Horst

SPERRY UNIVAC

PUBLICATIONS UPDATE

Operating System/3 (OS/3)

System Service Programs (SSP) User Guide (Series 90)

For System 80 see UP-8841

UP-8062 Rev. 8-B

This Library Memo announces the release and availability of Updating Package B to "SPERRY Operating System/3 (OS/3) System Service Programs (SSP) User Guide", UP-8062 Rev. 8.

This update documents the following new features for the 8.1 release:

- a new parameter for the SMCLIST canned job control stream to sort the SMC listing by the time alone that SMCs were applied, not time and date; and
- Change in the default for the FMT parameter on the SMCLIST job control stream from full to condensed.
- an enhancement to the condensed listing produced by SMCLIST to show which SMCs were backed out, replaced, or not installed because of an error during installation.
- Additions to the load code phase definition record format and block load module header record format tables to identity Base 0 and Key 0 shared code.

All other changes are corrections or expanded descriptions applicable to features present in SSP prior to the 8.1 release.

Copies of Updating Package B are now available for requisitioning. Either the updating package only or the complete manual with the updating package may be requisitioned by your local Sperry representative. To receive only the updating package, order UP-8062 Rev. 8–B. To receive the complete manual, order UP-8062 Rev. 8.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ and MZ	Mailing Lists A00, A01, 18, 18U, 19, 19U, 20, 20U, 21, 21U, 75, 75U, 76 and 76U (Package A to UP-8062 Rev. 8, 31 pages plus Memo)	Library Memo for UP-8062 Rev. 8–B RELEASE DATE: June, 1983

SPERRY UNIVAC

Horst.

PUBLICATIONS UPDATE

Operating System/3 (OS/3)

System Service Programs (SSP)

User Guide

UP-8062 Rev. 8-A

This Library Memo announces the release and availability of Updating Package A to "SPERRY UNIVAC Operating System/3 (OS/3) System Service Programs (SSP) User Guide", UP-8062 Rev. 8.

This maintenance update for Release 8.0 corrects assorted typographical errors in Sections 2, 9, 10, and 13.

Copies of Updating Package A are now available for requisitioning. Either the updating package only or the complete manual with the updating package may be requisitioned by your local Sperry Univac representative. To receive only the updating package, order UP-8062 Rev. 8-A. To receive the complete manaul, order UP-8062 Rev. 8.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ and MZ	Mailing Lists A00, A01, 18, 18U, 19, 19U, 20, 20U, 21, 21U, 75, 75U, 76 and 76U (Package A to UP-8062 Rev. 8, 23 pages plus Memo)	Library Memo for UP-8062 Rev. 8-A RELEASE DATE: January, 1983

SPERRY UNIVAC
COMPUTER SYSTEMS

PUBLICATIONS REVISION

Operating System/3 (OS/3)

System Service Programs (SSP)

User Guide (Series 90)
For System 80 see UP-8841

UP-8062 Rev. 8

This Library Memo announces the release and availability of "SPERRY UNIVAC® Operating System/3 (OS/3) System Service Programs (SSP) User Guide", UP-8062 Rev. 8.

Revision 8 of this manual incorporates the following changes:

- SAT Librarian:
  - Programming examples added
  - New PAGE statement
  - New // PARAM statements: ERROR, PRINT, PRTOBJ, TAPEFILES
  - COP statement expanded to replace ADD statement (ADD is still supported.)
  - New A option to process all groups in a file
  - Load module patch addresses can be expressed relative to start of phase.
  - New ORG directive for object and load module corrections
  - New syntax for canned librarian control streams
- New UNXFC parameter for disk and diskette preps
- New FDATA and PARTL parameters for diskette prep
- New COPYREL canned job control stream
- Executing SU$C16, SU$CSL, and DMPRST interactively.
- Other changes applicable to system service program routines for 8.0 and prior releases.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ and MZ	Mailing Lists A00,A01,18,18U,19,19U, 20,20U,21,21U,75,75U,76 and 76U (Cover and 419 pages)	Library Memo for UP-8062 Rev. 8
		RELEASE DATE:
		September, 1982

Destruction Notice: If you are going to OS/3 release 8.0, use this revision and destroy all previous copies. If you are not going to OS/3 release 8.0, retain the copy you are now using and store this revision for future use.

Copies of UP-8062 Rev. 7, UP-8062 Rev. 7-A and UP-8062 Rev. 7-B will be available for 6 months after the release of 8.0. Should you need additional copies of this edition, you should order them within 90 days of the release of 8.0. When ordering the previous edition of a manual, be sure to identify the exact revision and update packages desired and indicate that they are needed to support an earlier release.

Additional copies may be ordered by your local Sperry Univac representative.

System Service Programs (SSP)

OS/3

User Guide

Environment: 90/25, 30, 30B, 40 Systems

SPERRY UNIVAC

This document contains the latest information available at the time of preparation. Therefore, it may contain descriptions of functions not implemented at manual distribution time. To ensure that you have the latest information regarding levels of implementation and functional availability, please consult the appropriate release documentation or contact your local Sperry Univac representative.

Sperry Univac reserves the right to modify or revise the content of this document. No contractual obligation by Sperry Univac regarding level, scope, or timing of functional implementation is either expressed or implied in this document. It is further understood that in consideration of the receipt or purchase of this document, the recipient or purchaser agrees not to reproduce or copy it by any means whatsoever, nor to permit such action by others, for any purpose without prior written permission from Sperry Univac.

Sperry Univac is a division of the Sperry Corporation.

This document was prepared by Systems Publications using the SPERRY UNIVAC UTS 400 Text Editor. It was printed and distributed by the Customer Information Distribution Center (CIDC), 555 Henderson Rd., King of Prussia, Pa., 19406.

PRINTED IN U.S.A.

PAGE STATUS SUMMARY

ISSUE: Update B – UP-8062 Rev. 8
RELEASE LEVEL: 8.1 Forward

Part/Section	Page Number	Update Level
Cover/Disclaimer		Orig.
PSS	1	B
Preface	1 thru 3	Orig.
Contents	1 thru 6	Orig.
	7	B
	8 thru 10	Orig.
	11	B
	12	Orig.
PART 1		
	Title Page	Orig.
1	1 thru 12	Orig.
PART 2		
	Title Page	Orig.
2	1 thru 6	Orig.
	7	A
	8 thru 24	Orig.
	25	A
	26 thru 40	Orig.
	40a	Orig.
	41 thru 64	Orig.
	64a	Orig.
	64b	A
	64c	Orig.
	65 thru 84	Orig.
	85	A
	86 thru 104	Orig.
3	1 thru 13	Orig.
PART 3		
	Title Page	Orig.
4	1 thru 44	Orig.
5	1 thru 3	Orig.
6	1 thru 22	Orig.
7	1 thru 9	Orig.
8	1 thru 34	Orig.
PART 4		
	Title Page	Orig.
9	1, 2	Orig.
	3	A
	4 thru 14	Orig.
	14a	Orig.
	15 thru 20	Orig.
9 (cont)	21, 22	B
	22a thru 22d	B*
	23, 24	Orig.
	25 thru 27	A
	28	B
10	1, 2	Orig.
	3, 4	A
	5 thru 7	Orig.
11	1 thru 4	Orig.
12	1 thru 23	Orig.
13	1 thru 23	Orig.
	24	A
	25 thru 27	Orig.
14	1 thru 16	Orig.
15	1 thru 3	Orig.
16	1, 2	B
	2a	B*
	3 thru 5	Orig.
	6, 7	B
PART 5		
	Title Page	Orig.
Appendix A	1 thru 3	Orig.
Appendix B	1 thru 10	Orig.
	11	B
	12 thru 14	Orig.
	15	B
	16 thru 18	Orig.
Index	1	Orig.
	2, 3	B
	4 thru 10	Orig.
	11	B
	12 thru 14	Orig.
User Comment Sheet		

Part/Section	Page Number	Update Level

*New pages

All the technical changes are denoted by an arrow (⟹) in the margin. A downward pointing arrow (⇓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (⇑) is found. A horizontal arrow (⟹) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

Preface

This manual is one in a series designed to instruct and guide you in the use of the SPERRY UNIVAC Operating System/3 (OS/3). Specifically described are the OS/3 system service programs and their effective use. The system service programs include the system librarian, the linkage editor, and the standard system utilities.

This manual is intended for the novice programmer with a basic knowledge of data processing, but with limited programming experience, and for the more sophisticated programmer whose experience is limited to systems other than SPERRY UNIVAC systems. Two other manuals are available that cover the system service programs; one is an introductory manual and the other is a programmer reference manual (PRM). The introductory manual briefly describes the system service programs and their facilities. The PRM describes, in skeletal form, the characteristics of the system service programs and is intended as a quick-reference document for the programmer experienced in the use of the system service programs.

This user guide is divided into the following parts:

- PART 1. OS/3 SYSTEM SERVICE PROGRAM REPERTOIRE

  Introduces you to the various system service programs through descriptions of their intended purposes within the OS/3 operating system, their capabilities, and the terms peculiar to their functional operation.

- PART 2. THE LIBRARIANS

  Describes the functional characteristics of the system librarians relevant to you, the control statements you may use to direct their operation, and the various library mapping elements they are capable of producing.

- PART 3. THE LINKAGE EDITOR

  Describes the functional characteristics, programming considerations, and control statements required to allow you to effectively use the linkage editor as it is intended to be used. Also describes the link-edit mapping data produced by the linkage editor for every load module it produces.

- PART 4. SYSTEM UTILITIES

  Describes the utility programs provided by OS/3 to initialize disk, diskette, and tape volumes; copy disk, diskette, and tape volumes; and list software maintenance corrections.

- PART 5. APPENDIXES

  Appendix A presents the canned job control streams and the document numbers where they are described. Appendix B describes the code set components that, when combined in a particular sequence, make up a program source module, a macro/jproc source module, an object module, a load module, or a group code set module.

Each of these parts consists of one or more sections, which cover the different aspects of the subject matter contained in each part.

To fully understand and appreciate the functions performed by the system service program, you should be familiar with the information contained in the current version of the following SPERRY UNIVAC publications:

- 1974 ANS COBOL programmer reference, UP-8613
- Assembler user guide, UP-8061
- Basic COBOL supplementary reference manual, UP-8057
- Consolidated data management macro lanquage user guide/programmer reference, UP-8826
- Data management user guide, UP-8068
- Dump analysis user guide/programmer reference, UP-8837
- Extended COBOL supplementary reference manual, UP-8059
- File cataloging concepts and facilities, UP-8860
- Interactive services commands and facilities, UP-8845
- Job control user guide, UP-8065
- Operations handbook for operators, UP-8072 (for 90/30 and 90/40 systems) and UP-8511 (for 90/25 and 90/30B systems)
- Spooling and job accounting concepts and facilities, UP-8869
- Supervisor user guide, UP-8075

- System installation user guide/programmer reference, UP-8074
- System messages programmer/operator reference, UP-8076

The degree of familiarity required varies with the product in question. For example, the linkage editor user has to be familiar with almost all of the documents. On the other hand, those using the librarian and the system utilities require only a few of the documents. And, those interested only in the dump routines can find nearly all of the information they need in this document alone.

Contents

PART 2. THE LIBRARIANS

2. SAT LIBRARIAN FUNCTIONAL CHARACTERISTICS

3. MIRAM LIBRARIAN FUNCTIONAL CHARACTERISTICS

PART 3. THE LINKAGE EDITOR

4. FUNCTIONAL CHARACTERISTICS

5. PROGRAMMING CONSIDERATIONS

6. CONTROL STATEMENTS

PART 4. SYSTEM UTILITIES

10. ASSIGN ALTERNATE TRACK (AAT)

11. TAPE PREP (TPREP)

FIGURES

TABLES

PART 1. OS/3 SYSTEM SERVICE PROGRAM REPERTOIRE

1. Introduction

1.1. GENERAL

The system service programs are those programs required to support the operation and organization of the operating system in which your problem programs are to be executed. These programs allow you to construct and reorganize the program libraries in your system, create program modules for execution in your system, initialize tape and disk volumes for the storage of your program and data files, and obtain printouts of main storage.

The system service programs are introduced and outlined briefly in this section and discussed in full detail in the subsequent parts of this document. The common and program names of the system service programs are:

Common Name	Program Name
System librarian for SAT files	LIBS
System librarian for MIRAM files	MLIB
Linkage editor	LNKEDT
Initializing disk volumes	DSKPRP
Assign alternate track (nonsectored disk)	DSKPRP
Disk dump/restore	DMPRST
Tape prep	TPREP
System utility copy (sectored)	SU$C16
System utility copy (nonsectored)	SU$CSL
Hardware utilities (interactive DMPRST, SU$C16, and SU$CSL)	HU

Common Name	Program Name
Stand-alone disk copy (IDA)	SU$IDA
Stand-alone disk copy (SEL)	SU$SEL
System log accumulation utility	SY$LOG
JOBLOG report program	JBLOG
Catalog manipulation utility routine	JC$CAT
Diskette utility	CREATE

In addition, a system service program symbiont, SL$$SU, is initiated from the system console by the SU (system utility) or the TU (tape utility) command.

1.2. THE SYSTEM LIBRARIANS

There are two system librarians that can maintain and manipulate both your system and user libraries. For all non-MIRAM library files, you use the SAT librarian (LIBS). For MIRAM libraries, you use the MIRAM librarian (MLIB).

The librarians are also used during OS/3 system generation to tailor the SYSRES program libraries. The librarians are capable of manipulating the library files at your request and in the specific manner directed. The functions performed by the librarians are controlled by a set of integrated subroutines, file tables, and overlay segments associated with the supported indivdual functions.

Your OS/3 system can support several independent system and user program libraries, and the librarians can be used to maintain each one. A program library consists of one or more library files. A single library may contain both user and system files, or it may be used exclusively for one or the other. Each file within the program library contains a directory partition, the library file directory, and two data partitions.

The program library files can be composed of any combination of the following:

- Program source modules (language processor code)
- Macro/jproc source modules (language processor code/job control)
- Object modules (language processor output/linkage editor input)
- Load modules (linkage editor output)
- Module groups

- Screen format modules
- Saved run library modules

The library files may be composed of system or user code used for either program generation or execution. The code may be in any of the listed formats and may, from time to time, change in form, content, or relative position within a given file. The mixing and grouping of module types is a user option, and module groups can contain modules of the same or different types. Figure 1-1 depicts the structure of a SAT program library, showing various component configurations.

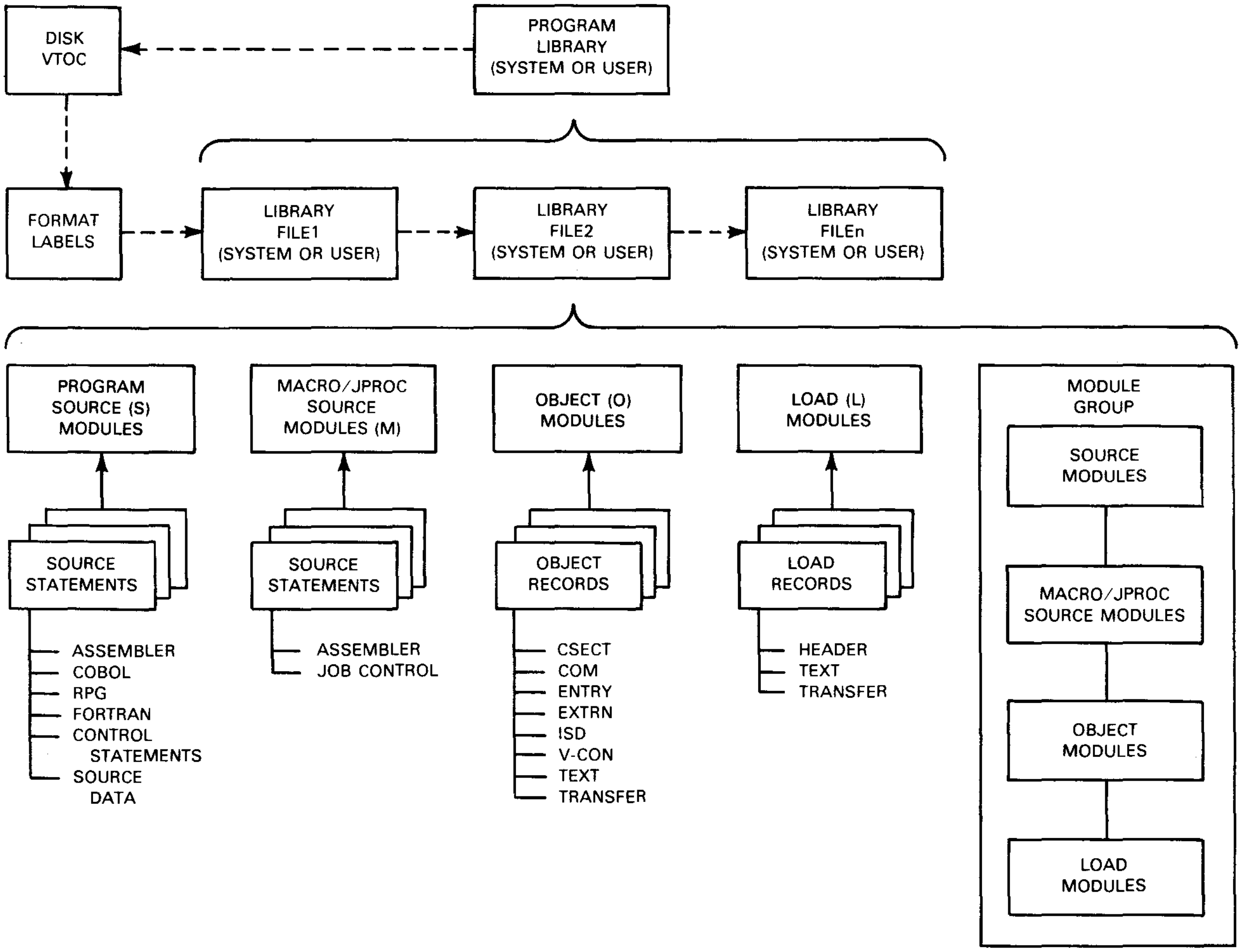


Figure 1-1. Program Library Structure

The librarians can perform, on all or specific portions of library files, such tasks as copying, merging, listing, or punching on cards the contents of specified files. The librarian can also add to or delete from a file or files. In fact, the OS/3 librarians can perform all the tasks you may be expected to require for program file management. The librarians, however, cannot perform these tasks on multivolume tape files. The tasks are initialized and directed through a set of control statements introduced to the librarian through the control stream. The librarians and the function associated with each task are fully explained in Part 2 of this document.

1.3. THE LINKAGE EDITOR

The OS/3 linkage editor converts and combines object modules and object module elements (control sections and common sections), produced by the OS/3 language processors, into modules that can be loaded into a system by the supervisor for execution. The modules produced by the linkage editor are called load modules. Only programs in load module form can be executed in an OS/3 environment, and the only way to convert object modules into a load module is by using the linkage editor.

The linkage editor produces three types of load modules:

- Single-phase (reentrant or nonreentrant)

- Multiphase (nonreentrant)

- Multiregion (nonreentrant)

A single-phase load module consists of a single program segment loaded into main storage each time the program is to be executed. Unless otherwise directed, the linkage editor will always produce a single-phase load module. Multiphase and multiregion load modules are composed of more than one program segment, each segment being a program phase loaded into main storage and executed individually, as required by the logic of the program. The linkage editor will create a multiphase or multiregion load module from one or more object modules only if directed to do so by the user through the linkage editor control statements. Savings in main storage space and increased system performance can be realized through proper application of multiphasing and multiregioning.

The capabilities of the linkage editor provide the system user with the following advantages:

- If a program logic error is discovered in a particular object module or control section of a program, only the incorrect element need be recompiled or reassembled. Afterward, the entire program can be relinked without extensive reassembling or recompiling.

- Subroutines or elements required in more than one program phase need be preserved only once as relocatable object code because a single module can be individually included in any number of load modules by the linkage editor.

- A single load module may actually consist of object elements produced by several different language processors because all processors generate compatible output object code acceptable to the linkage editor.

- Reentrant modules can be shared by other load modules, resulting in the overall reduction of main storage requirements.

Part 3 details the capabilities of the linkage editor.

1.4. THE SYSTEM UTILITIES

The system utilities are available to do the following:

- Test and prepare all tape, diskette, and disk volumes for use by OS/3.

- Manipulate the system catalog file, $Y$CAT.

- Create and maintain diskette files.

1.4.1. Disk Utilities

The disk utilities perform the following functions:

- Initialize sectored (SPERRY UNIVAC 8415, 8416, and 8418 Disk Subsystems) and nonsectored (SPERRY UNIVAC 8411, 8414, 8424, 8425, 8430, and 8433 Disk Subsystems) disk volumes; also 8413 diskettes

- Perform surface analysis for sectored and nonsectored disk volumes

- Assign alternate tracks on nonsectored disk volumes

- Dump, restore, or copy disk or tape volumes or files

- Place initial load control storage (ILCS) modules on disk

- Assign new volume serial numbers to active disks

1.4.2. Tape Utilities

The tape utilities initialize tape volumes for use in the system. Up to 36 tapes can be initialized at one time.

1.4.3. Hardware Utilities

The hardware utilities consist of the dump/restore and disk copy routines performed interactively. There are two types of disk copy routines: SU$C16 (used when copying sectored disks) and SU$CSL (used when copying nonsectored disks).

1.4.4. System Utility Symbiont

The system utility symbiont is a multipurpose utility enabling the operator to perform different utility operations from the system console, for example, reproducing cards and printing a tape.

1.4.5. Diskette Utility

The diskette utility is available to create files on the 8413 diskette and to write job control streams to the created diskette files. The diskette utility is executed by running a canned job control stream (RV WRT) and then directed through a series of queries appearing on the system console to which the user must respond.

1.4.6. List Software Maintenance Corrections (SMCLIST)

The SMCLIST canned job control stream produces a listing of all software maintenance corrections (SMCs) contained in the SMCLOG file.

1.5. LOGGING AND CATALOGING FACILITIES

The logging and cataloging facilities include the system log accumulation utility, the job log report program, and the catalog manipulation utility.

The system log accumulation utility is used to transfer job log and console log records from the system spool file to a user disk or tape file. Once in the user file, they are available for further processing by a job accounting and bookkeeping programs. The job log report program is used to produce a job accounting report from the SYSLOG file created by the system log accumulation utility. For a detailed description of the use and function of the system log accumulation utility and the job log report program, see the spooling and job accounting concepts and facilities manual, UP-8869 (current version).

The catalog manipulation utility (JC$CAT) is used to access the system catalog file $Y$CAT. Using JC$CAT, you can obtain a printout of the contents of $Y$CAT; assign, delete, or change a catalog password; and copy $Y$CAT to another disk or tape volume. For a detailed description of the use and function of the catalog manipulation routine, see the file cataloging concepts and facilities manual, UP-8860 (current version).

1.6. DUMP ROUTINES

Several dump routines are available to you as an aid in debugging the system or a single program if error conditions occur. The available dump routines are the SYSDUMP routine, JOBDUMP routine, and USE EOJ DUMP routine.

The system dump routine (SYSDUMP) is provided to you as a system debugging aid. Its primary function is to translate and print out the state of the operating system in the event the system terminates abnormally or is terminated by the operator because of abnormal operation. The abnormal termination of the system is commonly referred to as a system crash. The SYSDUMP routine translates the bits and bytes of information present in the system at the time it crashes into text and charts that can be recognized and analyzed by one familiar with the structure of OS/3.

SYSDUMP is a feature of the supervisor and is automatically included in the supervisor at system generation time unless specifically not included. Once included, it can be called upon by the system operator to translate the state of the operating system at any time. The execution of the SYSDUMP routine is always under the control of the supervisor (it is not a stand-alone routine) and is designed to run in a multiprogram environment.

JOBDUMP is a scaled down version of SYSDUMP designed to interpret the state of a single user job if the job terminates abnormally. JOBDUMP, just as with SYSDUMP, translates the state of the user job region into text and charts useful in interpreting and debugging the program. The output format is the same as that of SYSDUMP for job prologues and main storage hexadecimal/character dumps.

The user EOJ dump is a hexadecimal printout of the user job region initiated by either the DUMP macro or an abnormal termination of the job. The user EOJ dump can be used to determine the nature of an abnormal termination or as a diagnostic tool for program debugging.

For a detailed description of the use and function of the dump routines, see the dump analysis user guide/programmer reference, UP-8837 (current version).

1.7. PROGRAM ERROR CHECKING (UPSI BYTE)

The OS/3 system provides every job with a 12-byte communications region residing in the job preamble. The last byte of this region is the user program switch indicator (UPSI). The UPSI byte is used to pass information from one job step to the next job step and to indicate the presence of program errors. The librarian, the linkage editor, the utilities and dump routines, and other executable system components set the UPSI byte if errors are detected. You can test the UPSI byte during program execution to determine the nature and severity of any errors. The basic bit usage of the UPSI byte is:

- Bit 0

  A 1 in the first bit (X'80') indicates a catastrophic error. Subsequent job steps probably will not function and the job will terminate.

- Bit 1

  A 1 in this bit (X'40') indicates a serious error. A serious error could affect subsequent job steps or result in incomplete or erroneous processing results.

- Bit 2

  A 1 in this bit (X'20') indicates a statement format or syntax error. The affected statement will not function, and this may or may not have an effect on subsequent job steps.

The UPSI byte can be useful in contingency error processing. For example, the byte can be examined and, if certain conditions prevail, can cause a branch to error handling routines. The SKIP job control statement is used to perform the test. The following examples show how you can use the SKIP job control statement.

Example 1:

```
      1          10      16                                              72
    ----------------------------------------------------------------------------
 1. | // JOB DSKPRP
 2. | // DVC 2Ø // LFD PRNTR
 3. | // DVC 51 // VOL DSPØ28  // LFD DISKIN
 4. | // EXEC DSKPRP
 5. | /$
 6. |    SERNR=DSPØ28,PARTL=V
 7. | /*
 8. | // SKIP ENDS,1
 9. |    . other
10. |    . job
11. |    . steps go here
12. | //ENDS NOP
13. | /&
14. | // FIN
```

In example 1, you check the UPSI byte to see whether a fatal error (X'80') has occurred. If the leftmost bit (bit 0) of the UPSI byte contains a binary 1 (line 8), then all the other job steps are bypassed and control is transferred to the NOP job control statement with the label ENDS (line 12). The NOP job control statement provides you with an address for the SKIP with no function being performed. The /& job control statement terminates your job while the // FIN terminates the card reader operation.

Example 2:

```
       1        10    16                                                    72
    -------------------------------------------------------------------------------
1.  | // JOB DSKPRP
2.  | // DVC 2Ø // LFD PRNTR
3.  | // DVC 51 // VOL DSPØ28 // LFD DISKIN
4.  | // EXEC DSKPRP
5.  | /$
6.  |    SERNR=DSPØ28,PARTL=V
7.  | /*
8.  | // SKIP WARN,Ø1
9.  | // SKIP FATAL,1
10. | // SKIP EXIT
11. | //WARN OPR 'WARNING-A NON-FATAL ERROR HAS OCCURRED'
12. | // SKIP EXIT
13. | //FATAL OPR 'FATAL ERROR-JOB TERMINATED-CORRECT AND RERUN'
14. | // SKIP ENDOFJOB
15. | //EXIT NOP
16. |   .  other job steps
17. |   .  go here
18. |   .
19. | //ENDOFJOB NOP
20. | /&
21. | // FIN
```

In example 2, you check for both the fatal (X'80') and warning errors (X'40') and the display of appropriate messages on the system console. If a warning error has occurred – bit 1 of the UPSI byte is a binary 1 (line 8) – then you skip to the label WARN on the OPR job control statement and print the warning message (line 11). After processing the OPR statement, the SKIP job control statement (line 12) is the next job control statement processed. Here, you skip down to the label EXIT on the NOP job control statement (line 15). As mentioned earlier, the NOP acts as an ending point for the SKIP control statement. The remaining job steps follow the NOP statement and are processed accordingly. Following the last job step, the NOP statement on line 19 is processed with no action being performed. Your job then terminates normally through the /& and // FIN job control statements.

If a fatal error occurs, bit 0 of the UPSI byte is a binary 1 (line 9) and you skip down to the label FATAL on the OPR statement (line 13) and print the specified message. The SKIP job control statement (line 14) skips down to the label ENDOFJOB on the NOP statement, thus bypassing your remaining job steps, and terminates your job.

If no errors occurred, neither SKIP (lines 8 and 9) will be taken, and the SKIP job control statement (line 10) skips over the OPR statements to the remaining job steps.

The UPSI byte setting and the error count appear on the printout or map listing for the particular job. The UPSI byte value can also be retrieved by issuing the GETCOM supervisor macroinstruction in your BAL program. For more information on the GETCOM macro, refer to the current version of the supervisor user guide, UP-8075. For more information on the SKIP job control statement, refer to the current version of the job control user guide, UP-8065.

1.8. STATEMENT CONVENTIONS

The conventions used to illustrate the control statements and system console message displays presented in this manual are:

- Positional parameters must be written in the order specified in the operand field and must be separated by commas. When a positional parameter is omitted, the comma must be retained to indicate the omission, except for the case of omitted trailing parameters.

  Examples:

  Assume that LOADM is a linkage editor control statement with three optional positional parameters: A, B, and C.

  ```
  INCLUDE A
  INCLUDE A,B
  INCLUDE A,B,C
  INCLUDE A,C
  ```

- A keyword parameter consists of a word or a code immediately followed by an equal sign, which is, in turn, followed by a specification. Keyword parameters can be written in any order in the operand field. Commas are required only to separate parameters.

  Examples:

  Assume that LINKOP is a linkage editor control statement with three optional keyword parameters: ALIB, RLIB, and OUT.

  ```
  LINKOP ALIB=OBJFIL,RLIB=$Y$OBJ,OUT=$Y$LOD
  LINKOP ALIB=OBJFIL,RLIB=$Y$OBJ
  LINKOP RLIB=$Y$OBJ,ALIB=OBJFIL
  LINKOP OUT=$Y$LOD
  ```

- A positional or keyword parameter may contain a sublist of parameters, called subparameters, separated by commas and enclosed in parentheses. The parentheses must be coded as part of the list. The subparameters within the parentheses may be positional, in which case the comma must be retained if a parameter is omitted, except for the case of trailing parameters, or they may be nonpositional. The description of the subparameters indicates whether they are positional or nonpositional.

Examples:

```
FIELDS=([ADDR][,A2TD][,FREQ])
REDO=(MERGE,label,reel,to)
```

- Capital letters, commas, equal signs, apostrophes, and parentheses must be coded and displayed exactly as shown. The exceptions are acronyms, which are part of generic terms representing information to be supplied by the programmer.

  Examples:

```
CMcc NUMBCHAR=n
X'aa' (NOV)
ALIB=
```

- Lowercase letters and words are generic terms representing information that must be supplied by the user. Such lowercase terms may contain hyphens and acronyms (for readability).

  Examples:

```
lfn
name
group-name
comments
s1,   sn
```

- Information contained within braces represents mandatory entries of which one must be chosen.

  Examples:

```
⎧filename⎫
⎨(N)     ⎬
⎩$Y$RUN  ⎭
```

- Information contained within brackets represents optional entries that (depending upon program requirements) are included or omitted. Braces within brackets signify that one of the specified entries must be chosen if that parameter is to be included.

  Examples:

```
[sequence-no]
[ALIB=filename]
⎡⎧lfn   ⎫⎤
⎣⎩$Y$RUN⎭⎦
```

- An optional parameter with a list of optional entries may have a default specification that is supplied by the operating system when the parameter is not specified by the user. Although the default may be specified by the user with no adverse effect, it is considered inefficient to do so. For easy reference, when a default specification occurs in the format delineation, it is printed on a shaded background. If, by parameter omission, the operating system performs some complex processing other than parameter insertion, it is explained under an "If omitted" statement in the parameter description.

  Examples:

$$\left[,\begin{Bmatrix}\text{input-lfn}\\ \text{SYSRUN}\end{Bmatrix}\right]$$

$$\left[,\begin{Bmatrix}\text{73-80}\\ \text{1}\end{Bmatrix}\right]$$

- An ellipsis (series of three periods) indicates the omission of a variable number of entries.

  Example:

```
param-1,...,param-n
```

- Commas are required when positional parameters are omitted, except after the last parameter specified.

  Example:

```
positional-parameter-1,positional-parameter-2,,positional-parameter-4
```

NOTE:

There are three standard character sets used with SPERRY UNIVAC printers: two are 48-character print sets, and the third is a 63-character print set. Thus, not all characters are printable on all machines, and print code conversions are necessary to represent nonprintable characters when a 48-character print set is being used. The programming examples presented in this manual were produced by using the standard 48-character business print set and, therefore, make use of some of these conversion print characters. For example, an equals sign (=) is represented by a percent symbol (%), a left parenthesis by a number symbol (#), and a right parenthesis by an at symbol (@).

PART 2. THE LIBRARIANS

2. SAT Librarian Functional Characteristics

2.1. GENERAL

2.1.1. Capabilities

The SAT librarian of the SPERRY UNIVAC Operating System/3 (OS/3) manages the system and user libraries containing the modules making up the program environment for a given system. Although the SAT librarian is primarily a disk utility, library files may exist on magnetic tape, disk, diskette, or punched cards and may be converted from one medium to another. The SAT librarian facilitates merging of all or parts of existing library files, extending or adding to an existing library file, compressing fragmented files and reclaiming unused file space, deleting unwanted or nullified modules within a given library, and supplying appropriate printouts. (A map and associated listing can be provided for each library function performed.) The output of a given SAT librarian job can be an updated tape, disk library or diskette library, a new tape, disk library or diskette library, punched cards, listings, or some combination of these. Figure 2-1 illustrates the environments under which the librarian can be expected to function. These operational modes are normally selected at run time via parameter specifications. The program name of the librarian is LIBS.

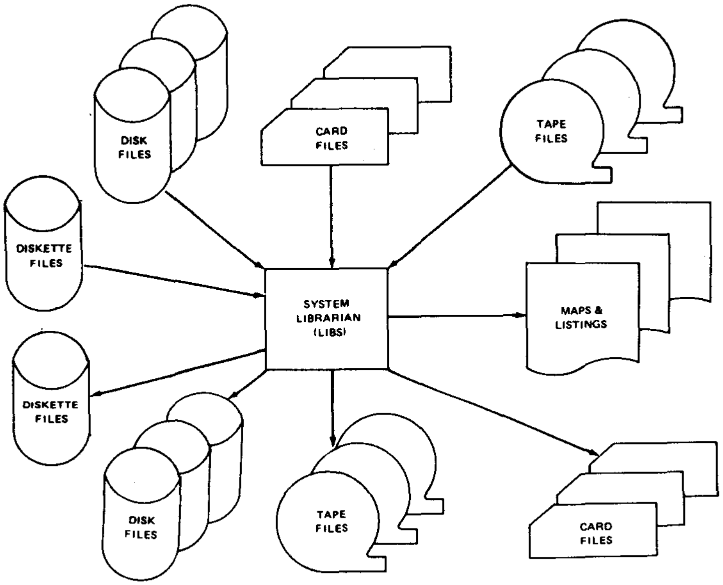


Figure 2-1. Librarian Input/Output File Environment

2.1.2. Additional Main Storage Requirements

Librarian performance can be significantly improved by allocating additional main storage space for the job in the // JOB control statement. This additional space is allocated in track-sized buffers. To determine the amount of main storage to allocate, compute the number of buffers needed using the following recommendations:

> Specify at least two buffers for disk access (four or more is optimal). Add one more buffer for each variable block tape used (one for each Tn file declared in the FIL librarian control statement).

Buffer Requirements	Decimal Bytes	Hex Bytes
Librarian base	28,672	7000
1	40,992	A020
2	51,264	C840
3	61,536	F060
4	71,808	11880
5	82,080	140A0
6	92,352	168C0

The second parameter in the // JOB statement must specify the amount of additional main storage in decimal or hex bytes. For example, each of the following // JOB statements allocates enough main storage for the librarian base plus one buffer:

```
// JOB SAMPLE,,A020
// JOB SAMPLE,,X'A020'
// JOB SAMPLE,,D'40992'
```

2.2. CONTROL FUNCTIONS

The following control functions are provided by the SAT librarian for user management of the program libraries in this system:

- BLK Convert standard load modules to block load modules
- BOG Write beginning-of-group record
- COM Compare elements
- COP Copy elements
- COR Correct elements
- DEL Delete elements

ELE	Add card file element (module)
EOD	Declare end-of-data
EOG	Write end-of-group record
ESC	Read control statements from user-created file
FIL	Declare file
LST	Print a file in alphabetic sequence
PAC	Pack (compress) files
PAGE	Cause the printing of a new page
// PARAM ERROR	Specify, in the event of an error, whether the librarian job should be canceled or just the librarian job step
// PARAM PRINT	Suppress the printing of the librarian map
// PARAM PRTOBJ	Print source module listings in hexadecimal format
// PARAM TAPEFILES	Allow multiple files to be written to the same tape volume
// PARAM UPDATE	Specify the data and time to be in effect during librarian execution
REC	Recycle source module current position pointer
REN	Rename elements, revise the comments field of header records, or mark object as reentrant or nonreentrant
REPRO	Produce or delete control statement records within object modules
RES	Reset file current position pointer
SEQ	Sequence or check sequence of elements
SKI	Skip source module records

See 2.8 for a detailed description of these statements.

2.3. MODES OF OPERATION

The SAT librarian operates in two modes. Functionally, these are:

1. an input file update and list/punch mode; and
2. an output file creation mode.

If no output file declarations are made, only input files may be updated and extended. The following functions initiate different operations, depending on the selected mode:

ELE
DEL
COP

The librarian maintains a set of file information tables (DTFs) for up to six logical files at one time. If a seventh file is accessed, the file information for the first is overlaid. Included in this information is the name, type, and address of the last module accessed on the file. Thus, while more than six files may be accessed by the user, six is the limit that may be accessed concurrently.

The librarian can, through options in the operation field, print or punch entire modules. The following functions allow such printing and punching:

COR
COP
DEL
ELE
REN
PAC
SEQ

With the ELE function, cards can be added to a library and be listed or punched at the same time.

Module headers are listed with each prime directive used unless the no-list header option (N) is specified by the user, in which case no listing of headers is supplied.

2.4. PROGRAM LIBRARY MANAGEMENT

2.4.1. Naming Conventions

Modules within library files (regardless of type) contain an 8-character EBCDIC identifier that is used as the name of the module. (Modules of the same name and type are not allowed in one file.) If the name assigned is less than eight characters, it is left-justified and space-filled. Naming of specific modules can be performed at:

- assemble or compile time for object modules;
- link-edit time for load modules;
- library services time for program source definition modules; and
- job run time for macro/jproc source modules.

The librarian also can be used to rename specific modules or module groups. It can:

- rename a program source or macro/jproc source definition module;
- rename an object module or a specific CSECT;

- rename common sections and ESD records in object modules;
- rename all phases of a load module (retaining phase numbers); and
- rename the alias phase name of a load phase.

2.4.2. Group Management

The librarian can process the elements in a file individually or by groups. Any number of modules, exclusive of type, can be grouped to form a single processing entity. Each module group is given a name and is bracketed by group demarcator records. Any number of module groups having the same name may reside in a single file. After a group is created, the librarian can process all modules in the group at one time. Gang operations also allow processing of all groups with a certain name or name prefix. If a gang operation is not specified, only the first group with the specified name is processed.

2.4.3. Gang Operations

Certain functions of the librarian are operable in gang mode in which several modules may be copied, deleted, punched, compared, or displayed at one time. These options are initiated via the appropriate command statements and the omission of the name parameter (or name and type parameters), in the operand field.

There are three types of gang operations available to the librarian user. The first depends on module type, the second on module name, and the third on the current position pointer of the file.

2.4.3.1. Module Type Gang Mode

Library files may contain mixed module types; that is, object code, load code, program source code, and macro/jproc source code can be intermingled within a given library file. When gang operations are to be processed on modules of a specified type, the module name is omitted and the type positional parameter is set as follows:

S	For program source modules
M	For macro/proc/jproc source modules
O	For object modules
L	For load module

By setting the type as shown and omitting the name, the user instructs the librarian to perform the designated operation on all modules of the type specified from the current position of the library file up through end-of-file.

The following functions permit specification of a module type gang operation:

COP COM
DEL

When the gang mode is initialized in one of the foregoing operations, the referenced file is scanned from its current position for the code set designated. When a module of the type indicated is detected, the requested operation commences.

2.4.3.2. Module Name Gang Mode

When gang operations are to be performed on modules with a specified name, the module type is omitted in the librarian control statement. These statements process all modules from the current position to the end of the file whose names match the module name field in the librarian control statement.

If gang operations are to be processed on modules with like name prefixes, the C option designator is appended to the librarian function code. This option instructs the librarian to compare the characters in the module name field of the control statement with the names of the modules in the designated file from its current position to the end of the file. Whenever a module is found whose name begins with the name prefix contained in the control statement and is of the type specified in the control statement, the requested operation is performed on the module. A module type need not be specified when operating in the module name gang mode, in which case all modules having the name prefix specified are operated on by the librarian.

Module name gang operations may be specified only for the COP and DEL functions.

2.4.3.3. Total Gang Mode

If the function to be performed does not concern itself with a specific module or code set, the type and name positional parameters can be omitted from the librarian function code. This instructs the librarian to perform the specific function on all the modules contained in the designated file, from its current position to the end of the file. In this manner, an entire library (or remainder of one pre-positional) may be manipulated via the facility desired.

Total gang mode operations may be specified only for the COP and DEL functions.

2.4.3.4. Current File Position

All gang operations process the library file from its current position as defined by the respective file table (DTF) contained within the program. The current position of a file can be affected:

- by the reset (RES) function; and
- by any librarian function except the EOD function.

The RES function can aim the current position pointer to the first logical record on the file specified, or the first record in a named module in the specified file.

All librarian functions except EOD affect the current position. When the function is completed, the current position pointer for the processed file is the address of the record immediately following the last record processed unless the pointer was at the end of the file. In this case, the pointer is positioned at the beginning of the file.

A COP function may be initiated with no output file specified. This effectively aims the current position pointer to the record after the last record of the module or module group specified in the COP function without actually copying the module or group.

If the librarian needs to find a module in a library file, the search begins at the current position of the directory and continues until the module is found, or the end of the file is reached. If the end of the file is reached, the search begins anew at the beginning of the file directory and continues until the module directory record is found or the original current position of the file directory is reached again. The current position being arrived at again signifies no find for that module on the file being searched.

2.4.4. Program Source Module Management

The librarian provides facilities for the maintenance of program source code modules. Program source code modules can be listed, filed, punched, corrected, and renamed, as well as manipulated, with the standard librarian-provided functions. Specific program source records can be added and deleted from a program source element. Updated program source modules may be mapped as corrections are applied. Program source records are printed individually in EBCDIC format, exactly as they were coded. If the // PARAM PRTOBJ statement is used, source modules are printed in hexadecimal format.

In addition to the librarian source module management, you can access assembler source modules via the // USE LIB job control statement. Here, you can update or create assembler source modules from another assembler program. For more information, see the job control user guide, UP-8065 (current version).

2.4.5. Macro/Jproc Source Module Management

Macro and jproc source modules are handled in much the same manner as program source modules by the librarian; that is, these modules can be copied, corrected, compared, renamed, added, and deleted. Since these modules can have more than one name associated with them, the librarian performs some additional processing. Thus, when one of these modules is added to a library, a separate directory entry (type A2) must be created for each name associated with the module. All directory entries reference the module being added so that the module may be located by any of its given names. Because macro and jproc source modules are functionally identical and have the same type code (type A3), a macro and a jproc source module of the same name may not coexist in the same library file. Macro and jproc source modules with the same directory entry name (type A2) may coexist in the same library file. A macro/jproc source module with a directory entry name (type A2) may also coexist with a macro/jproc directory entry name (type A3) in the same library file.

2.4.6. Object Module Management

Language processor output modules can be maintained by the librarian, in that object code can be patched, listed, punched, filed, and renamed. Specific CSECTs or ESDs also may be renamed. Patch corrections are inserted at the end of the object module. Listings of object modules are hexadecimal printouts of object records. All standard librarian functions regarding module manipulation apply to object elements. Whenever nonsource elements are serviced, they are checked for proper content and record sequence. Discrepancies trigger diagnostic processing.

All object modules produced by the various language processors are assumed to be nonreentrant modules. If in fact they are reentrant, they may be flagged as such by the librarian to enable them to produce reentrant load modules when they are link-edited.

2.4.7. Load Module Management

Load modules generated by the linkage editor also can be managed by the librarian. The facilities provided for load module management are much the same as those provided for object module management, except that specific load module phases may be patched. Applied patches are inserted at the end of the designated phase. Load modules also may be listed, punched, filed, and renamed. Load module listings are hexadecimal printouts of load module records. Load elements may be serviced via all standard librarian functions. Phases within a load module also can have an alias phase name given to it at link-edit time, in addition to the phase name assigned to the load segment. This alias phase name also can be renamed by the librarian.

Most load modules produced by the linkage editor can be converted to blocked load modules by the librarian. Blocked load modules can usually be loaded for execution faster than their standard counterparts. The exceptions to this are discussed in the description of the blocking load modules (BLK) control statement.

2.4.8. File Merging

The librarian can function in a library file merge mode; that is, one or more library files, module groups, or individual modules may be merged into a new output library (or libraries). Multiple file merging is permitted and the number of files involved is a function of the user requirements. The librarian can merge up to six files concurrently (including output files).

Reference to a seventh file (or more) causes the first file (and any succeeding files) to be reopened whenever a new, interspersed file reference is detected. Thus, merging of multiple files beyond a sixth may be more easily accomplished by first merging five files together to form a sixth file, and then merging the sixth file with the remaining files by repeating the desired merge operation.

2.4.9. File Extension

A current library file often can be updated (or effectively extended) without creating a new output file. This may involve replacement of a given element within the file with a new copy of the same element. Replaced elements are flagged as nullified and may be removed via a subsequent file compression operation. Directory entries for replaced elements in extended files are altered accordingly.

