ALLOC SETNAME,2400=FRESH

// EXEC UTILS

COPY*

(Cards to be copied)

/*

/&
```

This example assumes that SYSRDR and SYSIPT are assigned to a card reader data set that contains the control statements and the cards that are to be copied. It further assumes that the output magnetic tape data set is not to be labeled.

SETNAME is the name being assigned to the magnetic tape data set.

2400=FRESH requests a fresh magnetic tape. The system writes a message instructing the operator where to mount it.

Open and Close Options

The following options are used to open and close the input and output data sets. In general, they enable a programmer to specify whether a data set should be repositioned before or after a copying action.

These options correspond with the options an assembler language programmer has when he issues an OPEN or CLOSE supervisor call.

Repositioning may be requested when opening or closing a data set. A direct access data set is positioned to a point just before its first data block for either opening or closing. A magnetic tape, when opened, is positioned similarly, but on closing, it is rewound or rewound and unloaded.

In most cases, the alternative to repositioning a data set is to leave it exactly as it is. This might be done when a data set is being split into two or more smaller data sets or when two or more data sets are being combined into a single data set. For example, part of data set A could be copied into data set B as one utility action. When the copying ended, both data sets could be closed but data set A would not be repositioned. In a subsequent copying action, the rest of this data set could be copied into data set C.

OPENIP

This option specifies whether an input data set on SYS002 is to be repositioned when opened or left as it is.

Its format is:

$$\text{OPENIP=}\begin{Bmatrix}R\\S\end{Bmatrix}$$

where R means that it whould be repositioned, and S means it should stay as it is.

This option does not apply when input is on SYSIPT.

When input is on SYS002 and this option is omitted, OPENIP=R is assumed.

OPENOP

This option specifies whether an output data set on SYS003 is to be repositioned when opened or left as it is.

Its format is:

$$\text{OPENOP=}\begin{Bmatrix}R\\S\end{Bmatrix}$$

where R means the data set should be repositioned to its origin, and S means it should stay as it is.

This option may be used only with the COPY, COPY*, CPYMEM, and CPYMEM* actions. If it is not specified, OPENOP=R is assumed.

CLOSIP

This option indicates whether the input data set on SYS002 is to be

repositioned, repositioned and unloaded, or left as it is when closed.

Its format is:

CLOSIP={R / S / U}

where R means that the data set should be repositioned when closed, S means it should stay as it is, and U means it should be repositioned and unloaded.

For tape, repositioning means rewinding to load point, and unloading involves a physical disconnection. For direct access storage, repositioning means repositioning to the origin of the data set, and unloading involves a logical disconnection so that no input/output operations are accepted for the symbolic unit assigned to the data set until it is reassigned.

This option does not apply when input is on SYSIPT.

When input is on SYS002 and this option is omitted, CLOSIP=R is assumed.

CLOSOP

This option specifies whether an output data set on SYS003 is to be repositioned, repositioned and unloaded, or left as it is.

Its format is:

CLOSOP={R / S / U}

where R means reposition, S means it is to stay as it is, and U means it is to be repositioned and unloaded.

If there is an output data set on SYS003 and this option is not specified, CLOSOP=R is assumed. This option applies with only the COPY, COPY*, CPYMEM, and CPYMEM* operations.

Delimiter Options

The delimiter options enable a programmer to specify one or two characters that will appear at the beginning of a logical record in the input data set. When these characters are encountered in an input record, copying of the input data set terminates.

A hexadecimal zero (a 12-0-1-8-9 punch in extended card code) must never be used as a delimiter character.

ENDIN

This option specifies a one or two character delimiter which, when encountered in an input record, terminates a copying action.

Its format is:

ENDIN='c[c]'

where c represents a delimiter character specified by the programmer.

The delimiter character(s) must be in the first byte(s) of a logical record. When they are detected, copying stops. The last record to be copied is the logical record preceding the record containing the delimiter characters.

This option may be used with the open and close options to split a data set.

When the input data set is on SYSIPT and this option is omitted, ENDIN='/*' is assumed.

When this option is omitted and input is not on SYSIPT, the end of the input stream may be specified by the START option. Otherwise, it is the end of the data set.

ENDOUT

This option specifies a one or two character delimiter that is to be inserted at the start of a logical record at the end of the output data set.

Its format is:

ENDOUT='c[c]'

where c represents a delimiter character to be inserted.

The delimiter characters are inserted as the first byte or bytes in a dummy logical record. The remainder of this dummy record is filled with blanks. This dummy record immediately follows the last regular logical record copied into the output data set but it precedes any fill records that would be added through use of the FILL option.

If this option is omitted, then ENDOUT='/*' is assumed for PCHMEM and PCHMEM* actions. No delimiter record is produced for other operations.

Example:

The purpose is to convert a single direct access data set into two magnetic tape data sets. The characters /% at the beginning of a record indicate where the input data set is to be split.

```
//SAMPLE2 JOB

//SYS002 ACCESS SETA,SDSD='JTR703'

//SYS003 ALLOC SETB,2400=FRESH

// EXEC UTILS

COPY CLOSIP=S,CLOSOP=U,ENDIN='/%'

//SYS003 ALLOC SETC,2400=FRESH

// EXEC UTILS

COPY OPENIP=S

/&
```

SETA is the name of the direct access data set, SETB is the name of the first magnetic tape data set, and SETC is the name of the second magnetic tape data set.

'JTR703' is the identification number of a 2315 Disk Cartridge volume containing the data set to be copied.

/% are the delimiter characters in the direct access data set.

It is assumed that the installation does not use labels on magnetic tapes.

Since input is on SYS002, the IGJCL option is assumed for any job control statements in the data set being copied.

Size Options

SIZIN

This field specifies the size of input blocks.

Its format is:

$$SIZIN=\left\{\begin{matrix}a\\(a,n)\\(,n)\end{matrix}\right\}$$

where a is the size of the input block and n is the number of logical records per block. If n is omitted, it is assumed to be 1. If n is used, a must be divisible by n.

The input block size may be omitted if the * notation is used to indicate that input is on SYSIPT. The system assumes a block size of 80 bytes.

SIZOUT

This option specifies the size of the output record blocks.

Its format is:

$$SIZOUT=\left\{\begin{matrix}b\\(b,m)\\(,m)\end{matrix}\right\}$$

where b is the size of the output block, and m is the number of records per block. If m is omitted, it is assumed to be 1. If m is included, b must be divisible by m.

For COPY and CPYMEM, if this option is omitted, the system assumes the SIZOUT size is the same as the SIZIN size.

For all print operations, m must be 1. If b is omitted, it is assumed to be either equal to the number of characters resulting from the SIZIN, TRUNC, and PAD options or 120 if MODE=D is specified.

For PUNCH, PCHMEM, PRTPCH, or PPMEM, if this option is omitted, SIZOUT=(80,1) is assumed. If this option is specified, b must be either 80 or 81.

Record Manipulation Options

TRUNC

The TRUNC option indicates that each logical record is to be truncated before it is copied.

Its format is:

$$TRUNC=\left\{\begin{matrix}i\\(i,j)\\(,j)\end{matrix}\right\}$$

where i is the index of the first byte that is to be copied in each logical record and j is the number of bytes in each logical record to be copied.

If i is omitted, it is assumed to be 1, indicating that copying is to start with the first byte of each logical record. If j is omitted, the system assumes that byte i and all bytes following it in the logical record are to be copied. All bytes outside the specified boundaries are deleted from the output record.

For example, if TRUNC=(5,50) is specified, 50 bytes of each logical record, starting with the fifth byte, are copied. Any other material in the logical record is lost.

For directoried data sets, this option does not apply to any directory entries that are being copied.

PAD

This option is used to indicate that each logical record should be padded during the copying process.

Its format is:

$$PAD=\left\{\begin{array}{l}k\\([k],u)\\([k][,u],'h')\end{array}\right\}$$

where k is the number of padding characters to be inserted at the start of each logical record, u is the number of padding characters to be added to the end of each logical record, and h is the padding character to be used.

If either k or u is omitted, it is assumed to be 0. If h is omitted, it is assumed to be a blank for printed output and a 0 (hexadecimal '00' bytes) for all else.

The size of the logical record, after padding, cannot exceed the size indicated in the SIZOUT field, unless HEX is specified in the TYPE option field. That is, for CHAR format, the number of characters being copied plus the number of padding characters must be equal to the size specified for output logical records. When HEX format is specified, the output record size must be equal to the number of characters copied multiplied by 2 plus the number of padding characters.

For directoried data sets, this option does not apply to any directory entries that are being copied.

START

This option can be used to delete certain logical records during the copying process.

Its format is:

$$START=\left\{\begin{array}{l}x\\(x,y)\\(,y)\end{array}\right\}$$

where x specifies the first logical record in the data set that is to be copied and y is the number of logical records to be copied.

If x is omitted, it is assumed to be 1. If y is omitted, the xth logical record and all records following it are copied. All logical records preceding the xth record are deleted from the output data set.

This option applies to the input data set as a whole. Copying may start in the middle of one block and terminate in another. Each member or data set is considered separately. Records are reblocked, if necessary in the copying process.

For directoried data sets, this option does not apply to any directory entries that are being copied.

SEQIN

The SEQIN option indicates that each logical record contains an identification field for sequence checking.

Its format is:

$$SEQIN=\left\{\begin{array}{l}p\\(p[,q])\\(p,[q],'ident')\end{array}\right\}$$

where p identifies the first byte of the identification field, q is the number of bytes in the field, and 'ident' is a group of one or more fixed characters appearing in the left-most portion of every identification field.

The 'ident' characters are identical in each logical record identification field and usually would be followed by a sequence number, such as REC0005, REC0006.

A value for p must be specified. If q is not specified, the system assumes the field extends from the pth byte to the end of the logical record. When both p and an 'ident' field are specified, p designates the first byte of the 'ident' field. When the 'ident' field is omitted, p designates the first byte of the sequence number.

Specifying this option causes the system to perform sequence checking. Out-of-sequence records are reported on SYSLST, but copying is not otherwise affected. The identification field is reproduced in the output records unless it is removed or replaced through the use of other options.

For directoried data sets, this option does not apply to any directory entries that are being copied, but

numbering is expected to continue from member to member.

SEQIN is not permitted if the MODE=D option is specified.

SEQOUT

This option creates an identification field for sequence checking of output records.

Its format is:

SEQOUT={r / (r,s) / (r,[s],'ident')}

where r indicates where the first byte of the identification field is to be inserted, s is the number of bytes in the field, and 'ident' specifies a series of one or more characters that are to be inserted in the left-most portion of the identification field in every logical record.

An identification field is superimposed in the indicated position of each output logical record. The sequence value is incremented by 1 for each logical record copied. The size of the logical record is not changed.

If this option is used in a PCHABS statement, r must be 73. If s is included, it must be 8. The 'ident' field, if used, may contain any characters. This could be written:

```
SEQOUT=(73,8,'PHSE1')
```

This causes an eight characters identification field, starting with PHSE1, to be written in the last eight columns of the punched card or card image output.

For directoried data sets, this option does not apply to any directory entries that are being copied, except when PCHABS is used. Numbering continues from member to member.

SEQOUT is not permitted if the MODE=D option is specified.

FILL

The FILL option is used to specify a character to be used for augmenting the final block when the number of logical records it contains is not an exact multiple of the blocking factor.

Its format is:

```
FILL='f'
```

where f is the character to be used.

The programming system requires that all blocks within a data set be the same size. Manipulation of records during the copying process sometimes creates a situation where the final block does not contain enough logical records to make up a complete block. In such cases, the block is filled on the end with enough records containing the fill character to bring it up to the same size as other blocks.

If fill is required and this option is omitted, the last block is padded with binary zeros.

This option is disregarded if fill is not needed.

Type

This option is used to indicate whether output is in character or hexadecimal format.

One of the two parameters, CHAR or HEX, is specified. CHAR indicates character output, and HEX implies hexadecimal.

If neither is specified, CHAR is assumed.

Example:

The purpose is to copy part of a data set from magnetic tape to punched cards. The data set contains 300 logical records, but only records 100 through 199 are to be copied. They are to be truncated and padded in the process.

```
//SAMPLE3 JOB

//SYS002 ACCESS SETD,2400='CRX545'

// EXEC UTILS

  PUNCH  SIZIN=(240,3),START=(100,100),    X
                TRUNC=(,50),PAD=(,30)

/&
```

SETD is the name of the data set to be copied.

CRX545 is the identification number of the volume containing data set SETD.

SIZIN=(240,3) indicates that the input data set contains 240 byte blocks with three logical records per block. Since no SIZOUT option is specified, and the operation is PUNCH, SIZOUT=(80,1) is assumed.

START=(100,100) indicates that copying is to start with the 100th logical record in the data set and 100 records are to be copied.

TRUNC=(,50) indicates that only the first 50 bytes of each logical record are to be copied.

PAD=(,30) indicates that the 30 bytes on the end of each logical record are to be padded with binary zeros.

Example:

As in the previous example, a data set is to be copied from magnetic tape to punched cards. Each input record contains a sequence field in its last five bytes that is to be checked. A new sequence field is to be written in the output records.

```
//SAMPLE4 JOB

//SYS002 ACCESS SETD,2400='CRX545'

// EXEC UTILS

  PUNCH  SIZIN=(240,3),START=(100,100),    X
                SEQIN=(76,,'ID'),          X
                SEQOUT=(76,,'ID')

/&
```

The job control statements and the SIZIN and START options are the same as in the previous example.

SEQIN=(76,,'ID') indicates that the 76th through 80th bytes of each input record contain an identification field that starts with the characters ID.

SEQOUT=(76,,'ID') indicates that a similar identification field is to be written in each output logical record. The count starts from 1.

Output Format Options

MODE

The MODE option designates a format for print output.

Its format is:

$$\text{MODE}=\begin{Bmatrix}\text{D}\\\text{L}\end{Bmatrix}$$

where D indicates that the data set is to be printed in display mode, and L causes printing in list mode.

In display mode, each logical record starts on a new line with its block and record number indicated. List mode output provides one unlabeled line for each logical record.

If list mode is specified and the logical records are too long for one print line, an error message is written and no copying is performed.

The mode of printing of PRTPCH and PPMEM must be list.

If this option is not used, list mode is assumed if the output blocks are larger than the input logical records. Otherwise, display mode is assumed.

By using the SPACE option, single, double, or triple spacing can be obtained for list mode printouts. Display mode records always are single spaced with a blank line between blocks.

SPACE

The SPACE option is used to specify spacing for list mode printouts.

Its format is:

$$\text{SPACE}=\begin{Bmatrix}1\\2\\3\\\text{P}\end{Bmatrix}$$

where 1, 2, and 3 indicate single, double, or triple spacing and P indicates that the first byte in each logical output record is a forms control character. A programmer uses such a character to specify a nonstandard printout operation. The character is inserted as the first byte in the channel command used to print the record. (See SIZOUT option.) P may not be specified for PRTPCH or PPMEM.

If P is specified, the NUM, LINES, and HEAD options may not be used.

This option is permitted only for data sets printed in the list mode. If it is not specified, single spacing is assumed.

Numbering

The NUM option provides page numbering for output printing.

One of the two parameters, NUM or NONUM, may be specified.

If NUM is given, numbering starts on page 1 and an incremented number appears at the top right of each succeeding page.

NONUM indicates that no page numbering is desired. NONUM is assumed if this option is omitted.

If NONUM is specified or assumed, and either the HEAD or LINES option, or both, is specified, each page of the printed output contains one blank line above the heading line.

The numbering option may not be used if the SPACE=P option is specified.

HEAD

This option supplies a standard heading at the top of each page of printed output.

Its format is:

```
HEAD='heading'
```

where 'heading' represents the text of the desired heading. As indicated, the heading must be enclosed within single quote marks, which are not included in the printed output.

If single quote marks are wanted in the heading, they must be specified twice. For example:

```
HEAD='''USER''S REPORT'''
```

This would be printed as 'USER'S REPORT'.

Only one line of heading can be used. Its length cannot be greater than the size of an output block. It is printed, left-adjusted, on each page.

A special form of this option may be used when printing directoried data sets. It is written:

```
HEAD="
```

When this is specified, the name of the member currently being printed is written at the top of the page. No other heading can be specified if this form is used.

The HEAD option cannot be used if the SPACE=P option has been specified.

LINES

This option specifies the number of lines to be printed on each page with one logical record per line.

Its format is:

```
LINES=n
```

where n indicates the number of lines desired per page.

This option may not be used if the SPACE=P option has been specified.

If the HEAD and NUM options are being exercised to obtain a heading and numbering on each page, the heading is counted as one line on the page, but the page number line is not counted. If either the HEAD or NUM option, or both, is specified, and the LINES option is omitted, a page length of 56 lines is assumed.

Example:

The purpose is to print a direct access data set.

```
// JOB

//SYS002 ACCESS SETM,SDSD='VOL005'

  PRINT  SIZIN=(720,9),IGRED,NUM,                X
                HEAD='June Status Report'

/&
```

SIZIN=(720,9) indicates that input blocks are 720 bytes each with nine logical records per block. SIZOUT=80 is assumed.

IGRED instructs the system to continue copying if an irretrievable read redundancy is encountered.

The HEAD option causes the system to print June Status Report at the top of each page.

The system assumes list mode, single spacing, character format, and a page depth of 56 lines.

Output is written on SYSOPT.

Example:

The purpose is to print a directoried data set in which each logical record contains a printer control character.

```
// JOB

//SYS002 ACCESS SETL,SDSD='VOL14'

   PRTMEM SIZIN=(720,9),SPACE=P

/&
```

The entire directoried data set is written on SYSOPT. If this is assigned to a printer data set, printing is controlled by the control character in each logical record. The text of each member is preceded by one line for each of its member names.

Member Selection Options

EXCL

The EXCL option is used when copying a directoried data set to exclude one or more members and their directory entries.

Its format is:

```
EXCL=name

EXCL=(name[,name...])
```

where name is the name of a directoried data set member that is not to be copied.

The names cited in this field may be those of several different members. If a member has more than one entry in the directory, all its names must be cited in this list if it is not to be copied.

A member is not copied if the directory of the output data set already contains an entry with the same name. This member is copied, however, if it has one or more names in the input directory that do not appear in the output directory and are not listed in this field.

Names in this option field need not be listed in any particular order. Nevertheless, members are copied in the order in which their entries appear in the directory of the input data set. (The INCL option must be used if members are to be rearranged during copying.)

No member is copied more than once in a CPYMEM or CPYMEM* operation, regardless of how many directory entries it has. Any directory entries that are not listed in this field are copied with the member they represent.

If this option is specified, the * notation and the INCL option may not be used. If neither the EXCL option nor the INCL option is given, the entire directoried data set is copied.

INCL

The INCL option is used to copy specific members of a directoried data set and, optionally, to rearrange their positions in the output data set.

A programmer can list each member to be copied individually, or he can specify a range of names. When the range technique is used, the programmer lists two names, and the system copies all members whose directory entries fall between the two names specified. Both techniques may be used in the same option.

The format for the first technique, in which members to be copied are listed individually, is as follows:

```
INCL=name

INCL=(name[,name,...])
```

All members whose names appear in this list are copied. The members are copied in the same order as their names appear in the list. Directory entries also are copied and are entered in the directory of the output data set or are printed or punched as header statements for each member, depending upon the operation.

The second technique is based upon the order in which entries appear in the directory. A range of names is given, and all entries and members falling within this range are copied. If an asterisk precedes any name in this list, the corresponding member is not copied unless it has another name that is included within the range. This format is as follows:

```
INCL=[*]name1-[*]name2

INCL=([*]name1-[*]name2[,[*]name3-
[*]name4,....])
```

Each name in such a list is the name of a directoried data set member. The first name in each pair must appear before the second name in the input directory. Except for this, names may be listed in any order, and they are copied in the order in which they are listed.

For example, an input directory may contain the following entries:

NAMEA, NAMEA1, NAMEB, NAMEC, NAMED, NAMEE, NAMEF, NAMEG, and NAMEH.

NAMEA and NAMEA1 refer to the same member.

If INCL=(NAMED-NAMEG,*NAMEA-NAMED) is used, the following members are copied in the order indicated:

NAMED, NAMEE, NAMEF, NAMEG, NAMEA1, NAMEB, NAMEC.

NAMEH is not copied because it falls outside both the specified ranges. The member represented by NAMEA and

NAMEA1 is copied, but the NAMEA entry does not appear in the output directory because it was preceded in this list by an asterisk. NAMED appears in the list twice, but it is copied only once; no member is copied if the output directory already contains a member with the same name.

It is possible, however, to copy a member twice using this option. This would happen if the following option were specified:

```
INCL=(NAMEA,NAMEH,NAMEA1)
```

In the output data set, NAMEA1 would no longer be a name for member A, but would represent a totally independent member which happens to duplicate NAMEA. If INCL=(NAMEA,NAMEA1,NAMEH) had been specified, the member would be copied with both its directory entries, but it would be copied only once. If a member is to be copied only once, its names must be listed consecutively or omitted.

No member or directory entry is copied if its name is not included in this list or is not included in a range of names in this list.

The two methods of listing names may be combined, as in the following:

```
INCL=(NAMEA,NAMEC,NAMEE-NAMEH,
      NAMEB)
```

This would result in the copying of members NAMEA, NAMEC, NAMEE, NAMEF, NAMEG, NAMEH, and NAMEB in that order.

For PRTMEM, if the name of the directoried data set is one of the names cited in the INCL option, the input directory is printed separately as though it were a member. The HEX option should be used since some directory bytes are not otherwise printable.

When the INCL option is specified, neither the * notation nor the EXCL option may be used. If neither INCL nor EXCL is given, the entire directoried data set is copied.

Example:

The purpose is to print selected members of a directoried data set.

```
// JOB

//SYS002 ACCESS SETM,SDSD='ZAN55'

  PRTMEM  INCL=(NAMEA,NAMEC,NAMEE-NAMEH,   X
                NAMEB),SIZIN=(320,4),      X
                MODE=L,SPACE=2,NUM,HEAD=", X
                LINES=40

/&
```

SETM is the name of the directoried data set. It is located on direct access volume ZAN55.

INCL=(NAMEA,NAMEC,NAMEE-NAMEH,NAMEB) indicates which members are to be printed and the order of printing.

SIZIN=(320,4) means that block sizes are 320 bytes with four logical records to a block.

MODE=L requests list mode.

NUM requests page numbering.

HEAD=" specifies that the name of the member being copied is to be printed as the page heading.

LINES=40 specifies a maximum of 40 lines per page.

VOLUME UTILITIES

The volume utilities operations fall into two categories: volume initialization and disk organization. The initialization functions apply to both disks and magnetic tapes. The disk organization functions provide listings of the contents of disk volumes, identification of expired data sets, and condensation to remove deleted and expired data sets.

There are three volume utilities control statements.

The INITIAL statement is for initialization of disk and magnetic tape volumes. This involves assignment of volume serial numbers, checking a disk for defective tracks and assigning alternates for them, and preparation of an initial volume table of contents.

The INITIAL statement also is used for assigning alternate tracks when defective tracks are detected in a previously initialized disk volume.

The SQUEEZE statement updates the contents of disk volumes. Expired data sets are eliminated, and remaining data sets are

shifted to fill any gaps left by deleted data sets toward the beginning of the volume.

The MAP statement provides a listing of the contents of a disk volume, indicating which data sets on it have expired and which still are active. It also updates a disk volume's format 5 space management label. This utility should be executed before using any disk volume created under another System/360 programming system.

INITIAL Statement

The INITIAL statement prepares a volume for use by the system.

For 1316 Disk Pack volumes, the surface is checked for defective tracks, and, if any are found, alternate tracks are assigned. Home addresses and track descriptor records are written, and a volume label and an initial volume table of contents are written.

For a 2315 Disk Cartridge, the surface is checked for defective tracks, and alternate tracks are assigned, if necessary. Addresses are assigned to the fixed sectors, and the volume label and an initial volume table of contents are written.

For either type of disk, this statement may be used to assign an alternate track for a defective track in a previously initialized volume.

For a magnetic tape volume, an ALLOC job control statement must be used to assign symbolic unit SYS003 to the tape to be initialized. A dummy data set name may be used. The system uses information from the ALLOC statement to write a volume label. The system also writes a data set header label. Finally, end-of-file marks and a trailer label are written, where necessary, and the tape is rewound.

When re-initializing a disk, as when changing the size of the volume table of contents, the system checks for unexpired data sets. If any are discovered, it writes a message asking the operator whether to proceed with initialization or terminate the job step.

An INITIAL statement would have one of the formats shown in Figure 10.

TYPE=xxxx
This field identifies the type of device upon which the volume is mounted.

The entry must be one of the following:

SDSD — Single disk storage drive (2315 Disk Cartridge)

1316 — 1316 Disk Pack mounted on a 2311 Disk Storage Drive

TAPE — Any magnetic tape unit supported by the system

DVADR=cuu
This field indicates the physical location of the volume to be initialized.

The cuu field denotes its channel and unit address, where:

c = 0 for the standard multiplex channel

c = 1 or 2 for one of the optional high speed multiplex channels

uu = 00 to FE -- the unit address in hexadecimal

For disk:

```
INITIAL    TYPE=xxxx,DVADR=cuu,VOLID='yyyyyy',VTOC=e
```

For assigning an alternate track for a defective track on a previously initialized disk:

```
INITIAL    TYPE=xxxx,DVADR=cuu,VOLID='yyyyyy',TRACK=t
```

For initializing a magnetic tape volume:

```
INITIAL    TYPE=TAPE
```

Figure 10. INITIAL Statement

VOLID='yyyyyy'

This field is used to assign a serial number to a magnetic tape or disk volume.

For previously initialized disk volumes, this field is used to identify the volume. This applies when the TRACK=t option is specified to obtain assignment of an alternate track for a defective track.

A serial number consists of up to six alphameric characters. These characters are left adjusted in the volume identification field of the volume label and padded on the right, if necessary, with blanks. An identification field containing only blanks is acceptable.

This serial number may be used to identify the volume in ACCESS and ALLOC job control statements when referring to data sets on it.

When specified in the INITIAL statement, the six characters of this field must be enclosed in single quote marks.

VTOC=e

This field indicates the number of entries, e, that will be in the volume table of contents. The value of e is written as a decimal number.

The system uses this information to determine how much space on disk volumes must be reserved for the table of contents. The value of e must be at least 2 greater than the maximum number of data sets that will be stored on this disk.

The system does not permit a program to create a new data set on a disk if there is no space for its entry in the volume table of contents.

TRACK=t

This field is used to assign replacement tracks for defective tracks on a disk.

Detection of a defective track causes the system to cancel a job. Any data written previously on the track is lost.

Defective tracks may be discovered during an installation's routine operations. When this happens, the system writes a message identifying the bad track. The address in this message is used as the value of t in this statement.

When an alternate is assigned, all references to the defective track are treated as referring to the alternate.

No alternate is assigned if the disk's supply of alternate tracks has been exhausted. The defective track is never used again.

When this option is specified, the VOLID option must cite the volume serial number that was assigned to the disk when it was initialized. The serial number cannot be changed at this time.

SQUEEZE Statement

The SQUEEZE statement eliminates expired and deleted data sets from a disk volume and relocates remaining data sets to fill any gaps.

This statement may be used only for data sets on a 2315 Disk Cartridge.

When the SQUEEZE statement is executed, the system treats the space occupied by expired and deleted data sets as vacant. Expiration is based on the expiration date specified in a LABEL job control statement when the data set was created.

Remaining data sets on the volume are shifted to fill any gaps left by the expired and deleted data sets. They are, in a sense, left adjusted so that all vacant space in the volume follows the last remaining data set. The order of data sets in the volume is not changed.

Volume table of contents entries for deleted or expired data sets and old format 5 space management labels are zeroed out, but the table of contents is not otherwise affected.

Members of directoried data sets are not affected unless the entire directoried data set has expired or been deleted. Members of directoried data sets are eliminated with the DELETE and CONDENSE job control statements.

The format 5 label for disk space management is updated to reflect the elimination of any data sets.

The statement's format is:

```
SQUEEZE   DVADR=cuu,VOLID='yyyyyy'
```

where the DVADR and VOLID entries are specified in the same way as for the INITIAL utilities statement. In this case, the VOLID entry is the serial number previously assigned to the volume.

MAP Statement

The MAP statement provides a listing of the contents of a disk volume.

The listing identifies all data sets resident on the volume and indicates which, if any, have expired. Each data set is identified by name, and its extent information also is provided. (Name of data set, type, extent number, low limit of extent, high limit of extent, and expiration indication.)

Execution of this statement also causes the system to update the volume's format 5 space management label to reflect the current status of the disk. This should be one of the first steps in processing a volume that has been in use under another System/360 programming system.

The MAP statement also is valuable in determining whether a volume contains any data sets that have expired.

The statement's format is:

```
MAP DVADR=cuu,VOLID='yyyyyy'
```

where the DVADR and VOLID fields are specified in the same way as for the INITIAL utilites statement. The VOLID entry must be the serial number previously assigned to the volume.

There are two stand-alone disk initialization programs; one for 2315 Disk Cartridges and the other for 2311 Disk Packs. These programs are used to initialize disks for use with the Model 44 Programming System and to assign alternates for defective disk tracks.

These are self-loading programs that do not operate under system control. They perform the same functions as the system's disk initialization utilities routines.

INITIALIZATION

Initialization includes checking disk surfaces for defective tracks and assigning alternates, writing home addresses and track descriptor records, and writing a volume label and an initial volume table of contents.

ALTERNATE TRACK ASSIGNMENT

When the programming system discovers a defective disk track that does not have an alternate, it cancels the current job and writes a message giving the location of the defective track. A stand alone program can then be used to assign an alternate for the defective track. No alternate is assigned if the disk's supply of alternate tracks has been exhausted.

After an alternate has been assigned, the programming system treats all references to the defective track as referring to the alternate track. The defective track is not used again, but any data that had been written on it is lost.

CONTROL STATEMENTS

The stand-alone programs use only one control statement, an INITIAL statement. It has two forms, both of which are illustrated in Figure 11.

The first form, which requires a continuation card, is used for initialization. The second is is used for alternate track assignment.

The fields are as follows:

TYPE=xxxx

This field identifies the type of disk to be initialized. Either 1316 or SDSD must be specified.

DVADR=cuu

This field specifies the physical location of the disk pack. The cuu field provides the channel and unit address, where:

c = 0 for the standard multiplex channel.

c = 1 or 2 for one of the optional high speed multiplex channels.

uu = 00 to FE -- the unit address in hexadecimal notation.

VOLID='yyyyyy'

This field is used to assign an identification code to the disk volume.

An identification code may consist of up to six characters. These characters are left adjusted in the volume identification field of the volume label and padded on the right with blanks, if necessary.

When assigning an alternate for a defective track, this identification field is used to ensure that the correct volume has been mounted. If the identification code in the statement and the code in the volume label do not agree, the operator is given the chance to cancel the job and mount the correct volume.

VTOC=e

The field specifies the number of entries to be made in the volume table of contents. The value of e is written as a decimal number.

The program uses this information to determine how much disk space to reserve for the volume table of contents. The value of e must be at least 2 greater than the maximum number of data sets that will be stored on the disk at one time.

SYSLOG=cuu

This field is used to specify the physical address of a device to be used for communication with the operator.

The cuu field is specified in the same manner as for DVADR=cuu.

EDATE=yyddd

This field specifies an expiration date, where yy indicates the year and ddd the day of the year.

If the disk already contains data sets, their expiration dates are checked against this date. A message is written to the operator when the comparison indicates that a data set has expired. He has the option of cancelling the job or of instructing the program to continue and destroy the expired data set. Data sets that have not expired are not disturbed.

TRACK=ccccchhhh

This field specifies the address of a defective track.

When the programming system discovers a defective track, its address is included in a message to the operator. This address is used as the value of ccccchhhh.

Statement Format

The format for the INITIAL statement is subject the same rules as the programming system's utility statements. The operation code, INITIAL, must be preceded and followed by at least one blank. No blanks are permitted in the operand field, except as part of the volume identification and in the spaces preceding column 72 when a continuation card is being used. A non blank character in column 72 indicates that a continuation card is being used. The text in a continuation card must start in column 16.

```
For disk initialization:
                1                                                   7
1               6                                                   2
 INITIAL TYPE=xxxx,DVADR=cuu,VOLID='yyyyyy',VTOC=e,                 x
                SYSLOG=cuu,EDATE=yyddd
For alternate track assignment:
 INITIAL TYPE=xxxx,DVADR=cuu,VOLID='yyyyyy',SYSLOG=cuu,TRACK=ccccchhhh
```

Figure 11. Stand-alone Disk Program INITIAL Statement

This section contains information about the system's dump facilities and how to use them. It also contains examples of the various forms of output listings, status reports, and card formats produced by the system.

DUMP FACILITIES

A programmer may request a listing of the contents of the general registers and main storage at any time during execution of his program. Similar listings may be obtained at the end of the program. These listings show the status of the system and the program at the time the dump request is made.

Use of the DUMP parameter on the JOB job control statement provides a listing if a program terminates abnormally. Termination is considered abnormal when either the system or the problem program issues a CANCEL supervisor call. The system takes this action when a condition, such as a programming error, develops that requires termination of the job before its planned completion.

A dump can be obtained when a job terminates normally if the dump routine from the module library has been included in the problem program. With this routine, a programmer can also call for a dump at any time during execution of his program without stopping execution. The name of this routine is BOAFDUMP.

The abnormal termination dump provides a listing of the general and floating-point registers and all of main storage. The module library dump routine permits a programmer to specify which portions of main storage should be listed.

The abnormal termination dump is in hexadecimal format. Programmers using the module library routine may specify any of ten formats, and each section of main storage may be listed in a different format.

DUMP ROUTINE

The module library dump routine has two entry points, DUMP and PDUMP. When the DUMP entry point is used, the routine terminates the job step after providing its listing. Use of PDUMP enables a programmer to return to his program after the dump with conditions exactly as they were when the dump was requested. Except for this, the routine functions in the same way regardless of which entry point was used.

The entry point names, DUMP and PDUMP, must be defined as external symbols in any program using the routine. If either of the entry points is not used by the program, it need not be so defined.

The dump routine is incorporated into the program automatically during linkage editing if one of the entry point names is defined as an EXTRN unless the linkage editor's NOAUTO option has been specified. With the NOAUTO option in effect, the name of the routine or one of its entry points must be given in an INCLUDE linkage editor control statement.

The module library routine permits a programmer to specify which portions of main storage should be listed, and he may designate a different format for each portion. A programmer may change these specifications each time he uses PDUMP.

A FORTRAN programmer enters this routine with a CALL DUMP or CALL PDUMP statement followed by variables indicating the areas to be dumped and the format for each.

Assembler language programmers must provide certain additional information and use the calling sequence discussed in the following text.

Assembler Language Coding

To use the dump or any other FORTRAN library subprogram, an assembler language programmer supplies an entry address, a parameter list, and a save area. The following conventions must be observed:

1. Register 15 must contain the address of the library routine's entry point. For the dump program, this is the address of either the DUMP or PDUMP entry point.

2. Register 14 must contain a return address. Except for DUMP, which terminates the job step, the library

routine returns to the address specified in this register.

3. Register 13 must contain the address of a save area.

4. Register 1 must contain the address of a parameter list.

5. The save area must be assembled on a full word boundary. Its size depends on the subprogram, but, in general, it should be large enough for at least 18 words. The subprograms use this area to store the contents of the general registers and other information until they are ready to return control to the calling program.

6. The parameter list also must be aligned on a full word boundary. It consists of a series of four-byte address constants. The last parameter must have a 1 in its high-order bit to denote the end of the list.

7. If information in the floating point registers is to be preserved, their contents must be stored before the library routine is entered.

For additional information about general use of the FORTRAN library subprograms, see IBM System/360 Operating System, FORTRAN IV Library Subprograms, Form C28-6596.

For the DUMP and PDUMP routines, the parameter list contains address of the areas to be listed and of the format for the listing. Any number of areas may be specified.

The first parameter gives the starting address of the area to be listed. The next parameter gives the address of the last byte of this area. The third parameter gives the address of the byte specifying the format for the listing. Multiple areas and formats may be specified.

There are no restrictions on the size of the area to be listed, and it is not necessary to specify more than one. If none is specified, all of main storage is listed.

The save area is used to store the contents of the general registers. This is the first action taken by the dump routine.

Register 13, which at this point contains the address of the save area, is not stored. Other registers are saved in the order 14, 15, 0, 1, and on through 12. When the PDUMP entry is used, these registers are restored before the routine branches back to the main program. The PDUMP routine returns to whatever address is in register 14. This usually would be the first instruction after the branch instruction that was used to enter the dump routine.

Figure 12 contains an example of the coding that could be used to enter the dump routine so that it would return to the main program when the listing is completed.

Dump Format Codes

Following are the codes used to specify dump formats:

0 = hexadecimal

1 = LOGICAL * 1

2 = logical * 4

3 = INTEGER * 2

4 = INTEGER * 4

5 = REAL * 4

6 = REAL * 8

7 = COMPLEX * 8

8 = COMPLEX * 16

9 = literal

Listings using these formats are shown in Figure 13.

```
          EXTRN   PDUMP                 Define external symbols
          EXTRN   DUMP
          .
          .
          .
          LA      1,PARAM               Load parameter list address
          LA      13,SAVE               Load save area address
          L       15,APDUMP             Load PDUMP entry point address
          BALR    14,15                 Branch to PDUMP routine
          .
          .
          .
          LA      1,PARAM               Prepare for end of job dump
          LA      13,SAVE
          L       15,APDUMP
          BR      15
          .
          .
          .
          DS0F
SAVE      DS      18F                   Define save area
APDUMP    DC      AL4(PDUMP)            Adcon for PDUMP entry point
ADUMP     DC      AL4(DUMP)             Adcon for DUMP entry point
PARAM     DC      AL4(FIRST)            Define first area to be listed
          DC      AL4(LAST)
          DC      AL4(F1)               Address of format byte for first area
          .
          .
          .
          DC      AL4(BEGIN)            Define last area to be listed
          DC      AL4(FINAL)
          DC      XL1'FF'               Identify last word in parameter list
          DC      AL3(FX)               Address of format byte for last area
          .
          .
          .
F1        DC      XL1'00'               Request hexadecimal listing
          .
          .
          .
FX        DC      XL1'09'               Request literal format listing
          .
          .
          .
```

Figure 12. Sample Calling Sequence for Module Library Dump Routine

```
CALL PDUMP WITH HEXADECIMAL FORMAT SPECIFIED

00A3E0     485F5E10  00000000  485F5E10  10000000  42100000

006DC8     42B00000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000
006DF8     C0000000  00000000  41200000  41566666  0000000C  41100000

CALL PDUMP WITH LOGICAL*1 FORMAT SPECIFIED

006E1E     T  F

CALL PDUMP WITH LOGICAL*4 FORMAT SPECIFIED

006E10     F  T

CALL PDUMP WITH INTEGER*2 FORMAT SPECIFIED

006E18       10

006E1A      -100

006E1C       10

CALL PDUMP WITH INTEGER*4 FORMAT SPECIFIED

006E20             1          2          3          4          5          6          7          8          9         10
006E48            11         12

CALL PDUMP WITH REAL*4 FORMAT SPECIFIED

006E00      0.20000000E 01   0.53999996E 01

CALL PDUMP WITH REAL*8 FORMAT SPECIFIED

006DC8      0.17599999999999990 03

CALL PDUMP WITH COMPLEX*8 FORMAT SPECIFIED

006DD0               (3.0000000,4.0000000)             (4.0000000,8.0000000)

CALL PDUMP WITH COMPLEX*16 FORMAT SPECIFIED

006DE0             (0.99999999999999900,0.99999999999999900)         (-0.99999999999999900,-0.99999999999999900)

CALL PDUMP WITH LITERAL FORMAT SPECIFIED

006E5C     THIS ARRAY CONTAINS ALPHAMERIC DATA
```

Figure 13. Sample Dump Listings

SAMPLE LISTINGS

The remainder of this section contains sample output listings. Figure 14 shows an assembly listing and Figure 15 shows a linkage editor listing.

ASSEMBLER LISTING

The following text is keyed to the listing shown in Figure 14.

1. Job control statements are listed.

2. A message indicates the time of day, in hours, minutes, and seconds, that execution of the job started.

3. The external symbol dictionary lists each entry point, control section name, and external reference in the module. The dictionary indicates its type, ESD identification number, assembled address, and length.

4. Error flags and their severity codes are listed in this column. The number at the left is the highest severity encountered in the statement.

5. The location counter column indicates the relative addresses assigned by the assembler.

6. The OBJCODE and ADDR columns list the hexadecimal object code generated by the assembler for each machine-executable statement. If the statement contains errors, portions of these fields may be filled with zeros.

7. Program statements are numbered consecutively in the listing even though some of them do not generate object code.

8. Each statement in the source program is listed.

9. These are job control comments statements used, in this case, to aid program documentation. The source program statements also may contain comments that would appear on this listing to the right of the statement.

10. This literal origin code was generated by the assembler.

11. The relocation dictionary contains RLD card information. This includes the relocation and position identifiers, the RLD card flags, and the assembled address. This is used by the linkage editor to identify address constants and their positions in the module.

12. The cross reference dictionary lists each unique symbol used in the program. The DEFN column gives the number of the statement that defines the symbol, and the REFERENCES column gives the number of each statement in which the symbol appears in the operand field.

13. The following information is provided at the end of an assembly listing:

    a. The identification codes of any system error conditions encountered during the job step.

    b. The highest error severity reported during the assembly.

    c. The number of entries in the assembler's symbol table.

    d. The number of statements receiving error flags.

    e. The statement numbers of up to 15 statements that were flagged.

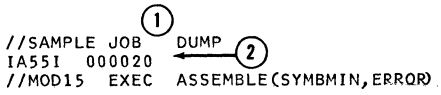


Figure 14. Assembler Listing
(Part 1 of 5)

EXTERNAL SYMBOL DICTIONARY (3)

SYMBOL	TYPE	ID	ADDR	LENGTH	LD ID
EOF	ER	01	000000		
ERR	ER	02	000000		
NODTMT	ER	03	000000		
IROST	ER	04	000000		
ANALYS	SD	05	000000	000060	
SAVE	LD		000034		05
CONTER	LD		000030		05

Figure 14. Assembler Listing
(Part 2 of 5)