2.4.10. File Compression

The librarian can compress fragmented files (interspersed voided elements) and reobtain dormant file space. The compression is automatic if merging or copying involving the file in question occurs. If not, an existing file may be compressed by using the PAC librarian function. File compression can be specified anywhere within a given librarian job stream. Any associated directories also are compressed in the update job.

2.4.11. File Deletion

Individual modules, or entire code sets, may be deleted from library files by using the facilities of the librarian. Deletions can occur while updating existing files or while creating new ones. Deletions applied to existing files can cause file fragmentation (as in the case of module replacement), which can, in turn, be remedied by later file compression.

2.5. RUN LIBRARY MANAGEMENT

The job run library is processed in the same way as all other libraries by the librarian. The job run library can be specified by designating $Y$RUN as the file name on the control statement FIL.

The job run library also is used as the default file if, on certain librarian commands, a file is not specified. These commands are ELE, COP, REN, COR, SEQ, DEL, REPRO, and PAC. If, for the DEL, ELE, PAC, REPRO, or REN functions, a logical file is not specified, the job run library is used as the logical file to be processed. Because output files do not need to be specified in the COP, COR, and SEQ functions, the default use of the job run library applies only to the input file.

2.6. MAPPING FACILITIES

Each time the librarian is executed, a map of the functions it performs is output on the system printer for the user. The map normally includes:

- a listing of all the librarian control statements processed;
- a printout of all the header records processed; and
- any appropriate diagnostic messages.

Additionally, the map can include:

- source module listings;
- object and load module listings; and
- module correction results (insertions versus deletions).

The map normally reflects the state or content of the output library files if one or more were produced; otherwise, it reflects the state or content of the input file serviced by the respective librarian function. In comparison functions, discrepancies are listed on a record-by-record or block-by-block basis.

2.6.1. Standard Map Layout

The librarian map lists all the control statements input to the librarian in the order they were processed, followed by any module data to be listed relative to each statement (Figure 2–2). Diagnostic messages are listed as close as possible to the control statements that initiated their generation, and are prefixed with a unique librarian message number. These messages and their meanings are described in the system messages manual, UP-8076 (current version). Unless suppressed by the user through a control statement option, module and module group header records are listed in their respective formats, as described in Appendix B. The location of each of these records within its respective file also is printed on the map as a function of its block location and record displacement within that block. (The OS/3 program library format also is described in 2.7.)

2.6.2. Source Module Listings

Whenever a source module is listed, each source record is listed in standard EBCDIC format exactly as it appears within the source module. A one-to-one relationship exists between the number of source statements in a source module and the number of lines printed for the source module. When source modules are being updated, lines deleted, lines preceding insertions, and insertions are listed in the same format. Figure 2–2 illustrates an example of a source module printout.

If the // PARAM PRTOBJ statement is used, source modules will be listed in hexadecimal format.

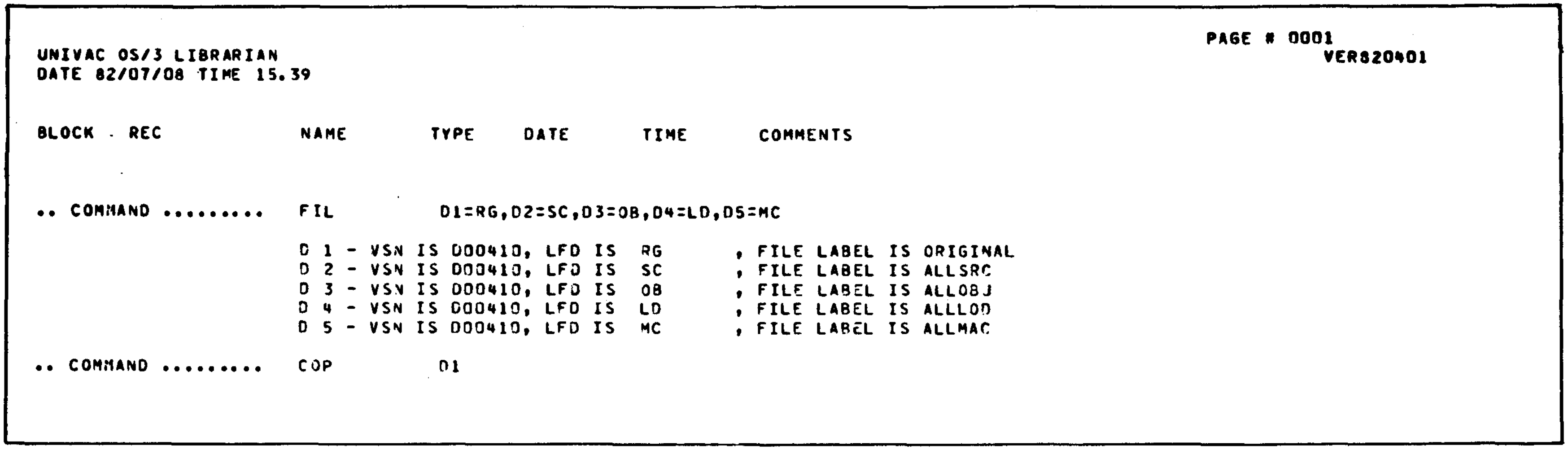
2.6.3. Object and Load Module Listings

Object and load modules are listed in hexadecimal form. Each byte appears on the map as two printed hexadecimal digits. Because object and load module records are not fixed in length, the listing is on a record-by-record basis. If patch records exist within the module, they are flagged appropriately. Figure 2–3 illustrates an example of an object module listing.

2.6.4. Diagnostic Message Listings

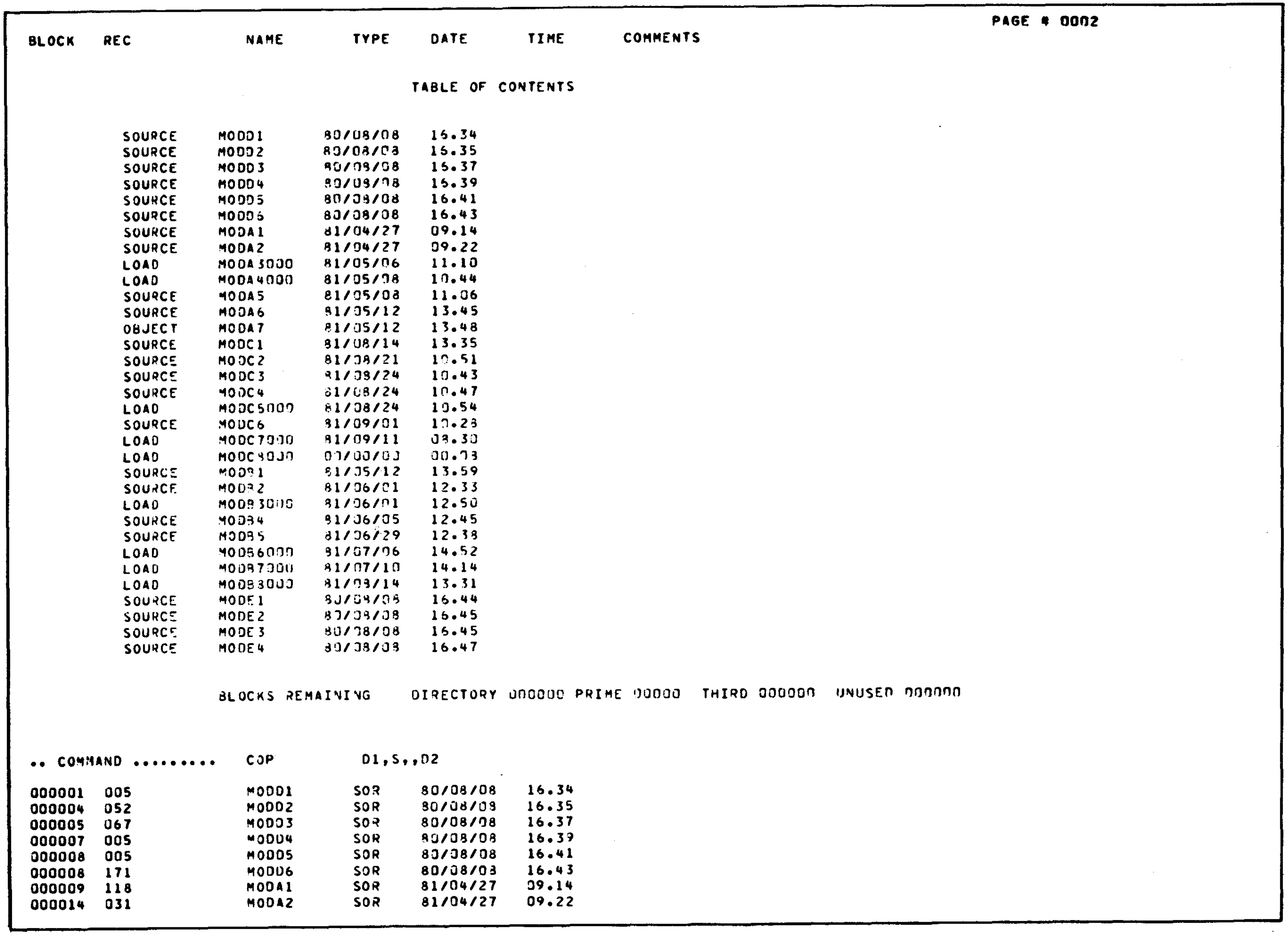
Diagnostic messages are listed on the librarian map whenever a processing error is detected by the librarian. The printed message identifies the type of error detected and the message number identifier. All the messages capable of being produced by the librarian are listed in the system messages manual, UP-8076 (current version), as well as the meaning of each message and the corrective action required to remedy the cause of the processing error. The librarian job is never aborted unless the processing error detected is sufficiently critical to preclude continuing.

Figure 2–2 shows an example of a typical diagnostic message at the bottom of page 0003 of the librarian map. It reads B060***** NOTHING FOUND.



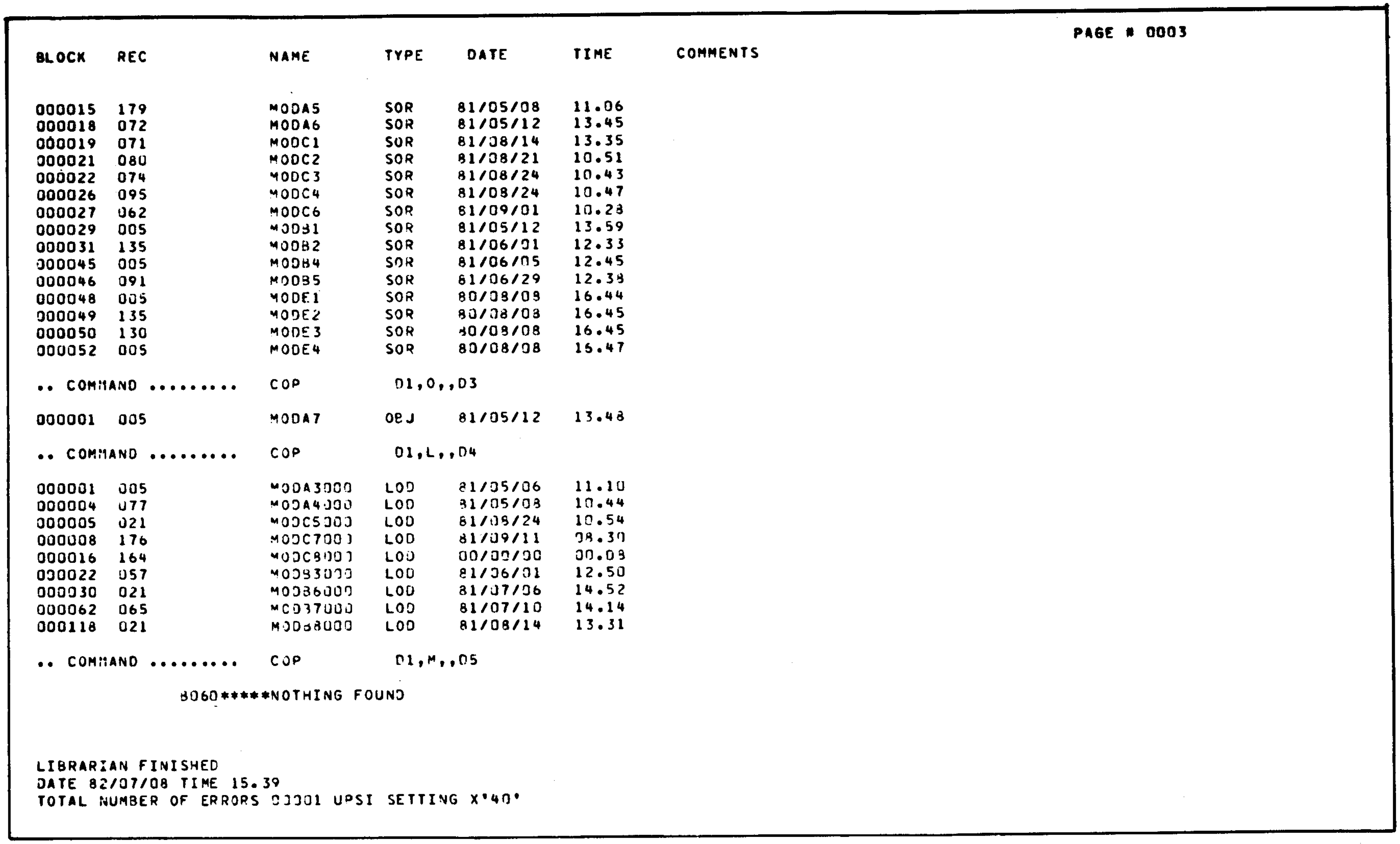
```
                                                                                PAGE # 0001
UNIVAC OS/3 LIBRARIAN                                                                  VER820401
DATE 82/07/08 TIME 15.39

BLOCK . REC             NAME       TYPE     DATE      TIME      COMMENTS


.. COMMAND .........    FIL          D1=RG,D2=SC,D3=OB,D4=LD,D5=MC

                        D 1 - VSN IS 000410, LFD IS  RG      , FILE LABEL IS ORIGINAL
                        D 2 - VSN IS 000410, LFD IS  SC      , FILE LABEL IS ALLSRC
                        D 3 - VSN IS 000410, LFD IS  OB      , FILE LABEL IS ALLOBJ
                        D 4 - VSN IS 000410, LFD IS  LD      , FILE LABEL IS ALLLOD
                        D 5 - VSN IS 000410, LFD IS  MC      , FILE LABEL IS ALLMAC

.. COMMAND .........    COP          D1
```

Figure 2—2. Typical Librarian Map (Part 1 of 3)



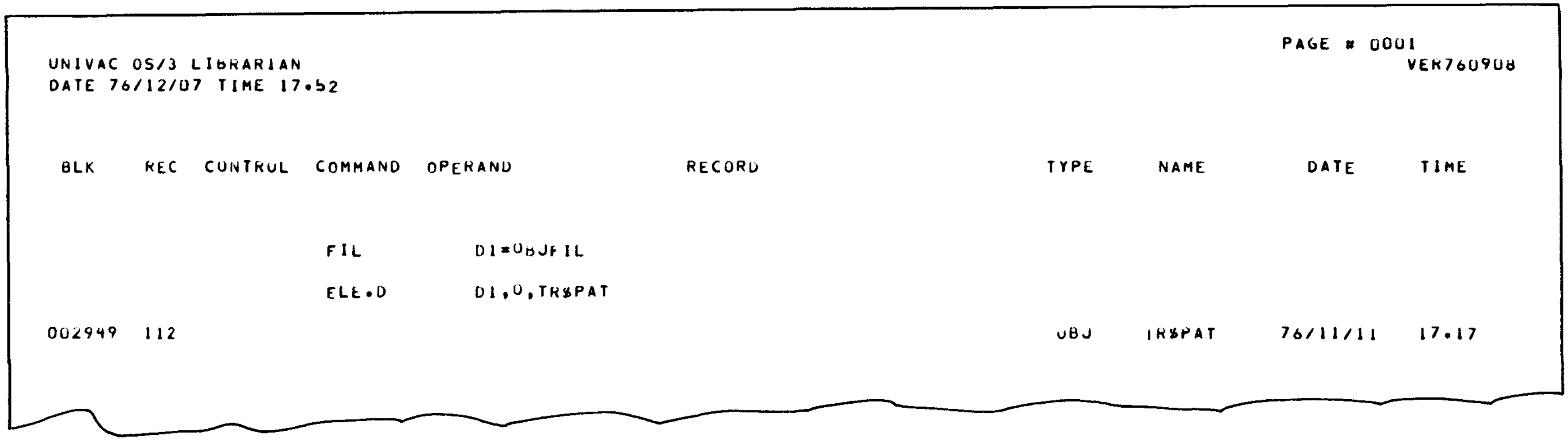
```
                                                                                      PAGE # 0002
BLOCK   REC           NAME        TYPE     DATE        TIME       COMMENTS

                                         TABLE OF CONTENTS


          SOURCE     MODD1      80/08/08    16.34
          SOURCE     MODD2      80/08/08    16.35
          SOURCE     MODD3      80/08/08    16.37
          SOURCE     MODD4      80/08/08    16.39
          SOURCE     MODD5      80/08/08    16.41
          SOURCE     MODD6      80/08/08    16.43
          SOURCE     MODA1      81/04/27    09.14
          SOURCE     MODA2      81/04/27    09.22
          LOAD       MODA3000   81/05/06    11.10
          LOAD       MODA4000   81/05/08    10.44
          SOURCE     MODA5      81/05/08    11.06
          SOURCE     MODA6      81/05/12    13.45
          OBJECT     MODA7      81/05/12    13.48
          SOURCE     MODC1      81/08/14    13.35
          SOURCE     MODC2      81/08/21    10.51
          SOURCE     MODC3      81/08/24    10.43
          SOURCE     MODC4      81/08/24    10.47
          LOAD       MODC5000   81/08/24    10.54
          SOURCE     MODC6      81/09/01    10.28
          LOAD       MODC7000   81/09/11    08.30
          LOAD       MODC8000   00/00/00    00.03
          SOURCE     MODB1      81/05/12    13.59
          SOURCE     MODB2      81/06/01    12.33
          LOAD       MODB3000   81/06/01    12.50
          SOURCE     MODB4      81/06/05    12.45
          SOURCE     MODB5      81/06/29    12.38
          LOAD       MODB6000   81/07/06    14.52
          LOAD       MODB7000   81/07/10    14.14
          LOAD       MODB8000   81/08/14    13.31
          SOURCE     MODE1      80/08/08    16.44
          SOURCE     MODE2      80/08/08    16.45
          SOURCE     MODE3      80/08/08    16.45
          SOURCE     MODE4      80/08/08    16.47

                     BLOCKS REMAINING    DIRECTORY 000000 PRIME 00000  THIRD 000000  UNUSED 000000


.. COMMAND .........  COP          D1,S,,D2

000001  005           MODD1       SOR      80/08/08     16.34
000004  052           MODD2       SOR      80/08/08     16.35
000005  067           MODD3       SOR      80/08/08     16.37
000007  005           MODD4       SOR      80/08/08     16.39
000008  005           MODD5       SOR      80/08/08     16.41
000008  171           MODD6       SOR      80/08/08     16.43
000009  118           MODA1       SOR      81/04/27     09.14
000014  031           MODA2       SOR      81/04/27     09.22
```

Figure 2-2. Typcial Librarian Map (Part 2 of 3)



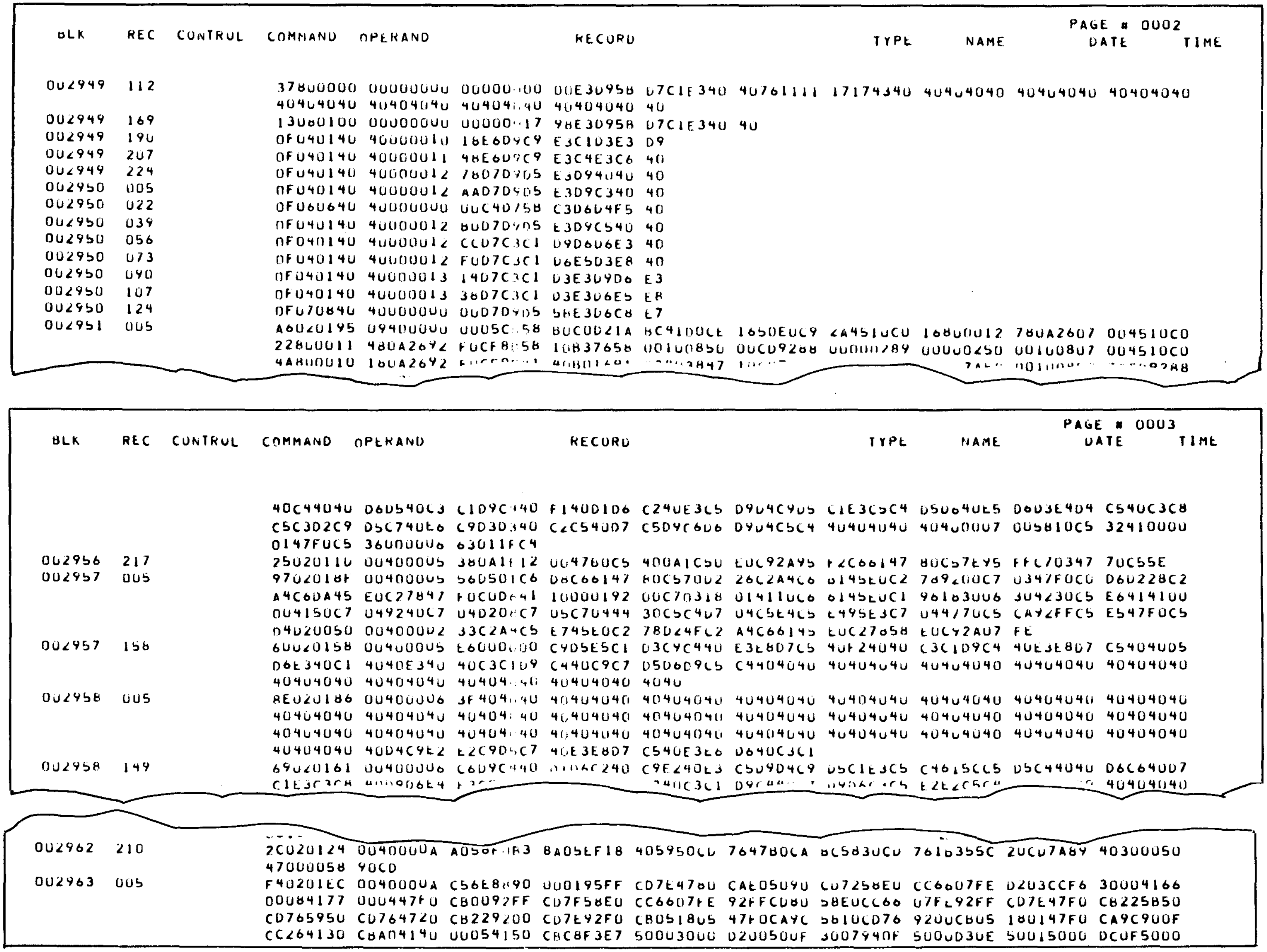
```
                                                                                                PAGE # 0003
BLOCK   REC                 NAME       TYPE     DATE        TIME       COMMENTS


000015  179                 MODA5      SOR     81/05/08    11.06
000018  072                 MODA6      SOR     81/05/12    13.45
000019  071                 MODC1      SOR     81/08/14    13.35
000021  080                 MODC2      SOR     81/08/21    10.51
000022  074                 MODC3      SOR     81/08/24    10.43
000026  095                 MODC4      SOR     81/08/24    10.47
000027  062                 MODC6      SOR     81/09/01    10.28
000029  005                 MODB1      SOR     81/05/12    13.59
000031  135                 MODB2      SOR     81/06/01    12.33
000045  005                 MODB4      SOR     81/06/05    12.45
000046  091                 MODB5      SOR     81/06/29    12.38
000048  005                 MODE1      SOR     80/08/08    16.44
000049  135                 MODE2      SOR     80/08/08    16.45
000050  130                 MODE3      SOR     80/08/08    16.45
000052  005                 MODE4      SOR     80/08/08    16.47

.. COMMAND .........        COP          01,0,,D3

000001  005                 MODA7      OBJ     81/05/12    13.48

.. COMMAND .........        COP          01,L,,D4

000001  005                 MODA3000   LOD     81/05/06    11.10
000004  077                 MODA4000   LOD     81/05/08    10.44
000005  021                 MODC5000   LOD     81/08/24    10.54
000008  176                 MODC7000   LOD     81/09/11    08.30
000016  164                 MODC8000   LOD     00/00/00    00.08
000022  057                 MODB3000   LOD     81/06/01    12.50
000030  021                 MODB6000   LOD     81/07/06    14.52
000062  065                 MOD37000   LOD     81/07/10    14.14
000118  021                 MOD88000   LOD     81/08/14    13.31

.. COMMAND .........        COP          01,M,,05

                8060*****NOTHING FOUND


LIBRARIAN FINISHED
DATE 82/07/08 TIME 15.39
TOTAL NUMBER OF ERRORS 00001 UPSI SETTING X'40'
```

Figure 2-2. Typical Librarian Map (Part 3 of 3)



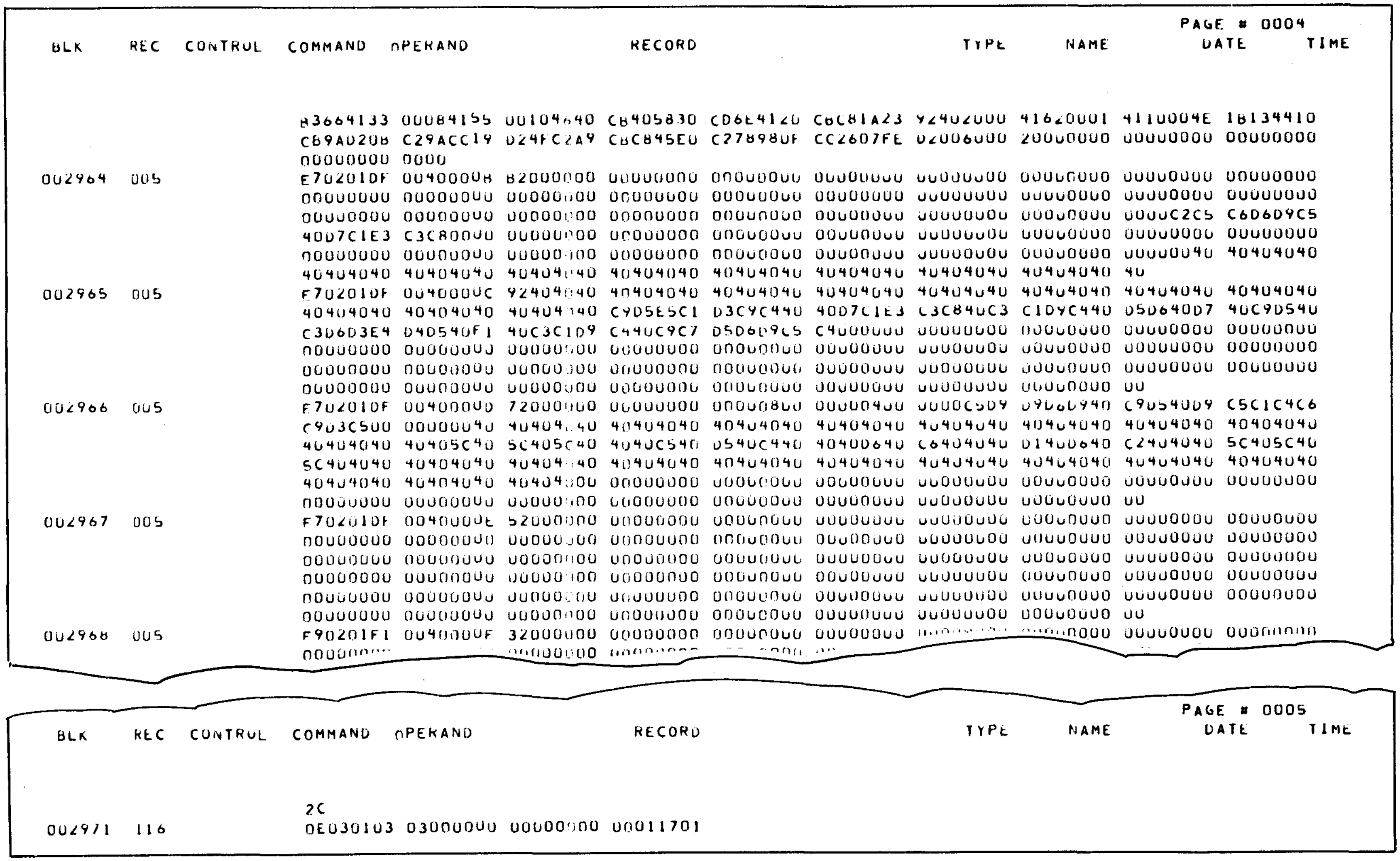
```
                                                                                             PAGE # 0001
UNIVAC OS/3 LIBRARIAN                                                                               VER760908
DATE 76/12/07 TIME 17.52


 BLK     REC  CONTROL  COMMAND  OPERAND              RECORD                  TYPE     NAME       DATE      TIME


                        FIL          D1=OBJFIL

                        ELE.D        D1,0,TR$PAT

002949   112                                                                  OBJ     TR$PAT     76/11/11   17.17
```

Figure 2—3. Object Module Lising (Part 1 of 3)



```
                                                                                                      PAGE # 0002
 BLK     REC  CONTROL  COMMAND  OPERAND                  RECORD                        TYPE      NAME          DATE      TIME


002949   112           37800000 00000000 00000000 00E3D95B D7C1E340 40761111 17174340 40404040 40404040 40404040
                       40404040 40404040 40404040 40404040 40
002949   169           13080100 00000000 00000017 98E3D95B D7C1E340 40
002949   190           0F040140 40000010 18E6D9C9 E3C1D3E3 D9
002949   207           0F040140 40000011 48E6D9C9 E3C4E3C6 40
002949   224           0F040140 40000012 78D7D9D5 E3D94040 40
002950   005           0F040140 40000012 AAD7D9D5 E3D9C340 40
002950   022           0F060640 40000000 00C4D75B C3D6D4F5 40
002950   039           0F040140 40000012 B0D7D9D5 E3D9C540 40
002950   056           0F040140 40000012 CCD7C3C1 D9D6D6E3 40
002950   073           0F040140 40000012 F0D7C3C1 D6E5D3E8 40
002950   090           0F040140 40000013 14D7C3C1 D3E3D9D6 E3
002950   107           0F040140 40000013 38D7C3C1 D3E3D6E5 E8
002950   124           0F070840 40000000 00D7D9D5 5BE3D6C8 E7
002951   005           A6020195 09400000 0005C058 B0C0D21A BC41D0CE 1650E0C9 2A4510C0 16800012 780A2607 004510C0
                       22800011 480A2692 F0CF8058 10837658 00100850 00CD9288 00000289 00000250 00100807 004510C0
                       4A800010 180A2692 F        40801         3847 10                        7A    0010               8288

                                                                                                      PAGE # 0003
 BLK     REC  CONTROL  COMMAND  OPERAND                  RECORD                        TYPE      NAME          DATE      TIME



                       40C44040 D6D540C3 C1D9C440 F140D1D6 C240E3C5 D9D4C9D5 C1E3C5C4 D5D640E5 D6D3E4D4 C540C3C8
                       C5C3D2C9 D5C740E6 C9D3D340 C2C540D7 C5D9C6D6 D9D4C5C4 40404040 40400007 005810C5 32410000
                       0147F0C5 36000006 63011FC4
002956   217           25020110 00400005 380A1F12 004760C5 400A1C50 E0C92A95 F2C66147 80C57E95 FFC70347 70C55E
002957   005           9702018F 00400005 56D501C6 D8C66147 80C57002 26C2A4C6 6145E0C2 789200C7 0347F0C0 D6D228C2
                       A4C6DA45 E0C27847 F0CD0641 10000192 00C70318 014110C6 6145E0C1 96163006 304230C5 E6414100
                       004150C7 049240C7 04D208C7 05C70444 30C5C4D7 04C5E4C5 E495E3C7 044770C5 CA92FFC5 E547F0C5
                       04D20050 00400002 33C2A4C5 E745E0C2 78D24FC2 A4C66145 E0C27658 E0C92A07 FE
002957   156           60020158 00400005 E6000000 C9D5E5C1 D3C9C440 E3E8D7C5 40F24040 C3C1D9C4 40E3E8D7 C54040D5
                       D6E340C1 4040E340 40C3C1D9 C440C9C7 D5D6D9C5 C4404040 40404040 40404040 40404040 40404040
                       40404040 40404040 40404040 40404040 4040
002958   005           8E020186 00400006 3F404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
                       40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
                       40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
                       40404040 40D4C9E2 E2C9D5C7 40E3E8D7 C540E3E6 D640C3C1
002958   149           69020161 00400006 C6D9C440 [illegible]6C240 C9E240E3 C5D9D4C9 D5C1E3C5 C4615CC5 D5C44040 D6C640D7
                       C1E3C3C8 40  9D6E4 F               40C3C1 D9C4         C5 E2E2C5C4                   40404040


002962   210           2C020124 0040000A A05[illegible]B3 8A05EF18 405950CD 764780CA BC5830CD 7616355C 20CD7A89 40300050
                       47000058 90CD
002963   005           F40201EC 0040000A C56E8890 000195FF CD7E4780 CAE05090 CD7258E0 CC6607FE D203CCF6 30004166
                       00084177 000447F0 CB0092FF CD7F58E0 CC6607FE 92FFCD80 58E0CC66 07FE92FF CD7E47F0 CB225B50
                       CD765950 CD764720 CB229200 CD7E92F0 CB051805 47F0CA9C 5610CD76 9200CB05 180147F0 CA9C900F
                       CC264130 CBA04140 00054150 CBC8F3E7 50003000 D200500F 3007940F 5000D30E 50015000 DC0F5000
```

Figure 2—3. Object Module Listing (Part 2 of 3)



```
                                                                                                          PAGE # 0004
  BLK     REC   CONTROL   COMMAND   OPERAND                  RECORD                          TYPE       NAME         DATE        TIME


                          B3664133 00084155 00104640 CB405830 CD6E4120 CBC81A23 92402000 41620001 4110004E 18134410
                          CB9AD20B C29ACC19 D24FC2A9 CBC845E0 C27898DF CC2607FE D2006000 20000000 00000000 00000000
                          00000000 0000
002964    005             E70201DF 00400008 B2000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
                          00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
                          00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000C2C5 C6D6D9C5
                          40D7C1E3 C3C80000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
                          00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000040 40404040
                          40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40
002965    005             E70201DF 0040000C 92404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
                          40404040 40404040 40404040 C9D5E5C1 D3C9C440 40D7C1E3 C3C840C3 C1D9C440 D5D640D7 40C9D540
                          C3D6D3E4 D4D540F1 40C3C1D9 C440C9C7 D5D6D9C5 C4000000 00000000 00000000 00000000 00000000
                          00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
                          00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
                          00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00
002966    005             E70201DF 00400000 72000000 00000000 00000800 00000400 0000C5D9 D9D6D940 C9D540D9 C5C1C4C6
                          C9D3C500 00000040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
                          40404040 40405C40 5C405C40 4040C540 D540C440 40400640 C6404040 D1400640 C2404040 5C405C40
                          5C404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
                          40404040 40404040 40404000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
                          00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00
002967    005             F70201DF 0040000E 52000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
                          00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
                          00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
                          00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
                          00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
                          00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00
002968    005             F90201F1 0040000F 32000000 00000000 00000000 00000000          0000 00000000 00000000
                          0000000           00000000 0000000    000
```

```
                                                                                                          PAGE # 0005
  BLK     REC   CONTROL   COMMAND   OPERAND                  RECORD                          TYPE       NAME         DATE        TIME


                          2C
002971    116             0E030103 03000000 00000000 00011701
```

Figure 2-3. Object Module Listing (Part 3 of 3)

2.7. PROGRAM LIBRARY DETAILS FOR SAT FILES

The system program library files, composed of program source, macro/jproc source, object, and load modules, are created and used by the various components of the SPERRY UNIVAC Operating System/3 (OS/3) during the normal course of system operation. It is these library files that the librarian services and maintains based on particular system needs and constraints determined by the user.

For you to realize the full extent of the capabilities of the librarian, you must be aware of the organization and content of the program libraries in the system. You also may elect to establish a program library of your own. If so, the librarian also can be used to maintain the object, program source, macro/jproc source, and load code sets contained in this library, under the same guidelines it uses when servicing the system program library files.

2.7.1. Library File Layout

The system library is composed of five permanent disk files and one temporary disk file for each job being processed in the system. All the files consist of at least a label, a single element, and an end-of-file marker; they are structured to support fixed-length block, variable-length record data and contain a directory partition. The directories are in fixed-length block, fixed-length record format.

Each of the five permanent files are 3-partition SAT files. One partition is used to maintain a directory for the file, and the other two are used to store the program modules contained in the file. When these files are initialized by the librarian, the space allocated for each file is distributed as follows:

- Two percent is allocated for the directory partition.
- Forty-eight percent is allocated for the prime data partition.
- No space is allocated for the second data partition.
- Fifty percent of the space allocated to each file is initially unassigned.

This initial allocation technique allows the librarian to assign file space to the various partitions in a file on an as-needed basis, and thus prevents space from being allocated for a partition that may never be used. (At present, only block load modules require the use of a third partition.) Thereafter, when a partition becomes full and requires more space, the librarian extends the partition by using some of the free space it has in reserve. Only the partition that was full is extended, and the amount of the extension is based on the file extension increment specified on the EXT job control statement used to create the file. When all the free space is allocated, the dynamic file expansion capability of the supervisor is called on to provide additional free space for the file in the same increments previously used to effect the file extensions performed by the librarian.

The job temporary library files are special files established by job control at the time jobs are input to the system for processing. These files are dynamic in nature, in that their size and structure are variable and they exist only until the job is terminated. Any programs or data that may be in these files are unrecoverable once their associated jobs have been terminated.

A program should not be executed from a library that is being restored, updated, or packed. In addition, it should be remembered that your files, excluding system files, may be sharable (depending on the FILELOCK parameter you specified during supervisor generation). See the system installation user guide/programmer reference, UP-8074 (current verison). Because OS/3 allows multiple writers to concurrently access sharable files, these files could be destroyed in a multiprogramming environment. It is recommended, therefore, that critical user files be prefixed by $LOKnn to prevent them from being accessed concurrently by multiple writer programs.

Providing information needed to create new files or extending existing files on disks is the function of the EXT job control statement. See job control user guide, UP-8065 (current version) for details on this and other job control statements.

2.7.1.1. Library Blocks

Library blocks are fixed-length, 256-byte blocks (Figure 2-4). Each block is composed of a 5-byte block prefix and up to 251 bytes of variable-record data. The block prefix includes a 3-byte logical block number, a 1-byte value indicating a block length (not including the block prefix), and a 1-byte check sum reflecting an exclusive OR for relevant data. Records within the block are variable in length up to a maximum size of 251 bytes for any given record including the record prefix.

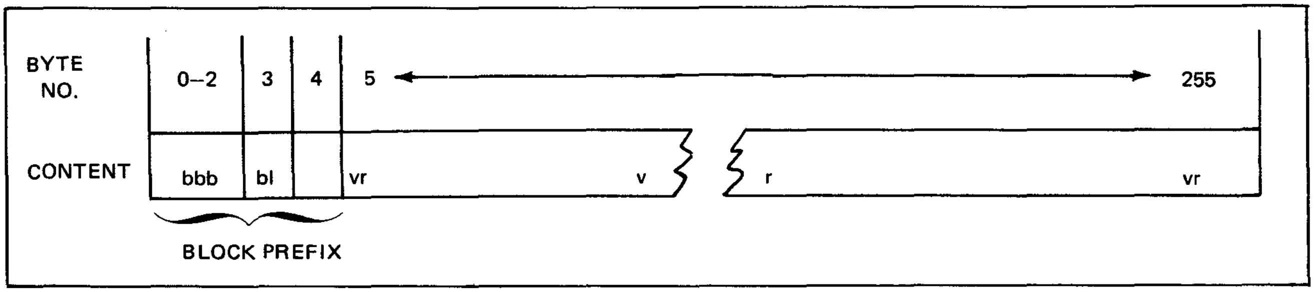


Figure 2-4. Library Block Format (Part 1 of 2)

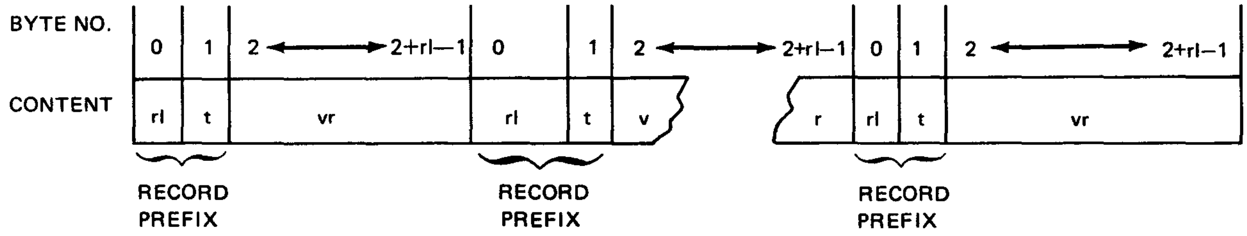
BLOCK FIELD DESCRIPTIONS

Byte Position	Field	Contents
0—2	Block number (bbb)	Starting with 1 for the initial block, this is the logical block sequence number.
3	Block length (bl)	This is a binary value less than or equal to 251, indicating the number of bytes of relevant record data within the body of this block, not including the block prefix.
4	Unused	
5 — 5+bl-1	Variable records (vr)	Variable-length records comprising the body of data contained in this block

Figure 2-4. Library Block Format (Part 2 of 2)

2.7.1.2. Library Records

Library records are variable in length. Each record is composed of a 2-byte record prefix and up to 249 bytes of record data (Figure 2-5). The record prefix includes a length byte and a type byte. The type byte indicates the specific type of record that follows the record prefix. The length byte indicates the size of the respective record (not including the record prefix) up to a maximum of 249 bytes.



RECORD FIELD DESCRIPTIONS

Byte Position	Field	Contents
0	Record length (rl)	This is a binary value, less than or equal to 249, indicating the length of the respective record (not including the record prefix).
1	Type (t)	This is a type byte indicating the specific type of record. (Refer to Table 2—1.)
2 — 2+rl-1	Variable-length record data (vr)	Body of the particular record (up to 249 bytes each)

Figure 2-5. Library Record Format

2.7.1.3. Record Type Byte

Associated with each record within a given library file is the type byte occurring in the respective record prefix. This byte is used to identify the record as to its code set and record particulars. Table 2-1 lists the record type byte values possible in an OS/3 system library file and their meanings. Note that the type byte field also exists in disk library directory items.

Table 2-1. Record Type Byte Descriptions (Part 1 of 2)

Type Byte Value (hexadecimal)	Description
00	Nullified item records
02	TEXT/RLD records in object modules
03	Transfer records in object modules
04	Standard ENTRY records
06	Standard EXTRN records
07	V-CON records
08	Named CSECT records
09	Unnamed CSECT records
0A	Named common records
0B	Unnamed common records
0C	Object code ISD records
12	TEXT/RLD records in load modules
13	Transfer records in load modules
16	Load code ISD records
1C	Load code ISD records
24	Program source or macro/jproc source module records
25	Compressed source code item
32	Blocked text or RLD records
40	Control statement records
80	Object module header records
90	Load module header/phase header records
A0	Beginning of group demarcator records
A1	EOF sentinel records

Table 2-1. Record Type Byte Descriptions (Part 2 of 2)

Type Byte Value (hexadecimal)	Description
A2	Marco/jproc name header records (in directory only)
A3	Marco/jproc module header records
A4	Program source module header records
A8	End of group demarcator records
B0	Blocked load module header/phase header records
C4	Shared code ENTRY (SENTRY) records
C6	Shared code EXTRN (SEXTRN) records
C8	Resource records

2.7.2. Disk Library Directories

Library files existing on disk are supplemented with a disk file directory composed of 13-byte records, each of which points to a specific demarcation record in the file. The directory precludes the need for scanning the library file to obtain a needed record. Instead, directory scanning suffices until the program is located. The pointers existing within the directory explicitly designate the position of the required element within the library file data partition. The format of the library file disk directories exists as a function of the needs of the prime routines accessing the directories. The directory format differs in record layout from the prime data partition of a library file, in that directory records are fixed, 13-byte blocked items. The directory block prefixes are identical to those of the file partition.

Disk directory records are composed of:

- a name field;
- a type indication; and
- a file pointer

Directory entries are made whenever the respective file record is:

- a module header for program source, macro/jproc, or object code;
- a phase definition for each phase of a load module;
- an entry ESD record for object code;
- a beginning-of-group (BOG) or end-of-group (EOG) demarcator;

- a named CSECT record for object code; or
- a procedure name for a macro module in proc format. (This is the directory entry for which there is no unique corresponding record in the prime data partition. This item points to the module header record.)

2.7.2.1. Directory Format

System libraries are built and managed by using the system access technique (SAT) access method. Thus, the first partition of each standard library file in the system consists of an index of pointers to the prime data area of the file described by the second partition. This directory index consists of a series of 13-byte slots, each pointing to the corresponding record in the prime data area. The directory blocks may be 251 bytes in length; the last four bytes of each directory block are unused when the block is full (contains 19 items). As many directory blocks as are needed to accommodate the needed number of index entries for a given library are available. The last index entry for each library directory is the index to the EOF record in the prime data partition. Figure 2-6 illustrates the disk library file structure and the format of each directory record.