```
(4)   (5)        (6)              (7)          (8)
FLGS  L.CTR      OBJCODE   ADDR   STMT         SOURCE STATEMENT

                                  (9) 1  * THIS IS A SAMPLE OF THE COMMENTS THAT CAN BE INCLUDED IN AN ASSEMBLY
                                      2  * LISTING FOR PROGRAM DOCUMENTATION
                                      3  *
                                      4  *
                                      5           ENTRY CONTER
                                      6           ENTRY SAVE
                                      7           EXTRN EOF
                                      8           EXTRN ERR
                                      9           EXTRN NODTMT
                                     10           EXTRN IROST
          000000                     11  ANALYS   CSECT
          000000   05A0              12  BEGIN    BALR  10,0              COMMENTS ALSO MAY BE
          000002                     13           USING *,10              INCLUDED IN INDIVIDUAL (9)
     (4)  000002   5050 A032  00034  14           ST    5,SAVE            STATEMENTS
     / \  000006   5850 A02E  00030  15           L     5,CONTER
   2  =   00000A   5A50 0000  00000  16           A     5,=1
          00000E   5050 A02E  00030  17           ST    5,CONTER
          000012   5850 A032  00034  18           L     5,SAVE
          000016   50E0 A032  00034  19           ST    14,SAVE
                                     20  *
                                     21  *              MAIN BRANCH TABLE
                                     22  *
          00001A   47FF A01C  0001E  23           BC    15,CKCOD(15)
          00001E   07FE              24  CKCOD    BR    14
          000020   47F0 A036  00038  25           BC    15,BEOF
          000024   47F0 A040  00042  26           BC    15,BERR
          000028   47F0 A04A  0004C  27           BC    15,BNODTMT
          00002C   47F0 A054  00056  28           BC    15,BIRQST
                                     29  *
          000030                     30  CONTER   DS    1F
          000034                     31  SAVE     DS    1F
   1  A   000038   58E0 A03C  0003E  32  BEOF     L     14,EOFA
          00003C   07FE              33           BR    14
          00003E   00000000          34  EOFA     DC    AL4(EOF)
          000042   58E0 A046  00048  35  BERR     L     14,ERRA
          000046   07FE              36           BR    14
          000048   00000000          37  ERRA     DC    AL4(ERR)
   1  A   00004C   58E0 A050  00052  38  BNODTMT  L     14,NODTMTA
          000050   07FE              39           BR    14
          000052   00000000          40  NODTMTA  DC    AL4(NODTMT)
          000056   58E0 A05A  0005C  41  BIRQST   L     14,IRQSTA
          00005A   07FE              42           BR    14
          00005C   00000000          43  IROSTA   DC    AL4(IROST)
                          (10)    +  44   LTORG
          000000                     45           END   BEGIN
```

Figure 14. Assembler Listing
(Part 3 of 5)

(11) RELOCATION DICTIONARY

REL.ID	POS.ID	FLAGS	ADDRESS
01	05	0C	00003E
02	05	0C	000048
03	05	0C	000052
04	05	0C	00005C

Figure 14. Assembler Listing
(Part 4 of 5)

```
                                CROSS-REFERENCE (12)

    SYMBOL     DEFN   REFERENCES

    ANALYS       11
    BEGIN        12     45
    BEOF         32     25
    BERR         35     26
    BIRQST       41     28
    BNODTMT      38     27
    CKCOD        24     23
    CONTER       30      5    15    17
    EOF           7     34
    EOFA         34     32
    ERR           8     37
    ERRA         37     35
    IRQST        10     43
    IRQSTA       43     41
    NODTMT        9     40
    NODTMTA      40     38
    SAVE         31      6    14    18    19

**  LA01I ERROR (13)A                    (13)B                    (13)C

                          HIGHEST SEVERITY   12     NUMBER OF SYMBOL TABLE ENTRIES  20

                  3 STATEMENT FLAGGED IN THIS ASSEMBLY
                (13)D
                    16    32    38 (13)E

/E
```

Figure 14. Assembler Listing
(Part 5 of 5)

LINKAGE EDITOR LISTING

All modules referred to in this listing were assembled in one or more previous jobs. Some are resident in the module library. MODULE statements are used to enter the others into the SDS000 system data set.

The following text is keyed to the listing shown in Figure 15.

1. Job control statements supplied by the programmer.

2. Name of the program; taken from the name field of the JOB statement.

3. The date.

4. Each linkage editor control statement supplied by the programmer is listed.

5. These REP statements are supplied by the programmer. They are for the module referred to by the immediately preceding INCLUDE A,L statement. The linkage editor alters the module as indicated by the REP statements.

6. An error message. This one means that there is no length in the last or only control section in the module incorporated into the phase by the immediately preceding INCLUDE D,L statement.

7. The AUTOLINK notation is written by the linkage editor. It identifies a module that is being incorporated into a phase as a result of an automatic search of the module library.

8. This column of the linkage editor MAP lists the phase names.

9. This column indicates the main entry point of each phase.

10. - 11. These columns provide the addresses of the first and last main storage bytes used by the phase.

12. This column gives the number of the first block in the phase library where the text for this phase is written.

13. This column identifies entry points and control sections within the phase. (If there were COMMON control sections or unresolved external symbols, they also would be indicated here.)

14. This column indicates the name the programmer has assigned to the entry point, control section, COMMON, or external symbol.

15. This column provides the main storage loading address of each entry point and control section.

16. This column provides the relocation factor for each control section. For a COMMON entry, this column represents the length of the COMMON control section.

17. This notation indicates that the phase was specified as a ROOT phase.

18. An asterisk indicates an entry point that was defined but not referred to.

19. This message indicates the severity of the most serious error detected during the job step. If the severity is greater than 4, the phases are not retained in the phase library, and the system will not accept them for execution.

Note: For the program that produced this listing, the beginning of the problem program area was set to 4FD0. This would not apply with the version of the programming system distributed by IBM.

```
//SAMPLE  JOB
//        EXEC (1) LNKEDT(KEEP)
```

Figure 15. Linkage Editor Listing
(Part 1 of 3)

```
                                                (2)
 67/134 (3)                                  SAMPLE   LINKAGE EDIT

LIST                MODULE A
LIST                MODULE B1
LIST                MODULE B2 (4)
LIST                MODULE C
LIST                MODULE D
LIST                MODULE E
LIST                PHASE TLEO013P1,ROOT,NOAUTO
LIST                INCLUDE A,L
LIST      REP  00051A 001A066           CORRECTION 1
LIST      REP  000542 001A06A           CORRECTION 2 (5)
LIST      REP  00056A 001A06E           CORRECTION 3
LIST      REP  000562 001ACB2           CORRECTION 4
LIST                PHASE TLE013P2,*+400
LIST                INCLUDE B1,L
LIST                INCLUDE B2,L
LIST                PHASE TLE013P3,CSECT22(TLE013P2)+400,NOAUTO
LIST                INCLUDE C,L
LIST                PHASE TLE013P4,ENTRYC32(TLE013P2)+1200
LIST          (6)   INCLUDE D,L
ERROR     KA04I     END

LIST      AUTOLINK     CSECTC22
LIST                PHASE TLE013P5,*-400
LIST          (7)   INCLUDE E,L
LIST      AUTOLINK     CSECTC22
LIST                ENTRY
```

Figure 15. Linkage Editor Listing
(Part 2 of 3)

	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)
67/134	PHASE	TRANSFER ADDR.	LOCORE	HICORE	BLOCK NO.	ESD TYPE	LABEL	LOADED	REL-FACTOR
ROOT	TLE013P1	004FD0	004FD0	005CEB	397	CSECT	CSECTC11	004FD0	004FD0
(17)	TLE013P2	005E80	005E80	0064C7	402	CSECT	CSECTC21	005E80	005E80
						ENTRY	FIRSTC21	005E80	
						ENTRY	MIDDLC21	006010	
						ENTRY	LASTC21	00619C	
						ENTRY	ENDC21	0061A0	
					(18)	* ENTRY	ENTRYC32	006010	
						CSECT	CSECTC22	0061A0	005E80
						ENTRY	FIRSTC22	0061A0	
						ENTRY	MIDDLC22	006330	
						ENTRY	LASTC22	0064C0	
						ENTRY	ENDC22	0064C4	
	TLE013P3	006330	006330	006333	405	CSECT		006330	006330
	TLE013P4	0064C0	006430	0067DF	406	CSECT		0064C0	0064C0
						CSECT	CSECTC22	0064C0	0064C0
						* ENTRY	FIRSTC41	0064C0	
						* ENTRY	MIDDLC41	006650	
						* ENTRY	LASTC41	0067DC	
						* ENTRY	ENDC41	0067E0	
	TLE013P5	006650	006650	006977	408	CSECT		006650	006650
						CSECT	CSECTC22	006658	006658
						* ENTRY	FIRSTC41	006658	
						* ENTRY	MIDDLC41	0067E8	
						* ENTRY	LASTC41	006974	
						* ENTRY	ENDC41	006978	

LINKAGE EDITOR HIGHEST SEVERITY WAS 4 (19)
/&

Figure 15. Linkage Editor Listing (Part 3 of 3)

This section lists messages written by system programs to the programmer. Such messages may document the progress of a job, indicate errors, or warn of possible error conditions.

Each message has a five character format. The first two characters identify the system program that is writing it. The next two characters, both digits, indicate which of several possible messages applies, and the final character indicates whether immediate action is required.

For example, in the message MA53I, the letters MA identify a linkage editor message, 53 indicates which linkage editor message, and the I means that the message is for information only. When the final character is A, instead of I, some operator action usually is required. Operator messages are written on SYSLOG. Programmer messages are written on SYSLST.

Some messages include descriptive text in addition to the code number. Some usually are accompanied by another message that provides additional information about a condition.

In the case of errors, a message is written as soon as possible after detection of an error condition. Whenever possible, a card image or descriptive text is provided to help locate the source of the error. Many times a statement or action that causes an error message would be all right if it were not for a real error that occurred earlier in the program but could not be diagnosed as one by the system at that time.

ASSEMBLER ERROR INDICATIONS

The assembler examines each statement in an input source program. It checks for violations of the language's syntactic rules and for gross inconsistencies that would interfere with the assembly process.

When errors are found, they are indicated by single-character flags in the first column of an assembly listing. These flags appear to the left of the statement causing the error condition. The meaning of each flag is explained in the following text.

At the end of an assembly listing, there is a count of the number of statements flagged and the identification numbers of up to 15 flagged statements. Another type of assembler error message also may appear at the end of a listing. These messages use a five character code, such as LA05I, to identify system errors detected during execution of the job step.

The discussion of errors in this section applies only to the assembling of a program. The assembler's UPDATE feature has its own error messages which are shown in Table 16 at the end of this section.

Assembler Diagnostic Process

The assembler makes two passes over the input program. The first pass scans the free-format card image and transforms it into an internal format. During this pass, it uncovers errors in punctuation, use of terms, and the number of operands in a statement.

The assembler also uses the first pass to allocate main storage and assign location values to symbols. For this reason, some symbols must be self-defining or must be defined before they are used in the operand of a statement. For example, the length specification and duplication fields of a DC statement must be processed during the first pass so that storage allocation can be completed. Therefore, the assembler must be able to evaluate immediately any symbols that appear in these fields of a DC statement.

The second pass completes the assembly process. In this pass, the assembler evaluates all symbolic terms and all expressions that were not resolved during the first pass. Errors that turn up during this pass include undefined symbols, erroneous relocatability, and values that are too large for their context.

When an error is detected in either scan, the assembler issues a flag and stops processing the statement. This means that another error later in the same statement may not be detected. Failure to process a statement may generate new errors in other statements. If, for example, an error is detected and the assembler stops processing a statement that defines a term, all other statements using that term are considered to contain an undefined symbol.

Error Flags and Severity

Error flags are shown in the first column of an assembly listing to the left of the statement to which they apply. Occasionally, an error condition may generate more than one flag for a single statement. The first flag usually represents the most serious condition.

Error conditions are ranked according to severity. This represents an attempt to evaluate the damage the error condition represents to the program. The codes are 1, 2, 3, and 4. One of these numbers appears to the left of the error code or codes for a line. It represents the highest severity condition in the statement. This could appear on a listing as 3 QN, when Q has a rating of 3 and N is rated at 1.

A severity of 1 is a warning message. It calls attention to a condition that appears to be an error. Whether it is actually an error would depend upon conditions that prevail when the statement is executed.

A severity of 2 is a more serious condition. It usually applies to errors that can generate other error conditions throughout a large part of the program. An assembled module containing errors of this level will be accepted for linkage editing, but it cannot be executed successfully unless REP cards are inserted in the assembler output module deck to correct the errors. If the errors are not corrected before linkage editing, the system will not accept the edited module if an attempt is made to execute it using an EXEC statement with a blank program name field.

A severity of 3 is assigned to errors that make the program impossible to execute.

A severity of 4 applies to conditions, such as permanent input/output errors, that affect the assembler's ability to continue. The job is cancelled.

A message at the end of an assembly listing notes the highest severity level encountered during the assembly. This rating includes any of the LAxxI series of system error messages that may have occurred during the job step. (These messages are documented in the alphabetical listings later in this section.) In this message, the severity levels used for the assembler flags are multiplied by 4 to conform to system standards. If the highest severity flag is 2 and there are no LA system errors, this message gives the highest severity for the job step as 8.

Another message at the end of the assembly listing gives the numbers of up to 15 flagged statements. This message is intended to make it easier to find statements containing errors. Only statements that appear after a START instruction can be included in this list.

Assembler Flags Summary Table

Table 15, which follows the list of flags and their meanings, relates several of the most serious error flags to the requirement for each type of assembler statement. The table is arranged by statement type and, further, by operand type within each statement.

Most of the assembler flags have a general meaning. An F flag, for example, means that that statement has an error in format, such as incorrect punctuation. The F flag columns of the table indicate the major format requirements for each statement. An erroneous statement can be checked against this listing to locate the cause of the error.

The "Name Requirement" column indicates whether the statement requires a name field, whether the name field must be blank, or whether a name is optional.

The "Operand" column indicates whether the operand field of a statement must be a term or an expression, must be blank, or may, optionally, be blank.

An F or T flag is issued if an operand is specified properly but is not one of those indicated in the columns headed "Types of Term Allowed." The F flag, in this case, applies only to literals. The T flag applies to normal symbols, self defining symbols, variable symbols, sequence symbols, and the asterisk.

The "Leading Character" column indicates when an operand field must start with certain characters and it identifies the required characters.

The "Punctuation Allowed" column indicates the choice of characters that may follow an operand. An F flag is issued if other punctuation is used.

The "Relocation Requirement" column indicates the permissible relocation attribute for each type of operand.

The "Previously Defined Symbol Required" column shows which operands must meet this requirement. This means the symbol must be defined by appearing in the name field of a

statement or as the operand of an EXTRN statement before it can be used in the operand field of any other statement. This is required so that the assembler can complete storage allocation in its first pass.

An S or V flag is issued when the value of an operand is greater than the limit shown in the column "Maximum Value of Operand." The S flag appears when the value is required for the first pass. The V flag appears for values required during the second pass.

Flags and Their Meanings

The following text lists each error flag and a description of the condition it reflects. Severity is indicated in parentheses following the description of the error condition.

Error Code	Description
A	An address in a machine instruction appears to have improper boundary alignment. This is not an error if, during execution, indexing is used to provide correct alignment. (1)
B	There is no base register within the range needed to address an operand of a machine instruction. The range of a single base register is 4096 bytes. (3)
C	A symbol contains more than 8 characters. (2)
D	A DROP statement refers to a register that is not being used. Either the register was dropped previously, or it was never specified in a USING statement. (1)
E	A proper register is not specified in a machine instruction. (2) 1. A double-shift, full word multiply or divide instruction does not specify an even numbered general register. 2. A floating-point instruction does not specify floating-point register 0, 2, 4, or 6.
F	A statement contains a format or syntax error; the punctuation following a term is erroneous, the term cannot be classified, or a multi-term expression is specified incorrectly. Term punctuation: The punctuation following a term is invalid for the type of term. (2) (See summary) An imbedded blank terminates the operation code or operand. (2) Punctuation between terms is missing. (2) The operand is not followed by at least one blank. (1) Term classification: An error exists at the start of a term so that it cannot be classified. (2) A symbol starts with a numeric character. (2) The second character of a variable symbol (where the first character is &) or sequence symbol (where the first character is .) is not alphabetic. (2) Terms, miscellaneous: An operand, term, or expression is absent. (2) A term contains an error. (2) A literal is used as an operand where it is not allowed. (2) (See summary) Expressions: A multi-term expression is used where only a single term is allowed. (2) (See summary) Punctuation other than + - * or / appears in an expression. (2) A multi-term expression contains more than three terms. (2)

Error Code	Description
	An expression starts with an arithmetic operator. (2) An expression contains two or more operators in succession. (2) A multi-term or parenthetical expression contains a literal. (2) Specific Instructions: DC-DS: The type code cannot be determined. (2) TITLE: The operand field does not start with a '. (1)
G	The duplication or length specification field of a DC or DS statement contains an error. This flag is usually preceded by another flag that identifies the error more specifically. (2) 1. The term or expression specifying the length or duplication factor is written improperly. 2. The value of the duplication factor is greater than 65,535. 3. The first character of the length specification (following the L) is incorrect. 4. The length specification is greater than is allowed for the indicated type of DC or DS. 5. The length specification is equal to zero.
H	The constant subfield of a DC or DS statement contains an error. This flag is often preceded by another flag that identifies the type of error more specifically. (2)
I	Statement is ignored. This code appears with another error code that indicates the cause of the action. (1)
J	A statement is not properly positioned. It is ignored. (2) 1. ICTL: This is not the first statement of the program. 2. START: This statement follows some other statement that changes or depends upon the setting of the location counter.
K	An expression does not meet the rules of relocatability. 1. An expression contains an unpaired relocatable term preceded by a minus sign. (2) 2. An expression contains more than one unpaired relocatable term. (2) 3. A multiply or divide operation involves a relocatable term. (2) 4. DC address constant: A symbol in the constant is defined within a DSECT. (3) 5. ENTRY: The operand symbol is the name of a statement within a COM or DSECT or is absolute. (3)
L	A name field is present in a statement that does not allow one. The name field is ignored. (1)
M	A symbol is multiply defined. The symbol has been defined elsewhere in the program as the name of an assembler statement or as the operand of an EXTRN. (2)
N	The name field is erroneous. A blank name is assumed. (1) 1. A name is not followed by a blank. 2. A name contains an improper character.

Error Code	Description
	3. A name contains too many characters. 4. A name starts with an invalid character. 5. A name is composed of an invalid term. 6. TITLE: A name field contains more than four characters; some characters are not alphameric. For CSECT or START, the severity for an erroneous name field is 2 rather than 1.
O	The operation code is erroneous. (2) 1. The operation code is misspelled or unrecognizable. 2. The operation code is not followed by a blank. 3. The name field contains an embedded blank or starts after the Begin column.
Q	A statement that requires a correct name field does not have one. Either the statement does not have a name, or the name was blanked out or ignored because of an error. (3) 1. CSECT or START: The program contains more than one unnamed CSECT or START. 2. SETA or ANOP: The name field does not contain a correct symbol. (2)
R	An expression or term has an incorrect relocation attribute. (2) (See summary) USING: A statement specifying register 0 does not have a relocatable address expression.

Error Code	Description
	DC-DS A Type: A relocatable expression is specified when the length specification is 1 or 2.
S	The value of an expression is out of range and is not used. (2) 1. An expression has an intermediate value greater than $2^{31}-1$ 2. An expression has a final value greater than the maximum allowed; $2^{24}-1$ generally, or $2^{23}-1$ in SETA and AIF operand expressions. 3. A self-defining term has a value greater than $2^{16}-1$ when the operand is required to be a single term. 4. A self-defining term has a value greater than $2^{32}-1$ when the operand can be an expression.
T	The operand contains an invalid term. (2) (See summary) 1. An operand is composed of an improper type of term. 2. An error within a term causes it to be misclassified as another type of term.
U	A statement contains an undefined symbol. (2) 1. The symbol does not appear as the name of an instruction or as the operand of an EXTRN. 2. A statement that requires previously defined symbols contains a symbol that has not appeared as the name of an instruction or as the operand of an EXTRN. (See summary) For ENTRY, the severity is 1.

Error Code	Description
V	The value of an expression or term is too large. It has been truncated. (2) (See summary)
Y	The value of a relocatable expression is negative. (2)
1	Only one statement of this type is allowed per assembly. (3)
2	A statement contains an improper operand. (2) 1. CNOP: The w field is neither 4 nor 8; or if w is 4, b is neither 0 nor 2; or if w is 8, b is not 0, 2, 4, or 6. 2. ICTL: The operand is neither 1 nor 25.
3	The operand of an ORG statement contains an error. (3) 1. The value of the operand is absolute. 2. The value of the operand refers to a location in a different control section. 3. The value of the operand is less than the original address value of the control section.
4	DC-DS: The statement specifies a constant value that is too large or too small. (2)
7	There is an error at right end of statement card. 1. Column 72 is not blank. (1) 2. The operand field is not terminated correctly. (2)
.	Mask bits 37 - 39 of a channel command word are not zeros. (2)
(	There is a parenthesization error within an expression. (2)

Error Code	Description
	1. An expression contains more than one level of parentheses. 2. A parenthetical expression has not been closed with a matching right parenthesis.
+	The program contains too many statements of this type. (3) 1. ENTRY: The maximum is 100. 2. For control sections, the maximum is 255, including CSECT, DSECT, COM, START, and EXTRN.
'	A self-defining constant or character string is incorrect. (2) 1. A hexadecimal self-defining term is longer than six characters. 2. A character self-defining term is longer than three characters. 3. A self-defining term or constant contains no characters. 4. A character self-defining term, DC constant, or a TITLE character string does not contain a closing ' or it contains an unpaired &. 5. A hexadecimal self-defining term or constant contains a non-hexadecimal character.
*	The program contains too many symbol table entries. (3) 1. The assembler's symbol table is full. To correct, use the symbol table parameter on the EXEC statement to request a larger symbol table and then reassemble; or divide the program into several assemblies.

Error Code	Description
	2. The symbol table was found to be full while processing a literal.
=	A literal is not properly specified. (2)
)	The program or one of its control sections is too big. (3) 1. The assembler's location counter exceeds 16,777,215. 2. CSECT, COM, DSECT: The length of the previous control section exceeds 65,535 bytes. 3. END: The length of the current control section exceeds 65,535 bytes.

Table 15. Assembler Error Summary
(Part 1 of 2)

	L, N, or Q Flag		F Flag	T Flag					F Flag			R Flag	U Flag	V or S Flag[8]	
				Types of Term Allowed											
Statement type/ operand type	Flag	Name Requirement	Operand	Norm Symb	Self Def Symb	Var Symb	*	Seq Symb	Lit	Lead Char	Punct. Allowed	Reloc Rqmnt	Prev Def Symb Rqd	Flag	Maximum Value of Operand
Machine instructions	N	Optional[10]													
R1 and R2 fields			Term	NS	SD	VS					Blank or ,	Abs		V	15
I and I2 fields			Expr	NS	SD	VS	*				Blank	Abs		V	X'FF'
D1 and D2 fields [1]			Expr	NS	SD	VS	*				Blank or ,	Abs		V	4095
S1 and S2 fields [1]			Expr	NS	SD	VS	*		Lit[2]		Blank or ,	Abs or rel		V	$2^{24}-1$
(X,B) subfield set			Term[11]	NS	SD	VS				(	, or)	Abs		V	15
AGO	N	Opt sequence symbol	Term					SS			Blank				
AIF	N	Opt sequence symbol													
First arithmetic expression			Expr		SD	VS				(	Blank			S	-2^{23} to $+(2^{23}-1)$
Relational operator			Term	NS[3]							Blank				
Second arithmetic expression			Expr		SD	VS					)			S	-2^{23} to $+(2^{23}-1)$
Sequence symbol			Term					SS			Blank				
ANOP	Q	Req sequence symbol	Blank												
CCW	N	Optional													
Command code operand			Term	NS	SD	VS					,	Abs		V	X'FF'
Data address operand			Expr	NS	SD	VS	*		Lit		,	Abs or rel		V	$2^{24}-1$
Flag operand			Term	NS	SD	VS	*				,	Abs		V	X'F8'
Count operand			Term	NS	SD	VS					Blank	Abs		V	$2^{16}-1$
CNOP	L	Blank													
Byte operand			Expr	NS	SD	VS					,	Abs	Yes	2	6
Word operand			Expr	NS	SD	VS					Blank	Abs	Yes	2	8
COM	N	Optional	Blank												
CSECT	Q or N	Optional[6]	Blank												
DC/DS	N	Optional													
Duplication factor[9]			Term[11] or Expr[11]	NS	Dec int SD	VS				(	)	Abs	Yes	S	65,535
Type code			Term	NS[3,7]											
Length specification[9]			Term[11] or Expr[11]	NS	Dec int SD	VS				L(	)	Abs	Yes	S	See manual [4]
Address constant[5]			Expr	NS	SD	VS	*			(	, or)	Abs or rel		V	$2^{24}-1$
Other constants[5]	See manual[4]									'	, or '				

Table 15. Assembler Error Summary (Part 2 of 2)

	L, N, or Q Flag		F Flag	T Flag						F Flag		R Flag	U Flag	V or S Flag	
				Types of Term Allowed											
Statement type/ operand type	Flag	Name Requirement	Operand	Norm Symb	Self Def Symb	Var Symb	*	Seq Symb	Lit	Lead Char	Punct. Allowed	Reloc Rqmnt	Prev Def Symb Rqd	Flag	Maximum Value of Operand
DROP	L	Blank	Term	NS	SD	VS					Blank	Abs		V	15
DSECT	Q	Normal symbol	Blank												
EJECT	L	Blank	Blank												
END	L	Blank	Expr [11]	NS	SD	VS	*				Blank	Rel		V	$2^{24}-1$
ENTRY	L	Blank	Term	NS							Blank	Rel			
EQU	Q	Normal symbol	Expr	NS	SD	VS	*				Blank	Abs or rel	Yes	S	$2^{24}-1$
EXTERN	L	Blank	Term	NS							Blank	Rel			
ICTL	L	Blank	Term		1 or 25						Blank			2	25
LTORG	N	Optional	Blank												
ORG	L	Blank	Expr [11]	NS	SD	VS	*				Blank	Rel	Yes	S	$2^{24}-1$
PRINT	L	Blank	Term	NS[3]							Blank or ,				
REPRO	L	Blank	Blank												
SETA	Q	Variable symbol	Expr		SD	VS					Blank			S	-2^{23} to $+(2^{23}-1)$
SPACE	L	Blank	Term [11]		Dec int	VS					Blank				$2^{24}-1$
START	N	Optional [6]	Term [11]		SD	VS					Blank			S	$2^{24}-1$
TITLE	N	Optional (see manual)[4]	Char string							'	'				
USING	L	Blank													
Address operand			Expr	NS	SD	VS	*				,	Abs or rel		V	$2^{24}-1$
Register operand			Term	NS	SD	VS					Blank	Abs		V	15

Footnotes:

1. For this type of operand the (index,base) subfield set is optional. When it is present, the "punctuation allowed" column refers to the character following the right parenthesis.
2. A literal does not allow the (index,base) subfield set.
3. These symbols are limited to specific ones required by the statement.
4. IBM System/360 Model 44 Programming System, Assembler Language, Form C28-6811.
5. Most errors related to this operand will issue an H flag.
6. Optional, but there may be only one unnamed CSECT or START in an assembly.
7. If the type code cannot be determined, an F flag is issued.
8. Several, more specific flags may be issued for this error.
9. Most errors related to this operand will issue a G flag.
10. Any statement that allows an optional name field or requires that it be blank allows a sequence symbol in that field.
11. May be omitted. When the entire operand is optionally blank, comments must be preceded by a comma. If this is omitted, the first word of the comments is treated as an operand.

Table 16. UPDATE Errors
(Part 1 of 2)

Error Message	Applicable UPDATE Instruction	Mode of Run	Meaning	Action
INVALID INST	OMIT CPYTO SKPTO	UPDASMB1 UPDASMB3 UPDATE2	An end-of-file was encountered previously in the source input data set.	Statement is ignored.
		UPDASMB2 UPDATE2	Instruction is invalid for mode of run specified.	Statement is ignored.
	REWND	UPDASMB2 UPDATE2	Instruction is invalid for mode of run specified.	Statement is ignored.
	ENDUP	UPDASMB1 UPDASMB2 UPDASMB3	Instruction is invalid for mode of run specified.	An END card is generated if one is not encountered.
INVALID OPND	NUM	UPDASMB1 UPDASMB2 UPDASMB3 UPDATE1 UPDATE2	Operand is not a valid serial number or is not blank.	Statement is ignored.
	OMIT	UPDASMB1 UPDASMB3 UPDATE1	Operand is not a valid serial number or is not blank.	Statement is ignored.
			An end-of-file was encountered in the source input data set before the serial number was found.	Updating terminates.
	CPYTO SKPTO	UPDASMB1 UPDASMB3 UPDATE1	Operand is not a valid serial number.	Statement is ignored.
			Operand is blank.	Statement is ignored.
			Operand is equal to serial field.	Statement is ignored.
			An end-of-file was encountered in the source input data set. There is no statement with a serial number that matches the operand of the UPDATE instruction.	Updating terminates.

Table 16. UPDATE Errors
(Part 2 of 2)

Error Message	Applicable UPDATE Instruction	Mode of Run	Meaning	Action
INVLD SERFLD	NUM	UPDASMB1 UPDASMB3 UPDATE1	An end-of-file was encountered in source data set, but there is no matching serial number.	Statement is ignored.
		UPDASMB2 UPDATE2	Specifying a serial number is meaningless for this mode of run.	Statement is acted upon when encountered.
	OMIT	UPDASMB1 UPDASMB3 UPDATE1	Serial field is blank.	Statement is ignored.