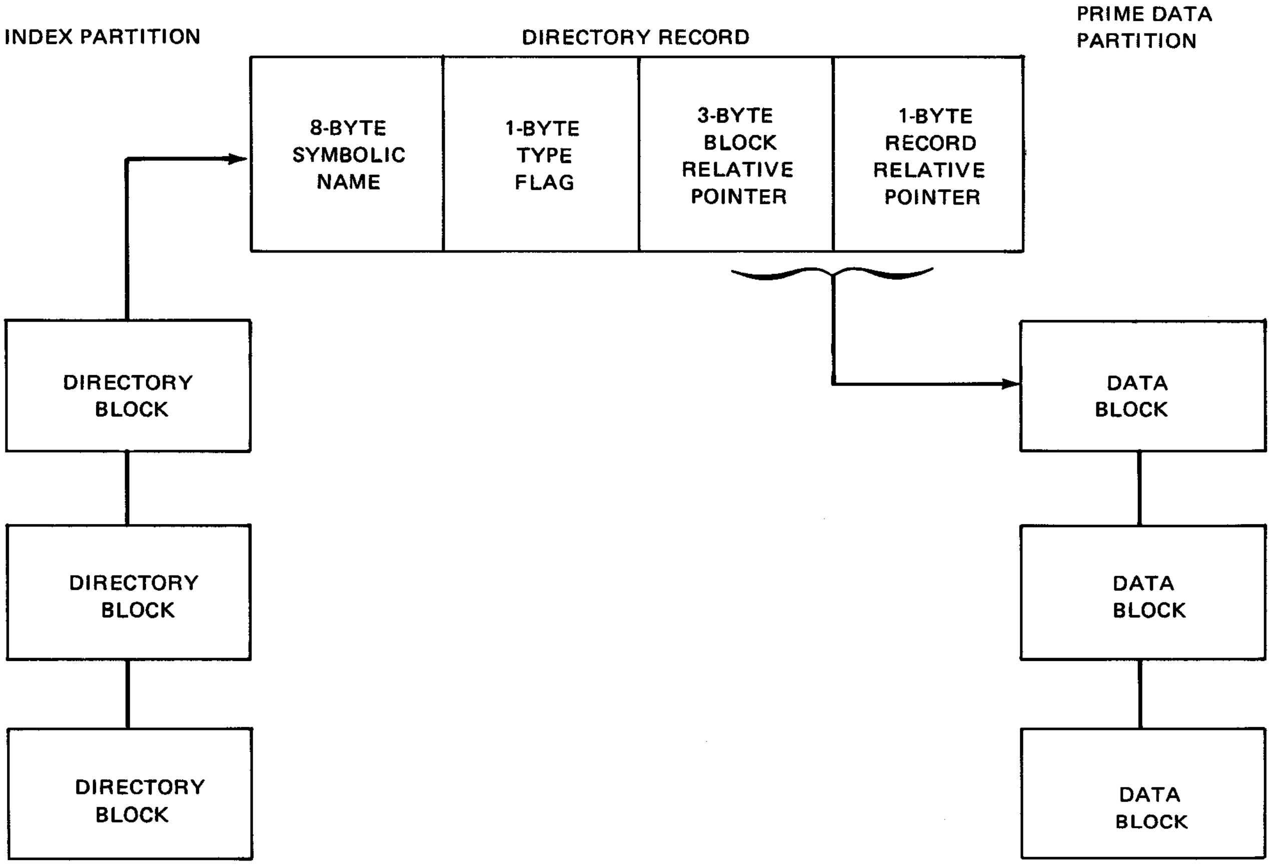


Figure 2-6. Disk Library File Structure

The symbolic name field (bytes 1 through 8) of a directory record is used as the identifier of the module or demarcator existing in the prime data partition. The type field specifies the demarcation flag for the respective record. The values of the type flag field correspond to the record type field in the prime data area. Table 2-2 lists the type flags possible in an index item.

The block relative pointer to the prime data area is a relative block number within the second file partition indicating the block containing the respective record. The record relative pointer is the number of bytes from the beginning of the block to the beginning of the record. The record relative pointer and block relative number are computed when the prime data area is constructed. The pointers for macro name header index items (in the proc format) always point to the beginning of the proc module regardless of where the *name* directive is contained within the body of the module.

Table 2-2. Disk Directory Index Type Flags

Hexadecimal Value	Demarcation
00	Nullified item
04	ENTRY name (object module)*
08	CSECT name (object module)*
80	Object module header
90	Phase header (load module)
A0	Beginning of group demarcator
A1	EOF sentinel
A2	Macro/jproc name header
A3	Macro/jproc module header
A4	Program source module header
A8	End of group demarcator
B0	Block module header record

*Multiple duplicate names can appear in a library file directory.

2.7.3. Card Libraries

The librarian can punch libraries into cards and, in turn, can access card files as input. Source module items, element headers, phase definitions, CSECT, ESD, ISD, PHASE, and TRANSFER records are punched directly. Text/RLD records in object and load elements are treated specially since the record size is variable. Thus, punched card formats for text/RLD records may require multiple punched card records.

Whenever object or load modules are punched into cards, a 5-digit sequence number is punched in columns 1 through 5, providing a card deck sequence check facility. When punching source modules, the librarian creates 80-byte source records (the source module header is eliminated) directly from the library.

When librarian functions require punched card output, the name PUNCH must be specified on the // LFD job control statement. With the punched card output, the librarian creates an ELE card to precede the module and an EOD card to end the module. The ELE card will be in the format:

LABEL	△OPERATION△	OPERAND
	ELE	D1, module-type,module-name

When filing object or load module card libraries, the librarian reconstructs the module from the card decks, checking the sequence number of each card and the record types within each module. When source modules are created from cards, the appropriate headers are created, prefixes attached, etc.

2.7.4. Tape Libraries

The formats for tape libraries are the same as those for disk libraries except that:

- tape libraries have only a data partition, no directory partition; and
- modules having the same name and type may exist in the same tape library. However, the first module encountered is the one processed.

Because of the structure of a tape library, once a module is written to a library, that module cannot be deleted or altered in any way in that same library. Therefore, the input library and a new output library must be specified when making changes to a tape library. This new library can be another tape, disk pack, or punched cards. The following control statements are not supported for a tape library because the operation takes place in the input file or involves the directory: DEL, LST, PAC, REC, REN, REPRO, and SEQ. If a load module on tape is blocked, the BLK control statement must specify an output file that is different from the input file.

The librarian provides the option of specifying a physical tape block length other than the standard length of 256 bytes. The variable block length is specified for each nonstandard length type in the job control stream via a DD job control statement. The format of this statement is:

```
// DD BKSZ=n
```

where:

n

Specifies the block length in bytes of the particular file. This block length can be:

- any multiple of 256,
- not greater than 8192; and
- for an input file, at least as large as the value used when the file was allocated.

The // DD BKSZ=n statement informs the librarian to either produce a physical tape output block or input a physical input tape block of the size specified. If the // DD BKSZ=n is omitted, a standard block size of 256 bytes is assumed.

Additional main storage space must be allocated when processing variable-length block tapes. The formula for computing the necessary main storage space is:

```
28,672 + 2048 + (10272 * number of tapes)
```

Whenever more than one variable-length block is specified in your job control stream, the I/O buffer space must be allocated for the largest combination of block lengths used in any single library function.

Your tape libraries must have the standard header and trailer label records and the name specified in the // LBL job control statement must agree with the file header 2 label of your tape library. The data management user guide, UP-8068 (current version) provides the information concerning the header and trailer label records associated with tape libraries.

All tapes can be prepped by using either the prep option of the // VOL job control statement or the tape prep routine (TPREP).

NOTE:

You should use extreme caution when specifying the PREP option of the // VOL job control statement for a tape file to be processed by the librarian. With this option, the tape is prepped every time it is opened as an output file. If a tape file is used as both an input file and an output file during a single job, every time the file is reopened as an output file, all data on the tape as a result of previous operations will be overwritten if the PREP option has been specified. If the operation was intended as a continual building of the tape file, the results would be disastrous.

When tape prep is specified in the same job step with a librarian file that contains more than six input files that output to the same tape, the seventh input will cause the tape to reset the output block to block number 1.

To avoid these problems, you should prep the tape in a separate job step.

2.7.5. Diskette Libraries

The librarian can be either input from a diskette or punched to a diskette. Diskette library processing is the same as card library processing.

2.8. CONTROL STATEMENTS AND PATCH CARDS

The system librarian of the SPERRY UNIVAC Operating System/3 (OS/3) is instructed in its task requirements via control statements presented by the user through the control stream. These statements designate information such as functions required, module or group name of code to be serviced, logical files associated with the various tasks, and options applying to the selected functions. The control statements and patch correction cards that permit the librarian to perform these various library file servicing jobs are described in this section. The control statements are presented in alphabetical sequence.

2.8.1. Control Statement Conventions and Format

All of the librarian control statements adhere to the following statement conventions:

- Control statements may be written in free form.
- Each operation code is composed of an identifier that describes the function. The operation code may be followed by a character string signifying options that alter normal processing of the function. The character string is separated from the operation identifier by a period.
- The operand field of each statement is composed of a variable set of positional parameters. Some positional parameters are optional. Optional parameters are indicated by brackets; choice alternatives are indicated by braces. Operands must be separated from the operation field by at least one blank space. Consecutive positional parameters must not contain embedded blanks.
- Prime librarian control statements may appear in any logical sequence within the librarian update control stream. Subfunction control statements must follow their associated prime control statements.
- File and module names may be composed of up to eight characters each. Inserted comments used to describe specific modules may consist of up to 30 characters including embedded blanks.
- Macro, proc, or jproc definition modules may be specified by using the letter M in the positional parameter describing the type of module.

The coding format of all the librarian control statements is:

LABEL	△OPERATION△	OPERAND	73 SEQUENCE
unused	function [.options]	p1,p2,p3,p4,p5	seq-no

where:

`function`
Is the mnemonic of the librarian process to be performed.

`options`

Is a string of one or more of the following letters, depending on the function specified.

A Specifies that all groups with a specified name are to be processed. (Must be used with a G option)

C Specifies that the *name* parameter in the operand field is a module name prefix or a group name prefix rather than a complete name.

D Specifies that the entire module or module group being processed is to be listed on the librarian map. (This may also be used to obtain a table of contents for a specified library.)

E Specifies that the card module is terminated when the librarian detects the first EOD statement following the ELE statement in the control stream.

G Specifies that the *name* parameter in the operand field is a group name rather than a module name. This option will initiate processing of only one group of the name specified, unless the C or A option also is specified. Whenever this option is used, the module type parameter should be omitted from the operand field.

M Specifies that the module identified in the parameter field is to be processed only if another module of the same name and type is in the output file.

N Specifies that the printing of header records on the librarian map is to be suppressed.

P Specifies that the entire module, or module group being processed, is to be reproduced in punched cards.

Q Specifies that the module identified in the parameter field is to be processed only if no other module of the same name and type is in the output file.

U Specifies that processing is to be performed on all modules from the current position of the file up to and including the module identified in the *name* parameter. Whenever this option is used, the type of the module identified in the *name* parameter must also be specified in the operand field, unless the G option is also being used.

X Extend an unblocked, single-phase, load module.

NOTES:

1. *If contradictory options are specified for a single librarian function, a diagnostic message is printed on the librarian map and the last option specified is honored.*

2. *Unless the RES control statement is specified, the file pointer is positioned at the current position of the file rather than at the beginning of the file. Therefore, when using options C, G, or U, specify the RES control statement to access the entire file.*

p1
Is a logical file name or a group name.

p2
Is a module type, a logical file name, a module name, or a sequence control field.

p3
Is a module name or a sequence control field.

p4
Is a logical file name, a comment, or a sequence control field.

p5
Is a comment.

seq-no
Is a 1- to 8-character alphanumeric sequence control number of which at least one character must be numeric.

2.8.2. Patch Card Formats

Because there is no standard format for a librarian patch card, but rather, several standard formats, the patch (correction) card formats recognized by librarian functions are described immediately after the librarian control statement that makes use of a particular type of patch card.

2.8.3. Blocking Load Modules (BLK) Control Statement

Function:

You use the BLK control statement to convert a standard load module to a block load module. Block load modules are intended to increase the efficiency of program loading in that all or large parts of the overlay phases may be loaded by a single I/O operation. When the load module is in block format, fewer disk accesses are required because the loader can read the entire phase at once (if the phase is less than or equal to one track in length), or one track at a time until the entire phase is loaded (if the phase is more than one track in length). If the load module were in standard format, then each phase would be loaded piece by piece, that is, 256 bytes at a time.

All files containing block modules must have three partitions, thus differing from the standard two-partition file. The block load module's first and second partitions are standard, but partition 3 is not. Partition 3 is not structured and is made up of contiguous text data, free of any control information, and is allocated by SAT when the file is first opened. The data in partition 2 describes the boundaries of each phase in partition 3. The text data in partition 3 is in sequential load order and is binary zero-filled when appropriate.

In standard load format, no text records can be overlaid; however, in block load format, they can. An example of this overlaying would be when the load module detects the following coding:

```
Loc       Operation       Operand
ØØØØ      CLI             R6,X'Ø1'
ØØØ4      BC              8,STOR1
ØØØ4      ORG             *-4
ØØØ4      BC              15,STOR2
```

The BC 15 overlays the BC 8. In standard load module format, the bytes of text for the BC 8, R1 continue to exist in the module although they are overlaid at load time.

Since the objective of converting to a blocked format is to increase the efficiency of program loading, the following considerations should be kept in mind when making a decision on load module format:

- Modules less than 4K bytes in length take longer to load if in blocked format, unless the resident loader (RESMOD.SM$LOD) is configured in your supervisor. In this case, block loading is as fast or faster than for standard load format.

- Do not block modules having information passed from one phase to the next in a DS area. All DS areas are zero filled.

- Patches to a blocked module phase will significantly slow down the loading of that phase. Patches do not affect the loading of standard load module phases.

- Loading blocked load modules from a selector channel disk is slower than loading from an IDA disk.

- Do not block modules written in assembly language and having address constants overlaid with text. This is the case where an unneeded address constant is used for patch space. It can also occur by using the assembler ORG statement and overlaying an address constant with instructions.

Format:

LABEL	△OPERATION△	OPERAND
unused	BLK	input-lfn,module-name[,output-lfn]

Positional Parameter 1:

input-lfn
Specifies the logical file name of the disk file on which the original load module resides.

Positional Parameter 2:

module-name
Specifies the name of the load module to be converted.

Positional Parameter 3:

output-lfn
Specifies the logical file name of the disk file to be used in the block operation.

If omitted, the input file contains the blocked module, and the original load module is nullified. This parameter is required if the input file is a tape.

Example:

```
1        10   16
--------------------------
         BLK  D1,STEP2,D2
```

This example converts the standard load module named STEP2 residing on file D1 to a block load module and places it on file D2.

NOTES:

1. *If a block load module with the same name is detected in the same file, the one already present is nullified and the new one added.*
2. *Load modules generated from ANSI 1974 COBOL source code that includes the dynamic CALL or CANCEL verbs cannot be converted to block format.*

2.8.4. Write Beginning-of-Group (BOG) Record Control Statement

Function:

This statement is used to begin a module group by writing a beginning-of-group record in a specified file. The modules that are to comprise the group must be added to the file before the end-of-group (EOG) record is written on the file.

Format:

LABEL	△OPERATION△	OPERAND
unused	BOG	group-name [,{lfn $Y$RUN}]

Options:

None

Positional Parameter 1:

group-name
A 1- to 8-character alphanumeric character string that specifies the name of the module group being started. Module groups within a given file may have identical names. Only one group, however, is processed each time a process group function is performed by the librarian, unless the C or A option also is specified.

Positional Parameter 2:

lfn
Specifies the logical file name of the disk or tape file on which the beginning-of-group record is to be written.

If omitted, the job run library ($Y$RUN) is used.

Examples:

```
     1        10    16
   ┌─────────────────────────
1. │          BOG   EXAMPLE1,D1
2. │          BOG   EXAMPLE2
```

1. Begins a module group named EXAMPLE1 on file D1.

2. Begins a module group named EXAMPLE2 on the $Y$RUN file.

2.8.5. Compare Elements (COM) Control Statement

Function:

This control statement permits the comparison of two source modules in two separate files on a record-by-record basis or the comparison of two complete files on a block-by-block basis. No other options are available with this command. The two source modules to be compared must have the same name and type designations.

When comparing two source modules, the librarian first locates them in the two files to be used. The comparison then occurs on a record-by-record basis. When a discrepancy is detected, the two source items are listed in EBCDIC. The sequence control fields (columns 73–80 unless altered by user specification) are then examined and the module with the lowest value has its file pointer advanced one record. The comparison is then redone. (Source modules so compared should be presequenced in some fashion so that a control field is available.) If the sequence control fields are equal when such a discrepancy occurs, both file pointers are advanced one record. The comparison continues until the end of a module is reached. Figure 2–7 illustrates an example of the librarian map produced during a source module compare operation.

If no source module name is provided, both files are compared in their entirety from beginning to end. This involves a block-by-block comparison. When a discrepancy occurs, both blocks are listed in hexadecimal, each file pointer is advanced one block, and the comparison continues. The process proceeds until end-of-file is detected on one of the libraries being scanned. Figure 2–8 illustrates an example of the librarian map produced during a file compare operation.

Format:

LABEL	△OPERATION△	OPERAND
unused	COM	[{prim-lfn }] [,{S}] [,{ n-n }][,name],sec-lfn [{$Y$RUN }] [{M}] [{73-80}]

Options:

None

Positional Parameter 1:

prim-lfn

Specifies the logical file name of the first disk or tape file to be used in the comparison.

If omitted, the job run library ($Y$RUN) is used.

Positional Parameter 2:

S,M

Specifies the type of modules being compared as either a program source module (S), or macro/jproc module (M).

If omitted, all modules in both files will be compared.

Positonal Parameter 3:

n-n

Two decimal numbers, separated by a hyphen, that specify the starting and ending columns of the sequence control field to be used if a source module is to be compared.

If omitted, columns 73-80 are assumed. If name is not specified, this parameter is ignored.

Positional Parameter 4:

name

Specifies the name of a source module to be compared. The module named will first be located in both files, then compared. Each must be a source level module.

If omitted, the files designated will be compared on a block-by-block basis from beginning to end, and positional parameters 2 and 3 are ignored.

Positional Parameter 5:

sec-lfn

Specifies the logical file name of the second disk or tape file to be used in the comparison.

Examples:

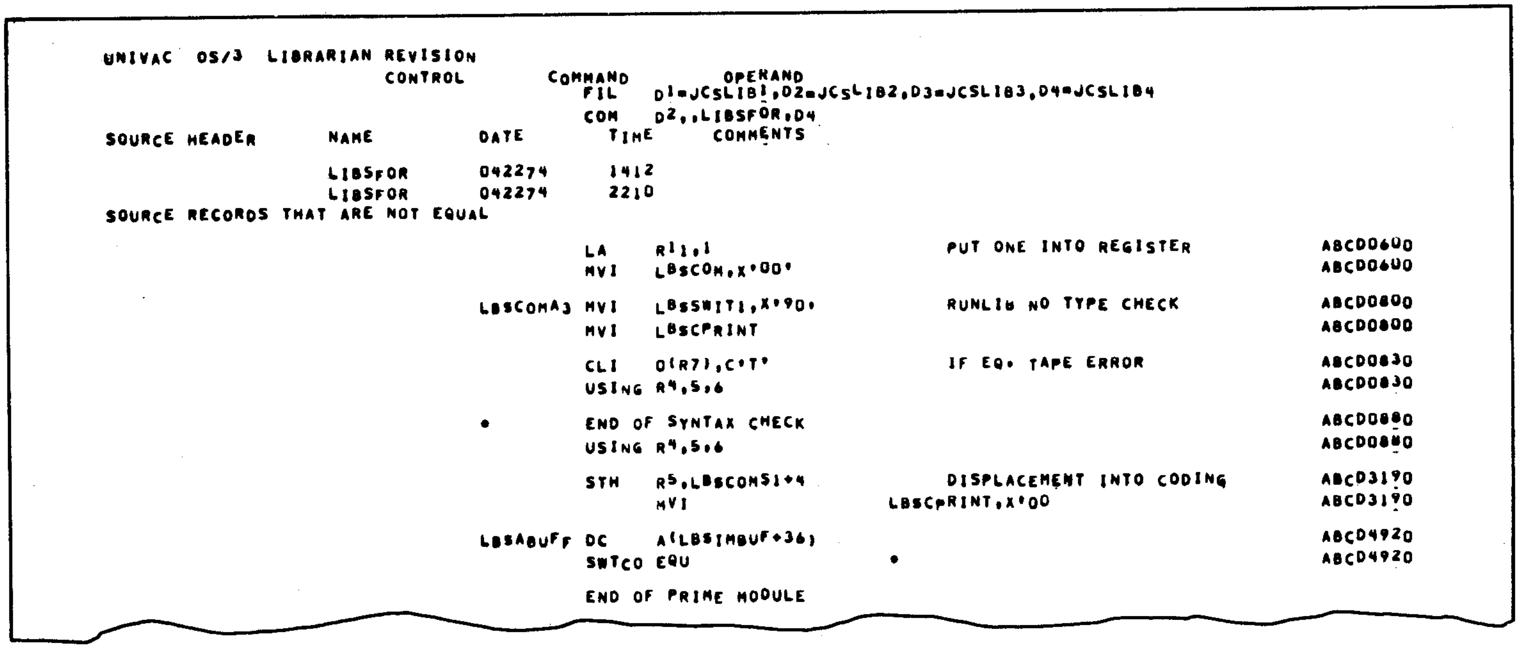
```
    1         10    16
1.            COM   D1,S,,EXAMPLE1,D2
2.            COM   D2,S,1-8,SORCMOD,D3
3.            COM   D11,S,,EXAMPLE3,D12
4.            COM   D5,,,,D6
```

1. Compares the source module named EXAMPLE1 in file D1 with the source module named EXAMPLE1 in file D2. The sequence control field used is positions 73-80 of the source module records compared.

2. Compares the source module SORCMOD in file D2 with the source module SORCMOD in file D3. The sequence control field used is positions 1-8 of the compared source module records.

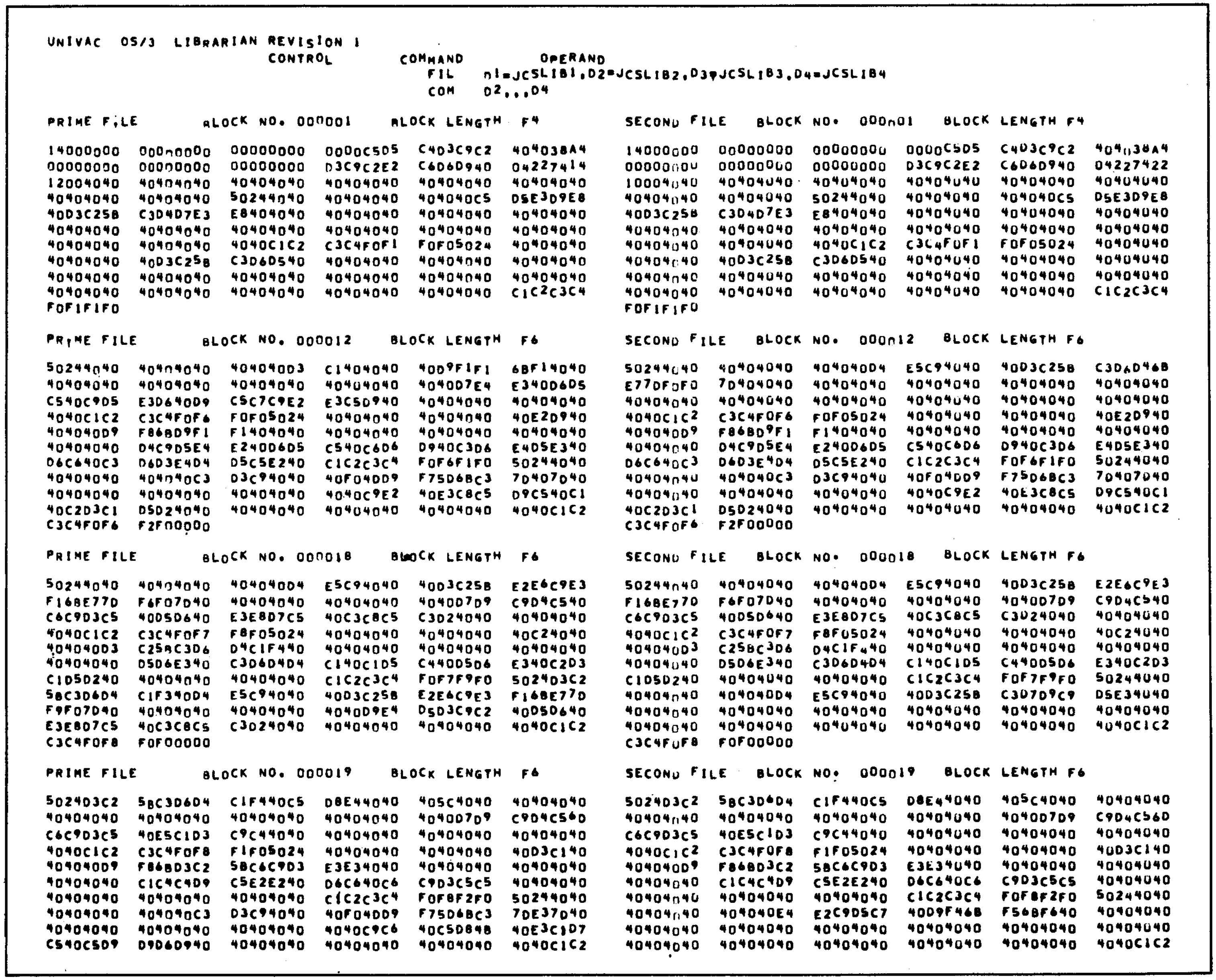
3. Compares the module EXAMPLE3 which exists in source code format in file D11 with the source module EXAMPLE3 in file D12. The sequence control field used in positions 73-80 of the source module records compared.

4. Compares all the modules in file D5 with all modules in file D6. The compare is terminated by end-of-file of either file. The compare is on a block-by-block basis from beginning to end of each file.



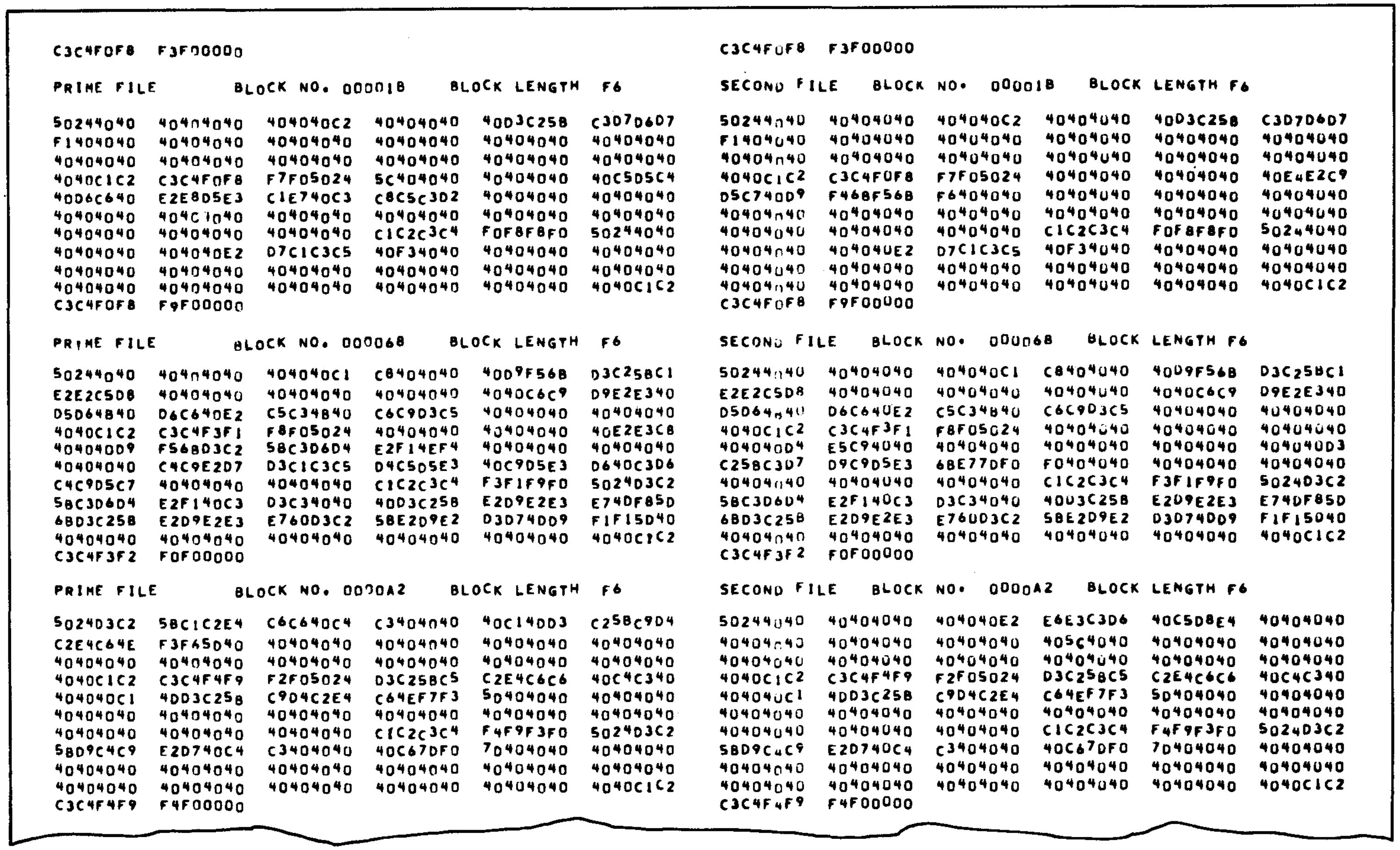
```
UNIVAC  OS/3  LIBRARIAN REVISION
                        CONTROL       COMMAND         OPERAND
                                         FIL   D1=JCSLIB1,D2=JCSLIB2,D3=JCSLIB3,D4=JCSLIB4
                                         COM   D2,,LIBSFOR,D4
SOURCE HEADER      NAME         DATE       TIME     COMMENTS

                   LIBSFOR      042274     1412
                   LIBSFOR      042274     2210
SOURCE RECORDS THAT ARE NOT EQUAL

                                         LA    R11,1                   PUT ONE INTO REGISTER            ABCD0600
                                         MVI   LBSCOM,X'00'                                             ABCD0600

                                LBSCOMA3 MVI   LBSSWIT1,X'90'          RUNLIB NO TYPE CHECK             ABCD0800
                                         MVI   LBSCPRINT                                                ABCD0800

                                         CLI   0(R7),C'T'              IF EQ. TAPE ERROR                ABCD0830
                                         USING R4,5,6                                                   ABCD0830

                                *        END OF SYNTAX CHECK                                            ABCD0880
                                         USING R4,5,6                                                   ABCD0880

                                         STH   R5,LBSCOMS1+4           DISPLACEMENT INTO CODING         ABCD3190
                                               MVI            LBSCPRINT,X'00                            ABCD3190

                                LBSABUFF DC    A(LBSIMBUF+36)                                           ABCD4920
                                         SWTCO EQU            *                                         ABCD4920

                                         END OF PRIME MODULE
```

Figure 2-7. Typical Librarian Map for Source Module Compare Operations



```
UNIVAC  OS/3  LIBRARIAN REVISION 1
                         CONTROL        COMMAND          OPERAND
                                          FIL    D1=JCSLIB1,D2=JCSLIB2,D3=JCSLIB3,D4=JCSLIB4
                                          COM    D2,,,D4

PRIME FILE        BLOCK NO. 000001    BLOCK LENGTH  F4          SECOND FILE    BLOCK NO.  000001    BLOCK LENGTH F4

14000000  00000000  00000000  0000C5D5  C4D3C9C2  404038A4      14000000  00000000  00000000  0000C5D5  C4D3C9C2  404038A4
00000000  00000000  00000000  D3C9C2E2  C6D6D940  04227414      00000000  00000000  00000000  D3C9C2E2  C6D6D940  04227422
12004040  40404040  40404040  40404040  40404040  40404040      10004040  40404040  40404040  40404040  40404040  40404040
40404040  40404040  50244040  40404040  404040C5  D5E3D9E8      40404040  40404040  50244040  40404040  404040C5  D5E3D9E8
40D3C258  C3D4D7E3  E8404040  40404040  40404040  40404040      40D3C258  C3D4D7E3  E8404040  40404040  40404040  40404040
40404040  40404040  40404040  40404040  40404040  40404040      40404040  40404040  40404040  40404040  40404040  40404040
40404040  40404040  4040C1C2  C3C4F0F1  F0F05024  40404040      40404040  40404040  4040C1C2  C3C4F0F1  F0F05024  40404040
40404040  40D3C258  C3D6D540  40404040  40404040  40404040      40404040  40D3C258  C3D6D540  40404040  40404040  40404040
40404040  40404040  40404040  40404040  40404040  40404040      40404040  40404040  40404040  40404040  40404040  40404040
40404040  40404040  40404040  40404040  40404040  C1C2C3C4      40404040  40404040  40404040  40404040  40404040  C1C2C3C4
F0F1F1F0                                                        F0F1F1F0

PRIME FILE        BLOCK NO. 000012    BLOCK LENGTH  F6          SECOND FILE    BLOCK NO.  000012    BLOCK LENGTH F6

50244040  40404040  404040D3  C1404040  40D9F1F1  6BF14040      50244040  40404040  404040D4  E5C94040  40D3C25B  C3D6D46B
40404040  40404040  40404040  40404040  404007E4  E34006D5      E770F0F0  7D404040  40404040  40404040  40404040  40404040
C540C9D5  E3D640D9  C5C7C9E2  E3C5D940  40404040  40404040      40404040  40404040  40404040  40404040  40404040  40404040
4040C1C2  C3C4F0F6  F0F05024  40404040  40404040  40E2D940      4040C1C2  C3C4F0F6  F0F05024  40404040  40404040  40E2D940
404040D9  F86BD9F1  F1404040  40404040  40404040  40404040      404040D9  F86BD9F1  F1404040  40404040  40404040  40404040
40404040  D4C9D5E4  E24006D5  C540C6D6  D940C3D6  E4D5E340      40404040  D4C9D5E4  E24006D5  C540C6D6  D940C3D6  E4D5E340
D6C640C3  D6D3E4D4  D5C5E240  C1C2C3C4  F0F6F1F0  50244040      D6C640C3  D6D3E4D4  D5C5E240  C1C2C3C4  F0F6F1F0  50244040
40404040  404040C3  D3C94040  40F040D9  F75D6BC3  7D407040      40404040  404040C3  D3C94040  40F040D9  F75D6BC3  7D407D40
40404040  40404040  40404040  4040C9E2  40E3C8C5  D9C540C1      40404040  40404040  40404040  4040C9E2  40E3C8C5  D9C540C1
40C2D3C1  D5D24040  40404040  40404040  40404040  4040C1C2      40C2D3C1  D5D24040  40404040  40404040  40404040  4040C1C2
C3C4F0F6  F2F00000                                              C3C4F0F6  F2F00000

PRIME FILE        BLOCK NO. 000018    BLOCK LENGTH  F6          SECOND FILE    BLOCK NO.  000018    BLOCK LENGTH F6

50244040  40404040  404040D4  E5C94040  40D3C25B  E2E6C9E3      50244040  40404040  404040D4  E5C94040  40D3C25B  E2E6C9E3
F16BE77D  F6F07D40  40404040  40404040  404007D9  C9D4C540      F16BE770  F6F07D40  40404040  40404040  404007D9  C9D4C540
C6C9D3C5  40D5D640  E3E8D7C5  40C3C8C5  C3D24040  40404040      C6C9D3C5  40D5D640  E3E8D7C5  40C3C8C5  C3D24040  40404040
4040C1C2  C3C4F0F7  F8F05024  40404040  40404040  40C24040      4040C1C2  C3C4F0F7  F8F05024  40404040  40404040  40C24040
404040D3  C25BC3D6  D4C1F440  40404040  40404040  40404040      404040D3  C25BC3D6  D4C1F440  40404040  40404040  40404040
40404040  D5D6E340  C3D6D4D4  C140C1D5  C440D5D6  E340C2D3      40404040  D5D6E340  C3D6D4D4  C140C1D5  C440D5D6  E340C2D3
C1D5D240  40404040  40404040  C1C2C3C4  F0F7F9F0  50240 3C2     C1D5D240  40404040  40404040  C1C2C3C4  F0F7F9F0  50244040
5BC3D6D4  C1F340D4  E5C94040  40D3C258  E2E6C9E3  F16BE77D      40404040  404040D4  E5C94040  40D3C258  C3D7D9C9  D5E34040
F9F07D40  40404040  40404040  4040D9E4  D5D3C9C2  40D5D640      40404040  40404040  40404040  40404040  40404040  40404040
E3E8D7C5  40C3C8C5  C3D24040  40404040  40404040  4040C1C2      40404040  40404040  40404040  40404040  40404040  4040C1C2
C3C4F0F8  F0F00000                                              C3C4F0F8  F0F00000

PRIME FILE        BLOCK NO. 000019    BLOCK LENGTH  F6          SECOND FILE    BLOCK NO.  000019    BLOCK LENGTH F6

5024D3C2  5BC3D6D4  C1F440C5  D8E44040  405C4040  40404040      5024D3C2  5BC3D6D4  C1F440C5  D8E44040  405C4040  40404040
40404040  40404040  40404040  40404040  404007D9  C9D4C56D      40404040  40404040  40404040  40404040  404007D9  C9D4C56D
C6C9D3C5  40E5C1D3  C9C44040  40404040  40404040  40404040      C6C9D3C5  40E5C1D3  C9C44040  40404040  40404040  40404040
4040C1C2  C3C4F0F8  F1F05024  40404040  40404040  40D3C140      4040C1C2  C3C4F0F8  F1F05024  40404040  40404040  40D3C140
404040D9  F86BD3C2  5BC6C9D3  E3E34040  40404040  40404040      404040D9  F86BD3C2  5BC6C9D3  E3E34040  40404040  40404040
40404040  C1C4C4D9  C5E2E240  D6C640C6  C9D3C5C5  40404040      40404040  C1C4C4D9  C5E2E240  D6C640C6  C9D3C5C5  40404040
40404040  40404040  40404040  C1C2C3C4  F0F8F2F0  50244040      40404040  40404040  40404040  C1C2C3C4  F0F8F2F0  50244040
40404040  404040C3  D3C94040  40F040D9  F75D6BC3  7DE37D40      40404040  404040E4  E2C9D5C7  40D9F46B  F56BF640  40404040
40404040  40404040  40404040  4040C9C6  40C5D848  40E3C1D7      40404040  40404040  40404040  40404040  40404040  40404040
C540C5D9  D9D60940  40404040  40404040  40404040  4040C1C2      40404040  40404040  40404040  40404040  40404040  4040C1C2
```

Figure 2-8. Typical Librarian Map for File Compare Operations (Part 1 of 2)



```
C3C4F0F8  F3F00000                                                C3C4F0F8  F3F00000

PRIME FILE       BLOCK NO. 00001B    BLOCK LENGTH  F6             SECOND FILE   BLOCK NO.  00001B   BLOCK LENGTH F6

50244040  40404040  404040C2  40404040  40D3C258  C3D7D6D7        50244040  40404040  404040C2  40404040  40D3C258  C3D7D6D7
F1404040  40404040  40404040  40404040  40404040  40404040        F1404040  40404040  40404040  40404040  40404040  40404040
40404040  40404040  40404040  40404040  40404040  40404040        40404040  40404040  40404040  40404040  40404040  40404040
4040C1C2  C3C4F0F8  F7F05024  5C404040  40404040  40C5D5C4        4040C1C2  C3C4F0F8  F7F05024  40404040  40404040  40E4E2C9
40D6C640  E2E8D5E3  C1E740C3  C8C5C3D2  40404040  40404040        D5C740D9  F468F56B  F6404040  40404040  40404040  40404040
40404040  404C1040  40404040  40404040  40404040  40404040        40404040  40404040  40404040  40404040  40404040  40404040
40404040  40404040  40404040  C1C2C3C4  F0F8F8F0  50244040        40404040  40404040  40404040  C1C2C3C4  F0F8F8F0  50244040
40404040  404040E2  D7C1C3C5  40F34040  40404040  40404040        40404040  404040E2  D7C1C3C5  40F34040  40404040  40404040
40404040  40404040  40404040  40404040  40404040  40404040        40404040  40404040  40404040  40404040  40404040  40404040
40404040  40404040  40404040  40404040  40404040  4040C1C2        40404040  40404040  40404040  40404040  40404040  4040C1C2
C3C4F0F8  F9F00000                                                C3C4F0F8  F9F00000

PRIME FILE       BLOCK NO. 000068    BLOCK LENGTH  F6             SECOND FILE   BLOCK NO.  000068   BLOCK LENGTH F6

50244040  40404040  404040C1  C8404040  40D9F56B  D3C258C1        50244040  40404040  404040C1  C8404040  40D9F56B  D3C258C1
E2E2C5D8  40404040  40404040  40404040  4040C6C9  D9E2E340        E2E2C5D8  40404040  40404040  40404040  4040C6C9  D9E2E340
D5D64B40  D6C640E2  C5C34B40  C6C9D3C5  40404040  40404040        D5D64B40  D6C640E2  C5C34B40  C6C9D3C5  40404040  40404040
4040C1C2  C3C4F3F1  F8F05024  40404040  40404040  40E2E3C8        4040C1C2  C3C4F3F1  F8F05024  40404040  40404040  40404040
404040D9  F568D3C2  58C3D6D4  E2F14EF4  40404040  40404040        404040D4  E5C94040  40404040  40404040  40404040  404040D3
40404040  C4C9E2D7  D3C1C3C5  D4C5D5E3  40C9D5E3  D640C3D6        C258C3D7  D9C9D5E3  6BE77DF0  F0404040  40404040  40404040
C4C9D5C7  40404040  40404040  C1C2C3C4  F3F1F9F0  5024D3C2        40404040  40404040  40404040  C1C2C3C4  F3F1F9F0  5024D3C2
58C3D6D4  E2F140C3  D3C34040  40D3C258  E2D9E2E3  E74DF85D        5BC3D6D4  E2F140C3  D3C34040  40D3C258  E2D9E2E3  E74DF85D
6BD3C258  E2D9E2E3  E760D3C2  5BE2D9E2  D3D74DD9  F1F15D40        6BD3C258  E2D9E2E3  E760D3C2  5BE2D9E2  D3D74DD9  F1F15040
40404040  40404040  40404040  40404040  40404040  4040C1C2        40404040  40404040  40404040  40404040  40404040  4040C1C2
C3C4F3F2  F0F00000                                                C3C4F3F2  F0F00000

PRIME FILE       BLOCK NO. 0000A2    BLOCK LENGTH  F6             SECOND FILE   BLOCK NO.  0000A2   BLOCK LENGTH F6

5024D3C2  5BC1C2E4  C6C640C4  C3404040  40C14DD3  C25BC9D4        50244040  40404040  404040E2  E6E3C3D6  40C5D8E4  40404040
C2E4C64E  F3F65D40  40404040  40404040  40404040  40404040        40404040  40404040  40404040  405C4040  40404040  40404040
40404040  40404040  40404040  40404040  40404040  40404040        40404040  40404040  40404040  40404040  40404040  40404040
4040C1C2  C3C4F4F9  F2F05024  D3C25BC5  C2E4C6C6  40C4C340        4040C1C2  C3C4F4F9  F2F05024  D3C25BC5  C2E4C6C6  40C4C340
404040C1  4DD3C25B  C9D4C2E4  C64EF7F3  5D404040  40404040        404040C1  4DD3C25B  C9D4C2E4  C64EF7F3  5D404040  40404040
40404040  40404040  40404040  40404040  40404040  40404040        40404040  40404040  40404040  40404040  40404040  40404040
40404040  40404040  40404040  C1C2C3C4  F4F9F3F0  5024D3C2        40404040  40404040  40404040  C1C2C3C4  F4F9F3F0  5024D3C2
5BD9C4C9  E2D740C4  C3404040  40C67DF0  7D404040  40404040        5BD9C4C9  E2D740C4  C3404040  40C67DF0  7D404040  40404040
40404040  40404040  40404040  40404040  40404040  40404040        40404040  40404040  40404040  40404040  40404040  40404040
40404040  40404040  40404040  40404040  40404040  4040C1C2        40404040  40404040  40404040  40404040  40404040  4040C1C2
C3C4F4F9  F4F00000                                                C3C4F4F9  F4F00000
```

Figure 2-8. Typical Librarian Map for File Compare Operations (Part 2 of 2)

2.8.6. Copy Elements (COP) Control Statement

Function:

The COP control statement is intended primarily to:

- Copy the contents of one entire library file to another library file

  Compression of the file being copied is performed as the new file is created, thus eliminating file fragmentation created by deleted modules and module groups. Only the input-lfn and output-lfn parameters are specified to obtain this function.

- Copy the contents of a library file from its current position up to and including a specified module or module group

  The U option must be specified in addition to the desired operands.

- Copy individual modules or module groups based on module names and types, or on module or module group name prefixes, from one library file to another

  If both the name prefix and module group options are specified, all the module groups with the specified name prefix are copied. If only the module group option is specified, only the first module group of the name specified is copied. When module group processing is requested, the module type parameter must be omitted, as it is not appropriate.

- Produce a table of contents for a library file, listing all the records contained in the directory partition if the D option is specified, or only the module header records if the D option is omitted

  When this function is desired, only the input-lfn parameter should be specified, with or without the D option code. All other option codes are invalid.

While performing any of the previously mentioned copy operations, the librarian may also be requested to list (D option) and punch (P option) the modules copied, or suppress the module header record listing (N option) it would normally produce. Also, if the output file already contains any modules of the same name and type as those being copied to it, the old modules are nullified.

The COP control statement also can be used to:

- List and punch one or more modules or module groups without performing a copy operation

- Position a library file pointer without performing a copy operation

These functions are obtained by simply omitting the output-lfn parameter. To produce listings of every module in a file, specify a D option, the input-lfn, and a trailing comma (COP.D DO,). This will distinguish it from the COP statement, which produces a table of contents for the file specified (COP DO).

When using the copy facility to create a new tape and you want the modules being copied to make up the new file, you must use a previously prepped tape free of any data. You must follow this procedure because the copy facility does not reinitialize an output file and all modules being copied are automatically written to the end of the output file. Also, your input and output tape files must be on separate volumes.

You should not attempt to copy an ICAM symbiont to a file that contains an active ICAM symbiont with the same name. The active ICAM symbiont is deleted.

NOTE:

You should use extreme caution when specifying the PREP option of the // VOL job control statement for a tape file to be processed by the librarian. With this option, the tape is prepped every time it is opened as an output file. If a tape file is used as both an input file and an output file during a single job, every time the file is reopened as an output file, all data on the tape as a result of previous operations will be overwritten if the PREP option has been specified. If the operation was intended as a continual building of the tape file, the results would be disastrous.

When tape prep is specified in the same job step with a librarian file that contains more than six input files that output to the same tape, the seventh input will cause the tape to reset the output block to block number 1.

To avoid these problems, you should prep the tape in a separate job step.

Format:

LABEL	△OPERATION△	OPERAND
unused	COP[.options]	[{input-lfn}] [,{S}] [,name][,output-lfn]
		[{YRUN}] [{M}]
		{O}
		{L}

Options:

A Process all groups in the input-lfn with the group name specified in the name parameter. (The G option must also be used.)

C Name specified in the *name* parameter is either a module name prefix or group name prefix.

D List all the modules copied, or if a table of contents is being produced, list all the records in the file directory.

G Name specified in the name parameter is a module group name or module group name prefix if the C option is also specified. If neither the C or A option is used, process the modules in only the first group found with the specified name. If the C option is used, process all groups with the specified prefix. If the A option is used, process all groups with the specified name. When goup processing is requested, the module type parameter should be omitted, as it is inappropriate.

M Copy a specified module from the input file to the output file only if the output file already contains a module with the same name and type.

N Do not list any header records on the librarian map.

P Punch the modules processed. This option cannot be used when requesting a table of contents for a file.

Q Copy a specified module from the input file to the output file only if the output file does not already contain a module with the same name and type.

U Process the modules from the current position of the input file, up to and including the specified module or module group. This option is ignored when producing a table of contents for a file.

Positional Parameter 1:

`input-lfn`

Specifies the logical file name of the disk or tape input file.

If omitted, the run library ($Y$RUN) of the job is used.

Positional Parameter 2:

`S,M,O,L`

Specifies the type of module being copied as either a program source module (S), macro/jproc source module (M), object module (O), or load module (L).

If omitted, all modules with the specified name from the current position to the end of the file are copied.

Positional Parameter 3:

`name`

Specifies the name of the module or module group (G option) to be copied, or a name prefix (C option), and may consist of up to eight characters.

If omitted, all modules from current position to the end-of-file of the specified type are copied. If no type or name is specified, all modules from current position to end-of-file are copied.

Positional Parameter 4:

`output-lfn`

Specifies the logical file name of the disk or tape unit output file to be used in the copy operation. The output file specification is not necessary to position a file, to list a disk file directory, or to list or punch specified modules in a file.

If omitted, only a subfunction (list, punch, position) of the COP statement can be performed. (See examples 1 and 3.) However, if you specify an input-lfn, module-type, and module-name and omit the output-lfn, the file pointer is positioned to the next module after the specified module in the file. (See example 7.)

Examples:

```
      1        10     16
    ┌──────────────────────────────
1.  │          COP    D1
2.  │          COP.D D2,S,MYMOD,D3
3.  │          COP.GP D1,,MYGROUP
4.  │          COP.UN DØ,M,MYMOD,TØ
5.  │          COP.C T2,L,MY,D1
6.  │          COP.D D1
7.  │          COP D1,S,COBOL4
```

1. Lists all the header records in the D1 (compare with example 6).

2. Copies source module MYMOD from file D2 to file D3 and provides a listing of module MYMOD.

3. Punches all modules in the module group MYGROUP, from file D1.

4. Copies to tape file TO all modules from current position of DO up to and including the procedure module MYMOD. Current position is reset to immediately follow MYMOD. The listing of header records is suppressed.

5. Copies any load module whose name begins with MY from the current position to the end of the file on tape file T2 to disk file D1.

6. Lists all directory records in file D1 (compare with example 1).

7. Positions the input-file to the next module following COBOL4.

2.8.7. Correct Module (COR) Control Statement

Function:

This statement is used to specify that the content of a source, object, or load module is to be corrected. Correction cards following the COR statement specify how the module is to be corrected. The librarian end-of-data (EOD) card indicates the end of the correction cards. Corrected modules may be output to either the same file or another file. For source modules, corrections are indicated via the sequence number field of the correction itself. Stand-alone deletions require the use of subfunction control statements. For object or load modules, the correction cards construct a text record containing the data and instructions required as patch corrections necessary to the specified object or load module. Text patches are inserted in the corrected module just ahead of the transfer record. Then, whenever the module is loaded in main storage or linked, its corrected text is inserted in the appropriate places in the module, overlaying any text that may have been nullified because of their replacement. When patched modules are listed, patches are flagged. When making changes to object and load modules, control section and phase sizes may not be altered. Patches must be correctly sequenced for phased load modules.

If errors are detected in the correction cards (for example, a wrong phase number), the librarian will not add these correction cards to your file.

The librarian will not terminate the card module until an unattached EOD card is detected, unless the E option is specified.

Format:

LABEL	△OPERATION△	OPERAND
unused	COR[.options]	[{input-lfn}],{S},name[,output-lfn] [{$Y$RUN}] M O L

Options:

E Terminate at the first EOD.

N Do not list header records, subfunction control statements, or records added or deleted.

P Punch module corrected.

X Extend the load module if any of the supplied patch addresses are beyond the end of the module. This option can be used only for unblocked single-phase load modules.

Positional Parameter 1:

input-lfn

Specifies the logical file name of the disk or tape file containing the module to be corrected. If a tape is specified as input, then a different output-lfn must be specified. The librarian cannot read and write from the same tape file.

If omitted, $Y$RUN is used.

Positional Parameter 2:

`S,M,O,L`
Specifies the type of module being corrected as either a program source module (S), macro/jproc source module (M), object module (O), or load module (L).

Positional Parameter 3:

`name`
Specifies the name of the module to be corrected.

Positional Parameter 4:

`output-lfn`
Specifies the logical file name of the disk file into which the corrected module is to be placed.

If omitted, the original module is deleted and the corrected version is added to the end of the input file. This parameter is required if the input file is a tape.

2.8.8. COR Correction Cards

2.8.8.1. Object or Load Module Corrections

Subfunction patch corrections for object and load modules must immediately follow the COR control statement. The last patch correction must be followed by a librarian EOD control statement. If the librarian detects an error within a correction card, it does not make the correction. Both text and relocation data (RLD) records may be supplied for the patch. RLD masks must be represented in hexadecimal 3-byte multiples exactly as required. Each patch supplied causes the generation of an appropriate text record. Contiguous patch addresses on succeeding patches do not cause the generated text to be merged. Load module patches must be correctly sequenced by phase number.

The format of a patch correction card for an object or load module is:

```
1
______________________________________________________________
{-}address [,{esid-no  }][,text[,RLD]]
{P}        [ {phase-no }]
           [ {ORG      }]
```

Column 1:

`-(hyphen)`
Indicates that the specified address is relative to the object or load module address.

P
: Indicates that the specified address is relative to the load module phase being patched.

Positional Parameter 1:

address
: Specifies the hexadecimal address that is relative to the base address of the object or load module and not to the address of the CSECT or phase area. This relative address is assigned to the generated text record. The address can be either positive or negative. A positive address begins in column 2, while a negative address has a hyphen in column 2 followed by the address.

Positional Parameter 2:

esid-no
: Specifies the external symbol identification number for the object module being patched. The number must be in the range of 01–255.

If omitted, 01 is used.

phase-no
: Specifies the phase number of the load module being patched. The number must be in the range of 00–99.

If omitted, 00 is used.

ORG
: Indicates that this is not a correction but specifies that the indicated address (positional parameter 1) is automatically added to all subsequent patch address fields until a new value is specified on another ORG correction card or an EOD statement is encountered in the control stream. When you use this parameter, text is not permitted.

Positional Parameter 3:

text
: Specifies a contiguous string of hexadecimal digits to be assigned at the resultant address (which is the sum of the specified address and, if specified, the most recent ORG address). The minimum amount of text patchable is one byte. Text is required unless ORG is specified in positional parameter 2. If text is not specified, the patch correction is flagged and the relocation data (RLD), if present, is disallowed.

Positional Parameter 4:

rld
: Indicates any relocation data for the specified object or load module text record being created. The rld data must be in 3-byte, 6-hexadecimal-digit multiples.

NOTES:

1. *Padding of zeros to the nearest half byte is automatic for the address, esid/phase number, and test specifications.*

2. *If the esid/phase number is omitted, the comma still must be coded.*

Example:

In this example, a multiphase load module named MYLOAD is corrected. The original version of the module resides in file D0, while the corrected version will reside in file D2.

```
    1        1Ø    16
    -------------------------
             COR   DØ,L,MYLOAD,D2
1.  -C9Ø,488ØDØ74
2.  P12D,1,954ØCØ12
3.  -Ø124,ORG
4.  P25Ø,3,AB
5.  --4B78,ORG
6.  -5672,4,ØA1C
7.  -Ø,ORG
8.  -D2E,6,ØØØ12E,Ø16FØØ
             EOD
```

1. The specified text is applied to load module address C90 of phase 0.

2. The specified text is applied to phase relative address 12D of phase 1.

3. The value 0124 is entered as the ORG value.

4. The specified text is applied at 250+124 bytes into phase 3.

5. The value -4B78 is entered as the ORG value.

6. The specified text is applied at the load module address 5672-4B78 of phase 4.

7. The ORG value is cleared.

8. The specified text and RLD are applied to load module relative address D2E of phase 6.

2.8.8.2. Source Module Corrections

To make source module insertions and replacements, the actual source module record to be inserted is used as the correction card. Replacements are performed by using a correction card with the same sequence number as the record to be replaced. Insertions are performed by using at least one correction card (always the first card) with a sequence number falling between the sequence number of the records between which the insertion is to be made. Any number of unsequenced correction cards may then follow.

Source module corrections will appear in the listing in the following manner:

- For source statement replacements, the statement being replaced and the replacement statement are printed.
- For insertions of new statements, the line preceding the insertion and the inserted line are displayed.

Figure 2-9 is an example of a source module correction showing the original source module, the librarian stream used to modify the module, and then the resultant corrected module as it appears after the librarian has completed its processing.

If the corrections to a source module include the /$–/* job control statements, they must be paired.

The source module always must contain record sequence identifiers for it to be corrected by the librarian; however, source modules are not required to carry sequence numbers to be in a given library. Sequence numbers optionally may be added to a source module whenever the user chooses, either at creation time from cards through the ELE control statement, or anytime afterwards, through the sequence (SEQ) control statement being used as a primary function. Cards that are out of sequence in a correction deck are inserted in the source module out of sequence (in the same order they appear in the correction deck), and the appropriate error message is printed on the librarian map.

There are three control statements that are used as subfunctions of the COR control statement to correct or reorder a source module. These are the skip (SKI), recycle (REC), and sequence (SEQ) control statements. They are to be used strictly as control statements.

```
SM$ERR3  START                                                        SM$00010
         PRINT NOGEN                                                  SM$00020
         SUPEQU                                                       SM$00030
         PRINT GEN                                                    SM$00040
         DC    'DATE SOURCE && REF   AMOUNT    DATE SOURCE && REF   '  SM$00050
         DC    '  REF-2  ORIG-DATE  DATE SOURCE && REF    AMOUNT    '  SM$00060
         LR    R1,R6                                                  SM$00070
         LR    R4,R5                                                  SM$00080
         END                                                          SM$00090
```

a. Original source module

```
1        10     16                                                      72
--------------------------------------------------------------------------
         COR    D1,S,SM$ERR3,D2
         MVC    LB$ERR,SM$ERR                                     SM$ØØØ2Ø
         L      R3,LB$MIKE                                        SM$ØØØ6Ø
         SKI    SM$ØØØ9Ø                                          SM$ØØØ9Ø
LB$BBBB                                                           SM$ØØ11Ø
         EOD
```

b. Correction card deck

```
------            PRINT NOGEN                                                SM$00020
++++++            MVC   LB$ERR,SM$ERR                                        SM$00020

------            DC    '  REF-2  ORIG-DATE  DATE SOURCE && REF   AMOUNT   '  SM$00060
++++++            L     R3,LB$MIKE                                           SM$00060

                  SKI   SM$00090                                             SM$00090

******            LR    R4,R5                                                SM$00080
++++++   LB$BBBB                                                             SM$00110
```

c. Librarian printout showing additions, deletions, and replacements

```
        SM$ERR3  START                                                        SM$00010
                 MVC   LB$ERR,SM$ERR                                          SM$00020
                 SUPEQU                                                       SM$00030
                 PRINT GEN                                                    SM$00040
                 DC    'DATE SOURCE && REF   AMOUNT    DATE SOURCE && REF   '  SM$00050
                 L     R3,LB$MIKE                                             SM$00060
                 LR    R1,R6                                                  SM$00070
                 LR    R4,R5                                                  SM$00080
        LB$BBBB                                                               SM$00110
```

d. Corrected source module

Figure 2—9. Example of Source Module Corrections

2.8.9. Delete Elements (DEL) Control Statement

Function:

This facility allows you to eliminate certain modules or module groups within a specified library. The deletion is of one module, or is inclusive up to and including the named module or group existing in the specified file. When elements of a given file are being deleted, all referenced code is effectively nullified. These dead modules or groups can then be removed through an eventual copy or pack function to eliminate the resulting fragmentation. You should not attempt to delete an ICAM symbiont while it is actively processing.

Format:

LABEL	△OPERATION△	OPERAND
unused	DEL[.options]	[{lfn SYSRUN}] [,{S M O L}][,name]

Options:

A Delete all groups with the group name specified by the name parameter. (The G option must also be used.)

C Delete all modules whose name begins with the prefix specified in the *name* parameter.

D List modules being deleted.

G Name specified in positional parameter 3 is a module group name rather than an individual module name. If neither the C or A option is used, delete all the modules in the first group encountered with that name. If the C option is used, delete all groups with the group name prefix specified by the name parameter. If the A option is used, delete all groups with that group name. Each operation starts from the current file position. When a group is deleted, the BOG and EOG records associated with that group are also deleted. When module group processing is requested, the module type parameter should be omitted, as it is inappropriate.

N Do not list header records.

P Punch modules being deleted.

U Delete from current position up to and including specified module. If a module name is specified, then a type must also be included. If no module name is specified, delete all modules after current position.

Positional Parameter 1:

`lfn`
Specifies the logical file name of the disk file in which the deletion is to occur.

If omitted, the job run library file ($Y$RUN) is utilized.

Positional Parameter 2:

`S,M,O,L`
Specifies the type of modules being deleted as either a program source module (S), macro/jproc source module (M), object module (O), or load module (L).

If omitted, all modules with the specified name from the current position to the end of the file are to be deleted.

Positional Parameter 3:

`name`
Specifies the module name, module group name, or module name prefix of the modules to be deleted.

If omitted, modules of the specified type are deleted. If both type and name specifications are omitted, all modules are to be deleted.

Examples:

```
    1        10    16
1.           DEL.D D1,S,EXAMPLE1
2.           DEL.P ,O
3.           DEL.C D2,O,EXA
4.           DEL.UN DØ,L,MYMOD
```

1. Deletes and lists the source module named EXAMPLE1 on file D1.

2. Deletes and punches all object modules from the job run library.

3. Deletes all object modules from current position to end-of-file in file D2 whose name begins with EXA.

4. Deletes all modules from current position to the load module named MYMOD in the D0. Also suppresses the listing of header records.

NOTES:

1. *The DEL control statement cannot be used if processing tape libraries.*

2. *Only root phase header records are printed during the delete operation; however, all overlays are deleted.*

2.8.10. Add Card File Element (ELE) Control Statement

Function:

This statement is used to add a source, object, or load module that is contained in cards to a disk or tape file. If a card element is being added to a disk file that already contains a module of the same name and type, the old module is replaced by the new module. The ELE control statement causes a module header record to be inserted in the specified output file. All cards immediately following the ELE card down to the end-of-data (EOD) card are assumed to comprise the module to be added. Librarian control streams are valid source modules, but each EOD card that is a part of that control stream must be associated with its own COR or ELE control statement. The librarian will not terminate the card module until an unattached EOD card is detected, unless the E option is specified.

Format:

LABEL	△OPERATION△	OPERAND
unused	ELE[.options]	[{lfn $Y$RUN}],{S M O L},name[,comments]

Options:

D List the module.

E Terminate at the first EOD.

N Do not list the header record.

P Punch the module.

Positional Parameter 1:

`lfn`

Specifies the logical file name of the disk, diskette, or tape file to which this card module is to be added.

If omitted, the job run library ($Y$RUN) is used.

Positional Parameter 2:

`S,M,O,L`

Specifies the type of the module being added as either a program source module (S), macro/jproc source module (M), object module (O), or load module (L).

Positional Parameter 3:

`name`

A 1- to 8-character alphanumeric string that specifies a name for the module being added.

For object and load modules, the name on the ELE card must be the same as the name of the module.

Positional Parameter 4:

`comments`

Up to 30 characters of comments to be inserted in the module header record.

If omitted, no comment is included in the header record.

Examples:

```
    1        10    16
   ---------------------------------------
1. |         ELE   D1,S,EXAMPLE1,NEW SOURCE MODULE
2. |         ELE   ,L,EXAMPLE2
3. |         ELE   D12,O,EXAMPLE3
```

1. Adds a source module name EXAMPLE1 to file D1 and, if a source module named EXAMPLE1 already exists therein, it will be nullified. The comment "new source module" will also be inserted into the comment field of the header record.

2. Adds a load module named EXAMPLE2 to the job run library and, if a load module of the same name already exists therein, it will be nullified.

3. Adds an object module named EXAMPLE3 to file D12 and, if an object module of the same name already exists therein, it will be nullified.

NOTES:

1. *The add, replace, or check sequence numbers (SEQ) control statement is supported as a subfunction command to the ELE control statement to: perform a sequence check on a source module being filed, sequence a source module being filed, or resequence a source module being filed.*

2. *A source module cannot have an EOD control statement as part of its coding since this statement is a terminator card. If detected in your source module, only that portion of your source module up to the EOD is added. Whenever an EOD control statement is used, it must be paired with a COR or ELE control statement.*

2.8.11. Declare End-of-Data (EOD) Control Statement

Function:

This statement is used to terminate the card data that follows an ELE, COR, or REPRO control statement. Each EOD card must be associated with one and only one ELE or COR card.

Format:

LABEL	△OPERATION△	OPERAND
unused	EOD	unused

Options:

None

2.8.12. Write End-of-Group (EOG) Record Control Statement

Function:

This statement is used to terminate a module group by writing an end-of-group record (Table B-2) in a specified file.

Format:

LABEL	△OPERATION△	OPERAND
unused	EOG	group-name [,{lfn SYSRUN}]

Options:

None

Positional Parameter 1:

`group-name`
Specifies the name of the module group being ended.

Positional Parameter 2:

lfn
: Specifies the logical file name of the disk or tape file on which the end-of-group record is to be written.

If omitted, the job run library ($Y$RUN) is used.

Examples:

```
     1         10     16
   ┌─────────────────────────────
1. │           EOG    EXAMPLE1,D1
2. │           EOG    EXAMPLE2
```

1. Places an end-of-group record on file D1 with the name EXAMPLE1.
2. Places an EOG record named EXAMPLE2 on the job run library.

2.8.13. ESCAPE (ESC) CONTROL STATEMENT

Function:

This statement causes the librarian to read all subsequent librarian control statements from either a SAM file or from a librarian disk source module rather than from the control stream. Your SAM file can reside on either a disk, diskette, or tape. You can think of the librarian control statements as a procedure module whereby the same control statements can be executed over again without change. You need to change only the FIL control statement to process different files.

ESC processing terminates when the end of module or end of file is detected. All statements read by ESC processing appear on the librarian map with *ESC* in the control field.

Format 1:

LABEL	ΔOPERATIONΔ	OPERAND
unused	ESC	$\texttt{filename,}\begin{Bmatrix}\texttt{TP}\\\texttt{DK}\\\texttt{DT}\end{Bmatrix}\left[,\begin{Bmatrix}\texttt{FU}\\\texttt{FB}\\\texttt{VU}\\\texttt{VB}\end{Bmatrix}\right]\left[,\begin{Bmatrix}\texttt{record-length}\\\texttt{80}\end{Bmatrix}\right]$ $\left[,\begin{Bmatrix}\texttt{block-length}\\\texttt{80}\end{Bmatrix}\right]$

Options:

None

Positional Parameter 1:

`filename`

Name of your SAM file containing the librarian control stream to be processed. The maximum allowable length is seven characters. The first character must be alphabetic.

Positional Parameter 2:

`TP,DK,DT`

Specifies the type of file to be read. The entries are TP, for a tape file; DK, for a disk file; and DT, for a diskette file.

Positional Parameter 3:

`FU,FB,VU,VB`

Specifies the record type for the file being read. Permissible entries are:

FU	Fixed, unblocked records
FB	Fixed, blocked records
VU	Variable, unblocked records
VB	Variable, blocked records

If this parameter is omitted, fixed, unblocked records are assumed.

NOTE:

SAM diskette files may not contain blocked records.

Positional Parameter 4:

`record-length`

Specifies the length in bytes of fixed records. Maximum permissible entry is decimal 128. For tape, the minimum permissible entry is decimal 18. If this parameter is omitted, a record length of 80 bytes is assumed. This parameter is not required for variable-length records. For fixed, unblocked records, this field is ignored.

Positional Parameter 5:

block-length
Indicates the length in bytes of the file blocks including all block header and record header fields. If the block length exceeds 1024 decimal bytes, see the section on additional storage requirements. For tapes, the minimum block length is decimal 18. For variable-length records, this entry indicates the maximum block size. If omitted, a block size of 80 bytes is assumed.

NOTES:

1. *The block length must equal or exceed the specified record length.*
2. *The ESC command can process tape files with or without block numbers.*
3. *For diskettes, the maximum block length is 1024 decimal bytes.*

Format 2:

LABEL	△OPERATION△	OPERAND
unused	ESC	filename,LD,modulename

Options:

None

Positional Parameter 1:

filename
Specifies the name of the file containing the librarian control stream module.

Positional Parameter 2:

LD
Indicates the control stream is in a librarian source module.

Positional Parameter 3:

modulename
Specifies the name of the librarian source module containing the librarian control stream to be processed.

Main Storage Considerations:

When you use the ESC control statement, you must specify additional main storage on the // JOB control statement. The amount of main storage required is dependent upon the file type being read and the extra storage required for block sizes in excess of 1024 bytes. To calculate the main storage amount, use the following equation:

`file type + excess block size + X'5C00'`

where:

`file type`
: Is bytes (in hexadecimal) required for a particular file type. The values are:

  X'1250' for a file type of TP (SAM tape)

  X'C50' for a file type of DT (SAM diskette)

  X'F50' for a file type of DK (SAM disk)

  X'780' for a file type of LD (librarian disk)

`excess block size`
: A hexadecimal value representing the number of bytes that the blocks in your program are in excess of 1024. Calculate this value by converting the decimal block size value to hexadecimal and subtracting X'400' from the resulting value.

Example 1. Using a librarian SAM disk file

Librarian control stream:

```
①  // JOB ESCRUN,,,8000
②  // DVC 20 // LFD PRNTR
③  // DVC 50  // VOL PUBRES
④  // LBL PRGFIL // LFD PRGFIL
⑤  // DVC 50 // VOL PUBRES
⑥  // LBL LIBFIL // LFD LIBFIL
⑦  // EXEC LIBS
⑧  /$
⑨              FIL   D1=PRGFIL,D2=LIBFIL
⑩              ESC   LIBFIL,DK
⑪  /*
⑫  /&
⑬  // FIN
```

SAM disk file:

```
⑭              PAC   D2
⑮              COP   D1,S,COBOL4,D2
⑯              LST   D2
```

Line 1 shows the JOB statement with the additional main storage required for the job.

Lines 2-6 show the device assignment set for the system printer and our disk files.

Line 7 shows the EXEC statement calling the librarian.

Lines 8 and 11 are the data delimiters for the librarian control stream.

Line 9 shows the FIL statement.

Line 10 shows the ESC statement identifying our file and file type.

Lines 12 and 13 end our job and card reader operation.

Lines 14–16 show the librarian control statements to be executed during ESC processing.

Example 2. Using a librarian source module

The following coding shows an example of ESC processing. Our source module COBOL4 residing in D1 is copied to D2 and all the header records are listed from D2.

Librarian control stream:

```
①  // JOB ESCRUN,,,8ØØØ
②  // DVC 2Ø // LFD PRNTER
③  // DVC 5Ø // VOL PUBRES
④  // LBL HAMMER // LFD HAMMER
⑤  // DVC 5Ø // VOL PUBRES
⑥  // LFD SRCFIL // LFD SRCFIL
⑦  // EXEC LIBS
⑧  /$
⑨                FIL  D1=HAMMER,D2=SRCFIL
⑩                ESC  SRCFIL,LD,LIBTEST
⑪  /*
⑫  /&
⑬  FIN
```

Librarian source module:

```
⑭                COP  D1,S,COBOL4,D2
⑮                LST  D2
```

Line 1 shows the JOB statement with the additional main storage required for ESC processing.

Lines 2–6 show the device assignment set for the system printer and our disk files.

Line 7 shows the EXEC statement calling the librarian.

Lines 8 and 11 are the data delimiters for the librarian control stream.

Line 9 shows the FIL statement.

Line 10 shows the ESC statement identifying the module.

Lines 12 and 13 end our job and card reader operation.

Lines 14 and 15 show the librarian control statements to be executed during ESC processing.

2.8.14. Declare File (FIL) Control Statement

Function:

The control statement is used to declare to the librarian all the tape and disk files that will be referenced subsequently in the control stream through // LFD control statement. At the same time, each file is assigned a type code (disk or tape) and a logical file number (0–15), which together form a logical file name that is to be used (rather than the file name) for all subsequent file references within the librarian control stream. File declarations may be strung out on one FIL card or be made individually on separate FIL cards. Up to 32 files can be declared: 16 tape files and 16 disk files. For each file described by the FIL statement, an appropriate job control file declaration card is required in the job control stream (unless a standard system or job run library file is being used).

Format:

LABEL	△OPERATION△	OPERAND
unused	FIL	$\left\{\begin{matrix}\texttt{Tn=filename-1}\\ \texttt{Dn}\end{matrix}\right\}\left[,\ldots,\left\{\begin{matrix}\texttt{Tn}\\ \texttt{Dn}\end{matrix}\right\}\texttt{=filename-n}\right]$

Options:

None

Tn Keyword Parameter:

`Tn=filename`
Is used to equate a tape file (LFD name) with a logical file name of T0 through T15.

Dn Keyword Parameter:

`Dn=filename`
Is used to equate a disk file (LFD name) with a logical file name of D0 through D15.

NOTE:

The file name specification may not exceed eight alphanumeric characters and must begin with an alphabetic character. When working with system files, you must equate the logical file name with the file identifier if DVC RES was used in the device assignment set for the resident volume.

Examples:

```
      1        10    16
   ┌─────────────────────────────────────────
1. │           FIL   TO=SCRTAPE,T1=MASTAPE,DØ=PROCLIB
2. │           FIL   T2=UPDATE,D1=LOADLIB
```

1. Declares the use of tape files SCRTAPE and MASTAPE, and of disk file PROCLIB, and assigns the logical file names TO, T1, and DO to the three files, respectively. Subsequent references to these files must specify the logical file names.

2. Declares the use of tape file UPDATE and disk file LOADLIB, and assigns the logical file names T2 and D1 to these files, respectively.

   NOTE:

   Using the FIL statement to equate files to be processed with logical file names allows a single LIBS control stream to be used to maintain any number of different files. The functions performed by the control stream use the logical file specifications declared in a FIL statement. When the needed files change, only the FIL statements need be modified. Thus, each command to the librarian need not specify the actual file name used.

2.8.15. Printing a File in Alphabetical Sequence (LST)

Function:

This command enables you to display a table of contents of a file in alphabetical sequence. The LST command has a built-in sort routine that sorts the directory records. At the completion of the sort, the LST command displays the listing in the form of module name, module type, date, and time. Groups and module header records are the only records printed; for example, the root phase of a load module.

Format:

LABEL	△OPERATION△	OPERAND
unused	LST	$\left[\begin{Bmatrix}\text{input-lfn}\\ \text{SYSRUN}\end{Bmatrix}\right]\left[,\begin{Bmatrix}\text{S}\\ \text{M}\\ \text{O}\\ \text{L}\end{Bmatrix}\right]$

Positonal Parameter 1:

input-lfn
Specifies the logical file name of the disk file containing the modules to be listed.

If omitted, the job run library is assumed to contain the modules.

Positional Parameter 2:

S,M,O,L
Specifies the type of modules being operated on as either program source modules (S), macro/jproc source modules (M), object modules (O), or load modules (L).

If omitted, the entire file is listed.

Examples:

```
      1        10    16
     ┌──────────────────────
  1. │         LST   D1,L
  2. │         LST
```

1. Prints an alphabetic listing of only the load modules residing in file D1.

2. Prints an alphabetic listing of the entire SYSRUN file.

2.8.16. Pack File (PAC) Control Statement

Function:

This operation compresses a library file by discarding any elements that are flagged as nullified and squeezing the remaining code sets together, thus eliminating any file fragmentation and pushing unused space toward the end of the file. This function may be used in conjunction with the delete (DEL) control statement to build a reordered, updated, and packed library file. The user should not attempt to pack a file that contains an ICAM symbiont while that symbiont is active.

The PAC printout shows both the modules being packed and the modules not being packed. The modules not being packed are printed first and are listed under the heading MODULES NOT MOVED. The modules being packed are then printed under the heading MODULES MOVED.

Format:

LABEL	△OPERATION△	OPERAND
unused	PAC[.options]	[{lfn SYSRUN}]

Options:

N Do not list header records.

Positional Parameter 1:

`lfn`

Specifies the logical file name of the disk file that is to be compressed.

If omitted, the job run library is compressed.

Example:

```
1        10    16
---------------------
         PAC   D1
```

Eliminates all nullified modules in file D1.

NOTES:

1. *The PAC control statement cannot be used if processing tape libraries.*
2. *The file being packed should be lockable, giving you exclusive use.*
3. *A file being packed cannot be updated.*
4. *When a load file is being packed, the pack operation must complete before a program can be executed from the file.*

2.8.17. Controlling Page Advancement for the Librarian Map (PAGE)

Function:

The PAGE librarian control statement starts a new page on the librarian map. It may also specify a header line to be printed at the top of each new page. This header line remains in effect for the duration of the librarian job step or until it is changed by another PAGE control statement.

If a PAGE librarian control statement is not used, the librarian starts a new page on the librarian map only when:

- the current page is full;
- an LST control statement is executed; or
- a COP control statement is used to print a file table of contents.

Format:

LABEL	△OPERATION△	OPERAND
unused	PAGE	['header-line']

Options:

None

Positional Parameter 1:

'header-line'
Specifies the header line to be printed at the top of each succeeding page. It can contain up to 64 characters and must be enclosed in single quotation marks.

This header line remains in effect for the duration of the job or until it is changed. If omitted, the current header line, if any, remains in effect.

Examples:

```
      1          10    16
   ___________________________________________
1.  |            PAGE
2.  |            PAGE  'PAYROLL MODULES (CONTINUED)'
3.  |            PAGE  ' '
```

1. Starts a new page on the librarian map, printing the current header line, if any, at the top of that page.

2. Starts a new page on the librarian map with new header line PAYROLL MODULES (CONTINUED) on that page and each succeeding page.

3. Starts a new page on the librarian map and ends the use of any previously specified header.

2.8.18. Specifying Error Handling during Librarian Execution (// PARAM ERROR)

Function:

This // PARAM statement specifies whether the librarian should stop the job step or cancel the entire job in the event of a librarian error.

Format:

```
// PARAM ERROR={STOP  }
               {CANCEL}
```

Keyword Parameter:

ERROR=STOP
Causes the librarian to stop processing the job step where the error occurred. Any librarian control statements following the error are not executed. However, any subsequent job steps are executed.

ERROR=CANCEL
Causes the librarian job to be terminated immediately. No subsequent job steps are executed.

NOTE:

The // PARAM statement cannot appear within the librarian control stream. It must be coded between the // EXEC LIBS and the /$ control statements.

2.8.19. Suppressing the Librarian Map (// PARAM PRINT=OFF)

Function:

This // PARAM statement suppresses printing of the librarian map. The printer device asignment set need not be present in your control stream when using this // PARAM statement. If a printer device assignment set is present, any subsequent job steps requiring the printer are not affected.

Format:

```
// PARAM PRINT=OFF
```

Keyword Parameter:

`PRINT=OFF`
Suppresses printing of the librarian map.

NOTE:

This must be the first parameter card in the job step and therefore must immediately follow the // EXEC LIBS statement in your control stream.

2.8.20. Printing Source Modules in Hexadecimal Format (// PARAM PRTOBJ)

Function:

This // PARAM statement causes any source module listings generated by the librarian to be printed in hexadecimal format.

Format:

```
// PARAM PRTOBJ=ON
```

Keyword Parameter:

`PRTOBJ=ON`
Causes any source module listings to be printed in hexadecimal format.

NOTE:

The // PARAM statement cannot appear within the librarian control stream. It must be coded between the // EXEC LIBS and the /$ control statements.

2.8.21. Creating a Multifile Tape (// PARAM TAPEFILES=MULTI)

Function:

This // PARAM statement allows the librarian to output more than one file to the same tape volume. If this statement is not used, only one file can be written to a tape. This statement is not required to read a file on a multifile tape.

A multifile tape is created by copying files from another storage medium to output files on the tape. The tape must already be prepped and it may or may not already contain some files. The files must be copied in sequence as they are to be arranged on the tape. These files cannot be extended later.

The librarian job, which copies the files to the tape, must contain this // PARAM statement. Also, the // LBL job control statement for each new tape file must include a file sequence number parameter (in positional parameter 4) to indicate the position of the output file on the tape. For example, if the tape already contains two files, the // LBL statement for the first new tape file must specify a 3 for the file sequence number. If the tape is a newly prepped tape that does not yet contain any files, the file sequence number for the first new tape file must be 1.

As the files are being output to the tape, only one tape file can be open at a time. Therefore, the librarian job step should use the same logical file name (Tn) for every tape file but redefine that logical file name in another FIL statement each time a new file is to be processed.

Normally, after each file is written to the tape and then closed, the librarian would rewind the tape to the load point. then it would reopen the tape and advance to the end of the tape again to write the next file. However, a // DD job control statement with a rewind parameter can be used in the device assignment set for each tape to eliminate unnecessary rewinding at each open and close operation.

The statement // DD OPRW=NORWD specifies no rewind at file open.

The statement // DD CLRW=NORWD specifies no rewind at file close.

Specifically, the device assignment sets for the tape files should specify:

- CLRW NORWD for the first tape file
- OPRW NORWD for the last tape file
- Both OPRW=NORWD and CLRW=NORWD for all other files

Format:

```
// PARAM TAPEFILES=MULTI
```

Keyword Parameter:

TAPEFILES=MULTI
Allows multiple files to be output to the same tape volume.

NOTE:

If used, this statement must be placed between the // EXEC LIBS statement and the /$ job control statement.

Example:

The following sample job copies five files from disk to the same new tape volume:

```
// JOB MULTFILE
// DVC 2Ø  // LFD PRNTR
// DVC 5Ø  // VOL DØØ41Ø  // LBL DISKFIL1  // LFD DISK1
// DVC 5Ø  // VOL DØØ41Ø  // LBL DISKFIL2  // LFD DISK2
// DVC 5Ø  // VOL DØØ41Ø  // LBL DISKFIL3  // LFD DISK3
// DVC 5Ø  // VOL DØØ41Ø  // LBL DISKFIL4  // LFD DISK4
// DVC 5Ø  // VOL DØØ41Ø  // LBL DISKFIL5  // LFD DISK5
// DVC 9Ø  // VOL SØ1841  // DD CLRW=NORWD
// LBL TFIL1,,,,1  // LFD TAPE1
// DVC 9Ø  // VOL SØ1841  // DD OPRW=NORWD,CLRW=NORWD
// LBL TFIL2,,,,2  // LFD TAPE2
// DVC 9Ø  // VOL SØ1841  // DD OPRW=NORWD,CLRW=NORWD
// LBL TFIL3,,,,3  // LFD TAPE3
// DVC 9Ø  // VOL SØ1841  // DD OPRW=NORWD,CLRW=NORWD
// LBL TFIL4,,,,4  // LFD TAPE4
// DVC 9Ø  // VOL SØ1841  // DD OPRW=NORWD
// LBL TFIL5,,,,5  // LFD TAPE5
// EXEC LIBS
// PARAM TAPEFILES=MULTI
/$
         FIL   DØ=DISK1,TØ=TAPE1
         COP   DØ,,,TØ
         FIL   DØ=DISK2,TØ=TAPE2
         COP   DØ,,,TØ
         FIL   DØ=DISK3,TO=TAPE3
         COP   DØ,,,TØ
         FIL   DØ=DISK4,TO=TAPE4
         COP   DØ,,,TØ
         FIL   DØ=DISK5,TØ=TAPE5
         COP   DØ,,,TØ
/*
/&
```

2.8.22. Specifying Date and Time during Librarian Execution (// PARAM UPDATE)

Function:

This // PARAM statement is used to specify the date and time to be in effect during the execution of a library job. This date and time is inserted in the header records of modules being corrected by the librarian. If a // PARAM UPDATE statement is not included in the librarian control stream, the date and time contained in the system information block (SIB) is used. This date and time remain in effect until the librarian job is terminated.

NOTE:

The // PARAM statement cannot appear within the librarian control stream. It must be coded between the // EXEC LIBS and the /$ statements.

Format:

```
// PARAM UPDATE=yymmdd/hhmm
```

Keyword Parameter:

`UPDATE=yymmdd/hhmm`

Specifies a date and time to be used for modules being corrected during the execution of the job.

2.8.23. Recycle Source Module Current Position Pointer (REC) Control Statement

Function:

The REC control statement is used only in conjunction with the COR control statement to make source module corrections. It causes the record pointer for the original source module to be repositioned to the first record in the source module. In conjunction with the SKI statement, it allows the user to rearrange major segments of a source module.

When a REC statement is processed, records are read from the original data set and written in the new data set up to and including the record whose sequence number matches the sequence number in the sequence field of the REC statement. Then, the record pointer for the original source module is reset to point to the first record in the module. If the sequence field of the REC statement is blank, repositioning of the record pointer takes place immediately.

Format:

		72	73
LABEL	**△OPERATION△**	**OPERAND**	**SEQUENCE**
	REC	unused	[last-sequence no]

Options:

None

Sequence Field Parameter:

`last-sequence-no`
Is a 1- to 8-character alphanumeric string that identifies the sequence number of the last record to be copied into the new data set before the record pointer is recycled to the first record in the module. This field begins in column 73 unless a SEQ control statement dictates otherwise.

If omitted, the recycling operation takes place without any records being copied into the new data set.

NOTE:

The REC control statement cannot be used if processing tape libraries.

Examples:

1. Figure 2-10 exemplifies how the REC and SKI control statements can be used to reorder a source module.

2. Figure 2-11 exemplifies how a source module can be corrected by using sequence data for control rather than for reordering purposes.

3. Figure 2-12 exemplifies how the SEQ statement can be used in a source module correction deck.

```
1          10     16                                                  72
--------------------------------------------------------------------------
EXAMPLE1        Source Statement                                  LIBSØ9ØØ
EXAMPLE2                                                          LIBSØ1ØØ
EXAMPLE3          .                                               LIBSØ2ØØ
EXAMPLE4                                                          LIBSØ3ØØ
EXAMPLE5          .                                               LIBSØ8ØØ
EXAMPLE6                                                          LIBSØ4ØØ
EXAMPLE7          .                                               LIBSØ6ØØ
EXAMPLE8                                                          LIBSØ7ØØ
EXAMPLE9                                                          LIBSØ5ØØ
EXAMPLEØ        Source Statement                                  LIBS1ØØØ
```

a. Original source module

Figure 2-10. Example of Source Module Reordering Operation (Part 1 of 2)

```
     1         10    16                                          72
 1.            SKI   LIBSØ9ØØ
 2.  UPDATE1           Source Statement                          LIBSØ2ØØ
 3.            SKI   LIBSØ8ØØ                                    LIBSØ8ØØ
 4.            SKI   LIBSØ7ØØ                                    LIBSØ6ØØ
 5.            REC
 6.            SKI   LIBSØ4ØØ                                    LIBSØ9ØØ
 7.  UPDATE2         Source Statement                            LIBSØ5ØØ
     UPDATE3         Source Statement                            LIBSØ55Ø
 8.            REC                                               LIBSØ7ØØ
 9.            SKI   LIBSØ3ØØ                                    LIBSØ9ØØ
10.            REC                                               LIBSØ8ØØ
11.            SKI   LIBSØ5ØØ                                    LIBSØ1ØØ
```

b. Correction deck

1. Skip source record LIBS0900.
2. Replace EXAMPLE3 source statement with UPDATE1 source statement.
3. Copy all records up to LIBS0800. LIBS0300 is copied. LIBS0800 is skipped.
4. Copy all records up to LIBS0600. LIBS0400 is copied. LIBS0600 and LIBS0700 are skipped.
5. Reposition record pointer back to the first record.
6. Start skipping at LIBS0900 and skip down to LIBS0400. No records are copied.
7. Insert UPDATE2 and UPDATE3 source records immediately after LIBS0400.
8. Before repositioning record pointer, copy down to and including LIBS0700; then reposition record pointer to the first record.
9. Start skipping at LIBS0900 and skip down to LIBS0300. No records are copied.
10. Before repositioning record pointer, copy down to and including LIBS0800; then reposition record pointer to the first record. LIBS0800 is the only record copied.
11. Start skipping at LIBS0100 and end skipping at LIBS0500. LIBS0900 and LIBS1000 are copied.

```
1          10    16                                          72
EXAMPLE2         Source Statement                            LIBSØ1ØØ
UPDATE1                                                      LIBSØ2ØØ
EXAMPLE4          .                                          LIBSØ3ØØ
EXAMPLE6                                                     LIBSØ4ØØ
UPDATE2           .                                          LIBSØ5ØØ
UPDATE3                                                      LIBSØ55Ø
EXAMPLE7          .                                          LIBSØ6ØØ
EXAMPLE8                                                     LIBSØ7ØØ
EXAMPLE5                                                     LIBSØ8ØØ
EXAMPLE1                                                     LIBSØ9ØØ
EXAMPLEØ         Source Statement                            LIBS1ØØØ
```

c. Corrected source module

Figure 2-10. Example of Source Module Reordering Operation (Part 2 of 2)

```
1        10    16                                                      72
-----------------------------------------------------------------------------
TESTEXAM EQU   *                                                       LINKØ1ØØ
         CR    RØ,W3                                                   LINKØ2ØØ
         BE    LK$3PA2Ø                                                LINKØ3ØØ
         BH    LK$3PA1Ø                                                LINKØ4ØØ
         M     W2,LK$CSGSZ                                             LINKØ5ØØ
         L     W2,LK$CSEGT                                             LINKØ6ØØ
         IC    W3,Ø(W2,W3)                                             LINKØ7ØØ
         N     W3,LK$CX7F                                              LINKØ8ØØ
         BNZ   LK$3PAØØ                                                LINKØ9ØØ
         LA    W3,LK$CROOT                                             LINK1ØØØ
TESTEXA1 EQU   *                                                       LINK11ØØ
         LR    RØ,W3                                                   LINK12ØØ
         LA    RRTNOD,4                                                LINK13ØØ
         B     LK$CPOP                                                 LINK14ØØ
         BAL   R14,LB$CSTK                                             LINK15ØØ
```

a. Source module

```
      1        10    16                                                      72
   -------------------------------------------------------------------------------
   |           COR   DO,S,TESTEXAM
1. |           CR    R1,W2                                                   LINKØ2ØØ
2. |           XR    W2,W2                                                   LINKØ45Ø
3. |           SKI   LINKØ8ØØ                                                LINKØ8ØØ
4. |           SKI   LINK14ØØ                                                LINK11ØØ
5. |           EOD
```

b. Correction deck

1. Replaces the source record with sequence number LINKO200 with this record.
2. Inserts this line between the lines with sequence numbers LINKO400 and LINKO500.
3. Deletes the line with sequence number LINKO800.
4. Deletes the lines starting with sequence number LINK1100 and ending with LINK1400.
5. Must be associated with the COR statement.

Figure 2-11. Example of Source Module Add/Replace/Delete Operation Using SKI Control Statements (Part 1 of 2)

```
1        10    16                                                      72
---------------------------------------------------------------------------
TESTEXAM EQU   *                                                       LINK0100
         CR    R1,W2                                                   LINK0200
         BE    LK$3PA20                                                LINK0300
         BH    LK$3PA10                                                LINK0400
         XR    W2,W2                                                   LINK0450
         M     W2,LK$CSGSZ                                             LINK0500
         L     W2,LK$CSEGT                                             LINK0600
         IC    W3,0(W2,W3)                                             LINK0700
         BNZ   LK$3PA00                                                LINK0900
         LA    W3,LK$CROOT                                             LINK1000
         BAL   R14,LB$CSTK                                             LINK1500
```

c. Corrected source module

Figure 2-11. Example of Source Module Add/Replace/Delete Operation Using SKI Control Statements (Part 2 of 2)

```
1        10    16                                                      72
---------------------------------------------------------------------------
         COR   DO,S,TESTEXAM
         SEQ   DO,S,TESTEXAM,1
LINK0200 CR    R1,W2
LINK0450 XR    W2,W2
LINK0800 SKI   LINK0800
LINK1100 SKI   LINK1400
         EOD
```

NOTE:

This example is the same as the one in Figure 2-10b except for the SEQ statement and the relocated sequence numbers.