	CPYTO SKPTO REWND	UPDASMB1 UPDASMB3 UPDATE1	An end-of-file was encountered in source data set, but no matching serial number was found.	Statement is ignored.
	ENDUP	UPDATE1	An end-of-file was encountered in source data set, but no matching serial number was found.	Updating terminates.
		UPDATE2	Specifying a serial number is meaningless for this mode of run.	Statement is acted upon when encountered.
SEQUENCE ERR	OMIT	UPDASMB1 UPDASMB3	Operand is less than serial field.	Statement is ignored.
			Serial number is less than previous edit serial number.	Statement is processed but action depends upon current serial in the source data set.
	CPYTO SKPTO	UPDASMB1 UPDASMB3 UPDATE1	Operand is less than serial field or serial number is less than previous edit serial.	Statement is processed, seeking a matching serial number.
	NUM REWND	UPDASMB1 UPDASMB3 UPDATE1	Serial number is less than previous edit serial number.	Statement is processed, seeking a matching serial.
	ENDUP	UPDATE1	Serial number is less than previous edit serial number.	Statement is processed seeking a matching serial number.
	Edit statements that are not UPDATE instruc-tions	UPDASMB1 UPDASMB3 UPDATE1	Serial number is less than previous edit serial number.	Statement is accepted as current edit input statement. Action depends upon the current serial number in source data set.
		UPDASMB2 UPDATE2	Serial number is less than previous edit serial number.	Statement is inserted when encountered.

SUPERVISOR MESSAGES

Supervisor messages may appear at any time during execution. They are written by the system's supervisor.

FA0CI - ERR LDING MESS WRTR

An input/output error occurred while the system was loading its message writer routine. The job is cancelled.

FA0DI - cuu NOT OPERATIONAL

where cuu is the physical address of an input/output device. An input/output operation was requested for a data set on a device that is not operational. The job is cancelled.

FA0EI - cuu SNSE UN CHK

where cuu is the physical address of an input/output device. A unit check interruption occurred in response to a sense operation on a device. The job is cancelled.

FA0FI - cuu I/O PROG CHK

where cuu is the physical address of an input/output unit. A program check occurred during execution of an input/output operation. This may be the result of a zero count in a data transmission request or an invalid data address. The job is cancelled.

FA10I - xxxx CAN'T BE LOADED

where xxxx is the name of a system routine. The routine is needed by a system program, but it cannot be found or it cannot be loaded because of an input/output error on SYSAB1. The job is cancelled.

FB0BI - OPRTR CNCLD

A job was cancelled by the operator.

FB11I - CNCL IN CNCL RTN

A CANCEL was requested by the operator while the system was executing a CANCEL routine.

GA06I - PD LST FULL
LAST SVC PSW xxxxxxxx

where xxxxxxxx is the new program status word for the last supervisor interruption. Too many supervisor calls have been issued in too short a time. The job is cancelled.

GA07I - ILLEG CODE - SVC x

where x is an invalid code that was used in a supervisor call. The job is cancelled.

GA08I - xxxxxx CAN'T BE FTCHD

where xxxxxx was used as the name of a phase. The system cannot find any phase with this name in the phase library. The job is cancelled.

PROG CHK INT CODE x
HA02I - IN USER PROG CHK RTN

A program check developed during execution of a user's program check interruption routine. x is the interruption code. The job is cancelled.

PROG CHK INT CODE x
HA03I - USER RTN NOT APPLICABLE

where x is the program check interruption code. On interruption codes 1 through 5, no user program check routine is entered. The job is cancelled.

PROG CHK INT CODE x
HA04I - NO USER RTN SPECIFIED

Where x is the program check interruption code. There is no user program check routine specified to handle this type of program check. The job is cancelled.

PROG CHK INT CODE x
HA05I - PSW xxxxxxxx IN SPVSR STATE

where x is the program check code and xxxxxxxx represents a program status word. A program check occurred in the supervisor state. The PSW is the last problem program PSW. Parameters in an input/output list are not properly aligned. The job is cancelled.

JA0AI - JOB CANCELLED

A job has been cancelled. An SVC 15 has been issued, either by the system or by the problem program. If by the system, another message usually appears giving the reason for the cancellation.

JOB CONTROL MESSAGES

Messages written by the system's job control processor are distinguished by the initial characters IA.

These messages are written on SYSLST. In the following text, they are grouped by type. Each group shares a common text message, but the identification code differs to indicate the source of the error condition.

When an error condition causes cancellation of a job, a dump of the general registers and main storage is provided if the DUMP parameter is specified in the JOB statement.

The messages are as follows:

IAxxx - STMNT FMT ERR

This message indicates an error in the text of a job control statement. The xxx portion identifies the problem area more specifically, as follows:

IA01I - Control field. The first two colums do not contain the proper characters for a job control statement. The job is cancelled.

IA02I - Name field. An invalid name has been specified. It may not be appropriate for the statement, as when something other than a symbolic unit name is specified in the name field of an ALLOC or ACCESS statement. The job is cancelled.

IA03I - Operation field. The system does not recognize the operation specified. The job is cancelled.

IA04I - Operand field. Either no operand field appears in a statement that requires one, or a required parameter is missing. The job is cancelled.

IA05I - Operand delimiter. An improper character has been used as a delimiter. The job is cancelled.

IA06I - Field size or count. A parameter is too long, or specifies an unacceptable size, or there are too many characters within a set of parentheses. The job is cancelled.

IA07I - Operand field. The operand field contains a parameter that cannot be recognized or that should not be used in this statement. The job is cancelled.

IA08I - Continuation error. The first two columns of a continuation statement do not contain the // characters, information starts before column 16, or a continuation statement is required but column 72 is not punched. The job is cancelled.

IA09I - VPS field. The VPS field of an EXEC statement contains an invalid entry, or a VPS setting has been specified for a system that is not equipped with this facility. The job continues, but the parameter is ignored.

IAxxx - STMNT SEQ ERR

This message indicates improper use of a job control statement. The xxx portion identifes the specific problem, as follows:

IA11I - A LABEL statement was misused. For a tape or unit record data set, it did not follow an ALLOC or ACCESS statement. For a disk data set, it did not follow an ALLOC statement. Otherwise, it appeared at an improper place in the input stream. The job is cancelled.

IA12I - An ALLOC statement for a direct access data set was not followed by a LABEL statement. The job is cancelled.

IA13I - The system read a // statement that is not a JOB statement and was not preceded by a JOB statement. The job is cancelled.

IA14I - The program name field of an EXEC statement is blank and the job step does not immediately follow a successful linkage editor job step, or the linkage editor reported an error severity level greater than 4. The job is cancelled.

IA15I - A DELETE, CONDENSE, or RENAME statement refers to a data set that was not cited in an ACCESS or ALLOC statement previously in the job. The statement is ignored.

IA16I - A data set or symbolic unit referred to in the SAME=parameter field of a LABEL statement was not defined previously in the job and is not a system data set. The job is cancelled.

IA17I - An invalid statement appears among the job control statements, or an EXEC statement is missing. The job is not cancelled.

IAxxx - VOL REQ ERR

The following messages apply to volumes requested in ALLOC or ACCESS statements.

IA21I - The system has no record of the volume or device referred to. The job is cancelled.

IA22I - A request for a particular type of device cannot be satisfied. Not enough devices of this type are available. The job is cancelled.

IA23I - The volume field of an ACCESS or ALLOC statement contains an entry that cannot be resolved. The job is canceled.

IA25I - An attempt has been made to remove the system residence volume. The job is cancelled.

IA26I - A statement has requested assignment of a device that is down. The job is cancelled.

IA27I - The volume field of an ACCESS or ALLOC statement specifies the address of a device that was assigned to another data set previously in the same job step. The job is cancelled.

IA28I - A job control maintenance statement has been detected for a data set on a volume that is not mounted. The statement is ignored.

IAxxx - DSNAME ERR xxxxxxxx

These messages apply to the names of data sets and members. The name causing the condition is printed with the message.

IA31I - The required data set cannot be found in the volume specified. The job is cancelled.

IA32I - The required member cannot be found in the data set specified. The job is cancelled.

IA33I - The data set named cannot be found in the system catalog. The condition is ignored.

IA34I - The name specified for a data set duplicates the name of a data set that is already in the same volume. The job is cancelled.

IA35I - The name of directoried data set member duplicates another name already in the directory. The job is cancelled.

IA36I - A data set name duplicates another name in the system catalog. The job is cancelled.

IA37I - The block length requested for the data set is too large for the device. The job is cancelled.

IA38I - An attempt has been made to close a new directoried data set member that was never written. The member is deleted.

IA41I - INSUFF SP xxxxxx

where xxxxxx is a volume identification number. This message indicates there is not enough room on a disk volume to permit a requested operation. The job is cancelled.

IA42I - INSUFF SP xxxxxx

where the x's represent a volume identification. The volume table of contents of a direct access storage volume is full. No new data sets can be added to the volume until some of those already on it are deleted or, if there is vacant space on the disk, the volume table of contents is enlarged through reinitialization. The job is cancelled.

IA43I - INSUFF SP xxxxxxxx

where the x's represent the name of a directoried data set. The directory of a directoried data set is full. No new members can be added until some directory entries are deleted. The job is cancelled.

IA44I - INSUFF SP xxxxxxxx

where the x's represent the name of a directoried data set. There is not enough room in a directoried data set to add another member, or there is not enough room in a data set of any type to write another block of data. The job is cancelled.

IA45I - INSUFF SP CATLG

There is not enough space in the system catalog to add another entry. The job is cancelled.

IA46I - INSUFF SP JOBTABLE

The job control processor's working space is full. The job is cancelled. Either the size of the job must be reduced or the size of the system's SDSUAS data set must be increased before the next run.

IA47I - INSUFF SP FCB

The system does not have enough space in main storage to construct a file control block for the symbolic unit cited in an ALLOC or ACCESS statement. The symbolic unit number may exceed the number that can be handled at the installation. The job is cancelled.

IA50I - ABN EOJ

The job did not include a /& end-of-job statement. The job is cancelled.

IA55I - hhmmss

This message, appearing after a JOB statement, gives the time that the execution of the job started, expressed in hours, minutes and seconds.

IA58I - CUU RW RR RN PW PR PN

IA59I - xxx x x x x x x

These messages report the number of input/output errors detected during the job. The count is listed in columns by device. The CUU column gives the device address; RW is the number of recovered writing errors; RR, recovered reading errors; RN, recovered non-data transmit errors; PW, permanent writing errors; PR, permanent reading errors; and PN, permanent non-data transmit errors.

IA61I - NEW NAME NOT CAT

A renamed data set cannot be cataloged. The name has been changed, as specified in a RENAME statement, but the new name cannot be entered in the system catalog.

IA62I - SYSERR

An unrecoverable system error has occurred. The operator must reinitiate the initial program loading procedure.

IA70I - DA FMT ERR xxxxxx

where xxxxxx is a volume identification. A volume label is unreadable or in an improper format. The volume cannot be used by the system until it is initialized by the Model 44 programming system's utilities program. The job is cancelled.

IA71I - DA FMT ERR xxxxxx

where xxxxxx is a volume identification. A volume label has been changed during the job. The job is cancelled.

IA72I - DA FMT ERR xxxxxx

where xxxxxx is a volume identification. The volume table of contents is not in the proper format. The volume can not be used until it is initialized by the Model 44 Programming System's utilities program. The job is cancelled.

IA73I - DA FMT ERR xxxxxxxx

where xxxxxxxx is the name of a data set. A directoried data set request has been made for a sequential data set. The job is cancelled.

IA75I - DISK I/O ERR

The system's standard error recovery procedure has failed. The system is unable to write on a disk volume during an access or allocate operation, either in handling the volume table of contents or a data set. The job is cancelled.

IA76I - DISK I/O ERR

The system's standard error recovery procedure failed while attempting to recover an input/output error during a DELETE operation. The job is cancelled.

IA79I - NO CATLG

A cataloging request has been made but cannot be executed because the system does not have a catalog.

IA82I - JC INIT DONE

The system has just completed an initial program loading procedure.

IA86I - CAUTION JOB TBL FULL

The job control processor's working space is full. This is only a warning message. Any additional job control statement will overlay a previous entry. If this happens, some references to data sets or symbolic units mentioned in previous statements may not be acceptable, and some symbolic unit assignments may not be made. The size of the job should be reduced, or the size of system data set SDSUAS should be increased.

IA88I - SYSxxx cuu dsname volid

where SYSxxx is a symbolic unit name, cuu is the unit's physical address, dsname is the data set to which the unit is assigned, and volid identifies the volume containing the data set. This format is used by the system in responding to a LISTIO request.

IA89I - M cuu volid

where M is an abbreviation for Mount, cuu is a device address, and volid is

a volume identification code. A disk device has just been assigned to a new volume. The operator can mount the volume to prepare for the IA90A message.

IA90A - M ALL REQ DISKS

This message instructs the operator to mount all disk volumes requested by preceding IA89I messages. When this is done, he signals the system to continue processing.

IA91D - VOL xxxxxx UNREADABLE

where xxxxxx represents a volume identification code. This message appears after an IA90A M ALL REQ DISKS message. It indicates that the system is unable to read the volume label of a disk that has been mounted. The operator can mount another volume, instruct the system to ignore the volume but continue operating, or cancel the job.

IA92I - JCT OFLOW

A LABEL statement uses the SAME parameter, but the reference cannot be resolved because the job control processor's working space was filled earlier in the program. This message follows an IA86I CAUTION JOB TABLE FULL message. The job is cancelled.

IA93I - OPEN ERR SYSxxx

where SYSxxx identifies a system unit. An error was detected while job control was opening a data set on the specified system unit. The data set is not opened, but processing continues.

IA94I - CLOSE ERR SYSxxx

where SYSxxx identifies a system unit. An error was detected while job control was closing a data set on the specified system unit. The data set is not closed, but processing continues.

LINKAGE EDITOR MESSAGES

Linkage editor error messages are written on SYSLST during the linkage editing job step. These messages apply to the ESD, TXT, REP, RLD, and END statements produced by the language processors and the linkage editor control statements.

In most cases, an error message is accompanied by a listing of the statement containing or causing the error. The "System Output" section of this publication contains a sample of these listings.

Some of the statements reproduced in an error listing do not correspond exactly to the actual input statement. This is because the linkage editor does some processing of the statements in the statement input area, and some fields have been altered by the time an error is detected. This applies mainly to the byte count, length, and type fields of the ESD statement. In no case, however, should there be any problem identifying the statement.

For TXT and RLD cards, only the first 36 columns of the variable field are printed. For a REP card error, other than a sequence error, the error code is printed immediately after the REP card listing. The notation FOR REP CARD is printed next to the error code.

Error messages fall into three categories:

1. Warning Messages. These are produced to call a programmer's attention to a condition that may or may not represent an error. They do not affect continuation of the job step.

2. Severe Errors. These messages are written when the linkage editor detects errors that would prohibit successful execution of the program. Linkage editing continues, but its output is flagged so that it will not be accepted for execution.

3. Termination Messages. These messages are written when conditions develop that require immediate termination of the job. Most of these are not the fault of the program, but represent an inability of the system to continue functioning properly.

Most of these messages are written in the format KAxxI, where KA identifies a linkage editor error message, xx represents a numeric code identifying a particular message, and I means the message is for information. A few messages are written out, as discussed in the following list of numeric codes and their corresponding messages.

The last line of any linkage editor listing contains the message LINKAGE EDITOR HIGHEST SEVERITY WAS xx, where xx indicates the severity level, as follows:

0 indicates no significant errors, and execution of the job may continue.

4 indicates that one or more warning messages have been printed, but execution may continue.

12 indicates that the program contains errors that prevent its execution. The phase or phases being linkage edited are not entered in the phase library.

16 indicates that a termination condition exists, and the linkage edit has not been completed. No phases have been entered in the phase library. The job is canceled.

Warning Messages, Severity Code 4

The following messages are designed solely to call a programmer's attention to an unusual condition.

Error Code	Condition
KA01I	A COMMON control section has the same name as a regular control section, but their lengths differ. Space has been reserved for the longer.
KA02I	Two or more control sections in different phases have the same name. The current one was not brought in by the automatic search of the phase library.
KA03I	The previous control section had a length of 0. If this condition is not intentional, it could have been caused by an error of the language processor.
KA04I	An END card that should indicate the length of a control section does not do so. The length of the last or only control section in the ESD dictionary is 0. This does not represent an actual error if the control section contains only instructions to the language processor that do not require any main storage space.
KA05I	A control section name in a CSECT list on an INCLUDE card is duplicated.

Severe Errors, Severity Code 12

The following messages document errors that prohibit execution of the program. Linkage editing continues.

Error Code	Condition
KA11I	The type field of an ESD statement contains an invalid entry. This usually represents a language processor error.
KA12I	A COMMON control section has the same name as an entry point.
KA13I	A label definition type entry in an ESD statement does not point to a section definition or private code type entry. This usually represents a language processor error.
KA14I	An origin for a control section that should be aligned on a double-word boundary is not so aligned. This usually represents a language processor error.
KA15I	An ESD statement indicates that a private code section is named. A private code section cannot be named. This usually represents a language processor error.
KA16I	An SD, LD, or ER type entry with a blank name field is invalid. This usually represents a language processor error.
KA18I	An entry point name improperly duplicates another entry point or control section name.
KA19I	Two or more ESD statements in the same input module have the same identification number. This usually represents a language processor error.
KA35I	System unit SYS000 or SYSREL contains a statement that is invalid or is out of order. Module cards must be in the order ESD, TXT, RLD, REP, and END.
KA36I	A MODULE statement was not followed by a statement with the 12-2-9 loader identification punch in its first column.
KA37I	The linkage editor has read beyond the last block of an input module. The input deck is out of sequence, or an END card is missing.

Error Code	Condition
KA38I	A statement on SYSIPT is invalid or out of sequence.
KA39I	A job control statement other than the /* end-of-data statement has been read. The /* statement is the only job control statement that should be read by the linkage editor.
KA40I	A hexadecimal field in a PHASE or REP card contains an invalid character.
KA41I	An object module contains an ESD identification number of 0 or greater than 255. Except for REP cards, this usually represents a language processor error.
KA42I	A TXT, REP, RLD, or END statement contains an ESD identification number that is not in the module's ESD dictionary. Except for a REP card, it may represent a language processor error. For a TXT or REP card, it also may mean that the ESD number does not point to a control section. This message is written only for the first TXT or REP card containing the error even though the following cards may contain the same erroneous number.
KA43I	The operand field of a control statement extends beyond column 71; the variable field of a REP card extends beyond column 71; or the last field in a REP card contains a number of characters that is not divisible by 4.
KA44I	An entry point in the ESD dictionary has an ESD number that should point to a control section, but the control section that it points to is not in the ESD dictionary. This may represent the loss of cards or a language processor error. This error is detected when an END card is processed, so the message is listed with the END card.
KA45I	The CSECT name list of an INCLUDE statement includes one or more control section names that are not in the module. This code is printed with the END card since the error cannot be detected earlier. In some cases, this message is given because the control section in the ESD dictionary was not processed because of another error condition, usually made by a language processor. In this case, the ESD card containing the control section's entry has been printed with another error code. When the other error condition has been corrected, this condition will be eliminated. If a MAP has been printed, the control sections which were included from the INCLUDE card are listed.
KA46I	An RLD statement contains a position pointer to an ESD number in the ESD dictionary that is not of the SD or PC type. This usually represents a language processor error.
KA47I	An entry in the operand field of a control statement contains too many characters.
KA48I	A required entry is missing from the operand field of a control statement.
KA49I	A control statement contains an invalid delimiter, or a required delimiter is missing.
KA50I	A decimal field in a PHASE statement contains a non-decimal character.
KA51I	The third entry in the operand field of a PHASE statement is invalid. Only NOAUTO can be specified in this field.
KA52I	A name in a PHASE or MODULE statement contains an invalid character.
KA53I	Two or more phases in the program have the same name.

Error Code	Condition
KA54I	A PHASE statement with an * or S origin also has a phase qualifier. This is permitted only when a control section or entry point is specified as the origin.
KA55I	A symbol specified in a PHASE statement for the origin of the phase was not defined previously.
KA56I	A PHASE statement specifies a negative origin.
KA57I	The END statement for the previous phase contains an invalid entry in its transfer address field.
KA58I	The previous phase contained no text. This may occur when the linkage editor is unable to find the modules named in an INCLUDE statement.