Figure 2-12. Example of the SEQ Statement in a Source Module Correction Deck

2.8.24. Rename Element (REN) Control Statement

Function:

This control statement is used to rename a specific module, module group, or record; to mark object and load modules as sharable or unsharable; or to change the comments field in a module header record. When a load module name is changed, the new name is reflected throughout each phase of the load module.

Format:

```
LABEL     △OPERATION△      OPERAND
unused     REN[.options]   [{lfn   }] [,{S}]
                           [{SYSRUN}]    {M}
                                         {O}
                                         {L}
                             ,old-name [.{record-type-and-name}]
                                         {RON                 }
                                         {ROFF                }
                           [,new-name][,comments]
```

Options:

G Names specified are group names. The first module group encountered with the name identified as the *old-name* is to be renamed.

N Do not list header records.

Positional Parameter 1:

`lfn`

Specifies the logical file name of the disk file that contains the modules to be renamed or identified as reentrant or nonreentrant.

If omitted, the job run library is assumed to contain the subject modules.

Positional Parameter 2:

`S,M,O,L`

Specifies the type of modules being operated on as program source modules (S), macro/jproc source modules (M), object modules (O), or load modules (L).

If omitted, all modules of the specified old name are affected.

Positional Parameter 3:

```
old-name [.{record-type-and-name}]
           {RON                 }
           {ROFF                }
```

Identifies the module, module group, or record to be processed, or the object module to be marked as reentrant (RON) or nonreentrant (ROFF). The record type codes that may be specified are as follows:

C Indicates COM.

E Indicates ENTRY.

N Indicates procedure name.

P Indicates alias phase name.

S Indicates CSECT.

V Indicates V-CON.

X Indicates EXTRN.

Record names can be from one to eight characters long. The record type and name specification cannot contain any embedded blanks. An example of how this parameter might be coded is:

MASTER.XTAG5

When load, source, or object modules are being renamed or their header record comments field is being changed, the first 1- to 8-character name is sufficient. If a record within an object module is being renamed, record type and old record name also must be provided. If an alias phase name is being changed, record type and old alias phase name must be specified.

Positional Parameter 4:

`new-name`

Specifies the new name to be substituted for the old name. If renaming a multiphase load module, only the first six characters can be changed; the last two remain the same. If you are changing the sharability status of a module or the comments field of a header record, the new name is not necessary.

Positional Parameter 5:

`comments`

A string of up to 30 characters of identification information that is to be inserted into the header record of the identified module.

If omitted, current comments remain unchanged.

Examples:

```
   1        10    16
1.          REN.N D2,,EXAMPLE2,NEWEXAM2
2.          REN   ,0,EXAMPLE3,NEWEXAM3
3.          REN   D3,0,EXAMPLE4.SEXAMPLE5,NEWEXAM5
4.          REN.G D5,,EXAMPLE6,NEWEXAM6
```

1. Renames all modules named EXAMPLE2 in file D2 to NEWEXAM2. No listing of headers is provided. Any old modules named NEWEXAM2 will be deleted.

2. Renames the object module named EXAMPLE3 in the job run library to NEWEXAM3. If an object module named NEWEXAM3 already exists in the job run library, nullify that module.

3. Renames the CSECT named EXAMPLE5 in the object module named EXAMPLE4 to NEWEXAM5.

4. Renames the group named EXAMPLE6 in the file D5 to NEWEXAM6.

NOTE:

The REN control statement cannot be used if processing tape libraries.

2.8.25. Produce or Delete Control Statement Records within Object Module (REPRO) Control Statement

Function:

This command is used to produce and delete control statement records within object modules. The named object module is recopied onto the original file. Insertion or deletion of control statement records may occur either after the object module header record or after the object module transfer record.

If no deletion is required, new control statement records will be added after the control statement record already present in the named object module is copied.

An EOD control statement delimits control statement insertions. Those seen prior to the first EOD are inserted in the object module header set. Those seen following the first EOD control statement are inserted in the object module transfer set. Both EOD control statements are always required, regardless of the presence of any insertion or deletion.

Format:

LABEL	△OPERATION△	OPERAND
unused	REPRO[.options]	⎡⎧lfn⎫⎤,module-name[,#deletions][,#deletions] ⎣⎩$Y$RUN⎭⎦

Options:

D List entire module.

N Do not list header records.

P Punch module.

Positional Parameter 1:

`lfn`
Specifies the logical file name of the disk file on which the subject module is located.

If omitted, the job run library file ($Y$RUN) is used.

Positional Parameter 2:

`module-name`
Specifies the name of the object module to be modified.

Positional Parameter 3:

`#deletions`
A decimal value indicating the number of control statement records to be deleted that currently follow the object module header record. This value represents the number of control statements to be dropped from the control statement set following the object module header record. Records are dropped from the end of the set.

Positional Parameter 4:

`#deletions`
A decimal value indicating the number of control statement records to be deleted that currently follow the object module transfer record. This value represents the number of control statements to be dropped from the control statement set following the object module transfer record. Records are dropped from the end of the set.

Examples:

1. Add the source records INCLUDE A and INCLUDE B to the end of the control statement set following the module header record. No changes are made to the control statement set following the transfer record. List and punch the module modified.

```
1        10    16
____________________________________
         REPRO.DP D1,EXAMPLE1
         INCLUDE A
         INCLUDE B
         EOD
         EOD
```

2. Add the source records INCLUDE A and INCLUDE B to the end of the control statement set following the object module header record and the source record INCLUDE C to the end of the control statement set following the object module transfer record.

```
         REPRO D1, EXAMPLE
         INCLUDE A
         INCLUDE B
         EOD
         INCLUDE C
         EOD
```

3. Add the source record INCLUDE A to the end of the control statement set following the object module transfer record. List the modified module.

```
         REPRO.D D1,EXAMPLE
         EOD
         INCLUDE A
         EOD
```

4. Delete the last control statement currently following the object module header record and then add the source record INCLUDE A. Also, delete the last three control statement records currently following the object module transfer record and then add the source record INCLUDE B. List the module modified.

```
         REPRO.D D1,EXAMPLE,1,3
         INCLUDE A
         EOD
         INCLUDE B
         EOD
```

5. Add the source record INCLUDE A to the end of the control statement set following the object module header record. Delete the last three control statement records currently following the object module transfer record and then add the source record INCLUDE B.

```
1       10    16
--------------------------
        REPRO D1,EXAMPLE,,3
        INCLUDE A
        EOD
        INCLUDE B
        EOD
```

NOTE:

The REPRO control statement cannot be used if processing tape libraries.

2.8.26. Reset File Current Position Pointer (RES) Control Statement

Function:

This statement is used to reset the current position pointer in disk files to the beginning of file or, for tape files, to rewind the tape to load point. If an output tape file is being rewound, a tape mark will be written before rewinding. If a module name and type are specified, the current position pointer in disk or tape files is aimed at the first record of the named module. If a module of the name and type specified is not found, the current position pointer remains as it was before the RES statement was processed, and an appropriate diagnostic is printed on the map.

The current position of library files is maintained via a set of relative pointers in the respective disk or tape files being managed by the librarian. As each librarian command is processed, the current position of the file directory partition and the prime file partition are updated accordingly. Each executed function is essentially serial in fashion in that the referenced file is processed from wherever it was last positioned up to the module or group specified. The processing involved may be inclusive or exclusive, depending on the function and the selection of various options. If a referenced module or group is in a file, ahead of the current position, the user may choose to perform the RES function prior to performing the function in question. If no RES is submitted, the file will eventually wrap around from end-of-file to the initial position, and then to the requested module or group. If the module or group cannot be located within the named file, the search terminates at the point of origin established when the process began.

Format:

LABEL	△OPERATION△	OPERAND
unused	RES[.options]	$\left[\begin{Bmatrix}\texttt{lfn}\\ \texttt{\$Y\$RUN}\end{Bmatrix}\right]\left[,\begin{Bmatrix}\texttt{S}\\ \texttt{M}\\ \texttt{O}\\ \texttt{L}\end{Bmatrix}\right]$[,name]

Options:

G *Name* parameter is the name of a group. The file position pointer points to the first record of the first group encountered with the name specified.

Positional Parameter 1;

`lfn`

Specifies the logical file name of the disk or tape file to be reset.

If omitted, the job run library file ($Y$RUN) is reset.

Positional Parameter 2:

`S,M,O,L`

Identifies a module type as a program source module (S), macro/jproc source module (M), object module (O), or load module (L).

If omitted, it is assumed that the reset operation is directed to a file rather than to a module or module group.

Positional Parameter 3:

`name`

Specifies the name of the module or group to which the current position pointer is to be aimed.

If omitted and a module type is not specified, it is assumed that the reset operation is directed to a file. Otherwise, an error message is listed to indicate its omission.

Examples:

```
      1        10     16
1.             RES    D1
2.             RES    D3,O,EXAMPLE1
3.             RES    T1,S,EXAMPLE2
4.             RES    ,L,EXAMPLE3
```

1. Resets the current position pointer of file D1 to the file start.

2. Resets the current position pointer for file D3 to the first record of the object module named EXAMPLE1.

3. Resets the current position pointer for tape file T1 to the first record of the source module named EXAMPLE2.

4. Resets the current position pointer for the job run library to the first record of the load module named EXAMPLE3.

2.8.27. Add, Replace, or Check Sequence Numbers (SEQ) Control Statement

Function:

The sequence function is provided to permit source modules to be sequenced or resequenced. This function does not apply to object or load modules. This function also is supported as a subordinate command to the ELE and COR control statement. When using the SEQ control statement with a tape library, you must use the SEQ control statement as a subfunction control statement to the COR or ELE control statement.

When the SEQ control statement is used in conjunction with the ELE control statement, you can perform a sequence check on a source module being filed, sequence a source module being filed, or resequence a source module being filed. When this statement is used in conjunction with the COR control statement, you can correct a source module by using sequence numbers for control. An example of each of these uses is given in the examples portion of this statement description. When you use the SEQ as a subfunction to an ELE or COR control statement, the options (if specified) are disregarded.

Format:

LABEL	△OPERATION△	OPERAND
unused	SEQ[.options]	$\left[\begin{Bmatrix}\text{lfn}\\ \text{SYSRUN}\end{Bmatrix}\right],\begin{Bmatrix}\text{S}\\ \text{M}\end{Bmatrix}[\text{,name}]$ $\left[,\begin{Bmatrix}\text{column-position}\\ \text{73}\end{Bmatrix}\right]\left[,\begin{Bmatrix}\text{content}\\ \text{SAME}\\ \text{00000000}\end{Bmatrix}\right]$ $\left[,\begin{Bmatrix}\text{increment}\\ \text{1}\end{Bmatrix}\right]$

Options:

D List sequenced module.

N Do not list header records.

P Punch sequenced module.

Positional Parameter 1:

`lfn`

Specifies logical file name of the disk file in which the source module to be sequenced or resequenced resides.

If omitted, the job run library ($Y$RUN) is assumed to contain the module to be sequenced or resequenced. If used with the ELE or COR statement, it must match the lfn in that statement.

Positional Parameter 2:

`S,M`

Specifies the type of module being sequenced as either a program source module (S) or a macro/jproc source module (M).

Positional Parameter 3:

`name`

Identifies the name of the source module to be sequenced or resequenced. This parameter is required when the SEQ control statement is being used to sequence or resequence a source module. If the SEQ control statement immediately follows a COR or ELE control statement, then the SEQ control statement is used to resequence the source module as it is corrected or sequenced as it is added, respectively. In this case, the name must match the name specified in the COR or ELE statement.

If omitted:

- and the SEQ control statement immediately follows an ELE control statement, the SEQ control statement can be used to check the sequence of a source module being filed. (See coding example 5.)

- and the SEQ control statement immediately follows a COR control statement, the SEQ control statement can be used to identify a sequence field, in the source module being corrected, that is to be used to insert corrections. (See coding example 6.)

Positional Parameter 4:

`column-position`

Specifies the first column position in the source module where the sequence field begins and where the sequence data is incorporated. A sequence number eight characters in length and beginning in column 73 is referred to as a standard sequence number.

If omitted, column 73 is assumed to be the first column of the sequenced field.

Positional Parameter 5:

`content`
A 1- to 8-character value that specifies the initial value to be placed into the sequence field of the first record in the module. The length of this value determines the length of the sequence field. The mixing of sequence numbers with alphabetic and numeric characters is permitted, provided the alphabetic and numeric string remain intact and the alphabetic characters are left-justified. For example, MA400 is a valid sequence number but M4A00 is not.

`SAME`
Indicates that the content of the sequence field of this first record of the module being resequenced is to remain as it was. This specification assumes that this field occupies eight character positions. If it does not, this parameter should not be specified. Instead, the initial sequence field content should be respecified.

If omitted, the initial sequence field contents is assumed to be 00000000 (eight zeros).

Positional Parameter 6:

`increment`
A decimal number, not to exceed 255, that specifies the sequence increment to be used in the sequencing process.

If omitted, the increment is assumed to be 1.

Examples:

```
     1        10     16                                                   72
    ------------------------------------------------------------------------
1.  |         SEQ.DP D14,S,EXAMPLE1,20,LNK000,10
2.  |         SEQ.N  D12,S,EXAMPLE2,,SAME
3.  |         ELE    D5,S,BALSORC
    |         SEQ    D5,S,BALSORC,,BAL00000,10
    |-SOURCE MODULE  CARD DECK-
    |         EOD
4.  |         ELE    D6,S,COBSORC
    |         SEQ    D6,S,COBSORC,1,COB00001
    |-SOURCE MODULE  CARD DECK-
    |         EOD
5.  |         ELE    D7,S,BALSORC
    |         SEQ    D7,S,,,SRC0000,1
    |-SOURCE MODULE  CARD DECK-
    |         EOD
6.  |         COR    D0,S,TESTEXAM
    |         SEQ    D0,S,,1,SRC00000
    |-SOURCE MODULE  CORRECTION CARD DECK
    |         EOD
```

1. Causes the source module EXAMPLE1 in file D14 to be sequenced in column positions 20–25; the initial content (sequence number) is to be LNK000; the increment is 10. The sequenced module is to be punched and listed.

2. Causes the source module EXAMPLE2 in the file D12 to be resequenced in column positions 73–80; the initial content is unchanged; the increment is 1. No listing of headers is to be provided.

3. Causes the source module named BALSORC to be added to file D5 and sequenced (or resequenced if the card file already has sequence numbers in it) in columns 73 through 80 with the initial value BAL00000, and each succeeding record to be incremented by a count of 10.

4. Causes the source module named COBSORC to be added to file D6 and sequenced (or resequenced if the card file already has sequence numbers in it) in columns 1 through 8 with an initial value of COB00001, and each succeeding record to be incremented, by default, by a count of 1.

5. Causes the source module BALSORC to be added to file D7 and its sequence numbers checked for agreement with the column position (73 through 79), content (SRC0000), and increment (1) specifications of the SEQ control statement.

6. Causes the source module TESTEXAM in file D0 to be corrected in accordance with the source records contained in the correction card deck. The sequence number field in the source module TESTEXAM that is being keyed on to incorporate the source module corrections begins in column 1 and has a length of eight column positions.

2.8.28. Skip Source Module Records (SKI) Control Statement

Function:

The SKI control statement is used only in conjunction with the COR control statement to make source module corrections. The SKI statement allows one or more original source module records to be bypassed by the COR function.

When a SKI control statement is processed, records are read from the old data set and written into the new data set until a sequence number is detected that matches the sequence number in the sequence field of the SKI command. The skip operation is then initiated and continues until a sequence number that matches the operand field of the command is detected. If the sequence field of the SKI control statement is blank, the skip operation is initiated immediately.

Format:

LABEL	△OPERATION△	OPERAND	SEQUENCE
	SKI[.options]	last-sequence-no	[starting-sequence-no]

Options:

D List the records skipped.

Positional Parameter 1:

`last-sequence-no`

Is a 1- to 8-character alphanumeric string that identifies the sequence number of the last source module record to be bypassed.

Sequence Field Parameter:

`starting-sequence-no`

Is a 1- to 8-character alphanumeric string that identifies the sequence number of the first source module record to be bypassed. This field begins in column 73 unless a SEQ control statement dictates otherwise.

If omitted, the skip operation is initiated immediately, starting with the source module record that immediately follows the last source module record operated on by the COR function.

Examples:

See examples under recycle source module current position pointer (REC) control statement.

2.9. LIBRARIAN CANNED JOB CONTROL STREAMS

The following librarian canned job control streams provide you with a more convenient method of performing certain library functions without having to punch the parameters and job control statements normally required to run them. These functions reside in the system load library file ($Y$LOD), and their corresponding job control streams reside in the system job control stream library file ($Y$JCS). The functions are initiated from the system console by keying in their associated job control stream name.

Table 2-3 shows the job names associated with the functions performed.

Table 2-3. Librarian Canned Job Control Streams

Job Name	Function
DRDP	Prints directory partition of a librarian disk file
LISTRES	Prints directory for SYSRES modules
MODLST	Lists the contents of the system libraries
PACKRES	Compresses all modules on SYSRES and prints diectory of compressed modules

2.9.1. Print Library Directory Partition (DRDP)

You can display the directory partition of any librarian disk file. A canned job control stream is provided that initiates the display for most files. Key in the following command on the system console to initiate the job.

```
RV DRDP,,V=vsn,L=file-identifier
```

The V (volume) keyword specifies the volume serial number of the volume containing the file. The L (label) keyword is the file identifier of that file; the maximum length is 11 characters for this keyin. For those files with a file identifier of more than 11 characters specified on the LBL job control statement, the following job control stream must be used.

```
1          10     16
--------------------------------------------------
// JOB     DRDP
// DVC     20 // LFD PRNTR
// DVC     50 // VOL vsn // LBL file-identifier
// LFD     LU$DTFI
// OPTION JOBDUMP
// EXEC    SULBD
// PARAM file-identifier
/&
// FIN
```

2.9.2. Print Directory for SYSRES Modules (LISTRES)

The LISTRES job control stream prints the directory for all the modules residing on your SYSRES pack or just the modules contained in a particular file on your release volume, depending on how you key in the RUN command. The format of the RUN command used to call LISTRES is:

```
RV LISTRES[,[,F=file-name][,V=vsn]]
```

The F parameter specifies the file names of all the modules on your SYSRES pack to be printed. If more than one file is specified, then the file names must be enclosed by parentheses and separated by commas.

If the F parameter is omitted, all modules contained in all system files are printed.

The V parameter specifies the volume serial number of your release volume.

The LISTRES supported file names are:

LOD
OBJ
MAC
SRC
JCS
SG$JCS
SG$LOD
SG$OBJ
SG$MAC
SMC
SCLOD
MIC
SAVE
DLG
FMT
HELP
SHR
SDF
IVP
MSG

The following is an example of a typical LISTRES keyin:

```
RV LISTRES,,F=(LOD,JCS,SRC)
```

Here, the system load, system job control stream, and system source files are listed.

NOTE:

LISTRES will not list the contents of any volume that is not a release volume.

2.9.3. List the Contents of the Release Volume System Libraries (MODLST)

MODLST lists the modules and macros in five system libraries ($Y$SRC, $Y$OBJ, $Y$LOD, $Y$JCS, and $Y$MAC). Each module and macro is given in alphanumeric sequence and is accompanied by a description of its function and its size. To run MODLST, key in the following command from the system console:

```
RV MODLST[,,VSN=vol-ser-no]
```

where:

`VSN=vol-ser-no`
Specifies an optional work disk. MODLST uses 30 cylinders on this disk for its work space. If you don't specify this option, the work space for the job is allocated on the disk containing $Y$RUN.

2.9.4. Pack SYSRES Modules and Print Directories (PACKRES)

The PACKRES job control stream packs and prints the directories of all modules residing on your release volume. The format of the RUN command to call PACKRES is:

```
RV PACKRES[,[,F=file-name][,V=vsn]]
```

The F parameter specifies the file names (Table 2-5) of all the files to be packed and printed. You can list the file names in any order. If the F parameter is omitted, then all files on your release volume are packed and printed.

The V parameter specifies the volume serial number of your release volume.

The PACKRES supported file names are:

LOD
OBJ
MAC
SRC
JCS
SG$JCS
SG$LOD
SG$OBJ
SG$MAC
SMC
SCLOD
MIC
SAVE
DLSG
FMT
HELP
SHR
SDF
IVP
MSG

The following is an example of a typical PACKRES operation:

```
    RV PACKRES,,F=(OBJ,SRC,LOD)
```

Here, the system object, source, and load files are packed and printed.

NOTE:

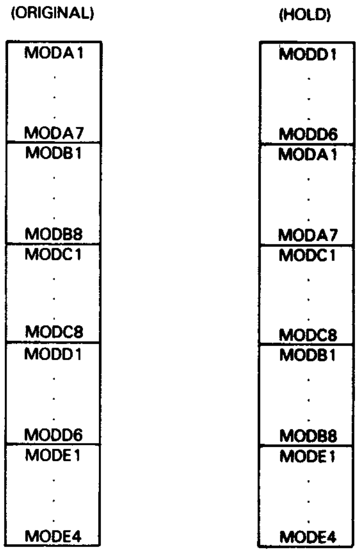
PACKRES assumes that a certain set of files exists on all release volumes. If your volume is not a release volume, open errors may occur if certain files are not present.

2.10. PROGRAMMING EXAMPLES

Some typical examples of librarian jobs follow. These jobs are illustrated as a function of the job control stream used to execute the librarian and the librarian maps produced for each job.

2.10.1. Repositioning Modules in a Disk Library File

This job rearranges modules in a disk file. it copies modules from the original file into a new file in the new sequence, as listed in the following job control stream. Names such as MODA1 and MODA7 are of the first and last modules in each series of consecutive modules that are copied with each COP statement. After all the modules are copied to the new file, the original file is scratched and the new file is renamed as the original. Figure 2–13 illustrates the librarian map for this job.



Job Control Stream:

```
//JOB SHUFFLE
// DVC 2Ø  // LFD PRNTR
// DVC 5Ø  // VOL DØØ41Ø
// LBL ORIGINAL  // LFD RG
// DVC 5Ø  // VOL DØØ41Ø
// EXT ST,,1,BLK,(256,4ØØØ)
// LBL HOLD  // LFD HD
// EXEC LIBS
/$
         FIL   D1=RG,D2=HD
         COP   D1
         RES   D1,S,MODD1
         COP.U D1,S,MODD6,D2
         RES   D1,S,MODA1
         COP.U D1,O,MODA7,D2
         RES   D1,S,MODC1
         COP.U D1,L,MODC8,D2
         RES   D1,S,MODB1
         COP.U D1,L,MODB8,D2
         RES   D1,S,MODE1
         COP.U D1,S,MODE4,D2
         COP   D2
/*
// SKIP END,11111111
// SCR RG
// REN HD,ORIGINAL
//END NOP
/&
```

1. Identifies the job.

2. Assigns a printer to the job.

3. Identifies logical unit number 50 for disk volume D00410, which contains the original file.

4. Declares file name ORIGINAL and logical file descriptor RG for the original file.

5. Identifies logical unit number 50 for disk volume D00410 to be used for the new file.

6. Allocates file space for the new file.

7. Declares file name HOLD and logical file descriptor HD for the new file.

8. Initiates execution of the librarian.

9. Indicates the start of the librarian control statements.

10. Assigns a type code and a logical file number to the files in the job. Thus, in the control statements that follow, D1 refers to ORIGINAL and D2 refers to HOLD.

11. Prints a sequential list of the modules in file ORIGINAL before the modules are rearranged.

12 through 21.
Copy modules from file ORIGINAL to file HOLD, moving a series of consecutive modules at a time. Each RES statement sets the pointer in file ORIGINAL to the first module in the series. Then a COP statement copies all modules from the pointer to the module named in the COP statement.

22. Prints a sequential list of the modules in file HOLD.

23. Identifies the end of the librarian control statements.

24 and 27.
Skip the scratch and rename operation if any errors occur in the job.

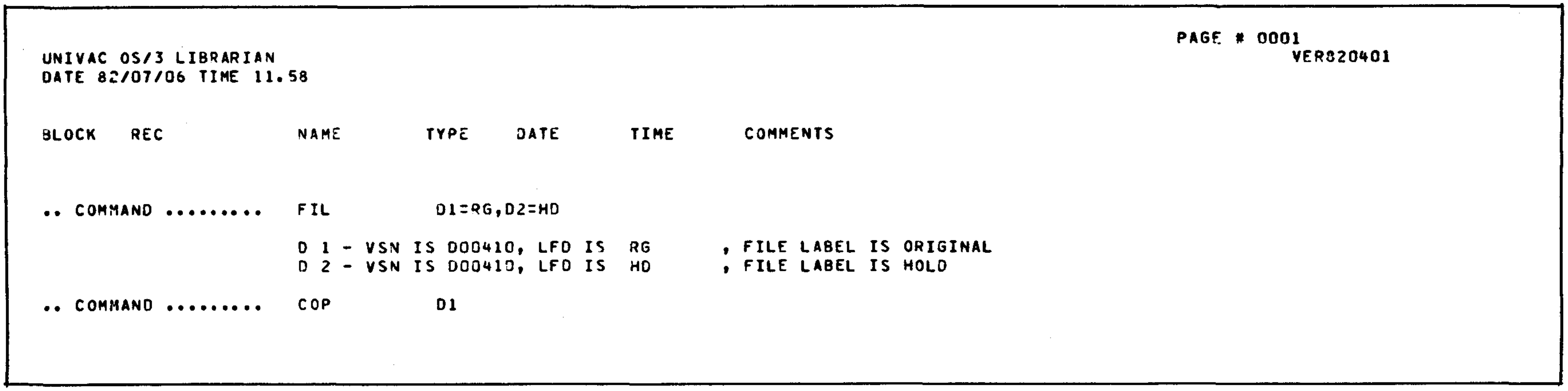
25. Scratches file ORIGINAL.

26. Renames file HOLD to ORIGINAL.

28. Indicates the end of job.

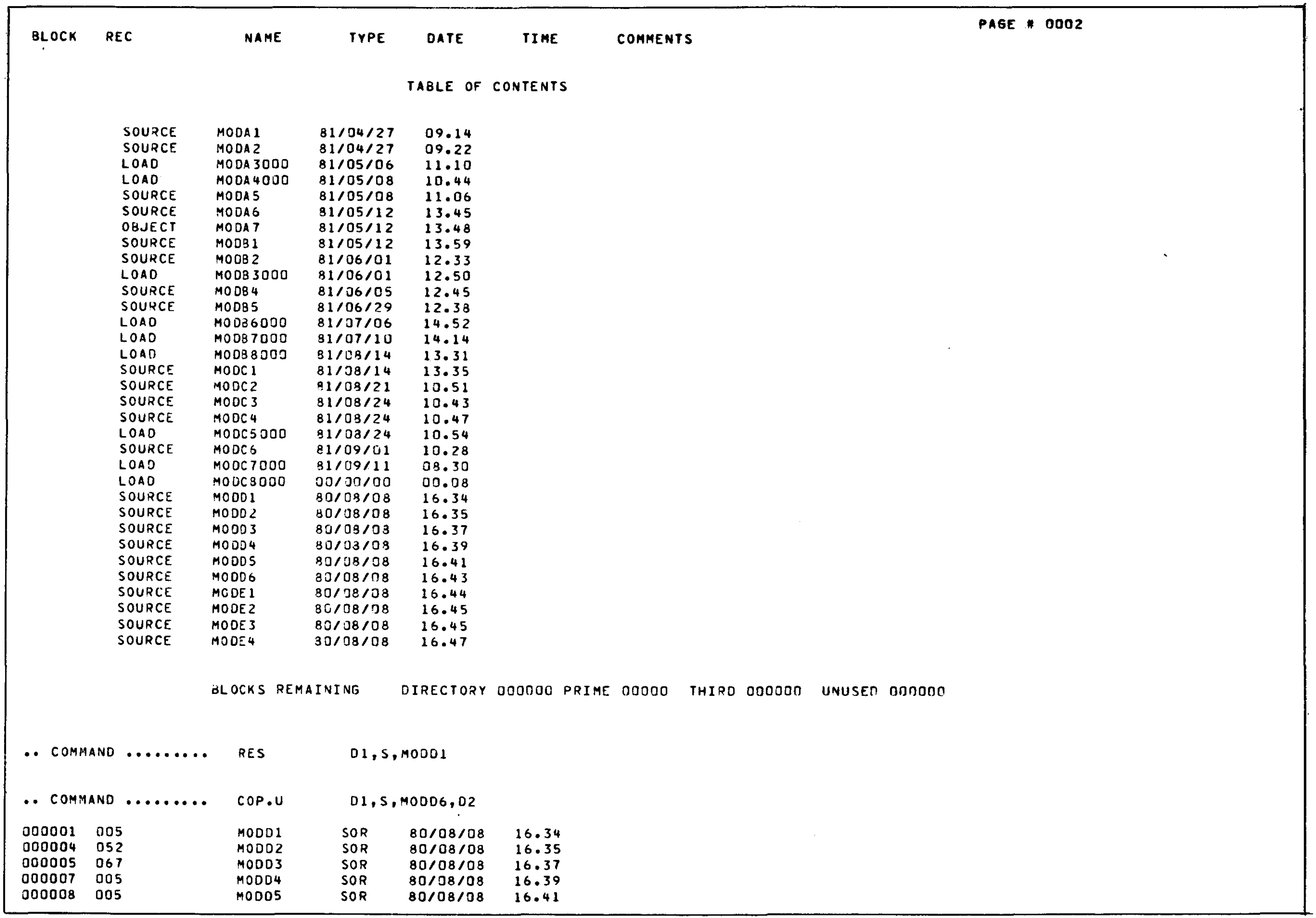
NOTE:

To save both the original file and the new file, omit lines 24 through 27.



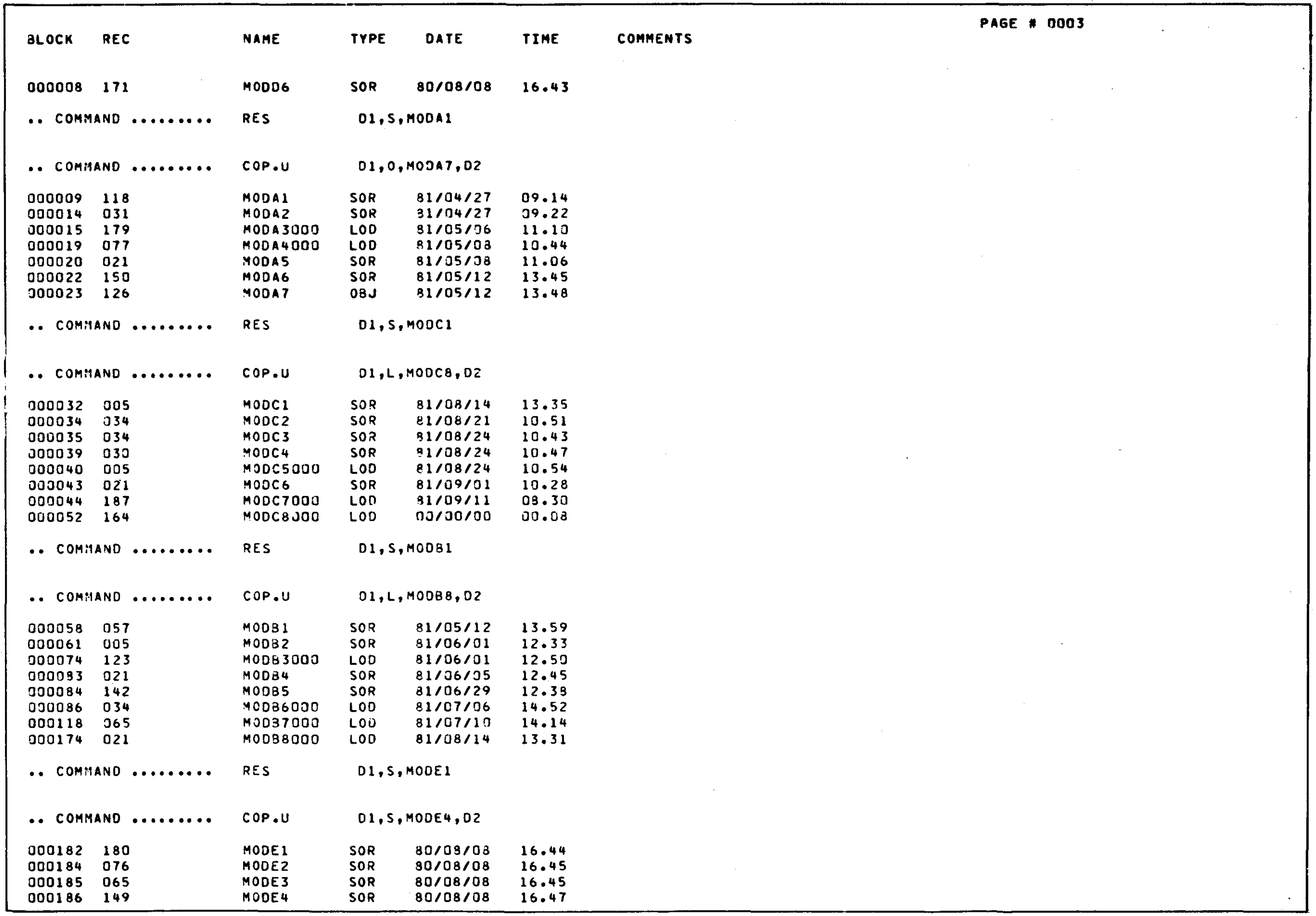
```
                                                                                            PAGE # 0001
UNIVAC OS/3 LIBRARIAN                                                                               VER820401
DATE 82/07/06 TIME 11.58

BLOCK   REC              NAME        TYPE    DATE       TIME       COMMENTS


.. COMMAND .........     FIL          D1=RG,D2=HD

                         D 1 - VSN IS 000410, LFD IS  RG        , FILE LABEL IS ORIGINAL
                         D 2 - VSN IS 000410, LFD IS  HD        , FILE LABEL IS HOLD

.. COMMAND .........     COP          D1
```

Figure 2–13. Librarian Map for Repositioning Modules (Part 1 of 5)



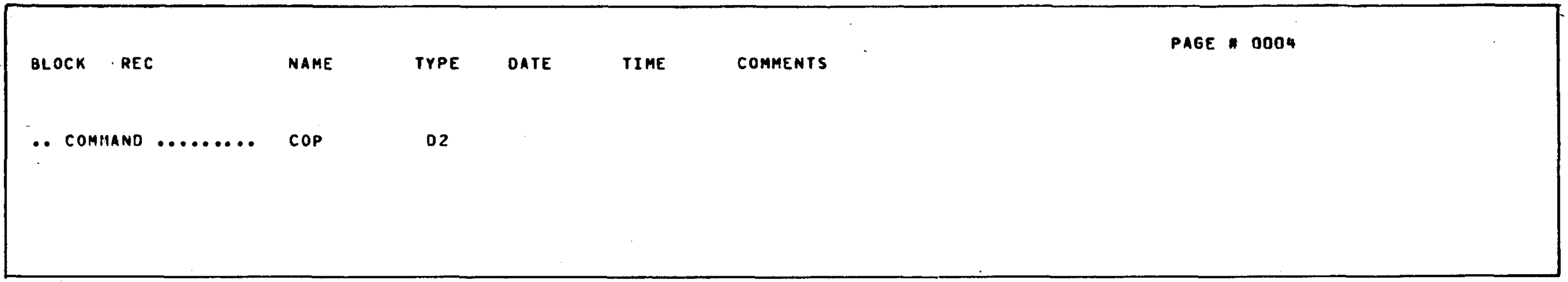
```
                                                                                          PAGE # 0002
BLOCK   REC              NAME       TYPE     DATE       TIME       COMMENTS

                                          TABLE OF CONTENTS

           SOURCE     MODA1        81/04/27    09.14
           SOURCE     MODA2        81/04/27    09.22
           LOAD       MODA3000     81/05/06    11.10
           LOAD       MODA4000     81/05/08    10.44
           SOURCE     MODA5        81/05/08    11.06
           SOURCE     MODA6        81/05/12    13.45
           OBJECT     MODA7        81/05/12    13.48
           SOURCE     MODB1        81/05/12    13.59
           SOURCE     MODB2        81/06/01    12.33
           LOAD       MODB3000     81/06/01    12.50
           SOURCE     MODB4        81/06/05    12.45
           SOURCE     MODB5        81/06/29    12.38
           LOAD       MODB6000     81/07/06    14.52
           LOAD       MODB7000     81/07/10    14.14
           LOAD       MODB8000     81/08/14    13.31
           SOURCE     MODC1        81/08/14    13.35
           SOURCE     MODC2        81/08/21    10.51
           SOURCE     MODC3        81/08/24    10.43
           SOURCE     MODC4        81/08/24    10.47
           LOAD       MODC5000     81/08/24    10.54
           SOURCE     MODC6        81/09/01    10.28
           LOAD       MODC7000     81/09/11    08.30
           LOAD       MODC8000     00/00/00    00.08
           SOURCE     MODD1        80/08/08    16.34
           SOURCE     MODD2        80/08/08    16.35
           SOURCE     MODD3        80/08/08    16.37
           SOURCE     MODD4        80/08/08    16.39
           SOURCE     MODD5        80/08/08    16.41
           SOURCE     MODD6        80/08/08    16.43
           SOURCE     MODE1        80/08/08    16.44
           SOURCE     MODE2        80/08/08    16.45
           SOURCE     MODE3        80/08/08    16.45
           SOURCE     MODE4        80/08/08    16.47

                BLOCKS REMAINING    DIRECTORY 000000 PRIME 00000  THIRD 000000  UNUSED 000000


.. COMMAND .........    RES        D1,S,MODD1

.. COMMAND .........    COP.U      D1,S,MODD6,D2

000001  005             MODD1     SOR     80/08/08     16.34
000004  052             MODD2     SOR     80/08/08     16.35
000005  067             MODD3     SOR     80/08/08     16.37
000007  005             MODD4     SOR     80/08/08     16.39
000008  005             MODD5     SOR     80/08/08     16.41
```

Figure 2–13. Librarian Map for Repositioning Modules (Part 2 of 5)



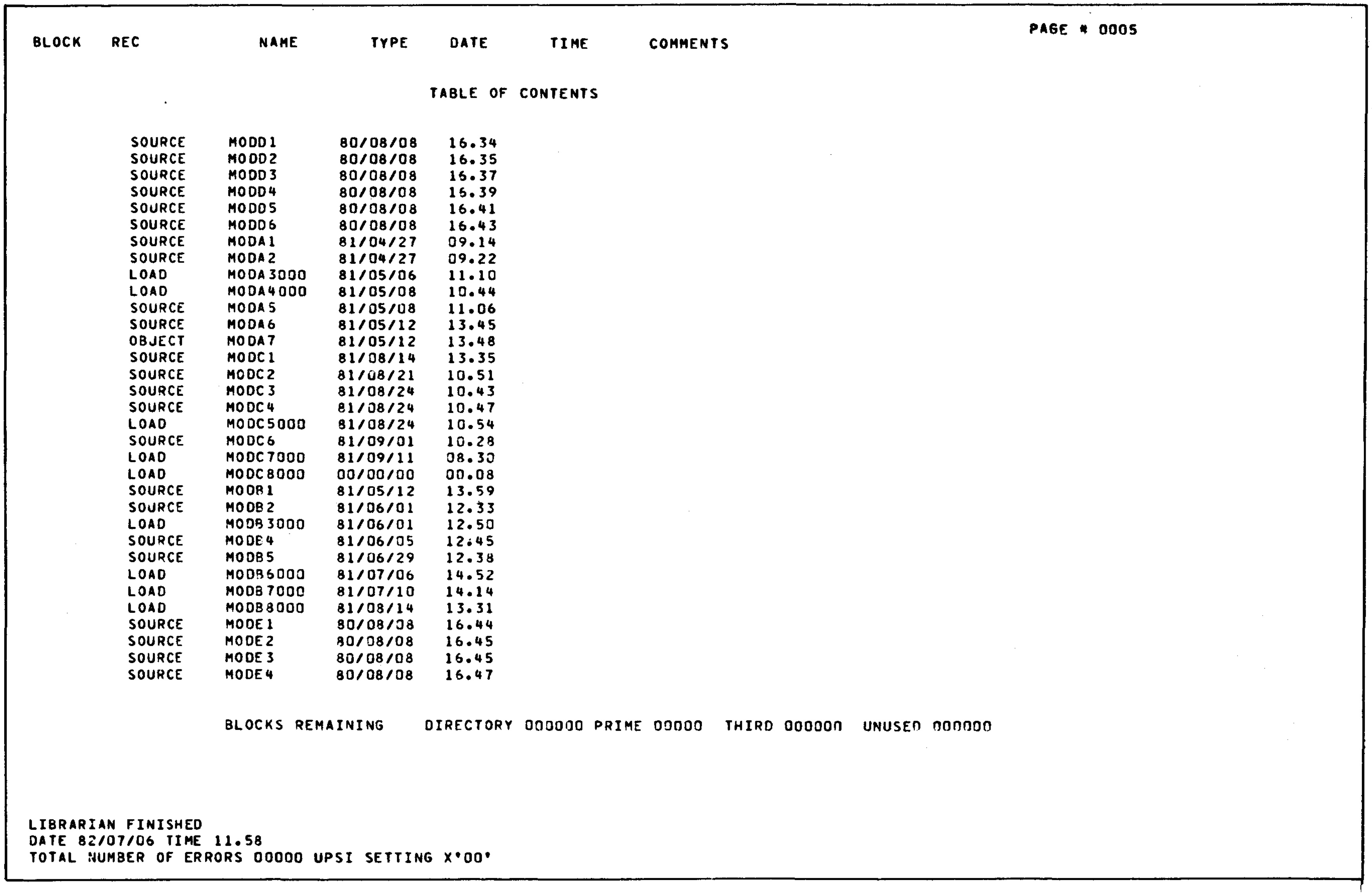
```
                                                                                   PAGE # 0003
BLOCK   REC            NAME        TYPE    DATE        TIME       COMMENTS

000008  171            MODD6       SOR     80/08/08    16.43

.. COMMAND .........   RES          D1,S,MODA1


.. COMMAND .........   COP.U        D1,O,MODA7,D2

000009  118            MODA1       SOR     81/04/27    09.14
000014  031            MODA2       SOR     81/04/27    09.22
000015  179            MODA3000    LOD     81/05/06    11.10
000019  077            MODA4000    LOD     81/05/08    10.44
000020  021            MODA5       SOR     81/05/08    11.06
000022  150            MODA6       SOR     81/05/12    13.45
000023  126            MODA7       OBJ     81/05/12    13.48

.. COMMAND .........   RES          D1,S,MODC1


.. COMMAND .........   COP.U        D1,L,MODC8,D2

000032  005            MODC1       SOR     81/08/14    13.35
000034  034            MODC2       SOR     81/08/21    10.51
000035  034            MODC3       SOR     81/08/24    10.43
000039  030            MODC4       SOR     81/08/24    10.47
000040  005            MODC5000    LOD     81/08/24    10.54
000043  021            MODC6       SOR     81/09/01    10.28
000044  187            MODC7000    LOD     81/09/11    08.30
000052  164            MODC8000    LOD     00/00/00    00.08

.. COMMAND .........   RES          D1,S,MODB1


.. COMMAND .........   COP.U        D1,L,MODB8,D2

000058  057            MODB1       SOR     81/05/12    13.59
000061  005            MODB2       SOR     81/06/01    12.33
000074  123            MODB3000    LOD     81/06/01    12.50
000083  021            MODB4       SOR     81/06/05    12.45
000084  142            MODB5       SOR     81/06/29    12.38
000086  034            MODB6000    LOD     81/07/06    14.52
000118  065            MODB7000    LOD     81/07/10    14.14
000174  021            MODB8000    LOD     81/08/14    13.31

.. COMMAND .........   RES          D1,S,MODE1


.. COMMAND .........   COP.U        D1,S,MODE4,D2

000182  180            MODE1       SOR     80/08/08    16.44
000184  076            MODE2       SOR     80/08/08    16.45
000185  065            MODE3       SOR     80/08/08    16.45
000186  149            MODE4       SOR     80/08/08    16.47
```

Figure 2–13. Librarian Map for Repositioning Modules (Part 3 of 5)



```
                                                                                PAGE # 0004
BLOCK  REC               NAME       TYPE    DATE      TIME      COMMENTS


.. COMMAND .........     COP         02
```

Figure 2–13. Librarian Map for Repositioning Modules (Part 4 of 5)



```
                                                                                  PAGE # 0005
BLOCK   REC         NAME      TYPE    DATE      TIME      COMMENTS

                                    TABLE OF CONTENTS

          SOURCE    MODD1       80/08/08   16.34
          SOURCE    MODD2       80/08/08   16.35
          SOURCE    MODD3       80/08/08   16.37
          SOURCE    MODD4       80/08/08   16.39
          SOURCE    MODD5       80/08/08   16.41
          SOURCE    MODD6       80/08/08   16.43
          SOURCE    MODA1       81/04/27   09.14
          SOURCE    MODA2       81/04/27   09.22
          LOAD      MODA3000    81/05/06   11.10
          LOAD      MODA4000    81/05/08   10.44
          SOURCE    MODA5       81/05/08   11.06
          SOURCE    MODA6       81/05/12   13.45
          OBJECT    MODA7       81/05/12   13.48
          SOURCE    MODC1       81/08/14   13.35
          SOURCE    MODC2       81/08/21   10.51
          SOURCE    MODC3       81/08/24   10.43
          SOURCE    MODC4       81/08/24   10.47
          LOAD      MODC5000    81/08/24   10.54
          SOURCE    MODC6       81/09/01   10.28
          LOAD      MODC7000    81/09/11   08.30
          LOAD      MODC8000    00/00/00   00.08
          SOURCE    MODB1       81/05/12   13.59
          SOURCE    MODB2       81/06/01   12.33
          LOAD      MODB3000    81/06/01   12.50
          SOURCE    MODB4       81/06/05   12.45
          SOURCE    MODB5       81/06/29   12.38
          LOAD      MODB6000    81/07/06   14.52
          LOAD      MODB7000    81/07/10   14.14
          LOAD      MODB8000    81/08/14   13.31
          SOURCE    MODE1       80/08/08   16.44
          SOURCE    MODE2       80/08/08   16.45
          SOURCE    MODE3       80/08/08   16.45
          SOURCE    MODE4       80/08/08   16.47

                    BLOCKS REMAINING    DIRECTORY 000000 PRIME 00000  THIRD 000000  UNUSED 000000



LIBRARIAN FINISHED
DATE 82/07/06 TIME 11.58
TOTAL NUMBER OF ERRORS 00000 UPSI SETTING X'00'
```

Figure 2–13. Librarian Map for Repositioning Modules (Part 5 of 5)

2.10.2. Sorting Modules into Separate Files by Type

This job sorts modules from one file into separate files for each module type. This job was run interactively, so the files for each module type were allocated with an interactive services ALLOCATE command before the job was run. Figure 2–14 illustrates the librarian map for this job. Notice that the last COP statement caused an error because there were no macro/jproc source modules in the original file. However, the job terminated normally.

Job Control Stream:

```
1.  // JOB TYPSRT
2.  // DVC 2Ø  // LFD PRNTR
3.  // DVC 5Ø  // VOL DØØ41Ø
4.  // LBL ORIGINAL  // LFD RG
5.  // DVC 5Ø  // VOL DØØ41Ø
6.  // LBL ALLSRC  // LFD SC
7.  // DVC 5Ø   // VOL DØØ41Ø
8.  // LBL ALLOBJ  // LFD OB
9.  // DVC 5Ø   // VOL DØØ41Ø
1Ø. // LBL ALLLOD  // LFD LD
11. // DVC 5Ø  // VOL DØØ41Ø
12. // LBL ALLMAC  // LFD MC
13. // EXEC LIBS
14. /$
15.           FIL  D1=RG, D2=SC,D3=OB,D4=LD,D5=MC
16.           COP  D1
17.           COP  D1,S,,D2
18.           COP  D1,O,,D3
19.           COP  D1,L,,D4
2Ø.           COP  D1,M,,D5
21. /*
22. /&
```

1. Identifies the job.

2. Assigns a printer to the job.

3. Identifies logical unit number 50 for disk volume D00410, which contains the original file.

4. Declares file ORIGINAL and logical file descriptor RG for the original file.

5 through 12.
Are device assignment statements for the files for each module type. All are on disk volume D00410 with logical unit number 50. The files are named ALLSRC, ALLOBJ, ALLLOD, and ALLMAC. They are assigned logical file descriptors SC, OB, LD, and MC, respectively.

13. Initiates execution of the librarian.

14. Indicates the start of the librarian control statements.

15. Assigns a type code and logical file number to the five files used in the job. Thus, in the control statements that follow, D1 refers to ORIGINAL, D2 to ALLSCR, D3 to ALLOBJ, D4 to ALLLOD, and D5 to ALLMAC.

16. Prints a table of contents of all modules in file ORIGINAL.

17. Copies all source modules from file ORIGINAL to file ALLSCR and prints a list of all of the modules copied.

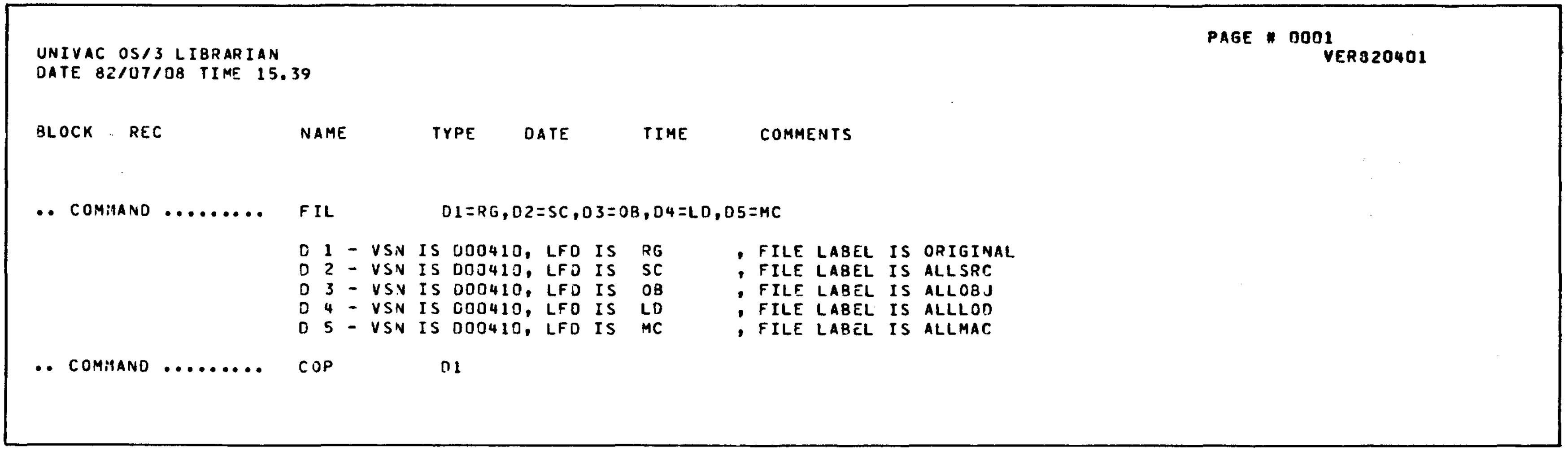
18. Copies all object modules from file ORIGINAL to file ALLOBJ and prints a list of all of the modules copied.

19. Copies all load modules from file ORIGINAL to file ALLLOD and prints a list of all of the modules copied.

20. Copies all macro/jproc modules from file ORIGINAL to file ALLMAC and prints a list of all of the modules copied.

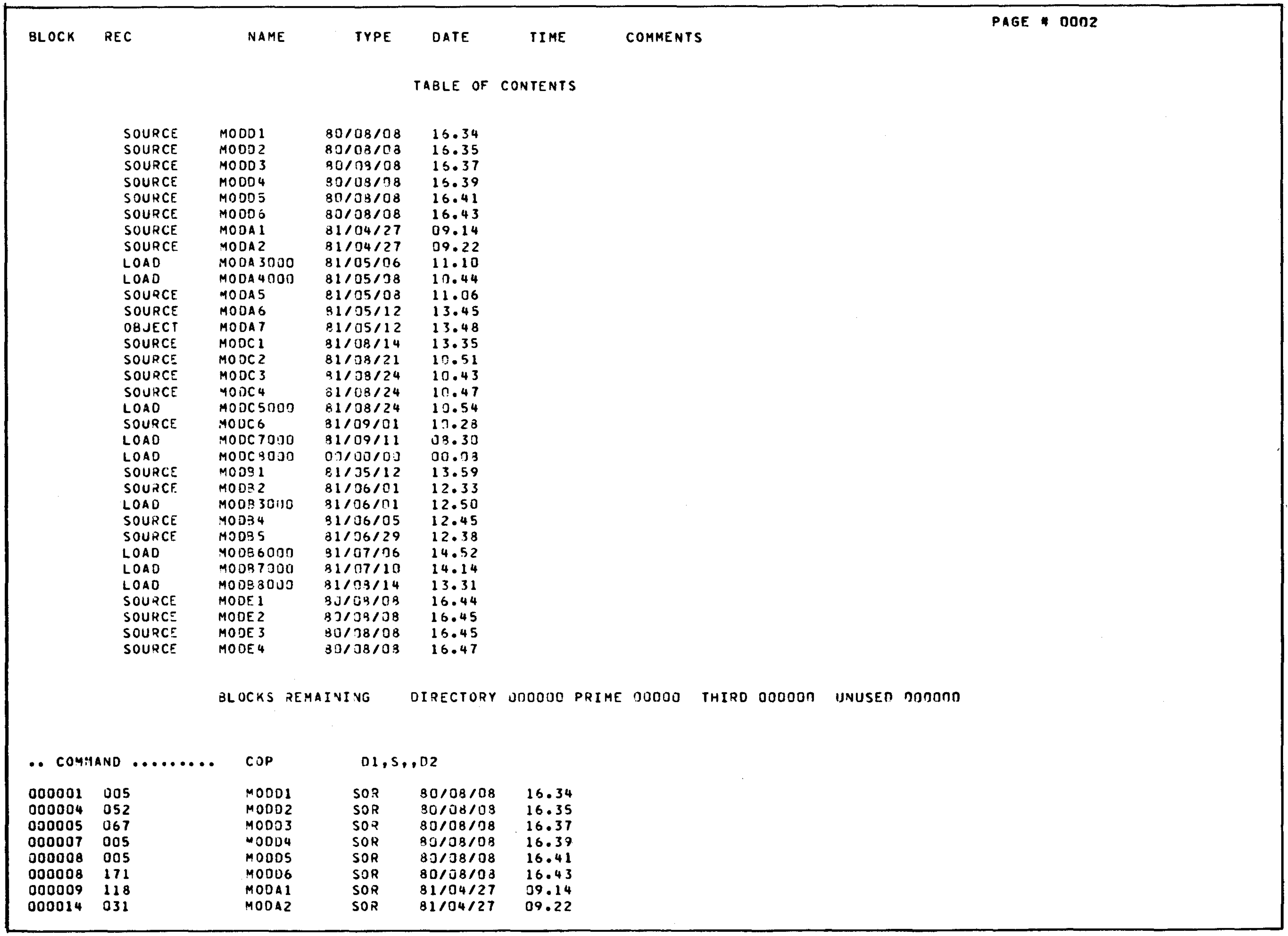
21. Indicates the end of the librarian control statements.

22. Indicates the end of job.



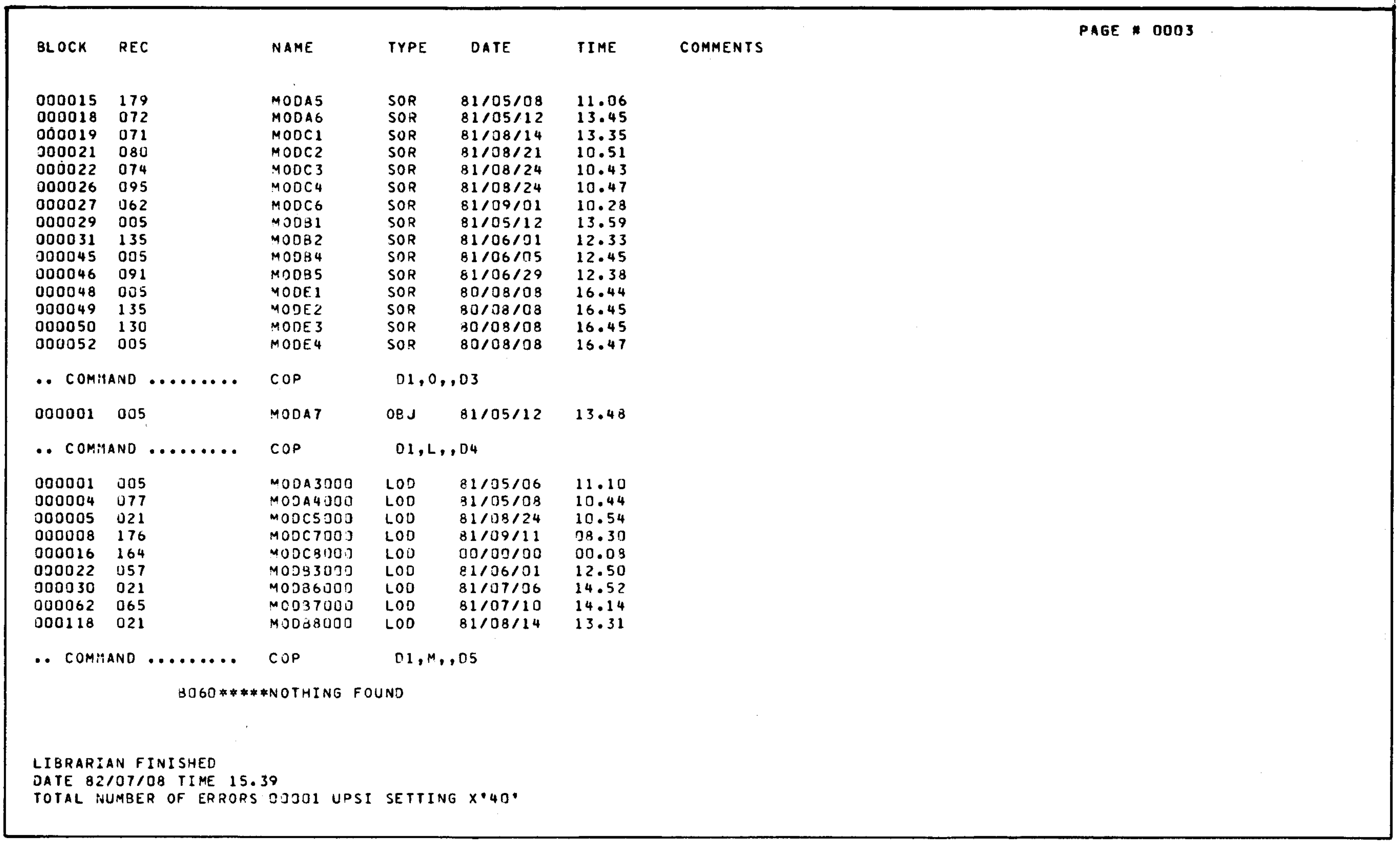
```
                                                                                PAGE # 0001
UNIVAC OS/3 LIBRARIAN                                                                  VER820401
DATE 82/07/08 TIME 15.39


BLOCK   REC              NAME       TYPE    DATE       TIME       COMMENTS


.. COMMAND .........     FIL         D1=RG,D2=SC,D3=OB,D4=LD,D5=MC

                         D 1 - VSN IS 000410, LFD IS  RG       , FILE LABEL IS ORIGINAL
                         D 2 - VSN IS 000410, LFD IS  SC       , FILE LABEL IS ALLSRC
                         D 3 - VSN IS 000410, LFD IS  OB       , FILE LABEL IS ALLOBJ
                         D 4 - VSN IS 000410, LFD IS  LD       , FILE LABEL IS ALLLOD
                         D 5 - VSN IS 000410, LFD IS  MC       , FILE LABEL IS ALLMAC

.. COMMAND .........     COP         01
```

Figure 2-14. Librarian Map for Sorting Modules by Type (Part 1 of 3)



```
                                                                                              PAGE # 0002
BLOCK   REC         NAME        TYPE    DATE       TIME      COMMENTS

                                           TABLE OF CONTENTS

          SOURCE    MODD1       80/08/08   16.34
          SOURCE    MODD2       80/08/08   16.35
          SOURCE    MODD3       80/08/08   16.37
          SOURCE    MODD4       80/08/08   16.39
          SOURCE    MODD5       80/08/08   16.41
          SOURCE    MODD6       80/08/08   16.43
          SOURCE    MODA1       81/04/27   09.14
          SOURCE    MODA2       81/04/27   09.22
          LOAD      MODA3000    81/05/06   11.10
          LOAD      MODA4000    81/05/08   10.44
          SOURCE    MODA5       81/05/08   11.06
          SOURCE    MODA6       81/05/12   13.45
          OBJECT    MODA7       81/05/12   13.48
          SOURCE    MODC1       81/08/14   13.35
          SOURCE    MODC2       81/08/21   10.51
          SOURCE    MODC3       81/08/24   10.43
          SOURCE    MODC4       81/08/24   10.47
          LOAD      MODC5000    81/08/24   10.54
          SOURCE    MODC6       81/09/01   10.28
          LOAD      MODC7000    81/09/11   08.30
          LOAD      MODC8000    00/00/00   00.08
          SOURCE    MODB1       81/05/12   13.59
          SOURCE    MODB2       81/06/01   12.33
          LOAD      MODB3000    81/06/01   12.50
          SOURCE    MODB4       81/06/05   12.45
          SOURCE    MODB5       81/06/29   12.38
          LOAD      MODB6000    81/07/06   14.52
          LOAD      MODB7000    81/07/10   14.14
          LOAD      MODB8000    81/08/14   13.31
          SOURCE    MODE1       80/08/08   16.44
          SOURCE    MODE2       80/08/08   16.45
          SOURCE    MODE3       80/08/08   16.45
          SOURCE    MODE4       80/08/08   16.47

                    BLOCKS REMAINING    DIRECTORY 000000 PRIME 00000  THIRD 000000  UNUSED 000000


.. COMMAND .........    COP         D1,S,,D2

000001  005             MODD1       SOR    80/08/08    16.34
000004  052             MODD2       SOR    80/08/08    16.35
000005  067             MODD3       SOR    80/08/08    16.37
000007  005             MODD4       SOR    80/08/08    16.39
000008  005             MODD5       SOR    80/08/08    16.41
000008  171             MODD6       SOR    80/08/08    16.43
000009  118             MODA1       SOR    81/04/27    09.14
000014  031             MODA2       SOR    81/04/27    09.22
```

Figure 2-14. Librarian Map for Sorting Modules by Type (Part 2 of 3)



```
                                                                                     PAGE # 0003
BLOCK   REC              NAME       TYPE    DATE       TIME       COMMENTS


000015  179              MODA5      SOR    81/05/08    11.06
000018  072              MODA6      SOR    81/05/12    13.45
000019  071              MODC1      SOR    81/08/14    13.35
000021  080              MODC2      SOR    81/08/21    10.51
000022  074              MODC3      SOR    81/08/24    10.43
000026  095              MODC4      SOR    81/08/24    10.47
000027  062              MODC6      SOR    81/09/01    10.28
000029  005              MODB1      SOR    81/05/12    13.59
000031  135              MODB2      SOR    81/06/01    12.33
000045  005              MODB4      SOR    81/06/05    12.45
000046  091              MODB5      SOR    81/06/29    12.38
000048  005              MODE1      SOR    80/08/08    16.44
000049  135              MODE2      SOR    80/08/08    16.45
000050  130              MODE3      SOR    80/08/08    16.45
000052  005              MODE4      SOR    80/08/08    16.47

.. COMMAND .........     COP          D1,O,,D3

000001  005              MODA7      OBJ    81/05/12    13.48

.. COMMAND .........     COP          D1,L,,D4

000001  005              MODA3000   LOD    81/05/06    11.10
000004  077              MODA4000   LOD    81/05/08    10.44
000005  021              MODC5000   LOD    81/08/24    10.54
000008  176              MODC7000   LOD    81/09/11    08.30
000016  164              MODC8000   LOD    00/00/00    00.08
000022  057              MODB3000   LOD    81/06/01    12.50
000030  021              MODB6000   LOD    81/07/06    14.52
000062  065              MODB7000   LOD    81/07/10    14.14
000118  021              MODB8000   LOD    81/08/14    13.31

.. COMMAND .........     COP          D1,M,,D5

             8060*****NOTHING FOUND


LIBRARIAN FINISHED
DATE 82/07/08 TIME 15.39
TOTAL NUMBER OF ERRORS 00001 UPSI SETTING X'40'
```

Figure 2-14. Librarian Map for Sorting Modules by Type (Part 3 of 3)

2.10.3. Building Module Groups

This job builds two module groups by copying modules from other files. All the files have already been allocated. Figure 2–15 illustrates the librarian map for this job.

Job Control Stream:

```
 1.  // JOB GROUP
 2.  // DVC 2Ø  // LFD PRNTR
 3.  // DVC 5Ø  // VOL DØØ41Ø  // LBL ORIGINAL  // LFD RG
 4.  // DVC 5Ø  // VOL DØØ41Ø  // LBL ALLLOD    // LFD LD
 5.  // DVC 5Ø  // VOL DØØ41Ø  // LBL ALLSRC    // LFD SC
 6.  // DVC 5Ø  // VOL DØØ41Ø  // LBL MIXED     // LFD MX
 7.  // EXEC LIBS
 8.  /$
 9.            FIL   D1=RG,D2=LD,D3=SC,D4=MX
1Ø.            BOG   GROUPMIX,D4
11.            COP   D1,S,MODA1,D4
12.            COP   D3,S,MODC3,D4
13.            COP   D1,O,MODA7,D4
14.            EOG   GROUPMIX,D4
15.            COP   D4
16.            BOG   LOADS,D1
17.            RES   D2,L,MODC5ØØØ
18.            COP.U D2,L,MODC8ØØØ,D1
19.            EOG   LOADS,D1
2Ø.            COP   D1
21.  /*
22.  /&
```

1. Identifies the job.

2. Assigns a printer to the job.

3 through 6.
Declare files for the job. All are on disk volume D00410 with logical unit number 50. Their names are ORGINAL, ALLLOD, ALLSRC, and MIXED with logical file descriptors RG, LD, SC, and MX, respectively.

7. Initiates execution of the librarian.

8. Indicates the start of librarian control statements.

9. Assigns a type and logical file number to each of the files used in the job. Thus, in the control statements that follow, D1 refers to file ORIGINAL, D2 to ALLLOD, D3 to ALLSRC, and D4 to MIXED.

10 through 14.
Build group GROUPMIX in file MIXED. The BOG statement writes the beginning-of-group record in file MIXED. Line 11 copies source module MODA1 from file ORIGINAL; line 14 copies source module MODC3 from file ALLSRC; and line 15 copies object module MODA7 from file ORIGINAL. The EOG statement writes the end-of-group record in file MIXED.

15. Prints a table of contents for file GROUPMIX.

16. Writes the beginning-of-group record for group LOADS in file ORIGINAL.

17. Sets the pointer in file ALLLOD to load module MODC5.

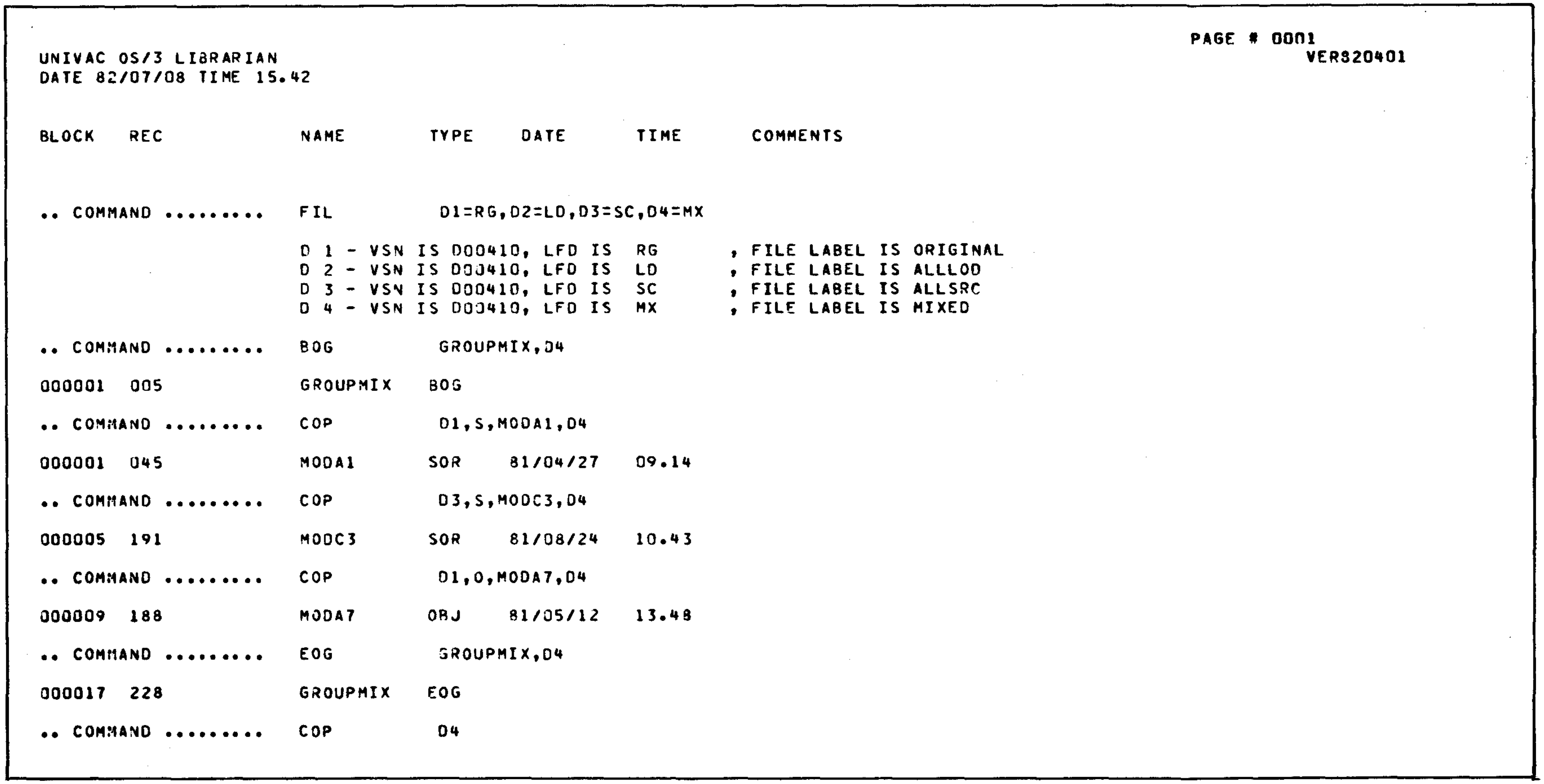
18. Copies all modules from MODC5 to MODC8 in file ALLLOD to group LOADS in file ORIGINAL.

19. Writes an end-of-group record for group LOADS in file ORIGINAL.

20. Prints a table of contents for file original.

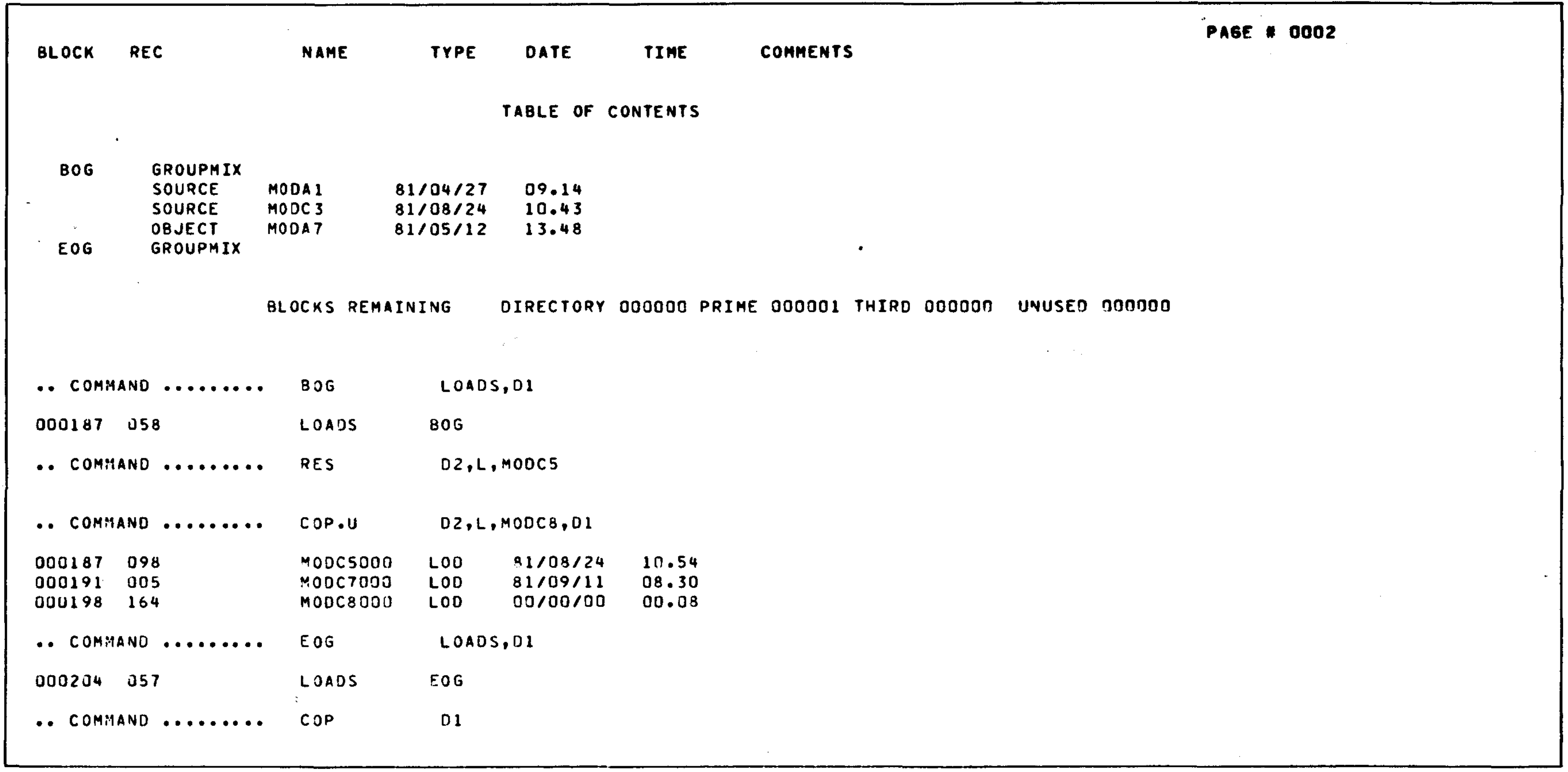
21. Indicates the end of the librarian control statements.

22. Indicates the end of job.



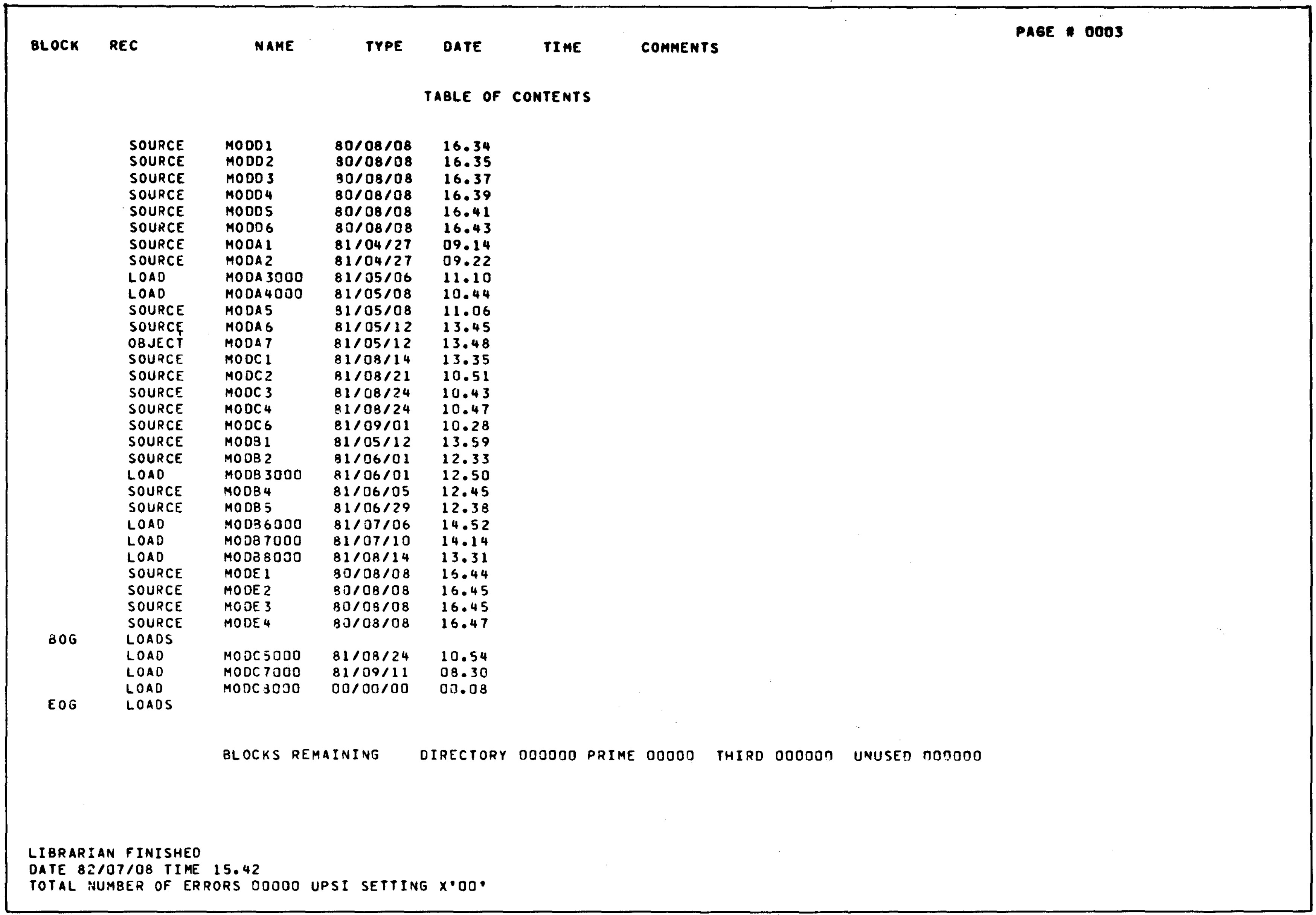
```
                                                                                       PAGE # 0001
UNIVAC OS/3 LIBRARIAN                                                                         VER820401
DATE 82/07/08 TIME 15.42

BLOCK   REC                NAME       TYPE    DATE       TIME       COMMENTS


.. COMMAND .........       FIL          D1=RG,D2=LD,D3=SC,D4=MX

                           D 1 - VSN IS D00410, LFD IS  RG      , FILE LABEL IS ORIGINAL
                           D 2 - VSN IS D00410, LFD IS  LD      , FILE LABEL IS ALLLOD
                           D 3 - VSN IS D00410, LFD IS  SC      , FILE LABEL IS ALLSRC
                           D 4 - VSN IS D00410, LFD IS  MX      , FILE LABEL IS MIXED

.. COMMAND .........       BOG          GROUPMIX,D4

000001  005                GROUPMIX   BOG

.. COMMAND .........       COP          D1,S,MODA1,D4

000001  045                MODA1      SOR    81/04/27   09.14

.. COMMAND .........       COP          D3,S,MODC3,D4

000005  191                MODC3      SOR    81/08/24   10.43

.. COMMAND .........       COP          D1,O,MODA7,D4

000009  188                MODA7      OBJ    81/05/12   13.48

.. COMMAND .........       EOG          GROUPMIX,D4

000017  228                GROUPMIX   EOG

.. COMMAND .........       COP          D4
```

Figure 2–15. Librarian Map for Building Module Groups (Part 1 of 3)



```
                                                                                    PAGE # 0002
BLOCK   REC             NAME       TYPE    DATE      TIME      COMMENTS

                                            TABLE OF CONTENTS

  BOG     GROUPMIX
          SOURCE    MODA1      81/04/27   09.14
          SOURCE    MODC3      81/08/24   10.43
          OBJECT    MODA7      81/05/12   13.48
  EOG     GROUPMIX

                    BLOCKS REMAINING    DIRECTORY 000000 PRIME 000001 THIRD 000000  UNUSED 000000


.. COMMAND .........    BOG         LOADS,D1

000187  058             LOADS      BOG

.. COMMAND .........    RES         D2,L,MODC5

.. COMMAND .........    COP.U       D2,L,MODC8,D1

000187  098             MODC5000   LOD     81/08/24   10.54
000191  005             MODC7000   LOD     81/09/11   08.30
000198  164             MODC8000   LOD     00/00/00   00.08

.. COMMAND .........    EOG         LOADS,D1

000204  057             LOADS      EOG

.. COMMAND .........    COP         D1
```

Figure 2–15. Librarian Map for Building Module Groups (Part 2 of 3)



```
                                                                                  PAGE # 0003
BLOCK   REC            NAME       TYPE    DATE      TIME      COMMENTS

                                   TABLE OF CONTENTS

        SOURCE    MODD1      80/08/08   16.34
        SOURCE    MODD2      80/08/08   16.35
        SOURCE    MODD3      80/08/08   16.37
        SOURCE    MODD4      80/08/08   16.39
        SOURCE    MODD5      80/08/08   16.41
        SOURCE    MODD6      80/08/08   16.43
        SOURCE    MODA1      81/04/27   09.14
        SOURCE    MODA2      81/04/27   09.22
        LOAD      MODA3000   81/05/06   11.10
        LOAD      MODA4000   81/05/08   10.44
        SOURCE    MODA5      81/05/08   11.06
        SOURCE    MODA6      81/05/12   13.45
        OBJECT    MODA7      81/05/12   13.48
        SOURCE    MODC1      81/08/14   13.35
        SOURCE    MODC2      81/08/21   10.51
        SOURCE    MODC3      81/08/24   10.43
        SOURCE    MODC4      81/08/24   10.47
        SOURCE    MODC6      81/09/01   10.28
        SOURCE    MODB1      81/05/12   13.59
        SOURCE    MODB2      81/06/01   12.33
        LOAD      MODB3000   81/06/01   12.50
        SOURCE    MODB4      81/06/05   12.45
        SOURCE    MODB5      81/06/29   12.38
        LOAD      MODB6000   81/07/06   14.52
        LOAD      MODB7000   81/07/10   14.14
        LOAD      MODB8000   81/08/14   13.31
        SOURCE    MODE1      80/08/08   16.44
        SOURCE    MODE2      80/08/08   16.45
        SOURCE    MODE3      80/08/08   16.45
        SOURCE    MODE4      80/08/08   16.47
  BOG   LOADS
        LOAD      MODC5000   81/08/24   10.54
        LOAD      MODC7000   81/09/11   08.30
        LOAD      MODC8000   00/00/00   00.08
  EOG   LOADS

              BLOCKS REMAINING    DIRECTORY 000000 PRIME 00000  THIRD 000000  UNUSED 000000


LIBRARIAN FINISHED
DATE 82/07/08 TIME 15.42
TOTAL NUMBER OF ERRORS 00000 UPSI SETTING X'00'
```

Figure 2–15. Librarian Map for Building Module Groups (Part 3 of 3)

2.10.4. Copying a Card Deck to Disk

This job copies cards to a module in a disk file. In this case, the cards contain the sample job control stream explained in 2.10.2, except that this time the job is called SRTFLS. Figure 2–16 illustrates the librarian map for this job.

Job Control Stream:

```
 1.  // JOB SAVEDECK
 2.  // DVC 2Ø  // LFD PRNTR
 3.  // DVC 5Ø  // VOL DØØ41Ø
 4.  // LBL YOURSRC  // LFD YR
 5.  // EXEC LIBS
 6.  /$
 7.           FIL    D1=YR
 8.           ELE.D D1,S,SRTFLS,SORTS MODULES BY TYPE
 9.  // JOB SRTFLS
1Ø.  // DVC 2Ø  // LFD PRNTR
11.  // DVC 5Ø  // VOL DØØ41Ø
12.  // LBL ORIGINAL  // LFD RG
13.  // DVC 5Ø  // VOL DØØ41Ø
14.  // LBL ALLSRC  // LFD SC
15.  // DVC 5Ø  // VOL DØØ41Ø
16.  // LBL ALLOBJ  // LFD OB
17.  // DVC 5Ø  // VOL DØØ41Ø
18.  // LBL ALLLOD  // LFD LD
19.  // DVC 5Ø  // VOL DØØ41Ø
2Ø.  // LBL ALLMAC  // LFD MC
21.  // EXEC LIBS
22.  /$
23.           FIL    D1=RG,D2=SC,D3=OB,D4=LD,D5=MC
24.           COP    D1
25.           COP    D1,S,,D2
26.           COP    D1,O,,D3
27.           COP    D1,L,,D4
28.           COP    D1,M,,D5
29.  /*
3Ø.  /&
31.           EOD
32.  /*
33.  /&
34.  // FIN
```

1. Identifies the job.

2. Assigns a printer to the job.

3. Identifies logical unit number 50 for disk volume D00410, which contains the file where the module is to be stored.

4. Declares file YOURSRC and logical file descriptor YR for the file where the module is to be stored.

5. Initiates execution of the librarian.

6. Indicates the start of the librarian control statements.

7. Assigns a type code and logical file number to file YOURSC. Thus, D1 in line 8 refers to file YOURSRC.

8. Indicates the beginning of the cards. It also supplies a name, SRTFLS, for the module once it is added to desk file YOURSRC. The comment SORTS MODULES BY TYPE is included in the module header record. The D option causes the records in the module to be listed on the librarian map.

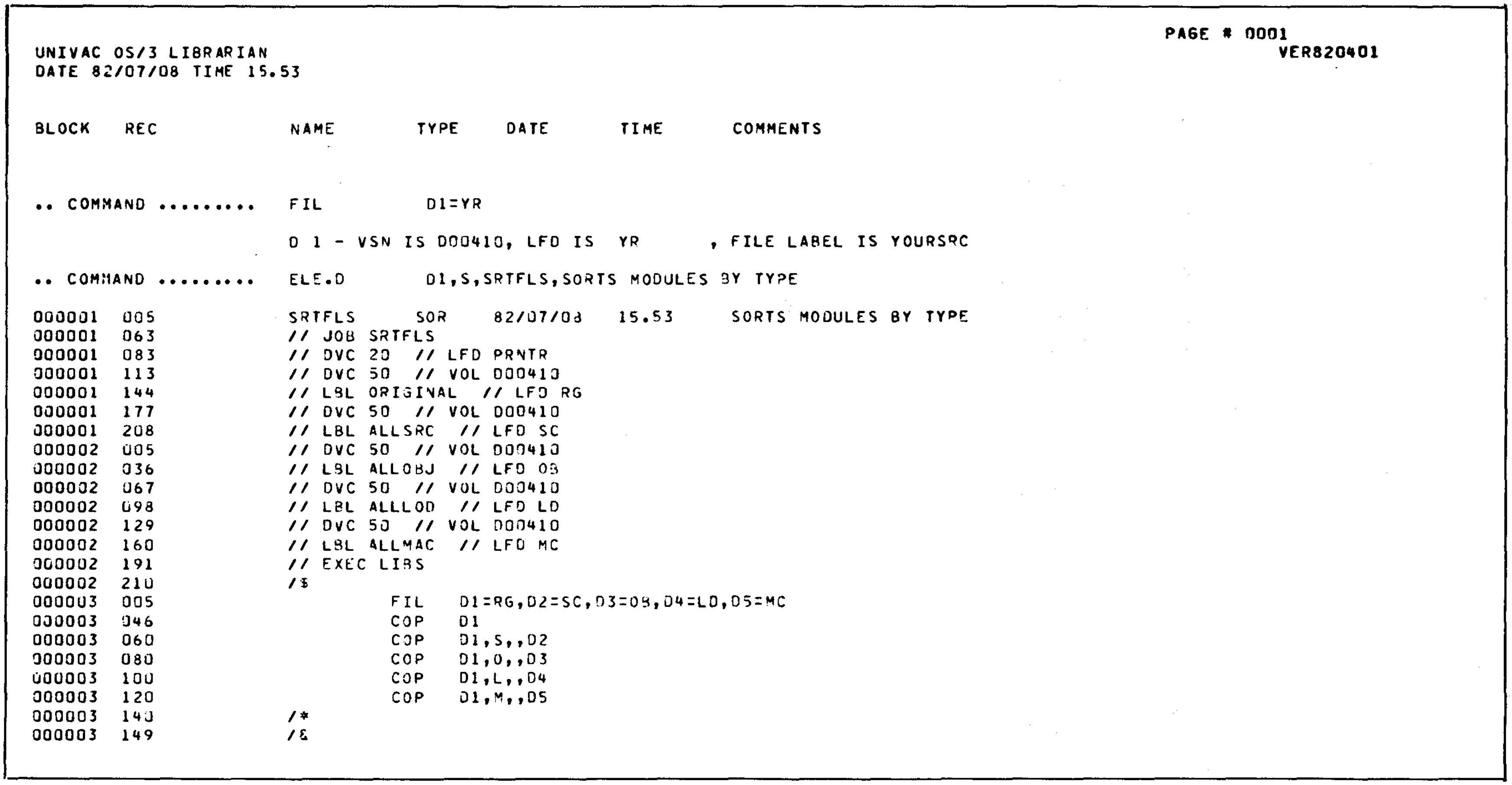
9 through 30.
Are the cards in the librarian control stream being saved.

31. Indicates the end of the cards.

32. Indicates the end of the library control statements for the job.

33. Indicates the end of job.

34. Ends card reader operation.



```
                                                                                      PAGE # 0001
UNIVAC OS/3 LIBRARIAN                                                                        VER820401
DATE 82/07/08 TIME 15.53


BLOCK   REC            NAME       TYPE    DATE       TIME      COMMENTS



.. COMMAND .........   FIL          D1=YR

                       D 1 - VSN IS D00410, LFD IS  YR      , FILE LABEL IS YOURSRC

.. COMMAND .........   ELE.D        D1,S,SRTFLS,SORTS MODULES BY TYPE

000001  005            SRTFLS     SOR    82/07/08   15.53     SORTS MODULES BY TYPE
000001  063            // JOB SRTFLS
000001  083            // DVC 20  // LFD PRNTR
000001  113            // DVC 50  // VOL D00410
000001  144            // LBL ORIGINAL  // LFD RG
000001  177            // DVC 50  // VOL D00410
000001  208            // LBL ALLSRC  // LFD SC
000002  005            // DVC 50  // VOL D00410
000002  036            // LBL ALLOBJ  // LFD OB
000002  067            // DVC 50  // VOL D00410
000002  098            // LBL ALLLOD  // LFD LD
000002  129            // DVC 50  // VOL D00410
000002  160            // LBL ALLMAC  // LFD MC
000002  191            // EXEC LIBS
000002  210            /$
000003  005                  FIL   D1=RG,D2=SC,D3=OB,D4=LD,D5=MC
000003  046                  COP   D1
000003  060                  COP   D1,S,,D2
000003  080                  COP   D1,O,,D3
000003  100                  COP   D1,L,,D4
000003  120                  COP   D1,M,,D5
000003  140            /*
000003  149            /&
```

Figure 2–16. Librarian Map for Copying a Card Deck on Disk

3. MIRAM Librarian Functional Characteristics

3.1. GENERAL

The MIRAM librarian provides limited file maintenance for the following module types:

- Screen format
- Help screen
- Saved run library
- Menu

The MIRAM librarian can copy modules from one file to another, delete modules from a file, print file directories and entire modules, change existing module names, and insert comments on the header record of the module.