KA59I	The entry point specified in an ENTRY statement is not the name of a properly defined entry point or control section.
KA60I	A TXT or REP statement contains a load address outside the limits of the current phase. This usually represents a language processor error, when it is in a TXT statement
KA61I	The program calls for a phase size greater than 368,640 bytes.
KA62I	The cs name field of an INCLUDE statement contains the names of more than five control sections.
KA63I	An entry other than R or L has been specified as the second operand of an INCLUDE statement.
KA64I	A module named in an INCLUDE statement cannot be found in the directory indicated by the R or L specification.
KA65I	The linkage editor has read a job control statement for the next job step and is unable to save it in the user communication region. When the linkage editor reads a job control statement at the end of the job step, it attempts to save it for the job control processor. This message is written when the attempt to store it in the user communication region results in an error return.
KA66I	A PHASE statement identifies a phase as ROOT but also specifies a phase qualifier or relocation factor.

Termination Messages, Severity Code 16

The following messages indicate error conditions so severe that the linkage editor cancels the job immediately. In addition to the error code, the message LINKAGE EDITOR CANNOT CONTINUE is printed.

Error Code	Condition
KA80I	End of extent was detected during a write operation. The output data set is not large enough.
KA81I	A permanent transmission error was detected during an input/output operation.
KA82I	An input/output operation terminated without transmitting any data.
KA83I	An input/output operation terminated because of an invalid Input
KA84I	An input/output operation terminated with an incorrect length condition.
KA86I	There is no room left in the SYSPSD directory to list the module specified in a MODULE statement; or an illegal end of extent was detected reading the last block of the directory. This means a module cannot be inserted on SYS000 with other modules for inclusion in a program.

Error Code	Condition
KA87I	An illegal end of extent condition was detected while reading SYSAB2 or the directories on SYSPSD or SYSREL.
KA88I	No phase can be created because there are no entries in the SYSPSD directory. This message also appears when the name field of an entry contains blanks. The EXEC statement name field was blank when the module was assembled or compiled.
KA90I	The linkage editor's control dictionary and linkage table are full. The program probably contains too many control sections and entry points. A maximum of 2047 control dictionary entries is permitted. If there is no ROOT phase, the maximum is 2048.
KA91I	The program specifies a phase name that duplicates the name of a phase already resident in the phase library.
KA92I	There is not enough room in the phase library directory for all the phases in this program.
KA93I	The system is unable to open the SDS000 or SDS001 data sets. The volumes containing these data sets may not be mounted, symbolic unit SYS000 or SYS001 may have been reassigned, or an error condition may have developed during opening.
KA94I	The system is unable to close SYS000 or SYS001. This indicates that a system error condition developed during the job step.
KA95I	SYS001 is assigned to a seven-track tape without the data convert feature on; or SYS000 and SYS001 are assigned to the same data set.

Text Messages

The following messages are written out by the linkage editor. In some cases, as indicated, the program is flagged so that it cannot be executed, but linkage editing is not interrupted.

xxxx ILLEGAL OPTION FOR LINKAGE EDITOR

This message appears when the EXEC LNKEDT statement contains an illegal or invalid parameter. The incorrect parameter is written at the start of the message.

xxxx UNRESOLVED ADDRESS CONSTANTS

This message appears if a control section contains an address constant for an external symbol in another module, and the linkage editor is unable to supply an address. The number of such unresolved external references in the program is written at the start of the message. If MAP is specified, a list of unresolved symbols is written. The program is flagged so it cannot be executed.

xxxx ADDRESS CONSTANTS OUTSIDE LIMITS OF PHASE

This message is written when the program contains address constants referring to points outside the phase that contains the address constant. This condition usually represents a language processor error. The program is flagged so it cannot be executed.

The following messages are written only if the MAP option has been specified in the EXEC LNKEDT statement. They are warning messages and do not prevent linkage editing or execution.

ROOT PHASE OVERLAID BY ANOTHER PHASE

The program specifies a phase origin that would overlay all or part of a phase that has been designated a ROOT phase. The phase that causes the overlay condition is marked by the word OVEROOT in the listing.

POSSIBLE INVALID ENTRY POINT DUPLICATION IN INPUT

The input contains possible duplication of entry point names. This may occur when control sections from a single module are being split among different phases, in which case the message can be ignored. When this message appears, one or more entry points in the input have been ignored. The MAP shows whether an entry point for a certain control section is missing. If so, any reference to the entry point has been resolved to the wrong location.

ASSEMBLER SYSTEM ERRORS

The following error codes apply to conditions that may develop during an assembly job step. These codes appear at the end of an assembly listing. They are ranked according to severity and are included in any determination of severity levels in the assembler's severity level message at the end of the listing. In the following list, the severity appears in parentheses at the end of each message description.

**LA01I ERROR

The EXEC ASSEMBLE statement contains a misspelled parameter. The assembly continues, but the module is flagged to prevent execution. (12)

**LA02I ERROR

The SYMBn option has been specified with a value for n that is too large for the space available. The option is changed to SYMBMAX. (4)

**LA03I ERROR

There is not enough main storage space available for the assembler's first pass to get started. The job step is terminated. (12)

**LA04I ERROR

There is not enough main storage space available for the assembler's second pass to get started. The job step is terminated. (12)

**LA05I ERROR

The LINK option was specified, but there was an unrecoverable input/output error on SYS000. NOLINK is assumed. (12)

**LA06I ERROR

The DECK option was specified, but there was an unrecoverable input/output error on SYSPCH. NODECK is assumed. (8)

**LA07I ERROR

The XREF option was specified, but there is not enough main storage space to compile a cross reference listing. NOXREF is assumed. (4)

**LA08I ERROR

The XREF option was specified, but there is not enough main storage space for a complete listing. A listing is provided, but some symbols are not cross-referenced. (4)

**LA09I ERROR

Logical records in a data set are larger than the block size specified for the data set. The job step is terminated. (12)

**LA10I ERROR

An unrecoverable input/output error was detected on SYSIPT. The job step is terminated. (12)

**LA11I ERROR

An unrecoverable input/output error was detected on SYS002. The job step is terminated. (12)

**LA12I ERROR

An unrecoverable input/output error was detected on SYS003. The job step is terminated. (12)

**LA13I ERROR

An unrecoverable input/output error was detected in the assembler's intermediate text data set on SYS001. The job step is terminated. (12)

**LA14I ERROR

An error was detected while opening a data set during the assembler's first pass. The job step is terminated. (12)

**LA15I ERROR

An error was detected while closing a data set during the assembler's first pass. The job step is terminated. (12)

UTILITIES MESSAGES

The messages in this section are written by the system's utilities routines. They are distinguished by the initial characters MA.

The action taken by the system upon detection of an error varies according to the severity of the condition. There are four possibilities:

1. Cancellation of the job.
2. Cancellation of the job step.

3. Cancellation of the utilities operation containing the error condition. The system moves on to the next operation, if any, in the job step.

4. No action. The message is considered a warning or notification of a possible error condition.

Utilities messages are written on SYSLST. The messages are as follows:

MA00I - UTILITIES UNSUCCESSFUL

This message is written on SYSLST and SYSLOG at the end of a utility job step if any of the utility operations within the job step was cancelled.

MA04I - I/O ERROR CODE xx ON SYSxxx

where xx is one of the input/output error codes shown in Table 9 and SYSxxx is a symbolic unit. An input/output operation on SYSxxx has terminated abnormally. The error code indicates the cause of termination. If the error occurred while reading a member of a directoried data set, the next member is processed. In all other cases, the operation is cancelled.

MA05I - JOB CONTROL CARD AMONG UTILITY INPUT

A job control statement, other than the /* end-of-data statement or the * comments statement, has been detected in the input stream. This constitutes an error condition only when the ABJCL parameter has been specified in a utilities control statement or assumed through default. If the job control statement was part of an input data set on SYS002, the operation is cancelled. If it was part of an input data set on SYSIPT or was among the utilities control statements, the job step is cancelled.

MA06I - ILLEGAL UTILITY VERB

The operations field of a utilities control statement, or of a card that is supposed to be a utilities control statement, contains an invalid parameter. This field specifies the type of operation, such as COPY, COPY*, PRTPCH, etc. The operation is cancelled.

MA07I - ILLEGAL OPTION NEAR COLUMN xx

A control statement contains an option that does not apply to the operation requested. The xx field in this message identifies the approximate card column of the invalid option in the control statement. The operation is cancelled.

MA08I - ILLEGAL SYNTAX OR OPTION ARGUMENT CLOSE NEAR COLUMN xx

A control statement contains an option with an invalid specification, such as specifying an output record size of XYZ; or blanks are not used properly in a control statement. The operation is cancelled.

MA09I - CORE SIZE EXCEEDED

There is not enough main storage space for the requested operation. Block sizes in the input data set may be too large. The operation is cancelled.

MA1AI - OPERATION CONTINUES

An input/output error occurred, but the utility operation continues if the condition was one of the following:

a. Incorrect length on reading. This is reported only the first time an incorrect length condition appears. Thereafter, the utilities program suppresses incorrect length when reading. The effective data length for copying purposes is the number of characters read, truncated, if necessary, to the nearest multiple of the logical record size.

b. Read redundancy. If the IGRED option has been specified, read errors do not cause termination of the utilities.

c. Print data check. When this condition develops, the utilities program attempts to write a blank line on SYSOPT. If it is able to do so without getting an error return code, the operation continues.

MA1DI - SEQUENCE ERROR {L / N}

The SEQIN option has been specified, and a sequence error has been detected. L indicates that the error is in the literal portion of the sequence field. N means it is in the numeric portion. In the latter case, the numeric control field is reset to the input numeric field. In both cases, the operation continues.

MA1EI - SYS003'S DIRECTORY FULL

A directoried data set is being copied, but the directory of the output data set is not large enough for all the required entries. The operation is cancelled.

MA11I - ILLEGAL OR UNSPECIFIED BLOCK SIZE

An invalid block size is specified. The operation is cancelled.

a. An input block size other than 80 is specified for an * operation.

b. An input block size other than 720 is specified for PCHABS.

c. An output block size other than 80 or 81 is specified for a punch operation.

d. An output block size larger than 144 is specified for a print operation.

MA12I - BLOCKING FACTOR DOESN'T DIVIDE BLOCK SIZE

This applies to either the SIZIN or SIZOUT option. A block size has been specified that cannot be divided evenly by the value specified as the number of logical records per block. The operation is cancelled.

MA13I - ILLEGAL SPACING SPECIFICATION

An invalid SPACE option is specified. The operation is cancelled.

a. SPACE=P is specified for a print-punch operation.

b. A SPACE option is specified for a non-print operation.

c. Both a SPACE option and the MODE=D option are specified.

d. A spacing character other than 1, 2, 3, or P is specified.

e. SPACE=P is specified with a HEAD, NUM, and/or LINES option.

MA14I - ILLEGAL MODE SPECIFICATION

An invalid MODE option is specified. The operation is cancelled.

a. MODE=D is specified for a print-punch operation.

b. A mode option is specified for a non-print operation.

c. A mode character other than D or L is specified.

MA15I - LOGICAL RECORD IMAGE EXCEEDS PRINT LINE

In printing, an output record exceeds the size specified in the SIZOUT option. The operation is cancelled.

MA16I - LOGICAL RECORD IMAGE DOESN'T EQUAL LOGICAL OUTPUT RECORD

An option requests action that is not consistent with the defined size of logical records. The operation is cancelled.

a. The value of i (index of the first byte to be copied) is specified as 0 in a TRUNC option.

b. The value specified for i in a TRUNC option is greater than the size of a logical record, as defined, explicitly or by default, in a SIZIN option.

c. A TRUNC option defines a field for copying that is too large for the logical record size, as defined in a SIZIN option.

d. For non-print operations, the logical record image is not equal to the size specified in a SIZOUT option. The logical record image is defined as the number of characters to be copied from an input logical record (multiplied by 2 if the HEX option is specified) plus the number of padding characters to be added.

MA17I - ILLEGAL OPEN OR CLOSE PARAMETER

An invalid open or close option is specified. The operation is cancelled.

a. A character other than R or S is specified in an open option.

b. A character other than R, S, or U is specified in a close option.

MA18I - ILLEGAL SEQUENCE OPTION

A SEQIN or SEQOUT option is not properly specified. The operation is cancelled.

a. A sequence index value other than 73 is specified in a SEQOUT option for a PCHABS operation.

b. The sequence index exceeds the logical record size.

c. The sequence index plus the sequence length exceeds the logical record size.

MA19I - ILLEGAL HEADING

An invalid heading is specified. The operation is cancelled.

a. A HEAD, LINES, and/or NUM option is specified for a non-print operation.

b. The number of characters in a heading exceeds 144.

c. A HEAD=" option is specified in a sequential data get statement.

MA1LI - NEW MEMBER NAMES

This message is written after the data has been copied in a CPYMEM or CPYMEM* operation. It is followed by a list of the member names that are being added to the destination data set. The operation terminates normally.

MA1OI - SYSxxx INVALID DIRECTORY ENTRY [1]

An error has been detected in an operation on a directoried data set. SYSxxx identifies the symbolic unit assigned to the data set in question. The operation is cancelled.

a. If a 1 appears at the end of the message, the first directory entry is invalid. This usually means that SYSxxx is not assigned to a directoried data set.

b. If there is no 1 at the end of the message, a directory entry other than the first is invalid.

MA1MI - TOO MANY MEM NAMES

A header card contains an invalid number of directoried data set member names. This usually indicates an invalid header card. The data for this member is bypassed, and the operation continues.

MA1NI - PACHABS PHASE MISSING

A PCHABS operation has been requested, but the system cannot locate utilities phase BMUTPCHA in the phase library. The operation is cancelled.

MA1RI - ILLEGAL RANGE SPECIFIED

An INCL option specifies a range for a directoried data set that does not accurately reflect the order of member names in the directory of the source data set. The operation is cancelled.

MA1TI - { OPEN / CLOSE } ERROR CODE xx -- SYSxxx

where xx is an OPEN or CLOSE supervisor call error code, and SYSxxx is the name of a symbolic unit. An error has been detected while trying to open or close a data set on SYSxxx. The code indicates the type of error. The operation is cancelled.

The following messages apply to the volume utilities routines.

MA20I - DVADR OPTION MISSING

The required DVADR device address parameter is missing in a SQUEEZE or MAP control statement. The operation is cancelled.

MA21I - VOLID OPTION MISSING

The required VOLID volume identification option is missing in a SQUEEZE or MAP control statement. The operation is cancelled.

MA22I - INCORRECT OR MISSING FORMAT 4 LABEL

The system cannot find a volume table of contents definition label, a format 4 label, on a direct access volume, or the label is incorrectly written. This message appears for SQUEEZE and MAP operations. The operation is cancelled.

MA23I - DEVICE NEITHER 2311 NOR 2315

A SQUEEZE or MAP operation has been requested, but the device is neither a 2311 nor 2315 Direct Access Storage Device. The operation is cancelled.

MA24I - VTOC CONTAINS AN INVALID LABEL

The volume table of contents for a direct access volume contains an invalid label. The system is unable to process the label. The operation is cancelled.

MA25I - SQUEEZE SPECIFIED FOR 2311

An invalid SQUEEZE operation has been requested. SQUEEZE can be executed only for 2315 Disk Storage volumes. The operation is cancelled.

MA26I - FORMAT 2 OR 3 LABEL POINTS INCORRECTLY

A format 2 or 3 label on a direct access volume is not properly written. This applies only to volumes prepared under other System/360 programming systems. The Model 44 system does not use these labels. The operation is cancelled.

MA27I - MISSING FORMAT 5 LABEL

The system is unable to find a format 5 space management label on a direct access volume. This label should be part of the volume table of contents. The operation is cancelled.

MA29I - INCORRECT TYPE OPTION

The type field of an INITIAL statement contains an invalid entry. Only SDSD or 1316 is permitted for a SQUEEZE or MAP operation. The operation is cancelled.

MA2AI - UTILITY ABORTED

The system has detected an error of such severity that it cannot continue. The operation is cancelled. This message usually appears with another message indicating the cause of the cancellation.

MA2BI - DISK UNAVAILABLE FOR FUTURE ALLOCATES

Input/output operations on a disk have resulted in errors of such severity that no further allocations should be attempted.

MA2CI - FOLLOWING DATA SETS ARE LOST

The system is unable to find certain data sets listed in the volume table of contents. The names of these data sets are listed.

MA2DI - READ ERROR BLOCK xxxx OF VTOC

The system detected an irretrievable read error in the volume table of contents. The xxxx field identifies the block in which the error was detected. The operation is cancelled.

MA31A - PERM I/O ERROR. ABEND

An unrecoverable input/output error occurred. The operation is cancelled.

MA33I - I/O ERROR READING VTOC OR VOL LBL. ABEND

The volume label or table of contents cannot be read or updated. The operation is cancelled.

MA36I - NO ALTERNATE TRACKS FREE. ABEND

No alternate tracks are available for assignment. The operation is cancelled.

MA37I - IPL RECORD CANNOT BE WRITTEN. ABEND

An unrecoverable error occurred while writing the initial program loading record. The operation is cancelled.

MA38I - Cxxx, Hx, WRITE HA or R0 ERR. ABEND

A home address or record 0 cannot be written on a disk. Cxxx and Hx identify the cylinder and track involved. The operation is cancelled.

MA39I - Cxxx, Hx, SEEK ERR. Cxxx, Hx REACHED. ABEND

A seek error occurred during a disk surface analysis. Cxxx and Hx identify the cylinder and head involved. The operation is cancelled.

MA3AI - Cxxx, Hx, R0 or HA READ ERROR. ABEND

Record 0 and a home address disagree as written. Cxxx and Hx identify the cylinder and head involved. The operation is cancelled.

MA3BI - Cxxx, Hx, TRK DESCRIPTOR REC BAD. ABEND

Record 0 written and record 0 read disagree. The operation is cancelled. Cxxx and Hx identify the cylinder and head involved.

MA3CI - Cxxx, Hx, ALT PERM ERROR. CONTINUING

An alternate track was found to be bad during disk surface analysis. Cxxx and Hx identify the cylinder and head involved. The operation continues.

MA3DI - Cxxx, Hx, PRIM PERM ERR. CONTINUING

A primary track was found to be bad during disk surface analysis. The operation continues.

MA3EI - Cxxx, Hx, ALT ASSIGNED. CONTINUING

An alternate track at the cylinder and head address specified in this message has been assigned for the bad primary track referred to in a previous MA3DI message. The operation continues.

MA3FI - ERR WRITING VTOC. ABEND

The utilities program is unable to write proper format 4 and format 5 records. The operation is cancelled.

MA3GI - UNEXPECTED ERR. ABEND

An input/output error occurred. The operation is cancelled.

MA3II - DISK SUCCESSFULLY FORMATTED. EOJ

A disk has been formatted successfully. The operation is completed.

MA3KI - I/O ERR WHILE INITIALIZING TAPE. ABEND

An input/output error occurred during tape initialization. The operation is cancelled.

All direct access volumes used with the programming system must be labelled. The use of labels on magnetic tape volumes is optional with each installation.

An installation determines at the time that its system is constructed whether magnetic tapes will contain data set and volume labels. If it decides to use labels, all tapes must contain standard labels. If it decides against labels, tapes may contain volume labels but no check is made for data set labels.

The programming system uses label information to improve its operating efficiency and to save programmers the time and trouble of providing certain data every time a program is run. Direct access data set labels, for example, are grouped together to form a volume table of contents. When a programmer needs to use a data set, he identifies it by name and volume. The system can then check its label to determine its exact track and cylinder address. Other information in the label includes the date the data set was created, the date it can be eliminated, the size of each record block, the number of blocks written, and whether there is room for more data blocks to be written.

The programming system writes direct access and magnetic tape data set labels with information supplied by programmers in ALLOC, ACCESS, and LABEL job control statements. The system's volume initialization utility routines write volume labels, using information from INITIAL control statements.

The system itself supplies some of the information that goes into these labels. The first field of a volume label, for example, must contain the entry VOL, which identifies a volume label. It is not necessary for a programmer to specify entries for fields such as this.

The supplying of label information is further simplified with the use of default options. In these cases, the system determines which of several possible entries will go into a label field unless a programmer specifically indicates that a different entry is wanted. The expiration date field of the data set label illustrates this type of operation. This field indicates when the data set may be eliminated. If a programmer does not provide an entry for this field, the system inserts the current date.

The descriptions of the ACCESS, ALLOC, LABEL, and INITIAL statements elsewhere in this publication indicate what information a programmer must supply and what default options, if any, apply to each field.

Some of the label information is not used by the Model 44 Programming System. Such fields are identified in the discussions of the statements. It is not necessary to supply information for these fields. They may be used, however, to make model 44 volumes and data sets compatible with the operating systems for other System/360 models. In most cases, this permits the interchange of data sets and volumes written under the control of the Model 44 Programming System with those written on other models.

A programmer also may use the label information fields of the job control statements to make certain information available to his program. Regardless of whether a data set label is written, the information from some of these fields is stored in a data set's file control block during execution. The system uses this information for its input/output operations, but the problem program also has access to it. This applies to data sets on any type of input/output device.

In addition to volume and data set labels, direct access volumes contain two other types of labels written and used solely by the programming system. One of these describes the size and address of the volume table of contents. The other is used to keep track of vacant space on the disk. Both are included as part of the volume table of contents.

A magnetic tape volume contains only a volume label in unlabeled installations. In other cases, data set header and trailer labels also are written.

DIRECT ACCESS LABELS

A direct access volume contains four types of labels.

1. A volume label. This is an 80-byte label that is written as the third record on cylinder 0, track 0 of a direct access volume. This label contains a volume identification serial number supplied by the installation

and indicates the address of the volume table of contents. The label is illustrated in Figure 16, and its fields are described later in this section.

2. Data set labels. One 140-byte label is written for each data set in the volume. It is included in the volume table of contents and is known as a format 1 label. It contains the name of the data set and other information necessary to identify, locate, and use the data set. It is illustrated in Figure 17 and described in detail later in this section.

3. A VTOC definition label. This 140-byte label is the first label in the volume table of contents. It describes the volume table of contents and the device upon which the volume is mounted. It is known as a format 4 label. Each direct access volume contains one of these labels. It is illustrated in Figure 18 and described later in this section.

4. A space management label. This is a 140-byte label that is included in the volume table of contents. The system uses it to keep track of vacant space that can be assigned to new data sets when an ALLOC statement is processed. Its contents are updated whenever a data set is created or deleted or the volume is condensed. It is known as a format 5 label and is illustrated in Figure 19. There is at least one on each direct access volume.

The volume, VTOC definition, and space management labels are created when a disk is processed by the system's volume initialization routines. Data set labels are written by the system whenever a new data set is added to the volume.

The Model 44 Programming System does not recognize or use format 2 and format 3 labels. These labels are used by other System/360 programming systems for indexed-sequential data sets and multiple extents.

Labels are updated at the end of the job in which changes are made. If, for example, additional blocks are written in a data set, the data set label is updated to reflect the additional material as part of the end-of-job processing by the system. This label also is updated at the end of any job step in which a CLOSE supervisor call has been executed for the data set.

MAGNETIC TAPE LABELS

Each installation determines whether its magnetic tape data sets will be labeled. If the installation standard is to use labels, each tape should contain a volume label and data set header and trailer labels.

Unlabeled Tapes

The system does no label processing on magnetic tapes when the installation standard is not to use labels. Any labels on a tape are treated as data. If a labeled tape is used for output, the labels will be written over unless the problem program positions them to avoid this.

Labeled Tapes

When the installation standard is to label tapes, the system checks each magnetic tape for a volume label. If there is none, it treats the tape as unlabeled. A warning message is written when the tape is opened for input/output operations, but no further attempt is made to read, write, or check labels.

The system's volume utilities program can be used to initialize a labeled tape. This program writes a volume label and dummy data set header and trailer labels.

Each of these labels requires 80 bytes. They are as follows:

1. A volume label. This is the first label on any magnetic tape volume. It contains a volume identification serial number specified by the installation when the volume was initialized. It is essentially the same as the direct access volume label and is shown in Figure 16.

2. A header label. This follows the volume label. It contains the name of the data set, its creation and expiration dates, and other data for system use. It is illustrated in Figure 20.

3. A trailer label. This label is similar to the header label. It also indicates the number of blocks written in the data set. It is illustrated in Figure 20.

LABEL FORMATS

This section contains illustrations of the label formats and descriptions of the individual fields. Most of the labels contain one or more reserved fields. These fields are being kept available for future use by the programming system and should be recorded as blanks. At present, these fields are ignored by the programming system.

In the illustrations, each field is numbered for identification in its upper right corner. A running count along the bottom indicates the number of bytes in each field and their relative position in the label.

The fields are described by number following the illustration.

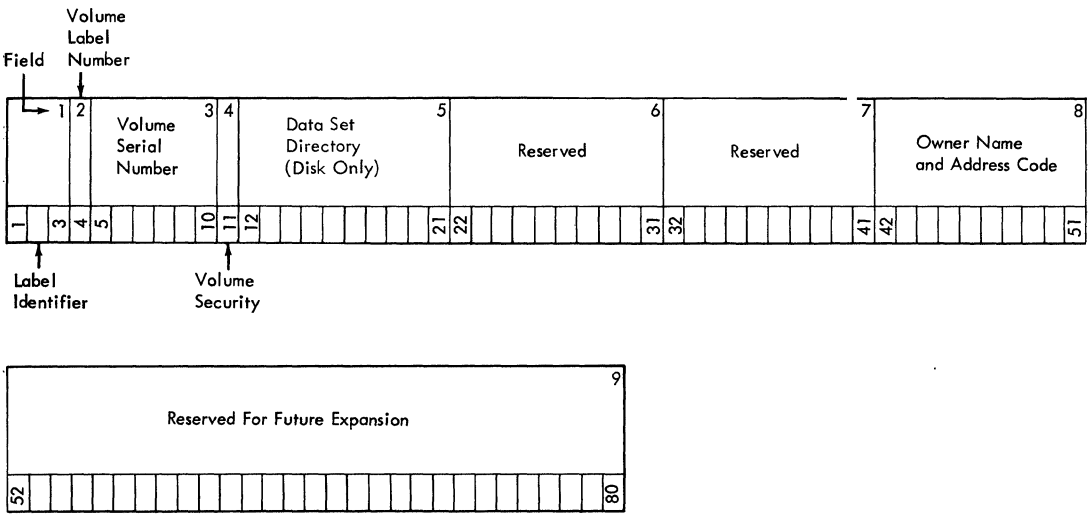


Figure 16. Volume Label Format

Field	Name and Length	Description
1	Label Indentifier, 3 bytes	Contains VOL.
2	Volume Label Number, 1 byte	Position of this label in a group of volume labels. For the Model 44 system, this field must be 1.
3	Volume Serial Number, 6 bytes	Identification code assigned to this volume by the installation.
4	Volume Security, 1 byte	Security status of a volume; not used by the Model 44 system.
5	Data Set Directory, 10 bytes	On direct access volumes, the address of the VTOC; on tapes, recorded as blanks.
6-7	Reserved, 20 bytes	Reserved.
8	Owner Name and Address Code, 10 bytes	A specific user, installation, and/or system to which the volume belongs.
9	Reserved, 29 bytes	Reserved.

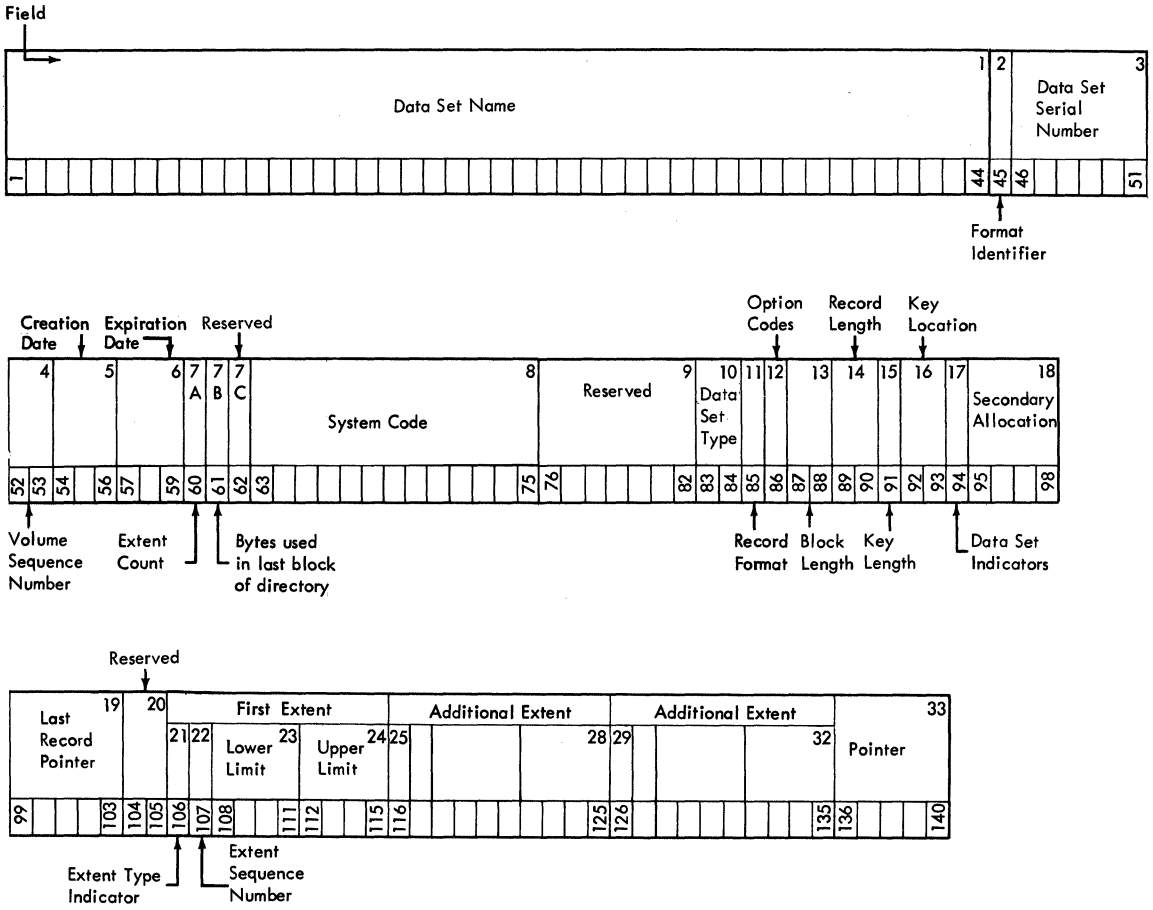


Figure 17. Direct Access Data Set Label, Format 1

Field	Name and Length	Description
1	Data Set Name, 44 bytes, EBCDIC	Each data set on the volume must have a unique name. Names are left-justified in the field and the remaining bytes are recorded as blanks. In the Model 44 system, data set names are restricted to eight characters.
2	Format Identifier, 1 byte, EBCDIC	1 = Format 1.
3	Data Set Serial Number, 6 bytes, EBCDIC	Identifies a data set-volume relationship; it is identical to the volume serial number.
4	Volume Sequence Number, 2 bytes, binary	The order of this volume relative to the first on which the data set resides. For the Model 44 system, this field must be 1.
5	Creation Date, 3 bytes, discontinuous binary	The date on which the data set was created

Field	Name and Length	Description
6	Expiration Date, 3 bytes, discontinuous binary	The date on which the data set may be deleted.
7A	Extent Code, 1 byte	The number of extents for this data set on this volume. For the Model 44 system, must be 1.
7B	Bytes Used in Last Block of Directory, 1 byte, binary	Used for directoried data sets.
7C	Reserved, 1 byte	Reserved.
8	System Code, 13 bytes	Identifies the programming system.
9	Reserved, 7 bytes	Reserved.
10	Data Set Type, 2 bytes	Identifies the type of data set.
11	Record Format, 1 byte	Identifies type of records in the data set: record length, track overflow (not used by the Model 44 system), record blocking, record truncation, control character, and record keys.
12	Option Codes, 1 byte	One-bit switches indicating options used in building data set. Bit 0 indicates whether data set was created using write validity check. Bits 1-7 are reserved.
13	Block Length, 2 bytes, binary	Block length used in data set.
14	Record Length, 2 bytes, binary	Record length used in data set.
15	Key Length, 1 byte, binary	Length of keys on data records in data set.
16	Key Location, 2 bytes, binary	High-order position of the embedded key in the data record.
17	Data Set Indicators 1 byte	One-bit flags, with the on position indicating: 0 The last volume on which data set resides. For the Model 44, this bit is always on. 1 This data set must always reside in the same absolute location on the volume. In the Model 44 system, this bit is always off. 2 The block length must always be a multiple of 8 bytes. Not used by the Model 44 system. 3 This data set is security-protected and requires a password for access. Not used by the Model 44 system. 4-7 Reserved.

Field	Name and Length	Description
18	Secondary Allocation, 4 bytes, binary	Indicates amount of storage to be requested for this data set at end-of-extent. Not used by the Model 44 system.
19	Last Record Pointer, 5 bytes, discontinuous binary	Points to the last record written in a sequential or directoried data set.
20	Reserved, 2 bytes	Reserved.
21	Extent Type Indicator, 1 byte	Indicates the type of extent. Not used by the Model 44 system.
22	Extent Sequence Number, 1 byte, binary	Indicates the extent sequence in a multi-extent data set. For the Model 44 system, must be 1.
23	Lower Limit, 4 bytes, discontinuous binary	Starting address of this extent component.
24	Upper Limit, 4 bytes, discontinuous binary	Ending address of this extent component.
25-32	Additional Extents, 20 bytes	Not used by the Model 44 system.
33	Pointer, 5 bytes, discontinuous binary	Pointer to next data set label within this label set. Not used by the Model 44 system.

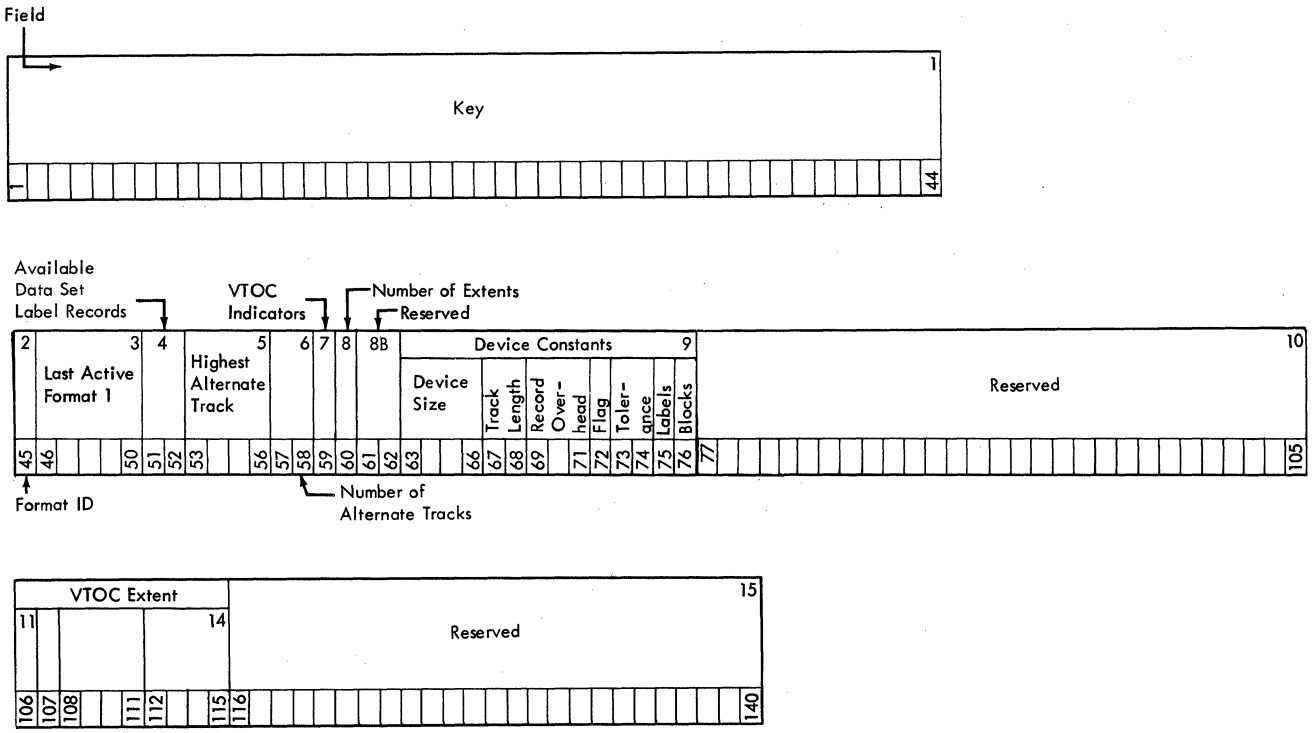


Figure 18. VTOC Definition Label, Format 4

Field	Name and Length	Description
1	Key Field, 44 bytes, binary	Each byte contains the hexadecimal code 04.
2	Format Identifier, 1 byte, EBCDIC numeric	4 = Format 4.
3	Last Active Format 1, 5 bytes	Address of the last active format 1 label; used to stop a search on a data set name.
4	Available Data Set Label Records, 2 bytes, binary	The number of unused records in the VTOC.
5	Highest Alternate Track, 4 bytes	The highest address of a block of tracks set aside as alternates for bad tracks.
6	Number of Alternate Tracks, 2 bytes, binary	The number of alternate tracks available.
7	VTOC Indicators, 1 byte	Bit 0, if on, indicates format 5 label does not reflect true status of volume.
8A	Number of Extents, 1 byte	Contains the hexadecimal constant 01 to indicate one extent in the VTOC.
8B	Reserved, 2 bytes	Reserved.
9	Device Constants, 14 bytes	Contains constants describing the device on which the VTOC was created, including device size, track length, record overhead, flags, tolerance, labels per track, and directory blocks per track.
10	Reserved, 29 bytes	Reserved.
11-14	VTOC Extent, 10 bytes	The extent of the VTOC; identical in format to fields 21-24 of the format 1 label.
15	Reserved, 25 bytes	Reserved.

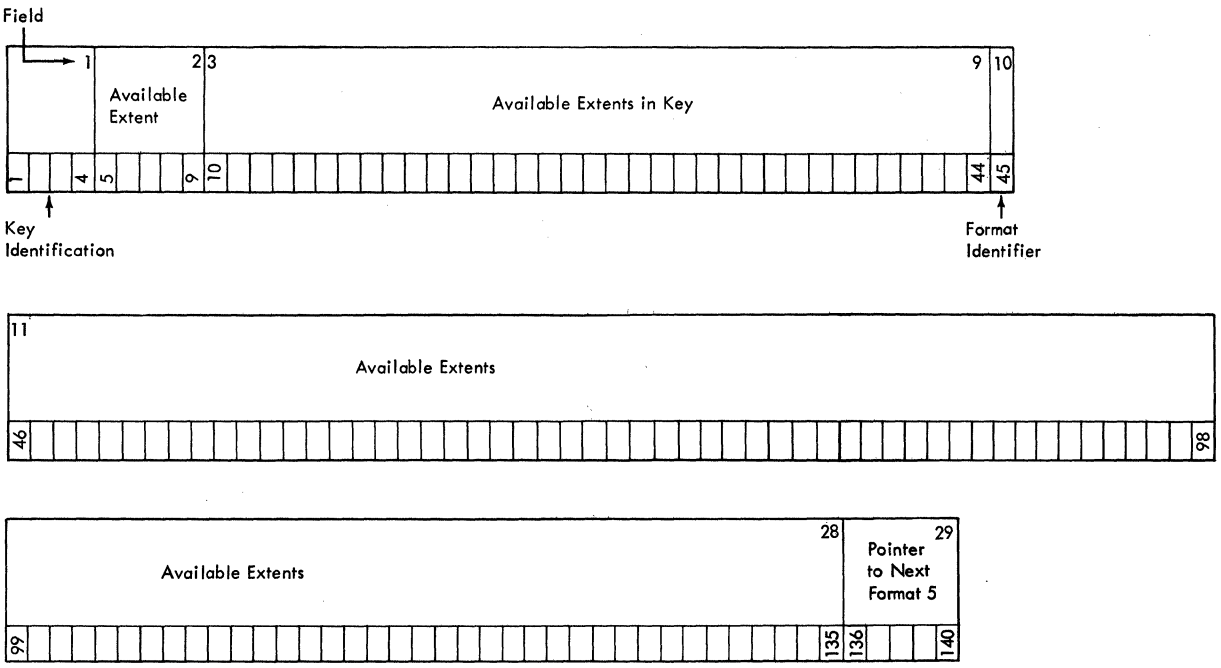


Figure 19. Direct Access Space Management Label, Format 5

Field	Name and Length	Description
1	Key Identification, 4 bytes	Each byte contains the hexadecimal code 05.
2	Available Extent, 5 bytes	Indicates an extent available for allocation to a data set. The first two bytes are the relative track address; the next two are the number of full cylinders in the extent. The last is the number of tracks in addition to the cylinders.
3-9	Available Extents in Key, 35 bytes	These fields, identical to field 2, are in relative track address sequence.
10	Format Identifier, 1 byte, EBCDIC	5 = Format 5.
11-28	Available Extents, 90 bytes	These fields are the same as field 2. There are altogether (fields 2-9, 11-28) 26 extent fields in the format 5 label.
29	Pointer to Next Format 5	Contains the address of the next format 5 label on the volume.

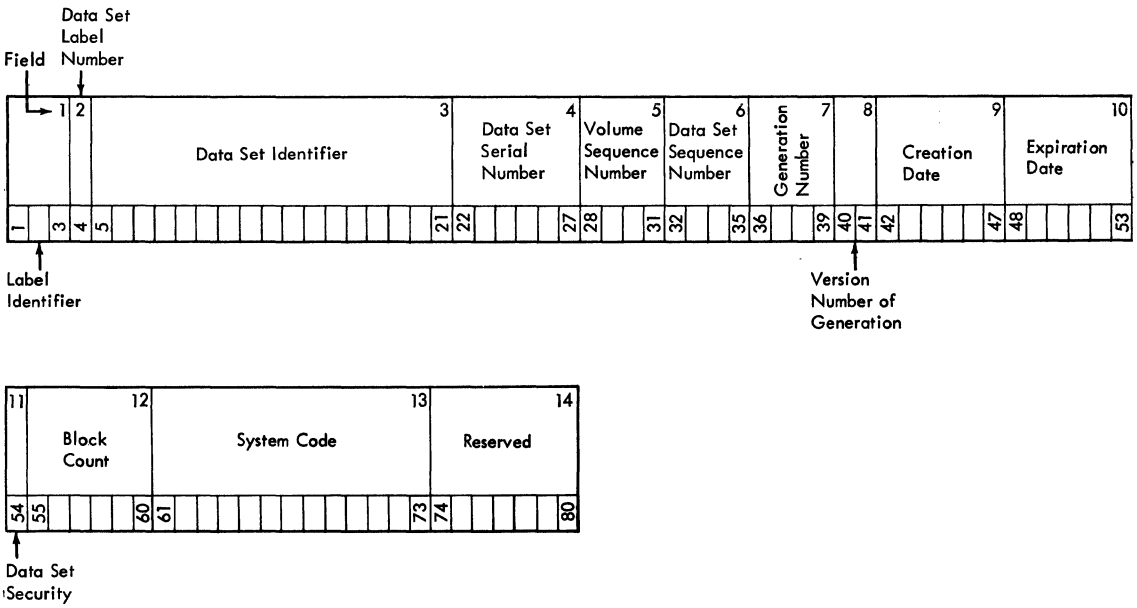


Figure 20. Standard Tape Data Set Header and Trailer Label Format

Field	Name and Length	Description
1	Label Identifier, 3 bytes, EBCDIC	HDR = Header Label -- beginning of a data set EOF = End of File -- end of a data set EOV = End of Volume -- end of a volume in a multivolume data set. Treated as EOF by the Model 44 system.
2	Data Set Label Number, 1 byte, EBCDIC	Always a 1.
3	Data Set Identifier, 17 bytes, EBCDIC	The data set name, left-justified, with remaining bytes written as blanks. In the Model 44 system, data set names are restricted to eight characters.
4	Data Set Serial Number, 6 bytes EBCDIC	Identifies a data set-volume relationship; it is identical to the volume serial number.
5	Volume Sequence Number, 4 bytes	Indicates the order of a volume in a data set. For the Model 44 system, must be 1.
6	Data Set Sequence Number, 4 bytes	Assigns numeric sequence to a data set on the volume. For the Model 44 system, must be 1.
7	Generation Number, 4 bytes	Not used by the Model 44 system.
8	Version Number of Generation, 2 bytes	Not used by the Model 44 system.
9	Creation Date, 6 bytes	The date the data set was created.
10	Expiration Date, 6 bytes	The date the data set may be deleted.

Field	Name and Length	Description
11	Data Set Security, 1 byte	Indicates whether data set has security protection. Not used by the Model 44 system.
12	Block Count, 6 bytes	Used for trailer labels only. Indicates the number of blocks written in the data set from the header label to the trailer label, exclusive of tape marks.
13	System Code, 13 bytes	Identifies the programming system.
14	Reserved, 7 bytes	Reserved; should be recorded as blanks.

An absolute loader is distributed with the programming system. It can be used to load and execute installation programs that are not meant to operate under control of the Model 44 Programming System.

The loader consists of six punched cards. These cards are self-loading from a card reader. They should be followed by the text of the program to be executed. The loader loads the program and transfers control to it. The loader has no further function.

The program cards must be in the format produced by the programming system's linkage editor. This can be obtained by linkage editing a program and copying it from the phase library with the PCHABS utilities function.

The loader clears up to 64,000 bytes of main storage and loads the program at the addresses supplied by the linkage editor. Loading terminates when the loader detects an END card. The END card must specify an entry point. The loader transfers control to this entry point.

Any ESD cards in the input deck are ignored. REP cards may be included anywhere before the END card. A REP card must not precede the text card that it is correcting.

Formats of the ESD, RLD, REP, END, and TXT cards are shown in Appendix B.

The END card format shown in Figure 24 is the type used by the absolute loader, except that any length given in columns 29 through 32 is ignored.

The REP card format for the absolute loader differs slightly from that shown in Figure 26. Any address specified in columns 7 through 12 must be aligned on a half word boundary. Also, for columns 17 through 71, each field must consist of exactly four characters.

APPENDIX A. INPUT/OUTPUT CONTROL BLOCK FORMATS

This section contains the formats of the control blocks used by the system's input/output routines at the read/write level.

At the read/write level, the file and unit control blocks are constructed and maintained by the system. The problem program must reserve space for the request control block and provide the SYSUNI index for its first byte.

REQUEST CONTROL BLOCK

The fields of the request control block are defined as follows:

Word	Byte	Description
0	0	SYSUNI index number of the system unit to be used in the input/output operation. This value can be determined from Table 2.
0	1-3	Address of the device dependent routine to be used to set up the channel commands and analyze interruptions.
1	4	Post request flag indicating whether the block currently is active: 00 = No operation pending 01 = Operation in progress
1	5-7	Address of the first channel command required to execute the operation.
2	8	Used by the system, when a permanent input/output error occurs, to pass an error code between system components.
2,3	9-15	Last seven bytes of Channel Status Word, stored when an operation is started and when an interruption occurs.
4	16-19	Sense information, stored when unit check condition occurs.
5	20-23	Name of program to be loaded from the phase library when necessary for interruption analysis.
6	24	Error recovery internal code identifying a type of error.
6	25-27	Counters used to keep track of number of attempts made to recover an error. The first two bytes keep track of the number of retries. The third byte is an operation counter used during multi-step operations.
7	28	Return code 00 = Operation completed 04 = End-of-file or end-of-volume 08 = Permanent transmission error 0C = No data transmitted 10 = Invalid request 14 = Incorrect length
7	29-31	Address of file control block being used for operation.
8	32	Request code identifying the type of operation to be set up by a device routine. 01 = Write 02 = Read 03 = Data transmission 07 = Rewind 0F = Rewind and unload 1F = Write end of file 3F = Point
8	33-35	Address of buffer to be used for transmission.
9*	36	Incorrect length control byte 20 = Suppress incorrect length indication 00 = Check for incorrect length
9*	37-39	Number of bytes to be transmitted.

*Following completion of an input/output operation, the contents of word 9 are replaced by the updated data set position block count. For a point operation, the block count is specified.

FILE CONTROL BLOCKS

The address of the file control block for a data set can be obtained from bytes 29 through 31 of the request control block that is being used for input/output operations on the data set.

The file control block for disk units is set up as follows:

Word	Byte	Description
0	0	Flag Byte 01 = Opened 02 = Disconnected 04 = EXT specified 08 = Header checked 10 = Labeled 20 = Update VTOC 40 = Fresh data set 80 = Standard unit
0	1	Flag byte 80 = Control character 40 = ASA control characters 20 = Write check 10 = Ignore (dummy data set) 08 = Data set deleted 04 = Opened for output 02 = Formatted data set
0	2	Reserved
0	3	Number of blocks per track
1	4-7	Current block count
2	8,9	Logical record length
2	10,11	Not used
3	12-15	Seek and search address for the current operation
4	16	Number of block being processed
4	17	Number of bytes in key area (2311 only)
4	18,19	Maximum number of bytes per block
5	20-23	Block number of first block of member in a directoried data set
6	24-27	Address of the first record in the data set
7	28-31	Number of the last block written
8	32-35	Number of blocks reserved for this data set.

Fields in the file control block for tape units are the same as for direct access devices, except as follows:

Word	Byte	Description
0	3	Mode of tape operation computed by job control
2,3	10-15	Expiration date of data set
4	16	Current record number
4	17	Not used
5	20,21	Current tape mark count
5,6	22-27	Volume identification in EBCDIC
7,8	28-35	Data set name in EBCDIC

Fields for the file control block for other units are the same as for tape except that bytes 28 through 35 are not used.

UNIT CONTROL BLOCK

Fields in the unit control block are defined as follows:

Word	Byte	Description
0	0,1	Physical device address
0	2	Device mode 01 = Burst mode 02 = Overrunable byte mode 03 = Non-overrunable byte mode
0	3	Type of unit 10 = 1052 Console Printer-Keyboard 20 = 2501 card reader 21 = 2540 card reader 22 = 2520 read-punch 23 = 1442 read-punch 28 = 2520P card punch 29 = 2540P card punch 2A = 1442P card punch 30 = 1403 printer 31 = 1403M7 printer 32 = 1443 printer, 120 characters 33 = 1443S printer, 144 characters 40 = 2400 magnetic tape, 800 bpi 41 = 2400H magnetic tape, 1600 bpi 42 = 2400D magnetic tape, dual density 48 = 2400T7 magnetic tape 49 = 2400T7C magnetic tape 50 = Single disk storage drive 51 = 2311 disk

Word	Byte	Description
1	4	Relative chain pointer (e.g., activity, event chain).
1	5	Channel Scheduler flags 01 = Device busy 02 = Event requested 04 = Attention signal has been detected 08 = Device not ready 10 = Retry in progress 20 = Device end signal received 40 = Channel end signal expected 80 = Give control to a device routine for an intervention-required condition
1	6	Job control flags 80 = Device inoperative 40 = System standard assignment 20 = Job control assigned 10 = Programmer assigned 08 = Assigned by job control
1	7	Read/Write flags 80 = Multiple operation required for error recovery 40 = Error message request has been made 20 = End of volume detected for tape 10 = System request operation requested 08 = Multiple operation requested by user 04 = Not used 02 = Volume label present for tape 01 = Required volume has been mounted
2	8-11	Address of input/output block used when an attention interruption occurs. (Applicable only to those devices that can signal attention.)
3	12	Request flag for device routine 00 = Setup requested 04 = Device end interruption routine requested 08 = Attention interruption occurred 0C = Program controlled interruption occurred
3	13-15	Input/output block address associated with this operation
4	16-19	Current physical position of the device in terms of the cylinder and head positions for direct access devices and block count for sequential devices.
5	20-23	Address of channel command word area associated with the device
6	24	Number of permanent no-data-transmitted errors that have occurred
	25	Number of permanent read errors that have occurred
	26	Number of permament write errors that have occurred
	27	Number of entries in the alternate track procedure
7	28	Number of no-data-transmitted errors corrected by recovery procedures
	29	Number of read errors corrected by recovery procedures
	30	Number of write errors corrected by recovery procedures
	31	Not used

APPENDIX B. LANGUAGE PROCESSOR CARD FORMATS

This section contains the formats of four types of cards (or card images) produced by the language processors. These are the external symbol dictionary, relocation dictionary, text, and module end cards. It also contains the format of the replace (REP) card used for patching a module that is being linkage edited.

Several fields in these cards are punched in extended card format. For information about the format codes, see IBM System/360 Reference Data, Form X20-1703.

ESD Card

Each language processor output module contains one or more external symbol dictionary cards. They are used for the following:

- To define the names of a module and any control sections within it.
- To define entry points within a module that may be referred to by other modules.
- To define symbols within a module that refer to other modules or to entry points in other modules.
- To define COMMON sections included in a program.

This information is used by the linkage editor.

The format of the ESD card is shown in Figure 21.

TXT Card

A text card contains the instructions and constants that make up a program. It also contains the address at which the first byte of the card's text is to be loaded. A card contains up to 56 bytes of text in extended card code.

The TXT card format is shown in Figure 22.

RLD Card

A relocation list dictionary card is produced when a language processor encounters a DC instruction or the second operand of a CCW instruction that defines an address as a relocatable symbol or expression. The address may be an internal symbol that occurs only within the same module, or it may be an external reference to another module.

An RLD card lists the address of up to 13 such symbols. If there are more than 13, additional RLD cards are produced.

The format of the RLD card is shown in Figure 23.

END Card

An END card indicates the end of all cards pertaining to a module. It may specify a location at which execution is to start.

Either of two similar formats may be used. They are shown in Figures 24 and 25.

Column	Contents
1	Loader identification (12-2-9 punch); identifies this as a card acceptable to the loader.
2-4	ESD -- identifies the card type.
5-10	Blank
11-12	Number of bytes in variable field of the card, written in extended card code.
13	Blank
14-16	External symbol dictionary identification of the first section definition, external reference, private code, or COMMON in the card. This is written in extended card code. If all items on the card are label definitions, this field is blank.
17	Beginning of variable field, which runs through column 64.
17-24	Name of control section, entry point, external symbol, or COMMON, to which the ESD entry applies.
25	ESD type; one digit in extended card code. Types — Hexadecimal Digit Section Definition (SD) — 00 Label Definition (LD) — 01 External Reference (ER) — 02 Private Code (PC) — 04 COMMON (CM) — 05 Code 00 identifies a control section. Its name, assembled origin, and length are given in other fields of the card. Code 01 identifies a label that may be used as an entry point. Other fields of the card identify its assembled address and the control section in which it appears. Code 02 identifies control sections and entry points that appear in other modules. Code 04, private code, is used for an unnamed control section. Other fields of the card indicate its address and length. Code 05 is used for a COMMON entry. Another field indicates its length.
26-28	This field specifies the address of a control section or entry point. The address is written in extended card code. This field is filled with zeros if the card is for an external reference or COMMON control section.
29	Blank