Although the MIRAM librarian is primarily a disk utility, the MIRAM library files may also exist on magnetic tape, diskette, or punched cards and may be converted from one medium to another. The output of a given MIRAM librarian job can be an updated tape, disk or diskette library, punched cards, listings, or any combination of these. The operational modes normally are selected at run time via parameter specifications. The program name of the MIRAM librarian is MLIB.

The MIRAM librarian can be executed via the workstation by using a canned job control stream or via punched cards. Examples of both methods are shown in 3.5.

3.2. CONTROL FUNCTIONS

Five control functions are provided by the MIRAM librarian for user management of the program libraries in this system. They are:

1. FIL

   Defines user files.

2. COP

   Copies screen format and saved run library modules from one file to another.

3. DEL

   Deletes screen format and saved run library modules from a file.

4. PRT

   Prints entire modules and file directories.

5. CHG

   Changes the name of an existing module; inserts comments to the header record.

These statements are described in detail in the following sections.

3.3. CONTROL STATEMENTS

The MIRAM librarian uses specific control statements within the job control stream to manipulate library information. These statements include such information as the module name, module type, logical file name, and any appropriate options.

3.3.1. Declare MIRAM File (FIL) Control Statement

Function:

The FIL control statement is used to declare to the librarian all MIRAM files that are referenced subsequently in the control stream. At the same time, each file is assigned a logical file number (0–29), which forms a logical file name that is to be used (rather than the file name) for all subsequent file references within the control stream. File declarations may be strung out on one FIL card or be made individually on separate FIL cards. Up to 30 files can be declared in your job stream.

Format:

LABEL	△OPERATION△	OPERAND
unused	FIL	Fn=filename-1 [,...,filename-n]

Options:

None

Positional Parameter 1:

`Fn=filename`

Specifies that the MIRAM file (file name) is equated with the logical file name (F0–F29).

NOTE:

The file name specification may not exceed eight alphanumeric characters and must begin with an alphabetic character.

Example:

```
1        10    16
_______________________________________
         FIL   FØ=FILØØ,FI=FILØ1,F2=FILØ2
```

Files FILOO, FILO1, and FILO2 are given the logical file names FO, F1, and F2.

3.3.2. Copy Modules (COP) Control Statement

Function:

The COP statement is used to:

- Create a compressed output file by copying the nondeleted contents of an entire library file to another library file
- Copy, from one file to another, individual modules based on module names and types or on module name prefixes

Format:

LABEL	△OPERATION△	OPERAND
unused	COP [options]	input-lfn,[module-type],[name],output-lfn

Options:

`C`
Specifies that the name specified in the name parameter is a module name prefix.

Positional Parameter 1:

`input-lfn`
Specifies the file number (as defined by the FIL statement) of the file to be used as input.

Positional Parameter 2:

`module-type`

Specifies the type of module being copied:

F and FC

Screen format modules. If you are copying a screen format module and the file contains other module types, you must specify two COP statements, one with F as the module type and the other with FC. It is unlikely that you would want to specify only one of the screen format module types, because you get both by default; however, you may refer to the screen format services concepts and facilities, UP-8802 (current version) for more information on the exact function of F and FC modules.

HELP

Help screen modules

J

Saved run library modules.

MENU

Menu modules.

If no type is specified, all modules with the specified name are copied.

Positional Parameter 3:

`name`

Specifies the name (up to eight characters) of the module. If no name is specified, all modules of the specified type are copied.

Positional Parameter 4:

`output-lfn`

Specifies the file number (as defined by the FIL statement) of the file to be used as output. No default is allowed.

NOTE:

If the name and type specifications are both omitted, the entire file is copied.

Examples:

```
    1        10    16
1.           COP   FØ,,,F2
2.           COP   FØ,F,ABC,F2
             COP   FØ,,ABC,F2
3.           COP.C FØ,,ABC,F2
4.           COP.C FØ,J,ABC,F2
```

1. All modules on file FO are copied to file F2.

2. Screen format module ABC on file FO is copied to file F2. Two COP statements are needed because both F and FC module types are necessary to copy screen format modules. Other module types are present in the file; otherwise, the COP statement on line 4 could have been used.

3. All modules on file FO having the prefix ABC are copied to file F2.

4. All saved run library modules on file FO having the prefix ABC are copied to file F2.

3.3.3. Print (PRT) Control Statement

Function:

The PRT control statement is used to:

- Print modules
- Print the file directory

Format:

LABEL	△OPERATION△	OPERAND
unused	PRT [.option]	input-lfn[,module-type][,name]

Options:

C

Indicates that the specified name is a prefix.

D

Indicates that a file directory is to be printed. If specified, the name parameter must be omitted.

Positional Parameter 1:

`input-lfn`
Specifies the file number (as defined by the FIL statement) of the input file. No default is allowed.

Positional Parameter 2:

`module-type`
Specifies the type of module to be printed:

F and FC
Specifies that a screen format module is to be printed. F and FC must be specified on separate PRT statements.

HELP
Specifies help screen modules.

J
Specifies saved run library modules.

MENU
Specifies menu modules.

If the type is omitted and the D option is specified, the entire directory is printed. If the type and the D option are both specified, only header records are listed. If both are omitted, all modules of the specified name are printed.

Positional Parameter 3:

`name`
Specifies the name or the prefix of the module to be printed. When you are using the D option, this parameter must be omitted. If the C option is used, the name is used as a prefix, and all modules of the specified type beginning with this prefix are printed. If this parameter is omitted, all modules of the specified type are printed.

NOTE:

If no options are specified, and name and type are omitted, all modules in the specified file are printed.

Examples:

```
     1           10     16
    ____________________________
1. |             PRT    FØ
2. |             PRT    FØ,F,ABC
   |             PRT    FØ,FC,ABC
3. |             PRT.C  FØ,J,ABC
4. |             PRT.C  FØ,,ABC
5. |             PRT.D  FØ
6. |             PRT.D  FØ,MENU
```

1. All modules in file F0 are printed.

2. Screen format module ABC in file F0 is printed. Two PRT statements are needed because there are two screen format module types (F and FC). If no other types are present in the file, use one PRT statement and omit the type parameter.

3. All saved run library modules having the prefix ABC are printed.

4. All modules having the prefix ABC are printed.

5. All active header records are printed.

6. All menu header records are printed.

3.3.4. Delete Module (DEL) Control Statement

Function:

The DEL control statement is used to:

- Delete the active contents of an entire MIRAM library file

- Delete individual modules based on prefixes, module names, and/or types

Format:

LABEL	△OPERATION△	OPERAND
unused	DEL [.options]	input-lfn[,module-type][,name]

Option:

`C`
: Indicates that the name specified in the name parameter is a module name prefix.

Positional Parameter 1:

`input-lfn`
: Specifies the logical file name of the input file.

Positional Parameter 2:

`module-type`
: Specifies the type of module being deleted:

F and FC
: Specifies screen format modules. If other module types are present in the file, both F and FC must be specified by using two DEL statements. If omitted, all screen format modules with the specified name are deleted.

HELP
: Specifies help screen modules.

J
: Specifies saved run library modules.

MENU
: Specifies menu modules.

If omitted, all modules with the specified name are deleted.

Positional Parameter 3:

`name`
: Specifies the name or prefix of the module being deleted.

NOTE:

If no type is specified, you must be sure to specify a name or the entire file will be deleted.

Examples:

```
       1        10     16
   ┌──────────────────────────
1. │           DEL    F1
2. │           DEL    F1,F
   │           DEL    F1,FC
3. │           DEL    F1,J,ABC
4. │           DEL    F1,,ABC
5. │           DEL.C  F1,,ABC
6. │           DEL.C  F1,MENU,ABC
```

1. All modules in file F1 are deleted.

2. All screen format modules in file F1 are deleted. Two DEL statements are needed for this example because you must specify both types of screen format modules (F and FC). If no other module types are present in the file, use one DEL statement and omit the type parameter.

3. Saved run library module ABC is deleted.

4. Modules ABC of any type are deleted.

5. All modules having the prefix ABC are deleted.

6. All menu modules having the prefix ABC are deleted.

3.3.5. Change Name and Comment (CHG) Control Statement

Function:

The CHG command is used to:

- Change the name of an existing module

- Insert comments on the header record

Only one change operation per command can be performed.

Format:

LABEL	△OPERATION△	OPERAND
unused	CHG	input-lfn,module-type,old-name,$\left\{\begin{matrix}\text{N}\\ \text{C}\end{matrix}\right\}$,$\left\{\begin{matrix}\text{new-name}\\ \text{comments}\end{matrix}\right\}$

Options:

None

Positional Parameter 1:

`input-lfn`

Specifies the file number (as defined by the FIL statement) of the file to be used as input. Also, the updated header record is written to this file.

Positional Parameter 2:

`module-type`

Specifies the type of module being processed:

F and FC

Specifies a screen format module. You need two CHG statements to process a screen format module, one specifying F and the other specifying FC for the type parameter.

HELP

Specifies a help screen module.

J

Specifies a saved run library module.

MENU

Specifies a menu module.

Positional Parameter 3:

`old-name`

Specifies the name of the module being processed. No default is allowed.

Positional Parameter 4:

`N,C`

Specifies the type of change being performed as either a name change (N) or a comment insert (C). No default is allowed.

Positional Parameter 5:

`new-name,comments`

Specifies the information required by the type of change parameter.

If N was specified, a valid module name must be specified. If a module with the same name and type already exists in the file, an error occurs and the change does not occur.

If C was specified, a comment (up to 30 characters) enclosed by single quotes must be specified.

Examples:

```
      1          10    16
    ------------------------------------------------------
1.|              CHG   FØ,F,VERSION1,N,VERSION2
  |              CHG   FØ,FC,VERSION1,N,VERSION2
2.|              CHG   FØ,J,ESCSRC,C,'MODULE ESCAPE SOURCE'
```

1. The screen format module named VERSION1 is renamed VERSION 2. Two statements are needed here, as explained in 3.3.4, example 2.

2. The header record for the saved run library module named ESCSRC will contain the comment 'MODULE ESCAPE SOURCE'.

3.4. MIRAM LIBRARY MODULE FORMATS

MIRAM library modules are unique in that they don't have the same structure as SAT modules. For example, SAT modules can exist in either source, object, or load code forms. MIRAM library modules, however, are created by their own software component and consist of a single form of code that is executed at run time. Table 3–1 shows the format of the MIRAM library module header record.

Table 3–1. MIRAM Library Module Header Record Format

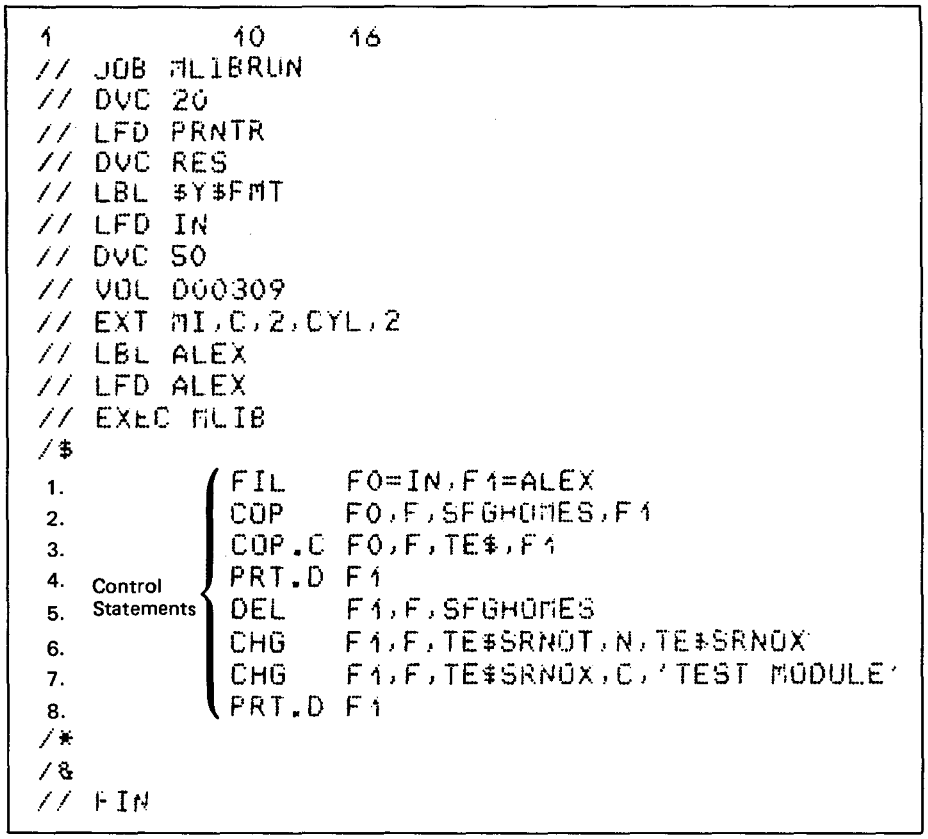
Byte Position	Contents
0-7	Module name
8-11	Module type
12–14	Creation date (yymmdd)
15–17	Creation time (hhmmss)
18–19	Number of active bytes in the last sector occupied by the module
20–23	Number of data sectors
24–27	Total number of sectors occupied by this module
28–30	Reserved for flags
31	Active flag: X'00' – Deleted module X'FF' – Active module
32–33	Record size
34–68	Unused
69–98	Comment field
99–255	Unused

3.5. PROGRAMMING EXAMPLES

A typical job stream is presented in this section to guide you in the execution of the MIRAM librarian. Also included in this section is a sample of the librarian map that is generated when you run this job stream.

3.5.1. Typical MIRAM Library Job Stream

Figure 3-1 shows a typical librarian job stream.



```
1         10     16
// JOB MLIBRUN
// DVC 20
// LFD PRNTR
// DVC RES
// LBL $Y$FMT
// LFD IN
// DVC 50
// VOL 000309
// EXT MI,C,2,CYL,2
// LBL ALEX
// LFD ALEX
// EXEC MLIB
/$
1.                    ( FIL    F0=IN,F1=ALEX
2.                    | COP    F0,F,SFGHOMES,F1
3.                    | COP.C  F0,F,TE$,F1
4.  Control           | PRT.D  F1
5.  Statements       <  DEL    F1,F,SFGHOMES
6.                    | CHG    F1,F,TE$SRNOT,N,TE$SRNOX
7.                    | CHG    F1,F,TE$SRNOX,C,'TEST MODULE'
8.                    ( PRT.D  F1
/*
/&
// FIN
```

Figure 3-1. Typical MIRAM Librarian Job Stream

In the job control portion of the job stream, you must always allocate a printer by assigning the name PRNTR via an // LFD job control statement. As shown, a printer an two disk files (IN and ALEX) are assigned in this job stream. The number of files that you must specify, however, depends on the actual number of files being used in your job. Note that the program name specified on the // EXEC statement is MLIB.

The MIRAM librarian control statements (lines 1 to 8) are included in the job stream as a set of embedded data inserted between the /$ and /* statements.

On line 1 of the embedded data set is the FIL control statement that assigns each file in your job stream a logical file number. It is this logical file number that is used in the subsequent control statements to reference each file. Here, the logical file number F0 is assigned to the file named IN, and F1 is assigned to the file ALEX.

NOTE

When you are working with system files on your resident pack and you specify // DVC RES in the device assignment set, you must include an // LBL statement in that set.

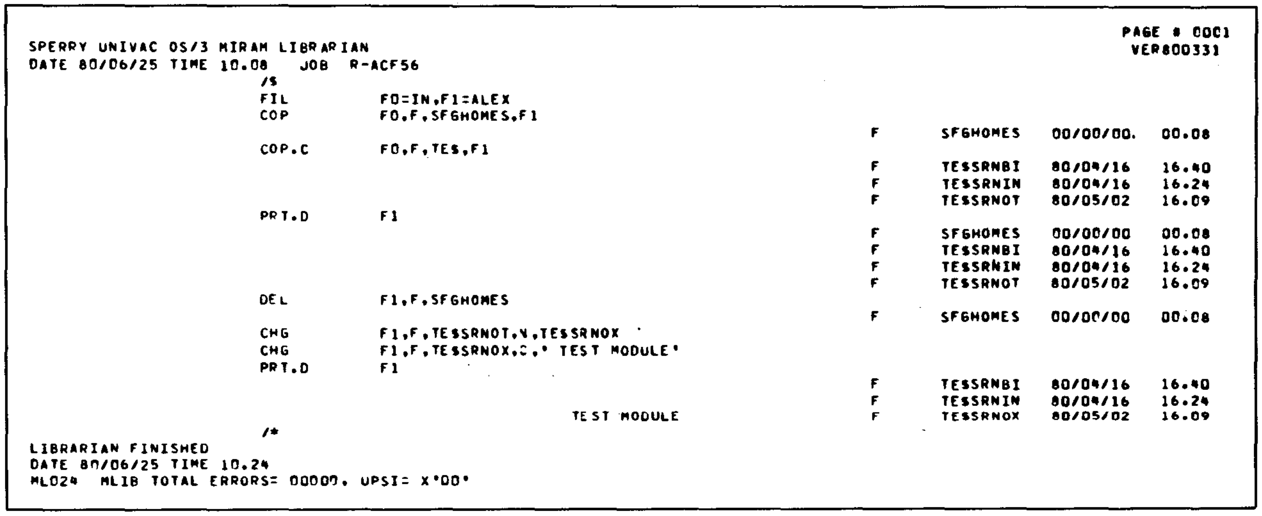
The COP control statement on line 2 copies the screen format module (F-type), SFGHOMES from the system file $Y$FMT on your resident volume to ALEX on volume D00309. On line 3, the COP control statement copies all of the modules beginning with TE$ from $Y$FMT to ALEX. Note here that the C option specifies that TE$ is the module name prefix.

Since the PRT statement in line 4 uses the D option, a directory listing of all the modules in $Y$FMT is printed. The DEL control statement on line 5 deletes the contents of SFGHOMES from the file ALEX.

Lines 6 and 7 contain CHG control statements. The statement in line 6 changes the name of the module TE$SRNOT on $Y$FMT to TF$SRNOX. The statement on line 7, however, doesn't change any names. Instead, it's used to insert the comment "TEST MODULE" on the header record.

Line 8 is the same as line 4. The PRT control statement on this line prints a directory listing of the modules contained in $Y$FMT. Notice that you can use duplicate statements in your job stream.

Figure 3-2 shows a sample librarian map for this job stream. This map is a printed record of the functions performed in your librarian job.



```
                                                                                                          PAGE # 0001
SPERRY UNIVAC OS/3 MIRAM LIBRARIAN                                                                         VER800331
DATE 80/06/25 TIME 10.08   JOB  R-ACF56
                              /$
                              FIL         F0=IN,F1=ALEX
                              COP         F0,F,SFGHOMES,F1
                                                                           F      SFGHOMES   00/00/00.   00.08
                              COP.C       F0,F,TE$,F1
                                                                           F      TE$SRNBI   80/04/16    16.40
                                                                           F      TE$SRNIN   80/04/16    16.24
                                                                           F      TE$SRNOT   80/05/02    16.09
                              PRT.D       F1
                                                                           F      SFGHOMES   00/00/00    00.08
                                                                           F      TE$SRNBI   80/04/16    16.40
                                                                           F      TE$SRNIN   80/04/16    16.24
                                                                           F      TE$SRNOT   80/05/02    16.09
                              DEL         F1,F,SFGHOMES
                                                                           F      SFGHOMES   00/00/00    00.08
                              CHG         F1,F,TE$SRNOT,N,TE$SRNOX
                              CHG         F1,F,TE$SRNOX,C,' TEST MODULE'
                              PRT.D       F1
                                                                           F      TE$SRNBI   80/04/16    16.40
                                                                           F      TE$SRNIN   80/04/16    16.24
                                                      TEST MODULE          F      TE$SRNOX   80/05/02    16.09
                              /*
LIBRARIAN FINISHED
DATE 80/06/25 TIME 10.24
ML024  MLIB TOTAL ERRORS= 00000. UPSI= X'00'
```

Figure 3-2. Sample Librarian Map

PART 3. THE LINKAGE EDITOR

4. Functional Characteristics

4.1. GENERAL

The linkage editor of the SPERRY UNIVAC Operating System/3 (OS/3) is a multiphase load module that resides in the system load library file ($Y$LOD). It relies on the allocation of both system disk and main storage free space for intermediate processing efficiency, as well as one or more system or user object libraries for obaining the desired object code. The load modules that the linkage editor produces subsequently can be stored in either the system load library file or a user load library file. All reentrant load modules must be stored in the system load library file ($Y$LOD). The system access technique (SAT) is used in the management of all files accessed by the linkage editor. Figure 4-1 illustrates the functional relationships between the linkage editor, SAT, and the various interfacing files.

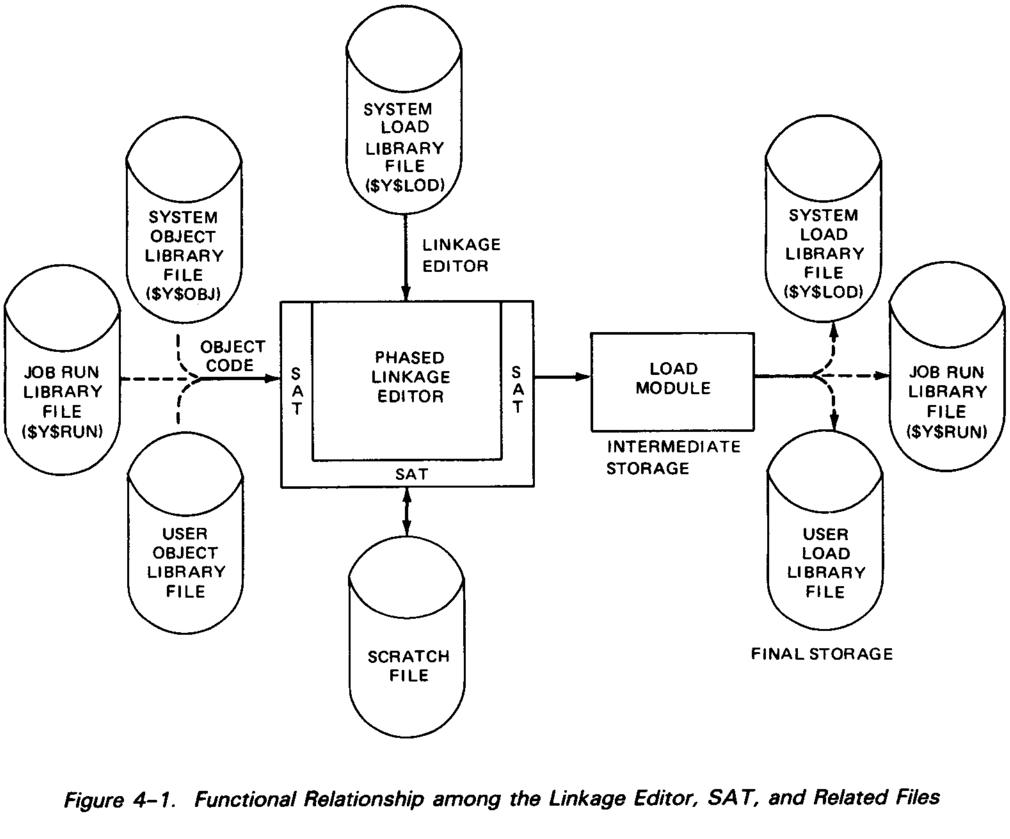


Figure 4-1. Functional Relationship among the Linkage Editor, SAT, and Related Files

The linkage editor relies on an input control stream for acquisition of control statement directives and parameter information. In the absence of such a control stream, the linkage editor defaults to the appropriate system job run library ($Y$RUN) file, to obtain the modules to be used in constructing the load modules. The program name of the linkage editor is LNKEDT.

4.1.1. The SAT Interface

The linkage editor uses the system access technique for the following purposes:

- Reading system and user libraries to obtain the necessary object modules to be included in the load module

- Managing an intermediate scratch file during the link-edit process

- Creating the final output load module or modules

Because the linkage editor produces standard load modules (as needed by the system) and accesses standard object modules (as produced by the various language processors), all of which are contained in disk program library files, the SAT file definitions (DTFs) describe three partitions for each file. The first partition is the directory partition used as an index into the second and third partitions, the data partitions of the library file. Because the output of the linkage editor is essentially a library file (load module type), it also contains both a directory partition and two data partitions (the directory is composed primarily of phase header pointers and the third partition is empty).

The load modules produced by the linkage editor contain phase header records (one for each phase segment); text/RLD records comprising the actual instructions, constants, and relocation information making up each individual phase; transfer records (one at the conclusion of each phase); ISD records and, optionally, shared code records. Each phase header contains an appropriate entry in the load library directory (first partition), which identifies the load module and the particular phase and points to the relative position within the data body of the library file (second partition). For each phase header entered in the body of the load module (second partition), the necessary directory entries also are entered in the directory index (first partition) with the relative pointer obtained from the necessary DTF description for the second partition. Both the directory and prime data areas of the file contain blocked logical records.

The object modules that the linkage editor uses in its construction of the load module also are retrieved through the SAT. These files are accessed through a file directory scan (first partition) in an attempt to locate the necessary module header, control section definition, or entry symbol definition. (Object library file directories are supplemented with named CSECTs, named COMs, and ENTRYs to facilitate this search.) When the desired directory item is found, the relative directory pointer is extracted from the record and copied into the second partition definition of the DTF to obtain the needed modules or control sections from the body of the module.

The intermediate scratch file used by the linkage editor in the course of the link-edit job and seen only by the linkage editor for its processing also is accessed via SAT. This file is scratched when the link-edit job is finished.

4.1.2. Temporary Storage Usage

The linkage editor uses open storage space at the end of its longest path for the management of temporary reference tables and buffers. This area is reserved by the linkage editor when it is first loaded into main storage. In a minimum storage system, the reserved area is always equal to the size of the maximum storage partition allowed any user program. In larger system configurations, this reserved area is dynamically expanded at linkage editor execution time.

If any job step in a job containing a linkage editor job step requires an amount of storage in excess of the minimum linkage editor storage requirements, the linkage editor uses the excess for reference table and buffer expansion. Because job control assigns storage requirements for a given job based on the largest requirement for any job step, when not specified explicitly on the // JOB control statement, the linkage editor adjusts its pointers (as originally established) to accommodate the availability of the extra tabling space (if any extra exists). The linkage editor determines the amount of storage space allocated for the job of which it is a step by examining the job prologue. The speed of the link edit is proportional to the amount of main storage space available to the linkage editor.

If a given link-edit job involves a number of definitions with table entries exceeding the current internal storage space available for that job step, the linkage editor expands the reference table by using data blocks on the temporary output medium via the DTF partition assigned to the intermediate file. Thus, table overflow does not cause the link-edit job to be aborted.

4.2. LINKAGE EDITOR INPUT AND OUTPUT

The linkage editor uses as input both job control stream data in the form of linkage editor control statements and object module elements and produces, as output, one or more load modules and a link-edit map for each load module (Figure 4–2). Under normal operating conditions (no normal linkage editor options suppressed or superseded), the linkage editor uses the control statements contained in the job control stream to construct one or more load modules per job. These statements, which comprise the primary control stream input to the linkage editor, specify the object modules to be included in the load modules, the structure of the load modules to be produced, and the linkage editor options to be in effect during construction of the load modules.

The object modules referenced in the primary control stream also may have linkage editor control statements embedded in them. When they do, these control statements are inserted in the primary control stream at the point the object module containing them was referenced, and thus become part of the primary control stream input.

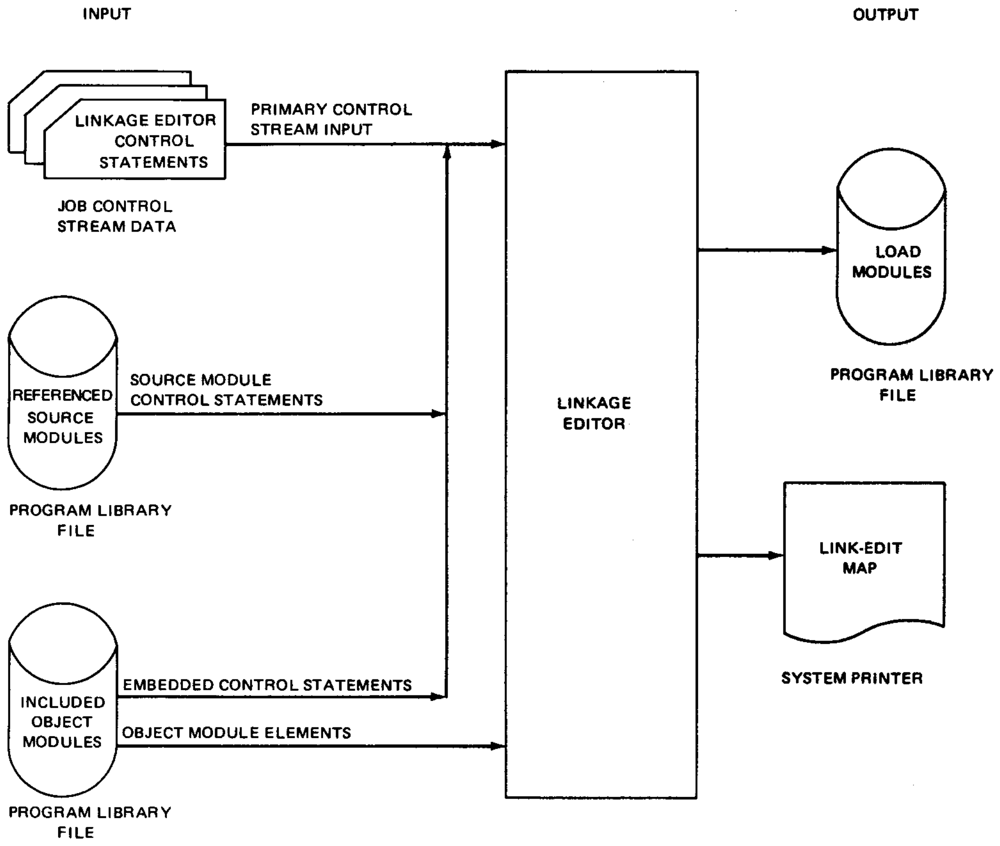


Figure 4-2. Linkage Editor Input and Output

The control statements contained in the primary control stream input also may identify source modules containing linkage editor control statements to be processed during the job step. When such statements are processed, the control statements they reference also are inserted in the primary control stream input, just like embedded control statements. These statements, however, have special processing significance attached to them and are thus referred to as source module control stream statements in this manual to differentiate them from the other two types of primary control stream statements (job control data and embedded control statements).

The load modules produced by the linkage editor normally are output to the system job run library file ($Y$RUN). The user, however, may direct his output to be stored in any library file he wishes, or he may elect to suppress this output entirely through the linkage editor control statement options available to him. The same is true for the linkedit map, which normally is output by the linkage editor for each load module it produces, whether or not the load module output is suppressed. This output also can be suppressed by the user through a linkage editor control statement option. Each link-edit map describes the control statements used to construct its associated load module, the symbols, or labels, detected in the object modules included in the load module, the structure of the load module produced, and any processing errors that might have been detected during construction of the load module. Section 7 gives detailed descriptions and illustrations of the map components.

4.3. CONTROL STATEMENT FUNCTIONS

Control statements are available to the user to direct the linkage editor in construction of a load module. These statements allow the user to generate a load module that is structured in accordance with his program requirements, as these requirements exist in the object elements that are to make up the load module. Linkage editor control statements normally appear as data in a job control stream following an execute linkage editor job control statement (// EXEC LNKEDT). As previously indicated, however, linkage editor control statements also can be embedded in object modules or contained in source modules referenced by linkage editor control statements. Therefore, if no control statements are present in the job control stream following a // EXEC LNKEDT control statement, the linkage editor constructs a load module by using all the object modules currently contained in the job's job run library file ($Y$RUN). If none of the object modules contained in this file contain embedded linkage editor control statements, the linkage editor constructs only one nonreentrant single-phase load module, named LNKLOD, consisting of all the object modules contained in the $Y$RUN library file; otherwise, if so directed by embedded control statements, the linkage editor may produce multiple load modules, reentrant or otherwise, with or without multiphase and multiregion structures.

The name and basic function of each of the linkage editor control statements are described in the following paragraphs. Section 6 gives a more detailed description of these control statements.

■ LINKOP and // PARAM Statements

These statements specify the linkage editor processing options that are to be in effect during construction of a load module. These options include:

- a method of determining file name assumption to be used by the include mechanisms of the linkage editor when such file names are not explicitly designated by the user;
- disallowing the automatic inclusion of object modules in the load module by the linkage editor;
- disallowing the automatic overlay mechanism of the linkage editor from being included in the load module;
- disallowing the promotion of common storage sections by the linkage editor;
- selecting the output file in which the load module created by the linkage editor is to be stored;
- terminating a link-edit job if a processing error is detected;
- selecting the components of the link-edit map to be output for a given link-edit job;

- inserting comments in the phase header records produced by the linkage editor;

- building reentrant load modules;

- ignoring the reentrancy of object modules; and

- suppressing the production of ISD records.

■ LOADM – Begin Load Module

This statement requests the linkage editor to begin construction of an executable load module. This statement initiates the creation of the start of the root phase segment and specifies a name for the load module. It normally is the first control statement in each link-edit job.

■ INCLUDE – Include Object Code

This control statement instructs the linkage editor to include, in the load module being constructed, an entire object module or specific object module sections. This statement specifies the name of the module and module sections, if applicable, required to be in the load module segment currently under construction. It may also identify the file in which the specified module is located.

■ OVERLAY – Begin Overlay Phase

This statement directs the linkage editor to begin construction of a new load module phase separate from the initial phase and any other previously defined phase. Any and all INCLUDE requests for object code following an OVERLAY statement are included in the phase initiated by the OVERLAY statement up until the detection of another OVERLAY or REGION control statement or termination of the immediate link-edit job. The phase name assigned to the overlay phase is a combination of the name assigned to the load module plus a 2-digit number from 00 to 99, which indicates the order in which the various phases of a load module were declared to the linkage editor. A unique 6-character alias phase name also may be assigned to each overlay phase.

■ REGION – Begin New Region

This statement causes a new phase to be created in a new region of the load module. This statement has all the attributes of the OVERLAY statement, in addition to initiating the construction of a new load module region.

■ ENTER – Define Phase Execution Entrance

This statement specifies the start-of-execution point for the phase currently under construction in a load module. This is the point to which control is optionally transferred once the phase has been loaded in main storage. The transfer point is optionally assigned by the linkage editor if no such statement is supplied for a phase. This statement is normally the last specified for each phase defined in a load module.

- EQU - Define Label

  This statement is used to define an otherwise undefined label in a load module. The normal method of defining and satisfying cross-references by the linkage editor is via the proper INCLUDE directives and external symbol dictionary (ESD) records in object code included in the load module. This statement, however, allows you to equate two symbols (a symbol and a value, or a value only) that could not otherwise be resolved in the link-edit run. If one symbol is being equated to another, the equating symbol must have been previously defined.

- MOD - Modify Location Counter

  This statement is used to modify the current program counter that the linkage editor maintains during construction of a load module. This statement permits the user to accomplish boundary adjustments at link-edit time outside the realm of the makeup of modules and code being included.

- RES - Reserve Storage

  This statement directs the linkage editor to reserve additional storage at the end of the longest path in the load module being constructed. The additional storage requested is included in the overall length of the load module as recorded in the load module header record, and may be referenced by the object code included in the load module.

4.4. OBJECT MODULE FORMAT

Figure 4-3 illustrates the format of the object modules produced by all the language processors and used as input by the linkage editor. As shown, an object module consists of:

- an object module header record that uniquely identifies the object module;
- one or more control section records, each of which defines the symbolic name, external symbol identification (ESID), and length of each control section (CSECT and COM) comprising the object module;
- possibly one or more external symbol dictionary (ESD) records, each of which is used by the linkage editor in satisfying cross-references (ENTRY, EXTRN, and V-CON references) between object modules during the link-edit operation;
- possibly one or more internal symbol dictionary (ISD) records used to describe the internal symbols of the user program;
- one or more text and relocation list dictionary (RLD) records containing the data and instructions making up the object code, together with relocation masks used by the linkage editor to modify specific areas of the object code at link-edit time;

- a transfer (TRF) record (normally the last record in any set of relocatable object code produced by a language processor) that may optionally indicate a start-of-execution address for the object module; or

- one or more linkage editor control statements that may be embedded in an object module. Embedded control statements may appear only immediately following the object module header record or transfer record, as shown in Figure 4-3.

OBJECT MODULE HEADER RECORD
LINKAGE EDITOR CONTROL STATEMENTS (OPTIONAL)
CONTROL SECTION RECORDS
EXTERNAL SYMBOL DICTIONARY (ESD) RECORDS (OPTIONAL)
ISD RECORDS (OPTIONAL)
TEXT/RELOCATION LIST DICTIONARY RECORDS
TRANSFER RECORD
LINKAGE EDITOR CONTROL STATEMENTS (OPTIONAL)

NOTE:

All control section and ESD records must have unique names within a given object module. For example, an entry point and a CSECT must not have the same name and exist in the same object module.

Figure 4-3. OS/3 Object Module Format

4.5. LOAD MODULE FORMAT

Figure 4-4 illustrates the format of the load modules produced by the linkage editor, as they are stored in a library file. As shown, all load modules consist of at least one phase, called the root phase. Multiphase and multiregion load modules consist of a root phase plus one or more additional phases. (A root phase is required in every load module because it contains the information necessary to allocate the load module space in main storage prior to its execution by the system.) The number of phases and regions comprising a load module is specified by the user in the linkage editor control stream that produces the load module. A load module may consist of up to 100 phases and up to 10 regions.

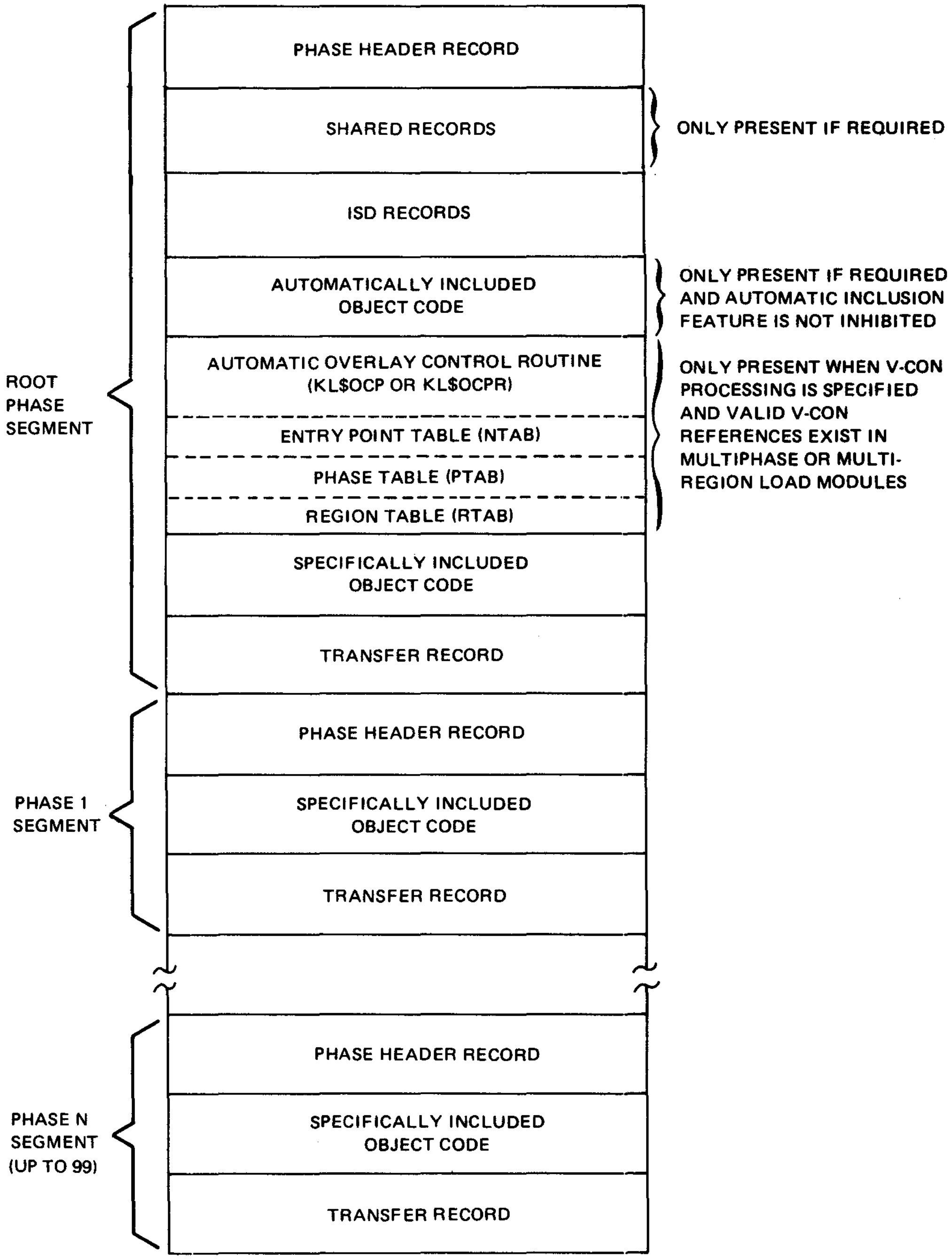

PHASE HEADER RECORD
SHARED RECORDS
ONLY PRESENT IF REQUIRED
ISD RECORDS
AUTOMATICALLY INCLUDED OBJECT CODE
ONLY PRESENT IF REQUIRED AND AUTOMATIC INCLUSION FEATURE IS NOT INHIBITED
AUTOMATIC OVERLAY CONTROL ROUTINE (KL$OCP OR KL$OCPR)
ENTRY POINT TABLE (NTAB)
PHASE TABLE (PTAB)
REGION TABLE (RTAB)
ONLY PRESENT WHEN V-CON PROCESSING IS SPECIFIED AND VALID V-CON REFERENCES EXIST IN MULTIPHASE OR MULTI-REGION LOAD MODULES
ROOT PHASE SEGMENT
SPECIFICALLY INCLUDED OBJECT CODE
TRANSFER RECORD
PHASE HEADER RECORD
PHASE 1 SEGMENT
SPECIFICALLY INCLUDED OBJECT CODE
TRANSFER RECORD
PHASE HEADER RECORD
PHASE N SEGMENT (UP TO 99)
SPECIFICALLY INCLUDED OBJECT CODE
TRANSFER RECORD

Figure 4-4. OS/3 Load Module Format

Each phase of a load module consists of at least the following:

- A phase header record that identifies:

  - the name of the phase;

  - the size of the phase in bytes and its origin; and

  - the amount of main storage required to load the longest path of the load module.

- At least one (and probably more) text/relocation (RLD) record that includes both data and instructions comprising the load module phase and possible relocation masks to be used to relocate the text at execution time

- A transfer record identifying the end of the phase, and containing a transfer address at which point phase execution is to optionally begin

In addition to these records, the root phase of a load module may contain:

- Any number of shared records

  A reentrant load module will have one or more SENTRY records indicating the availability of ENTRY points in that module. A nonreentrant load module will not have any shared records unless it references reentrant code. In this case, it will have a resource record and a SEXTRN record for each reference that was resolved to a reentrant object module.

  - Resource records contain the name of the reentrant module referenced.

  - SEXTRN records identify the EXTRNs that were resolved to the resource.

- Any number of ISD records

  The ISD records can be one of the following:

  - CSECT/COM ISD records identifying control and common sections included by the linkage editor; or

  - ISD records produced as a result of being generated by any of the language processors within object modules.

- Any automatically included object module sections needed to satisfy any cross-references not defined in any specifically included object module section

- An automatic overlay control routine if the load module produced requires the automatic loading of its phases

  Two versions of this routine exist: One handles single-region structures (SL$OCP) and the other, multiregion structures (KL$OCPR).

- An entry point table (NTAB) used in conjunction with the automatic overlay control routine to provide for automatic loading of phases

  This table contains the addresses of referenced ENTRY points and an index into the phase table (PTAB) that identifies the phases containing the ENTRY points.

- A phase table (PTAB), also used in conjunction with the automatic overlay control routine, to provide for automatic loading of phases

  This table contains the overlay structure information required to load the various phases of the load module.

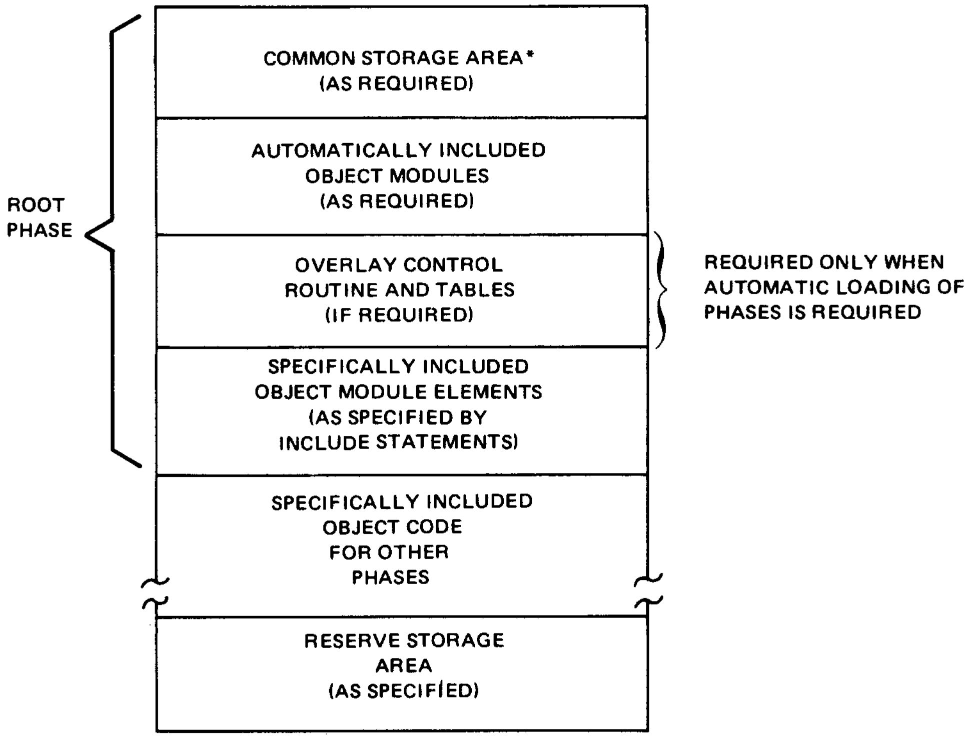
- A region table (RTAB) used in conjunction with the automatic overlay control routine to control automatic loading of the various regions that may be in the load module

  This table describes the region structure of the load module and is present only if the KL$OCPR automatic overlay control routine, with multiregion control, is present.