30-32	This field specifies, in extended card code, the number of bytes in a control section or COMMON, or for an entry point, the identification number of the control section containing it. This field is blank in external reference cards.
33-48	The format of this field is the same as for columns 17-32 if there is a second ESD entry on the card.
49-64	The format of this field is the same as for columns 17-32 if there is a third ESD entry on the card.
65-72	Blank
73-80	Deck sequence field.

Figure 21. ESD Card Format

Column	Contents
1	Loader identification (12-2-9); identifies this as a card acceptable to the loader.
2-4	TXT--identifies card type.
5	Blank
6-8	Assembled address, punched in extended card code, where first byte of text from the card is to be loaded.
9-10	Blank
11-12	Number of bytes of text in this card, punched in extended card code.
13-14	Blank
15-16	ESD identification number of control section to which the text in this card belongs.
17-72	Up to 56 bytes of assembled program text, written in extended card code.
73-80	Deck sequence field.

Figure 22. TXT Card Format

Column	Contents
1	Loader identification (12-2-9); identifies this as a card acceptable to the loader.
2-4	RLD--Identifies card type.
5-10	Blank
11-12	Number of bytes of information in the variable field, columns 17-72, of this card. This is written in extended card code.
13-16	Blank
17-72	The variable field, written in extended card code. Each entry contains the following subfields, except that the relocation and position identifiers may be omitted in certain cases, as described under "Flags." Relocation identifier: two columns containing a code for the ESD entry that determines the relocation factor of the address constant. Position identifier: two columns containing a code for the ESD entry that describes the control section in which the address constant occurs. Flags: one byte, encoded as 000xnnsc, the fourth bit of the flag byte indicates the type of load constant. If it is 0, it is a non-branch type of load constant. If the bit is 1, it is a branch type load constant. (This bit is not checked by the Model 44 linkage editor.) The fifth and sixth bits, nn, indicate the length of the load constant. Code 00 means that the load constant is one byte, 01 indicates two bytes, 10 means three bytes, and 11 represents a length of four bytes. The seventh bit, s, indicates the sign of the relocation factor. Code 0 means that the relocation factor is to be added, and 1 means it is to be subtracted. The final bit, c, is the continuation flag. This bit is 1 if this entry contains one of a series of addresses with the same relocation and position identifiers. It is 0 if the entry represents the only address to be adjusted or if it is the last in a series of adjustments with the same relocation and position identifiers. Address: three columns containing the address of the load constant in extended card code.
73-80	Deck sequence field.

Figure 23. RLD Card Format

Column	Contents
1	Loader identification (12-2-9); identifies this as a card acceptable to the loader.
2-4	END--identifies the card type.
5	Blank
6-8	Optional field. May contain address to which control should be transferred at the end of the loading process, written in extended card code.
9-14	Blank
15-16	ESD identification number for the control section containing the entry point referred to in columns 6-8. (Must be blank if columns 6-8 not used.)
17-28	Blank
29-32	Length of last or only control section in ESD dictionary if its length was not specified in its ESD card.
33-72	Blank
73-80	Deck sequence field.

Figure 24. END Card Format

Column	Contents
1	Loader identification (12-2-9 punch); identifies this as a card acceptable to the loader.
2-4	END -- identifies the card type.
5-16	Blank
17-24	Symbolic transfer label supplied to assembler if label not defined within the assembly.
25-28	Blank
29-32	Length of control section if its length was not specified in its ESD card.
33-72	Blank
73-80	Deck sequence field.

Figure 25. Alternate END Card Format

REP Card

Replace cards may be supplied by a programmer to change portions of a program that has already been assembled or compiled.

A REP card specifies the address of one or more bytes to be changed and supplies the substitute text. Any text can be changed, and new text can be added. The replacement text overlays whatever is at the specified address.

When instructions are being changed, the replacements must occupy the same number of bytes. Conditional no-operation instructions can be used to fill gaps.

For the linkage editor, REP cards must follow the module's TXT and RLD cards and precede its END card. If a REP card is used to change an address constant, the text should not reflect the value to be added to the assembled text. REP cards are processed before the value in the symbol referred to by the address constant is added to the text containing the address constant.

REP cards must be written in hexadecimal notation. The format is shown in Figure 26.

Column	Contents
1	Loader identification (12-2-9); identifies this as a card acceptable to the loader.
2-4	REP -- identifies the card type.
5-6	Blank
7-12	Assembled address, in hexadecimal, of first byte of text to be replaced Must be right justified, with leading zeros, if necessary, to fill the field.
13	Blank
14-16	ESD identification number of the control section into which the text is to be inserted. This number, obtained from the language processor output listing, must be punched in hexadecimal, and right justified with leading zeros, if necessary to fill the field.
17-71	Up to eleven fields of replacement text. Loading of these fields starts at the address specified in columns 7-12. The fields are written in hexadecimal characters. The number of characters in each field must be divisible by four. The fields are separated by commas. If the last field terminates before column 71, it must be followed by at least one blank.
72	Blank
73-80	Deck sequence field.

Figure 26. REP Card Format

This section contains a listing of phase names used by system programs. These phases are permanently resident in the phase library. The system does not permit duplication of names in this library, so user programs should not attempt to assign these names to phases.

In this list, the names ASSEMBLE, FORTRAN, LNKEDT, and UTILS are included although they are not actually the names of phases. They can be cited in an EXEC statement, and the system converts them into the appropriate name of the first phase of the corresponding program. If, for example, EXEC ASSEMBLE is cited, the system converts this into EXEC BLAST and loads the first phase of the assembler program. Therefore, ASSEMBLE cannot be assigned to any user program.

Phase Name	System Component
ASSEMBLE	Assembler Phase 1
BAAIN15	2315 Disk Initialization
BABIN11	2311 Disk Initialization
BACDPRS	Dump/Restore
BADPRPU	Print/Punch
BBLDR1	Absolute Loader
BBLDR2	Absolute Loader
BCASC15	System Construction
BDAIP15	IPL Phase 1
BFSUPVSR	System Supervisor
BGD1	Disk ERP Phase 1
BGD2	Disk ERP Phase 2
BGT1	Tape ERP Phase 1
BGT2	Tape ERP Phase 2
BGCD	Card ERP
BGPR	Printer ERP
BGMG	Error Message Writer
BHOPEN1	OPEN Phase 1
BHOPEN2	OPEN Phase 2
BHOPEN3	OPEN Phase 3
BHOPEN4	OPEN Phase 4
BHCLOSE1	CLOSE Phase 1
BHCLOSE2	CLOSE Phase 2
BHMCHCHK	Machine Check (32K)
BHCNCEL1	CANCEL Phase 1
BHCNCEL2	CANCEL Phase 2
BHDUMP1	Dump Phase 1
BHDUMP2	Dump Phase 2
BIAJBBAS	Job Control Basic Phase
BIBJBPH1	Job Control Phase 1
BICJBPH2	Job Control Phase 2
BIDJBPH3	Job Control Phase 3
BIEJBPH4	Job Control Phase 4
BIFJBPH5	Job Control Phase 5
BKLNKEDT	Linkage Editor
BKLNKED1	Linkage Editor
BKLNKED2	Linkage Editor
BLAST	Assembler Phase 1
BLAZE	Assembler Phase 2
BMUTILS	Utilities ROOT Phase
BMUTCOPY	Utilities Data Set Copy
BMUTPCHA	Utilities Data Set Copy
BMUTSQMP	Utilities SQUEEZE/MAP
BMUTINIT	Utilities INITIAL Basic
BMUTINTP	Utilities INITIAL Tape
BMUTIN11	Utilities INITIAL 2311
BMUTIN15	Utilities INITIAL 2315
BNAFORT	FORTRAN Phase 1
BNAALL	FORTRAN Phase 2

Phase Name	System Component
BNAGEN	FORTRAN Phase 3
BNAEXT	FORTRAN Phase 4
BNAADJST	FORTRAN Object Fix Up
BNAXPND	FORTRAN POPS Expander
FORTRAN	FORTRAN Compiler
LNKEDT	Linkage Editor
UTILS	Utilities Program

* comments statement 18,20,42
* utilities option 91
/* statement see: end-of-data statement
/& statement 18,24,135
// statements see: job control statements

ABJCL option 92
abnormal termination
  input/output 66
  program 22,82
ABRED option 93
absolute library see: phase library
absolute loader 157
ACCESS job control statement
  described 32
  discussed 9,18
  errors 133-135
  unit assignments 13,15
accounting information 22-25,83
additional names 9,32
address constants 53,57,82,157,166
ALLOC job control statement
  described 25
  discussed 9,18
  errors 133-135
  unit assignments 13,15
alphabetic characters 20
alphameric characters 20
alternate entry points 81
alternate loading address 81
alternate tracks 102,105,153
AND, logical 84
ASA control characters 37,67,91,159
ASSEMBLE parameter 47,130
assembler diagnostic process 121
assembler language coding 45
assembler options 45
assembler output modules 46,58
assembly listing 46
asterisk utilities option 91
attention interruptions 68
AUTOLINK 56,117
automatic module library search 56

BCD 51
blank COMMON 54
block numbers 57,117
block count 69,74,77
block sizes 10,30,35,68
blocking 67
branching conventions 108
buffers, input/output 48,67,74

cancellation, job 22,82
CANCEL supervisor call 80,82
card code 51,160
card punching 14,37,46,50,89-101
card reading 14,157
carriage spacing 37,67,91,98
catalog, system 11,31,42
CATLG job control statement 11,18,42
CATLG parameter 11,31
channel queue 67
channel scheduler flags 160
channel status word 158
character set 20
CHAR option 97
CHECK supervisor call 64,76
CLOSE supervisor call 64,69,71
closing data sets 71
CLOSIP utilities option 94
CLOSOP utilities option 94
comments 20,42
comments statement 18,20,42
COMMON control sections 54,55
communications
  region 82
  with operator 14,42
  with programmer 14
compiler, FORTRAN 45
CONDENSE job control statement 10,18,41
condensing data sets 41
conditional interruptions 85,88
console printer-keyboard 14,42,68
continuation statements
  defined 19
  job control 19
  linkage editor 57
  stand alone disk initialization 105
  utilities 89
control characters 37,67,91,159
control sections 54,60
control statements
  general format 19
  job control 25-43
  linkage editor 57
  stand alone disk initialization 105
  utilities 89,90,102
conventions
  branching 108
  publications 21
  register 17
  system 9
convert feature for tape 28
COPY utilities statement 89
copy utility 89
CPYMEM utilities statement 90
CPYTO instruction 130
creating data sets 25
cross reference dictionary 111
cross reference listing 46,127
CTLASA parameter 36,67,91
CTLCHR parameter 36,67,91
cylinder boundaries, disk 30

data sets
  blocking 35,67
  cataloging 11,31,42
  closing 71-73
  combining 93,94
  condensing 41
  copying 89
  creating 25,32
  creating with /* 31

defined 9
deleting 40,103
directoried 9,32,41,69
dummy 32,71,73
editing 47
eliminating 40,103
expired 40,103
formatting 31
identifying 32
labels 147,149
naming 27,32,41
opening 69-71
positioning 34,47,70,73,77,93
printing 37,75,89
punching 37,89
reading 74,89
renaming 41
repositioning 47,70,73,77,93
splitting 93,94
squeezing 41,103
system 11,15
truncating 95
unit record 27,37
writing 75
DECK parameter 46,50
deck serialization 47
defective tracks 102,105
DELETE job control statement 10,18,40
deleting data sets 40,103
delimiter utilities options 94
density, tape 28
devadr field 27,30
device independence 64
device initialization 102,105
disconnecting units 72,94
disk
blocking 35,67
condensing 41,103
initialization 102,105
labels 69,72,102,147
mapping 39,104
opening and closing 69,72
options 28,29
organization 10
sectors 68,102
squeezing 103
volumes 10
distributed system 13,69,83
directoried data sets
adding new members 32,69
condensing 41
copying 90-101
creating 25
deleting members 40
directory length 30
opening and closing 69-73
printing 90-101
punching 90-101
displacement, phase 59
display mode 98
double buffers 48
dual density feature 28
dummy assignments 32,34,71,73
dump and cancel 86
dump facilities 22,107
dump formats 108
DUMP parameter 21,22,107
DUMP routine 107
DVADR utilities parameter 102,104,105

early unit assignments 15
EBCDIC 51
EDATE utilities parameter 106
edit data sets 47
eliminating data sets 40,103
eliminating members 40
END cards 61,137,157,161
end-of-data statement (/*)
creating data sets 31
described 25
discussed 18
early unit assignments 15
use with linkage editor 52,57
use with utilities 90,91
end of extent 34,66,70,77
end of file marks
disk 34,66,70-73,102,105
tape 34,66,70-73,79
end of job statement (/&) 18,24,135
ENDIN utilities option 94
ENDOUT utilities option 94
ENDUP instruction 130
ENTRY linkage editor statement 57,61
entry points
absolute loader 157
interruption routine 86,87
phase 16,55,61,80-82
EOJS supervisor call 80,82
error flags 111,122
error codes 66,71,73
error counters 158,160
error severity
assembler 83,122
linkage editor 136
ESD cards 137,157,161
EXCL utilities option 100
exclude members 100
EXCP supervisor call 64
EXCP input/output level 64
EXEC job control statement
described 21
discussed 18
use with assembler 45-49
use with FORTRAN 45,50
use with linkage editor 52,57
use with utilities 89
execute channel program 64
expiration dates 37,70,106,148
expired data sets 103
external references 56
EXT parameter 34,70
external symbol dictionary 56,137,156,160
EXTRACT supervisor call 80,82,84

FETCH supervisor call 79,80
file control block 35,65,68,148,159
FILL utilities option 97
fixed point overflow 86
FMT parameter 30
format 1 label 37,149
format 4 label 152
format 5 label 40,103,104,154
forms control characters 37,67,91,98
FORTRAN
EXEC statement parameters 50

mathematics and service routines 16,108
maximum block size 35
publications 7
source program 45
FRESH parameter 27,30

general assignment units 14,15
GETIME supervisor call 80,85,87

HEAD utilities option 99
header cards 91
header labels 70,102,147
HEX option 97
High Resolution Timer 88
home addresses 102,105

identification code, disk 10,29,103,105
identification field 96,97
IGJCL option 92
IGN parameter 34
IGRED option 93,141
INCL utilities option 100
INCLUDE linkage editor statement 57,60
incorrect length condition 66,74,141
index 169
initialization, volume 102,105
initial entry point 61
initial program loading 84,87
INITIAL utilities statement 101,102
INITIAL stand alone utilities 105
input/output
abnormal termination 66
buffers 48
control blocks 65,158
error codes 66
execute channel program 64
maximum efficiency 68
operations 74-78
read/write level 64-78
INSERT supervisor call 80,82,84
interprogram communications area 83,84
interruptions
attention 68
conditional 85
input/output 66,75
program check 86
timer 87
interruption routines
input/output 66,75
user 76,85
interval timer 87
intraprogram communciations area 83,84
invalid request 66

job control flags 160
job control processor
described 18
use of SYSUAS 14,134
closing data sets 69
job control statements
ACCESS 32
ALLOC 25
CATLG 42
comments (*) 42
CONDENSE 41
DELETE 40
end-of-data (/*) 25
end-of-job (/&) 24
EXEC 22
JOB 21
LABEL 35
LISTIO 39
PAUSE 42
RENAME 41
RESET 39
REWIND 43
STOP 25
UNCATLG 42
UNLOAD 43
job control utilities option 91
JOB job control statement 18,21
jobname 22
job steps 7,23,82

KEEP parameter 57
key field 38,154
KEYLEN parameter 38
KEYLOC parameter 38

LABEL job control statement
described 35
discussed 10,18,67,147
errors 133-135
label processing 69-73,147
labels
data set 35,147
discussed 147
disk space management 148
header 70,147,155
trailer 69-72,147,155
volume 35,147
VTOC 148
language processors
assembler 45
FORTRAN 45
output modules 14,52,58
use of symbolic units 14
lines per page 46,99
LINES utilities option 99
LINK parameter 46,50
linkage editor
control statements 57
described 52-62
error messages 136
input 46,50,52
location counter 59
output 53
use of symbolic units 13
list mode 98
LIST parameter 46
LISTC parameter 46
LISTIO job control statement 39
literals 48,126
LNCTn parameter 46
load address 81
loading address 81
loading phases 21,79-81
load point 70,72,78,93
LOAD supervisor call 79-81
logical AND, OR 84
logical records
copying 94-98
deleting 96
printing 94-98
punching 94-98

magnetic tape
  convert feature 28
  initialization 102
  labels 69-72,102,147,155
  options 28
  translate feature 28
main entry point 61,79-81
main storage allocation
  assembly parameters 48
  problem program area 55,59,83
manipulating records 95
MAP parameter 51,57
mapping
  disk 104
  storage 57
MAP utilities statement 104
members
  adding 32
  copying 90-101
  names 9,32
  input/output 69-71
  opening and closing 69,72
  rearranging 100
messages
  error 121-146
  to operator 14,42
  to programmer 14,42,121
MODE utilities option 98
module library
  branching conventions 108
  defined 16
  dump routines 107
  FORTRAN routines 107
  use by linkage editor 53
  searching 56
MODULE linkage editor statement 57,58
multiphase programs 52
multiple names 32
multiplex channels 102

names
  control sections 54
  data set 27,32
  member 32
  multiple 9,32
  phase 58,79,167
  program 23
  44 character 41
NEW parameter 34
NOAUTO parameter 57,60,107
NODECK parameter 46,50
NODUMP parameter 21,22
NOKEEP parameter 57
NOLINK parameter 46,50
NOLIST parameter 46
NOMAP parameter 51,57
non data transmit 66
non sequential data sets 9,34
non sequential processing 9,31,77
NONUM option 98
NOSOURCE parameter 50
NOTE supervisor call 64,77
NOWRCHK option 28,38
NOXREF parameter 46
numeric characters 20
NUM instruction 130,131
NUM option 98
OMIT instruction 130
open and close utilities options 93
opening data sets 69
OPEN supervisor call 64,69
OPENIP utilities option 93
OPENOP utilities option 93
operator messages 14,42
OR, logical 84
output format options 98
output option parameters 45
overlay capability 52

padding records 96
PAD utilities option 96
page ejection 37,67,91
page numbering 98
parameter lists 19,23,67,107
parity, tape 28
patch cards 58,137,161
PAUSE job control statement 18,42
pauses 42
PCHABS utilities statement 90
PCHMEM utilities statement 90
PDUMP routine 107
permanent input/output errors 66,93
phase library
  block numbers 57,117
  copying 90
  defined 15
  entering phases 53
  fetching from 13,52,79
  loading from 13,81
  punching 90
PHASE linkage editor statement 57,58
phase names 58,79,167
phases
  copying 90
  entry point 61,79-81
  linkage editing 52-63
  loading 59,80
  origin 53,58
  relocation 53,81
POINT supervisor call 64,77
positioning data sets 34,70,73,77
post request flag 157
PPMEM utilities statement 90
printing format 98
printer manipulation 37,67,91
PRINT utilities statement 89
privileged state 86
problem program area 55,59,83
progname field 23
program check interruptions 86
programming error routines 86
program status word 86
program testing 32,34
PRTMEM utilities statement 90
PRTPCH utilities statement 89
pseudo directory 14,53
PUNCH utilities statement 89

queues
  input/output 67
  interruption 87

READ supervisor call 64,74
reassignment, unit 13,39
RECLEN parameter 38

record blocks 10,30,35
record sizes 35,38
records, logical 94-98
redundancy utilities option 93
register conventions 17,108
relocatable library see: module library
relocatable modules 52,58,107
relocation dictionary 111
relocation loader cards 137,157,161
RENAME job control statement 18,41
REP cards 58,137,157,166
replace cards 58,137,157,166
repositioning 70-73,77
request control block
  format 158
  use for input/output 65,74-78
reserialization 47
RESET job control statement 39
restoring symbolic units 39
return address register 17,108
return code register 17,66
return codes 17,65,66
REWIND job control statement 43
REWIND supervisor call 64,78
REWND instruction 130,131
RLD cards 137,157,161
ROOT parameter 53,59,60
ROUND parameter 30
RTXIPC supervisor call 80,85,86
RTXITC supervisor call 80,85,87

SAME parameter 27,34,38
save area register 17,85
save areas 85,108
SDSABS data set
  described 13,14
  phase library 15,79,81,167
  use by linkage editor 53
SDSCAT data set 15
SDSD 102
SDSIPL data set 15
SDSIPT data set 14,53
SDSLOG data set 14
SDSLST data set 14
SDSOPT data set 14
SDSPCH data set 14
SDSPSD data set
  described 14
  use by language processors 45
  use by linkage editor 53
SDSREL data set
  altering 17
  described
  module library 17,107
  use by linkage editor 53
SDSRDR data set 14
SDSUAS data set 14
SDS000 data set
  described 14
  use by language processors 45
  use by linkage editor 52
sectors, disk 68,102
self-defining symbols 121
sense data 158
SEQIN utilities option 96
SEQOUT utilities option 97
sequence checking 96,142
sequential data sets see: data sets
serialization, deck 47
serial numbers, volume 10,29,103
SETIME supervisor call 80,85,87
shifting data sets 103
SIZIN utilities option 95
SIZOUT utilities option 95
SKPTO instruction 130,131
source language listing 46,50
SOURCE parameter 50
space allocation parameters 48
space management labels 40,103,104,154
spacing, printer 98
SPACE utilities option 98
special characters 20
SQUEEZE utilities statement 103
squeezing volumes 103
standard heading 99
standard labels 147
standard unit assignments 13
START utilities option 96
stepname field 23
STOP job control statement 25
storage allocation 49,83
storage map 51,57
STXIPC supervisor call 80,85,86
STXITC supervisor call 80,85,87
supervisor, system 7,82
supervisor calls 49,64-89
switches
  user program 24,83,85
  variable precision 23
switch options 23
SYMBMAX parameter 49
SYMBMIN parameter 49
SYMBn parameter 49
symbolic units
  assigning 25,32
  defined 12
  disconnecting 72
  index 65
  listed 12-15
  listing assignments 39
  reassignment 13,25,32
  restoring 13,39
  standard assignments 13,39
symbol table, assembler 48,111,126
SYSAB1
  described 12,13
  errors 132
SYSAB2
  described 14
  reassignment 53
  use by linkage editor 53,140
  use by utilities 90
SYSDMY 15
SYSIPT
  described 14
  discussed 12,21
  use by linkage editor 52,53,58
  use by update 47
  use by utilities 89-94
SYSLOG
  described 14
  discussed 12,71,105,121
SYSLOG parameter 106
SYSLST
  described 14
  discussed 12,20,121

use by update 47
SYSOPT
described 14
use by assembler 46
use by FORTRAN 50,51
use by utilities 89,90
SYSPCH
described 14
use by assembler 46
use by FORTRAN 50
use by utilities 89,90
SYSPSD
described 14
use by language processors 45
use by linkage editor 53,58,139
SYSRDR
described 14
discussed 12,21,25
SYSREL
described 14
errors 137,142
module library 17,107
use by linkage editor 53
system data sets 11
system generation 105
system residence volume 12,134
system units 13,32,65
SYSUAS 14
SYSUNI index 65
SYS000
described 14
errors 137,140
use by assembler 45,46
use by FORTRAN 51
use by linkage editor 52,58,59
SYS001 14,140
SYS002 47,89-94
SYS003 47,89-94,102
SYS001-240 13-15,69

table of contents, volume 10,40,102,105,147
tape
convert feature 28
data sets 25,70,72
density 28
expiration date 37,70,106
initialization 102
labels 69-72,102,147
options 28
parity 28
positioning 34,47
rewinding 43,47,78
translate feature 28
unloading 43,73
text cards 137,157,161
timer interruptions 87
timer services 87
track descriptor records 102,105
TRACK parameter 103,106
tracks, alternate 102,105,153
tracks, defective 102,105
trailer labels 69,72,102,147
transfer point 61
translate feature 28
transmission errors 66
transient routines 16
TRUNC utilities option 95
type, device 27,29,159
type utilities option 97
TXT cards 137,157,161

UNCATLG job control statement 11,18,42
unit control block 65,68,159
unit record data sets 22,67,91
units, symbolic see: symbolic units
UNLOAD job control statement 43
UNLOAD supervisor call 64,78
unnamed control sections 54
unresolved external references 57
UPDASMB1 parameter 47,130
UPDASMB2 parameter 47,130
UPDASMB3 parameter 47,130
update parameters 47
update program 46,47
updating capabilities 46
UPDATE1 parameter 47,130
UPDATE2 parameter 47,130
UPSAND supervisor call 80,82,84
UPSOR supervisor call 80,82,85
user communications region 82
user program switch 24,83,85

variable precision switch 23
VOLID parameter 103,105
volumes
control statement field 27
defined 9
disk 10,67
labels 69-73,102,147
listing contents of 39,103
serial number 10,29,103
table of contents 10,40,102,105,147
tape 10

VTOC definition label 102,105,147

WAIT supervisor call 64
warning messages 57,122,136
WEF supervisor call 64,78
WRCHK option 28,38
WRITE supervisor call 64,75
write validity checking 29,37

XREF parameter 46

Printed in U.S.A. C28-6812-1

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

READER'S COMMENTS

Title: IBM System/360 Model 44 Programming System Guide to System Use

Form: C28-6812-1

Your comments assist us in improving the usefulness of our publications; they are a major part of the input used for technical newsletters and revisions.

Please do not use this form for technical questions about the system; it only delays the response. Instead, direct your technical questions to your local IBM representative.

Corrections or clarifications needed:

Page	Comment

If you wish a reply, please include your name and address below:

C28-6812-1

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, N.Y. 10020

Attention: PUBLICATIONS

fold fold

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

Printed in U.S.A. C28-6812-1