When a load module is loaded in main storage for program execution, only its instructions and data are loaded; phase header, shared records, ISD records, and transfer records are not included. These records are used only to:

- relate the amount of main storage required to store the load module and the particular phase being loaded, including any common storage areas and reserved storage areas required by the load module program;
- identify the phase load address;
- identify the program starting address;
- identify the resources (reentrant load modules) that are needed and SEXTRNs to be satisfied at execution time;
- identify the entry points that are available in reentrant load modules at execution time; and
- give JOBDUMP the necessary information to print a dump of segmented CSECTs of the program and internal symbols of the user program.

Figure 4-5 illustrates the format of a typical load module as it might appear in main storage. As shown, it might have a common storage area and a reserve storage area. These areas are storage areas that the linkage editor reserves for the load module program in accordance with the common storage area definitions contained in the object module code included in the load module and the reserve storage (RES) control statements contained in the linkage editor control stream that created the load module. The size of the common storage area reserved for a load module is the sum of all the common storage areas requested and described by both the specifically included and automatically included object modules in the load module. Common storage areas may be labeled or unlabeled in the object module code. The size of the reserve storage area reserved for a load module is the sum of all the reserve storage areas specified by the user in all RES control statements.



* Common storage areas also may be promoted to specific phases other than the root phase by the linkage editor. See 8.7.3.

Figure 4-5. Typical Load Module Format When Loaded in Main Storage

4.6. LOAD MODULE STRUCTURE

The structure of a load module is that form a load module takes when it is loaded in main storage by the program loader routine of the supervisor. This structure is defined by the phase and region data contained in the load module as placed there by the linkage editor in accordance with the control stream directives by the user. Three basic load module structures are capable of being constructed by the linkage editor:

1. single-phase load modules;

2. multiphase load modules; and

3. multiregion load modules.

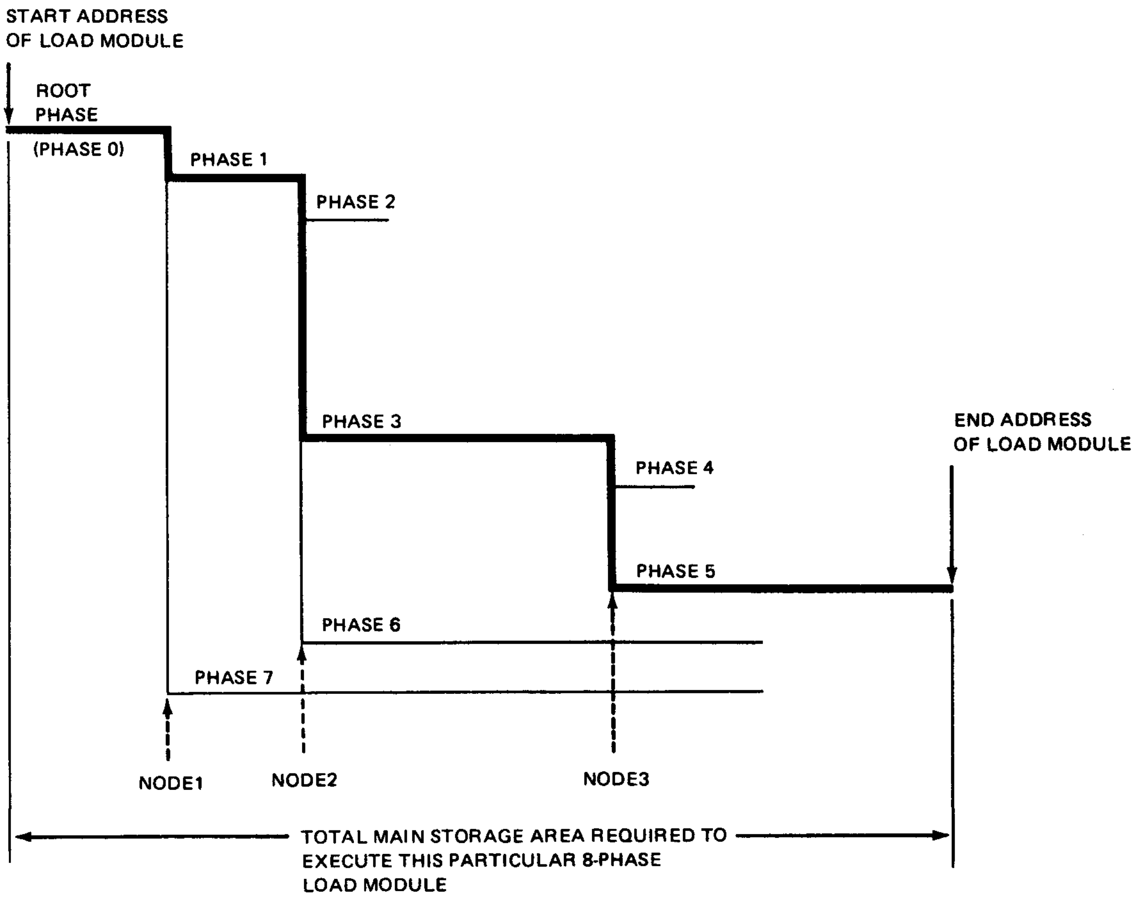
4.6.1. Single-Phase Load Modules

The storage structure of a single-phase load module can be represented by a single horizontal line whose length is relative to the amount of serial storage locations required to store the load module in main storage. All load modules generated by the linkage editor start out as single-phase load modules and are created only as multiphase or multiregion load modules if so directed through OVERLAY and REGION control statements in the input control stream. Thus, single-phase load modules are modules that consist solely of a root phase; multiphase and multiregion load modules consist of a root phase plus one or more additional phases. Reentrant load modules must be single-phase load modules.

4.6.2. Multiphase Load Modules

Multiphase load modules are constructed by a programmer to minimize the main storage requirements of a program. Multiphase load modules contain multiple phases, which can overlay each other in main storage. The structure of a typical multiphase load module is illustrated in Figure 4-6. As shown, multiphase load modules are represented by multiple horizontal lines, each of which represents a particular program phase and its relative length. The main storage location at which a phase is loaded is called a node point. Node points are shown as vertical lines whose lengths have no significance. All the phases in a multiphase load module excluding the root phase are loaded in main storage at a node point. Node points and phases are defined by the user through the linkage editor OVERLAY and REGION control statements.

The INCLUDE statements following an OVERLAY or REGION control statement identify the object module elements that are to comprise the phase. Ignoring the root phase, the number of phases in a multiphase load module coincides with the number of OVERLAY and REGION control statements present in the control stream that caused its generation. The root phase of all load modules is initiated with the initiation of the load module, normally in response to a LOADM control statement.


START ADDRESS OF LOAD MODULE
ROOT PHASE
(PHASE 0)
PHASE 1
PHASE 2
PHASE 3
PHASE 4
PHASE 5
PHASE 6
PHASE 7
END ADDRESS OF LOAD MODULE
NODE1
NODE2
NODE3
TOTAL MAIN STORAGE AREA REQUIRED TO EXECUTE THIS PARTICULAR 8-PHASE LOAD MODULE

NOTES:

1. Heavy lines indicate an example of a path.

2. The end address of the load module is at last addressable location of the longest path in the load module. This address is always defined and may be declared as an external reference (EXTRN) in any object module included in the program. The entry name assigned this end address is KE$ALP.

Figure 4-6. Typical Multiphase Load Module Structure

Figure 4-7 shows the OVERLAY control statements required to produce the multiphase load module illustrated in Figure 4-6.

```
LOADM      EXAMPLE
  .                 }  GENERATES
  .                 }  PHASE 00 (ROOT PHASE)
  .                 }
OVERLAY    NODE1
  .                 }
  .                 }  PHASE 01
  .                 }
OVERLAY    NODE2
  .                 }
  .                 }  PHASE 02
  .                 }
OVERLAY    NODE2
  .                 }
  .                 }  PHASE 03
  .                 }
OVERLAY    NODE3
  .                 }
  .                 }  PHASE 04
  .                 }
OVERLAY    NODE3
  .                 }
  .                 }  PHASE 05
  .                 }
OVERLAY    NODE2
  .                 }
  .                 }  PHASE 06
  .                 }
OVERLAY    NODE1
  .                 }
  .                 }  PHASE 07
  .                 }
LOADM      NEXT
```

NOTE:

The ellipses represent INCLUDE statements for object modules to be included in a particular phase. These statements must follow the proper LOADM or OVERLAY statement.

Figure 4-7. Typical Multiphase Load Module Control Stream

4.6.2.1. Phase Definitions

Phases are defined by the user beginning with an OVERLAY or REGION control statement (root phases excepted) and usually followed by one or more INCLUDE control statements that identify the object module elements that are to comprise the phase. A load module phase may be thought of as a program segment that can perform one or more specific processing tasks. The order in which phases are defined by the user does not dictate the order in which the phases in a load module must be executed; the logic of the program determines the sequence of execution of the phases contained in a load module and, likewise, the number of times any specific phase will be loaded and executed. In some programs, not all phases are executed each time the program is executed, again, depending on the logic of the program.

Thus, the order in which phases are defined, which is also the order in which they are numbered by the linkage editor, does not necessarily have any bearing on the order in which the phases of a load module are executed.

The node point assigned to each phase usually does, however, determine where the phase will be loaded in main storage. Thus, a program's logic must be considered before one phase of a load module is assigned the same origin as another phase in the load module, because these phases never can be in main storage simultaneously.

4.6.2.2. Phase Names

Phase names are used to identify the various phases of a load module when they are to be loaded in main storage for program execution. Programmers who wish to do their own program loading, rather than have the automatic overlay-region control mechanism of the linkage editor embedded in their load module, must reference a phase name in a FETCH or LOAD macroinstruction whenever a phase is to be loaded for execution.

The linkage editor automatically assigns a name to each phase of a multiphase load module, based on the name assigned to the load module either by the programmer or the linkage editor. Phase names consist of eight alphanumeric characters. The first six characters are the same as the load module name, and the last two characters are the decimal number of the phase (00 through 99). Phase numbers are assigned to each phase consecutively, in the order in which the phases are defined by OVERLAY and REGION control statements.

An alias phase name also may be assigned to each phase by the programmer through the OVERLAY or REGION control statement that causes its generation. The assignment of alias phase names allows the programmer to reference the phases of a load module in his subroutines or phrases, without knowing the order in which the phases will be defined in the linkage editor control stream. The linkage editor overlay control mechanism always refers to the linkage-editor-generated phase names.

4.6.2.3. Node Points and Paths

The starting address of each phase in a load module is called a node point. Node points are defined by the user as a symbol in the operand field of the OVERLAY or REGION control statements that are used to define each phase. The node point of the root phase is the name assigned to the load module. The starting address of one phase is normally the terminating address of a previous phase (it also could be a relative definition in the current path). The same node point may be the assigned origin of more than one phase in a load module; however, when an OVERLAY statement that refers to a node point previously defined is detected by the linkage editor, all intervening node points are eliminated. For example, in Figures 4-6 and 4-7, once phase 6 is defined, no additional phases may be defined starting at NODE3. Also, once phase 7 is defined, no additional phases may be defined starting at NODE2. If phase 7 were followed by an OVERLAY NODE2 statement, the linkage editor would construct a completely new NODE2 node point at the end of phase 7 for the new phase being defined.

References to a given label in the load module may be traced along a path from one phase back to the root phase. Reference x is said to be on the path of phase y if they are in the same phase, or if they are on the same path and reference x is closer to the root phase than phase y. The root phase is usually on the path of every other phase in a load module, except when it is overlaid by a subsequent phase. In Figure 4-6, phases 0, 1, 3, and 5 are on the path of phase 5, as indicated by the heavy line; however, phase 5 is not on the path of phases 0, 1, or 3. Phase 1 is not on the path of phase 7, and phase 7 is not on the path of phase 1. The maximum number of phases allowed on any path is 14.

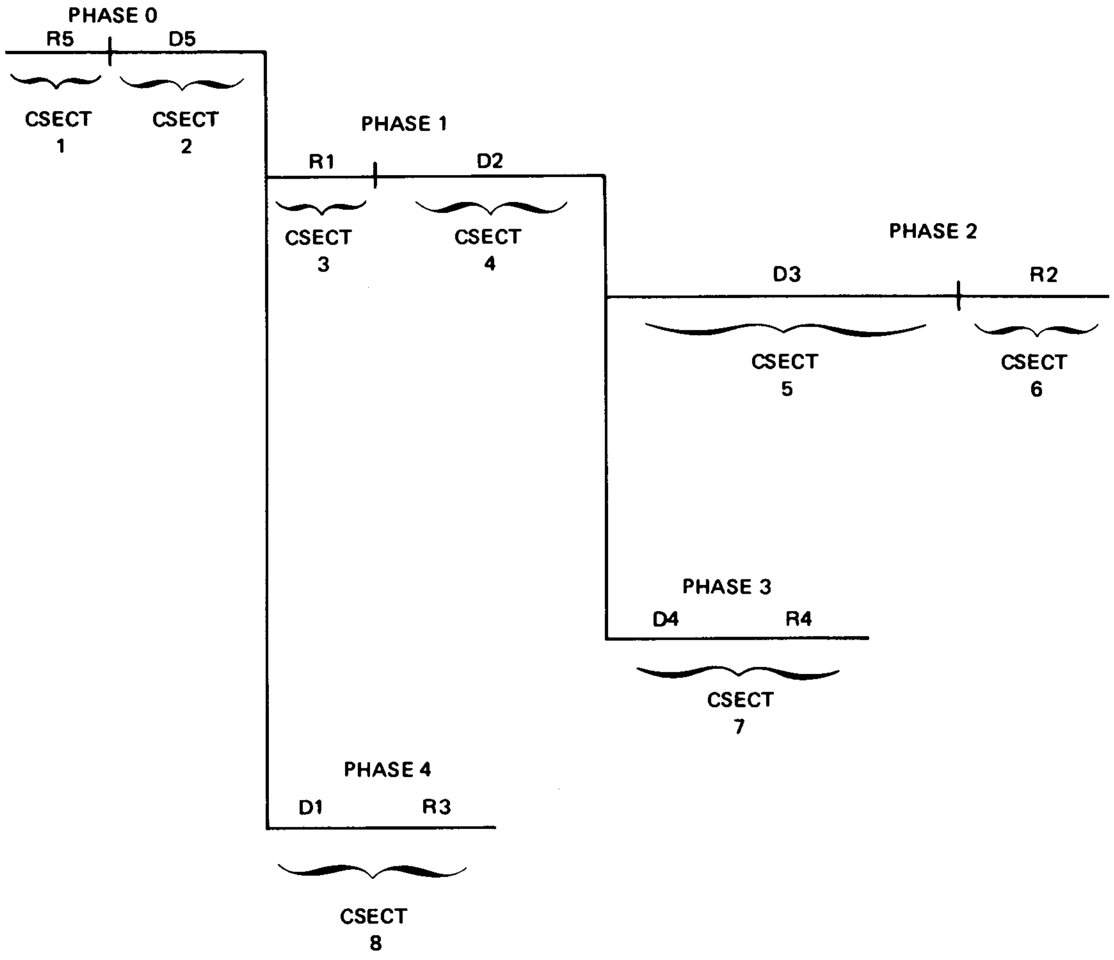
The path concept of a load module must be understood before the user can appreciate how the linkage editor satisfies references between various load module phases.

4.6.2.4. Communications between Phases

Communications between phases is accomplished by an external reference in one phase whose name matches a definition (CSECT or ENTRY point) in another phase. The language processors may produce two types of external references. The first is a standard (A-type) address constant which, at execution time, will contain the linkage-editor-assigned address of the associated definition. The second is a V-type address constant, which, in effect, requests that an automatic load mechanism ensure that the definition being referenced is resident at the time the reference is made. A linkage editor option allows the user to convert, at link-edit time, all V-type EXTRNs to standard address constants.

The phase structure of a particular load module determines another characteristic of a given reference. If the definition to which a particular EXTRN refers is on the path of the reference, the reference is said to be inclusive. All inclusive references are treated as standard address constants by the linkage editor regardless of the EXTRN type generated by the language processor. Conversely, if the definition that satisifies a particular reference is not on the path of the reference, the reference is said to be exclusive. It is these exclusive references that can be controlled by the user through the linkage editor (V/NOV) option. Figure 4-8 illustrates both inclusive and exclusive references.

If a V-type exclusive reference is made, the linkage editor puts the address of the definition in a table it generates (NTAB) in the root phase of the load module. The reference address is then made to point to an automatic overlay control routine. This control routine ensures that the required path is loaded before transferring control to the required definition. All other references require that the user ensure that the required definition is loaded when referenced, through supervisor FETCH and LOAD macroinstructions contained in the text of his program.



LEGEND:

D Definition
R Reference

Inclusive References

R2 to D2, D3, or D5
R1 to D2 or D5
R3 to D1 or D5
R4 to D2, D4, or D5
R5 to D5

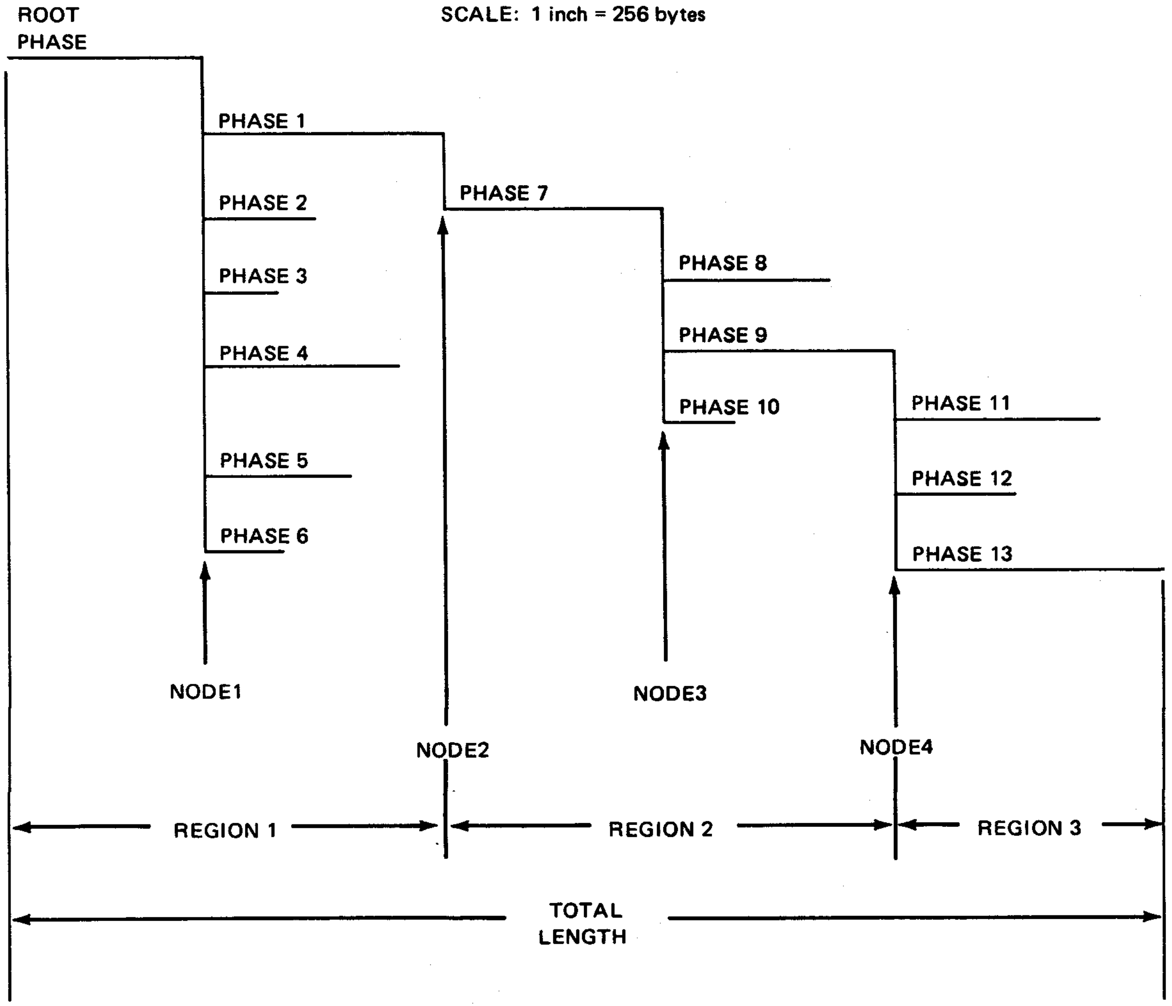
Exclusive References

R1 to D1, D3, or D4
R2 to D1 or D4
R3 to D2, D3, or D4
R4 to D1 or D3 .
R5 to D1, D2, D3, or D4

Figure 4-8. Examples of Inclusive and Exclusive References

4.6.3. Multiregion Load Modules

Multiregion load modules are basically the same as multiphase load modules, except that a multiregion load module is so constructed that the origin of the first phase of each region is at the end of the longest path defined in the previous region, rather than at the end of the phase previously defined. This feature prevents the phases in one region from overlaying any portion of a phase in any other region. Figure 4-9 illustrates the structure of a typical multiregion load module. As shown, the load module consists of 14 phases divided into three regions. Figure 4-10 illustrates the OVERLAY and REGION control statements required to produce this structure. If this same load module were defined by using only OVERLAY control statements, its structure would be as illustrated in Figure 4-11.



NOTES:

1. The origin of each region is the end of the longest path of the previous region.
2. The root phase is in the path of every region.
3. Up to 10 regions are permitted per load module.

Figure 4-9. Program SAMPLE as a Multiregion Load Module

```
LOADM        SAMPLE   }
  .                   }  GENERATES
  .                   }  PHASE 00
  .                   }  (ROOT PHASE)
OVERLAY      NODE1    }
  .                   }
  .                   }  PHASE 01
  .
OVERLAY      NODE1    }
  .                   }
  .                   }  PHASE 02
  .
OVERLAY      NODE1    }
  .                   }
  .                   }  PHASE 03
  .
OVERLAY      NODE1    }
  .                   }
  .                   }  PHASE 04
  .
OVERLAY      NODE1    }
  .                   }
  .                   }  PHASE 05
  .
OVERLAY      NODE1    }
  .                   }
  .                   }  PHASE 06
  .
REGION       NODE2    }
  .                   }  PHASE 07
  .                   }  AND REGION 2
  .
OVERLAY      NODE3    }
  .                   }
  .                   }  PHASE 08
  .
OVERLAY      NODE3    }
  .                   }
  .                   }  PHASE 09
  .
OVERLAY      NODE3    }
  .                   }
  .                   }  PHASE 10
  .
REGION       NODE4    }
  .                   }  PHASE 11
  .                   }  AND REGION 3
  .
OVERLAY      NODE4    }
  .                   }
  .                   }  PHASE 12
  .
OVERLAY      NODE4    }
  .                   }
  .                   }  PHASE 13
  .
```

NOTE:

The ellipses represent INCLUDE statements for object modules to be included in a particular phase. These statements must follow the LOADM, OVERLAY, and REGION control statements.

Figure 4–10. Control Stream Coding Required to Construct the Multiregion Load Module SAMPLE

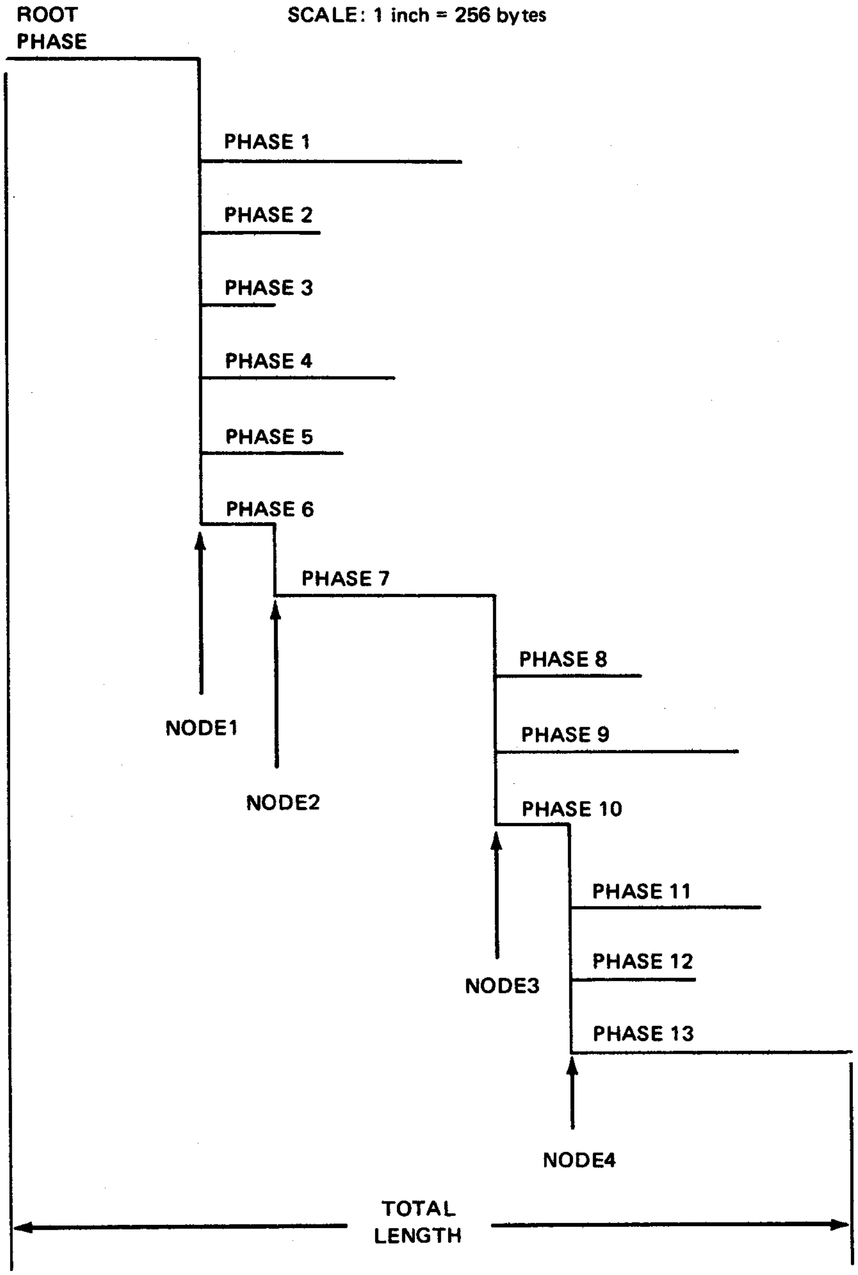


Figure 4–11. Program SAMPLE as a Multiphase Load Module

As can be seen by comparing Figure 4-9 with Figure 4-11, a load module constructed as a multiregion load module normally requires more main storage space for execution than the same program configured as a multiphase load module. Multiregion structures are most useful, however, when a need exists for a phase to reside in an area where it will not be overlaid by other phases that may not be directly associated with it. Also, it is sometimes possible to realize an actual saving in main storage space with a multiregion construction when one or more control sections are required for two or more distinctly separate phases but are not required by any other phases. In this case, these CSECTs could be placed in a separate region, rather than being embedded in a phase common to the phases requiring them. Placing them in a common phase could unnecessarily affect the origins of succeeding phases even though the majority of these phases do not require these CSECTs. The opposite is true when these CSECTs are placed in a separate region; however, inasmuch as region origins always are assigned at the end of the longest path of the preceding region, unnecessary placement of CSECTs in separate regions may have an adverse effect on the overall length of the load module.

Regions are declared by the programmer with the REGION control statement in much the same way a phase is declared with an OVERLAY control statement. Both statements initiate construction of a new phase at some symbolic starting address, or node point, specified in the control statement. The only difference between the two statements is in the way they cause the linkage editor to assign an origin to the phase being created. Region nodes are always logical and never reference a previously defined symbol because a new path is about to be constructed.

4.7. LINKAGE EDITOR OPERATION

When the linkage editor is called upon to perform its intended functions via a // EXEC LNKEDT job control statement, it searches the appropriate control stream data set for control statement data. If control statements specifying the object modules to be included in the load module are found, the linkage editor links these modules together in accordance with the control statements that affect the structure of the load module that also may be in the control stream. If no control statements that identify the object modules to be included are found, the linkage editor links together all the object modules currently in the job's run library file, as previously described.

Under normal operating conditions (no linkage editor capabilities suppressed or altered), the linkage editor performs the following operations for each link-edit job:

- automatically includes any object modules needed to satisfy any references not defined in the object modules specifically included in the load module by the user;

- automatically deletes control sections redundantly included in any nonunique path of a load module;

- determines in what phase each common storage area is best located in the load module;

- automatically includes an automatic overlay control routine and its associated control tables in the load module if it is required for execution of the program;
- resolves multiple definition problems in the load module;
- includes partial object module elements (CSECTs), if specified by the user, in the load module;
- detects object modules that are marked reentrant, deletes their text, and creates appropriate shared records to indicate the reentrant code requirement;
- relocates ISD records detected in object modules and passes them to the load modules being created; and
- produces CSECT/COM ISD records based on the CSECT/COM records being included.

The processing involved in performing these operations is described in the remainder of this section.

4.7.1. Automatic Inclusion Processing

The linkage editor is designed to automatically include object modules in a load module when such modules are needed to satisfy references in the object modules specifically included in the load module by the user. When a reference is found in a specifically included object module for which no definition exists, the linkage editor searches up to two object module library files looking for an object module containing the definition. The library files to be searched are specified by the user through the ALIB and RLIB keyword parameters of the // PARAM or LINKOP linkage editor control statements. If the definition is found, the entire object module containing the definition is automatically included in the load module being constructed. If the required definition is not found, all references to the undefined label are flagged as errors in the link-edit map. All automatically included object modules are placed in the root phase of the load module being constructed.

To facilitate use of the automatic inclusion feature and avoid redundant searches of object module files, library directories contain a cross-reference index that points to label definitions (CSECTs, COMs, and ENTRYs) in the object modules. These file directories are scanned for the required reference definitions, rather than the object module elements. The search for label definitions continues within the directories until all labels are defined or until the entire directory is scanned once without creating new, undefined references. All references that cannot be satisfied are flagged as errors in the link-edit map.

New, undefined references sometimes appear within an automatically included object module or in the object module elements introduced by INCLUDE control statements embedded in an automatically included object module. These embedded INCLUDE statements are processed in exactly the same way as INCLUDE statements embedded in specifically included object modules. Thus, an INCLUDE statement embedded in an automatically included object module could request the partial inclusion of an object module, as well as the inclusion of a complete object module.

All object module elements included in a load module as a result of an INCLUDE statement being embedded in an automatically included object module are placed in the root phase of the load module as part of the automatically included object modules.

If a label is defined in an automatically included object module, that definition is used to satisfy all references to that label appearing in the load module. If a label is defined in a specifically included object module and a second definition of the same label is contained in an automatically included object module, the second definition is ignored by the linkage editor. If a label is defined in two or more specifically included object modules that are in the same phase, the first definition obtained by the linkage editor is used to satisfy all references to the label. Figure 4-12 illustrates these three referencing techniques.

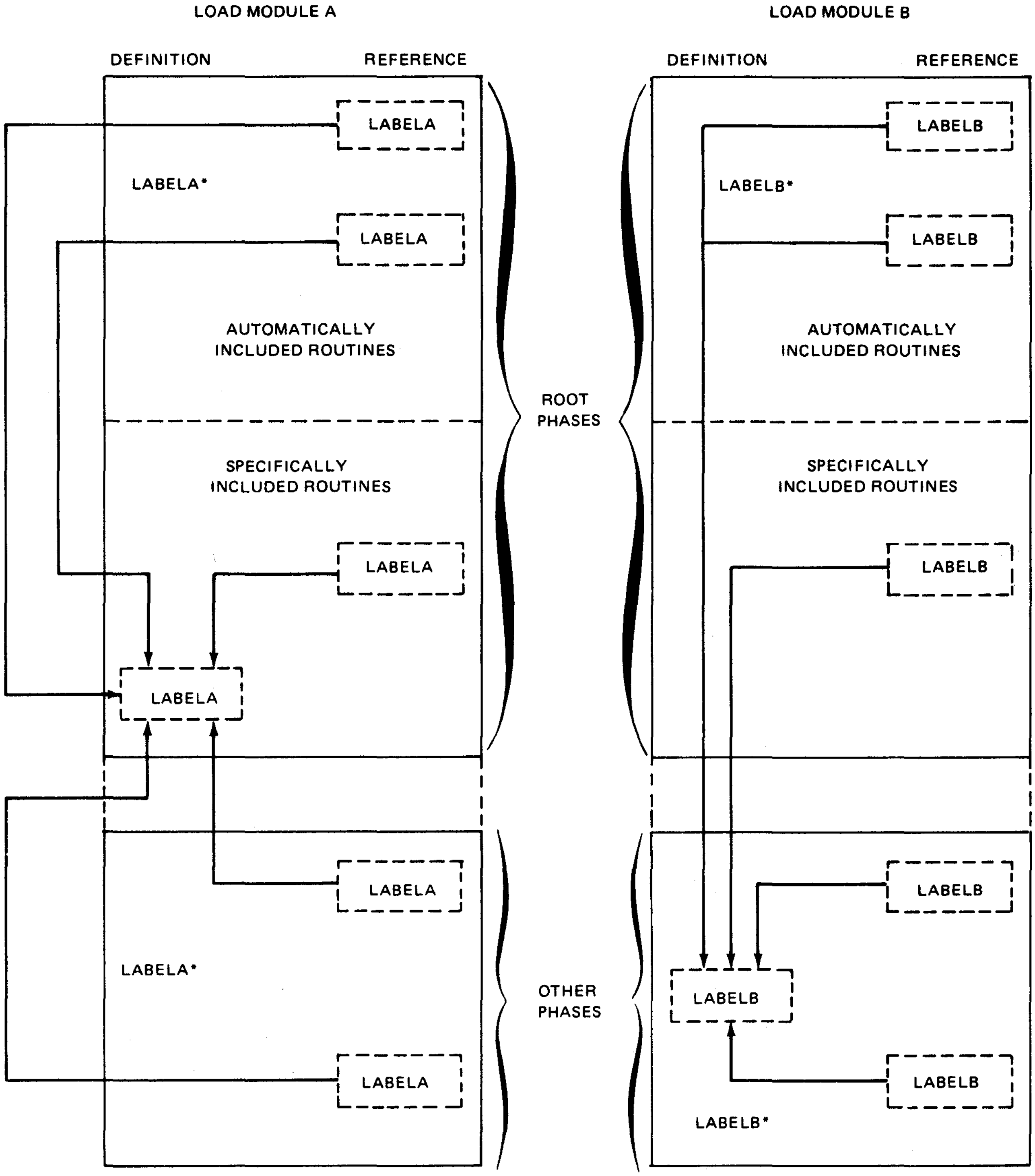
If an automatically included object module contains a control section whose name is already defined by a label in a previously included object module, that control section is not included in the load module even if that control section contains an entry point definition that may be required in the load module.

Another function of the automatic inclusion feature is to construct a single-phase load module from all the object modules in the system job run library ($Y$RUN) when the linkage editor is executed and no control statements defining the load module to be constructed are provided by the user.

The automatic inclusion feature can be inhibited by the user through the NOAUTO keyword of a // PARAM or LINKOP control statement. This capability is provided to enable the user to check the completeness of the definitions within his own program. In this instance, all references to undefined labels are flagged in the link-edit map without an attempt by the linkage editor to satisfy them.

4.7.2. Automatic Deletion Processing

The linkage editor automatically deletes control sections and entry points previously defined in the phase currently under construction, or in a phase that is on the path of the current phase. When a newly included object module is found to contain a CSECT, or an ENTRY point with the same name as a CSECT or ENTRY point previously defined, the linkage editor deletes the second definition from the phase under construction. Unlabeled control sections are not automatically deleted, because each of them is considered a unique entity for lack of a name. Shared definitions are considered to be in the root phase, so any subsequent duplicate definitions will always be deleted. However, nonshared definitions that are accepted before a shared definition is encountered are not deleted. Also, if such definitions are themselves in the root phase, they will cause the automatic deletion of all subsequent definitions, shared or unshared.



*These definitions are deleted by the linkage editor.

NOTE:

The arrows indicate which definition is used to satisfy each reference.

Figure 4–12. Referencing Label Definitions in a Load Module

The only exception to this rule for automatic deletion is when multiple CSECTs are found to have the same name as a labeled common storage area. In this instance, the CSECTs are treated as block data subprograms and are used to initialize the associated common storage area. These multiple CSECTs, however, must be included in the load module after the inclusion of the related common section; otherwise, the second and subsequent CSECTs could be deleted automatically. Common (COM) sections in reentrant object modules are treated as DSECTs. In nonreentrant object modules, unnamed COM sections are never deleted. Deletion of named COM sections is subject to the rule that block-data/common relations are not permitted across nonreentrant/reentrant code boundaries. Thus, if a shared CSECT has been accepted and, subsequently, a nonshared COM with the same name is encountered, the COM section is deleted. On the other hand, if a nonshared COM section has been accepted and a subsequent shared CSECT with the same name is found, the CSECT is deleted.

Thus, if a second definition for a name already defined on the path of the current phase is detected, the second definition is ignored. A single-phase program, therefore, contains only one definition for each label that does not match a common section name. Only one definition is accepted, even if the subsequent definition is for an absolute entry. All absolute entry items are treated as low-priority items and are always deleted whenever at least one duplicate definition exists in the load module.

4.7.3. Common Storage Processing

The linkage editor constructs a common storage area for each unique COM section contained in a load module. COM sections with the same name, however, are assumed to be referring to the same common storage area; therefore, the linkage editor creates a single common storage area for these COM sections. The size of each storage area is equal to the largest size requested by all object module elements referring to a COM section. Thus, if one object module indicates that COM section A requires 256 bytes of main storage, and another object module indicates that COM section A requires 1024 bytes of main storage, the linkage editor creates a single common storage area of 1024 bytes to satisfy all object module references to COM section A. Blank (unlabeled) common storage is allocated in the same way.

The linkage editor assumes that the following rules concerning COM section specification have been adhered to by the object module element programmer:

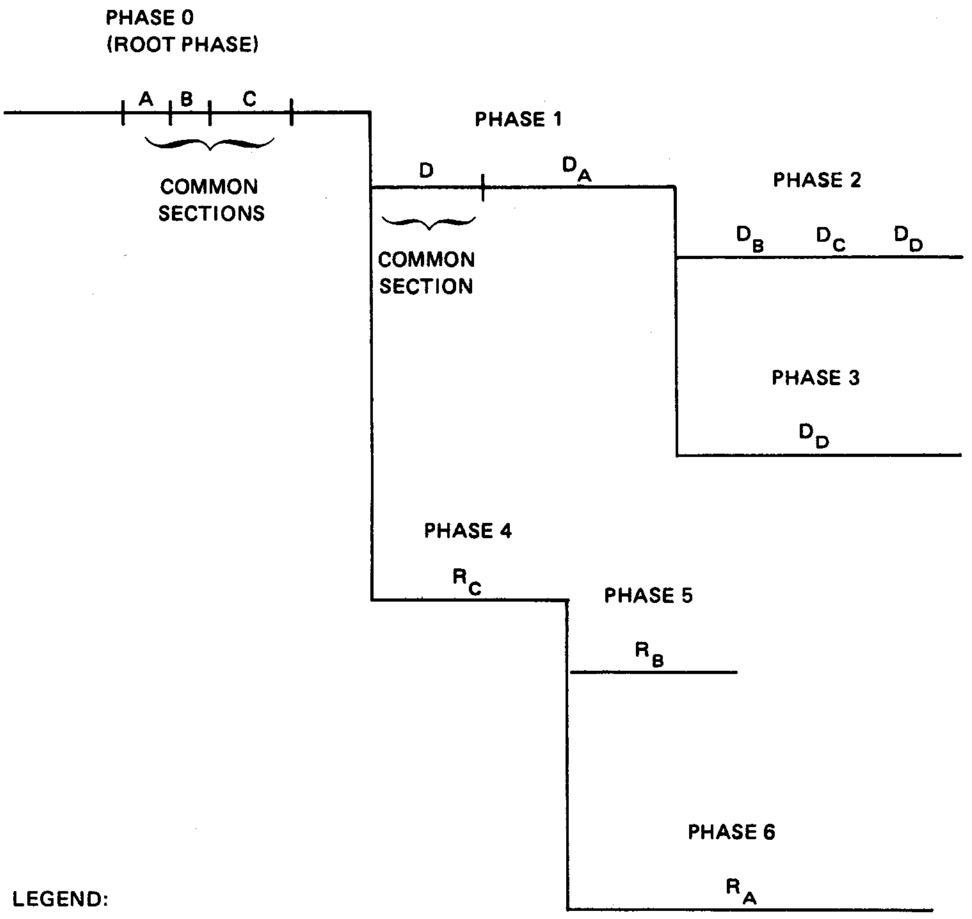
- An entry point in a load module cannot bear the same name as a labeled common storage area included in the load module.

- When a phase containing a CSECT with the same name as a common storage area is loaded for execution, that section is treated as a block data subprogram and is loaded in all or a portion of the common storage area with the same name. (Block data subprograms can be used to initialize common storage areas. Blank common storage areas cannot be initialized during loading unless the text following the common storage area declaration is for that COM ESD.)

- All COM sections defined in an object module are processed by the linkage editor, even if only a partial include of an object module is taking place and the included object module element does not refer to any common storage area.
- COM sections encountered in reentrant object modules are treated as DSECTS. They do not result in any space allocation during the creation of a reentrant load module.
- Automatic deletion of nonshared common sections is subject to rules established for deletion processing.

In the past, most linkage editors placed all common storage areas in the root phase of all load modules, regardless of the structure of the phases requesting access to the common storage area. This allocation scheme forces all phases of a load module to include the space required by all common storage areas in their main storage space requirements, even if they do not use any common storage space. The OS/3 linkage editor, however, is designed to promote the allocation of common storage areas. The promotion of a common storage area refers to its placement within a nonroot phase segment of a load module. The OS/3 linkage editor allocates common storage areas only as required by the structure of the phases referencing them. Figure 4-13 illustrates a load module in which a common section was promoted. As shown, common storage areas A, B, and C are located in the root phase because they are referenced by phases whose only common phase is the root phase. But, because common storage area D is referenced only in phases 2 and 3, it was promoted to the phase 1 segment because this segment also is common to both phases. In this way, phases 4, 5, and 6 do not need to pay for the storage space required by common section D because it will be loaded only when phase 1 is loaded. Thus, this promotion technique allows a load module to require less main storage space than would otherwise be required. It should be noted, however, that common storage area D is overlaid each time phase 4 is loaded, and the use of common storage area D must be planned carefully by the programmer to ensure that required common storage area data is not overlaid before that data is processed. Nonroot phase common sections normally are feasible only when phased programs are executed in a straightforward fashion. For example, let's assume phases 2 and 3 of Figure 4-13 were executed consecutively, followed by the execution of phases 4, 5, and 6, in that order, and the program was concluded without ever having to repeat either phases 2 or 3, common section D would probably fit very nicely in phase 1. On the other hand, if we assume phase 2 was executed and stored data in common section D for use by the phase 3 program segment, but the phase 4 program segment was loaded next, the common storage data for the phase 3 program would be overlaid and its data rendered useless to the subsequent execution of the phase 3 program segment.

This capability of the linkage editor can be enabled and disabled through the PROM and NOPROM keyword parameters of the // PARAM and LINKOP control statements. Thus, to inhibit the promotion and force root phase assignment of common storage areas, the user need only specify the NOPROM keyword in his input control stream to the linkage editor; otherwise, the promotion of common storage is enabled.



LEGEND:

D Definition of a common storage area
R Reference to a common storage area

Figure 4-13. Example of Common Storage Promotion Scheme

4.7.4. Automatic Overlay Control Processing

The linkage editor can automatically load the various phases of a multiphase or multiregion load module in main storage in accordance with the execution requirements of the program. Normally, the loading and execution of the phases of a program is done by the programmer through the object code in each phase of the load module. If the programmer wishes, however, he may leave this responsibility up to the linkage editor for the minor cost of the storage space required to include the automatic overlay control routine and its associated control tables in his load module. The exact amount of storage varies with the number of control table entries required to describe the load module.

When the programmer wishes to have his program phases loaded automatically, he must reference the ENTRYs he wishes to transfer control to with V-CON (V-type address constants) references. Then, when he proceeds to link-edit his program as either a multiphase or multiregion load module, the linkage editor decides whether the automatic loading of phases is required based on:

- the types of EXTRNs (A-type or V-type) contained in the object modules being included in the load module;
- the types of references (inclusive or exclusive) in the load module; and
- the V-CON processing option is effect (V-CON processing is normally enabled).

If the V-CON processing option is in effect and at least one exclusive V-CON reference is in the object code being included in the load module, the linkage editor constructs a set of tables depicting the load module structure and includes them in addition to an automatic overlay control routine in the root phase of the load module. Also, it modifies the V-CON references in the load module to make them reference the automatic overlay control routine. The automatic load routine (KL$OCP or KL$OCPR) that is copied into the load module depends on the number of regions comprising the load module. The KL$OCPR routine is capable of handling the automatic overlay requirements of a multiregion load module; KL$OCP is not.

When a load module containing the automatic overlay facilities is executed, its root phase is loaded in main storage, as is any other load module, by the supervisor, and program control is transferred to the program entry point for the phase. As program execution proceeds, V-CON references cause the needed phases to be loaded, thus satisfying the V-type address constant that referenced the label of the required entry point. A branch instruction is executed through register 15, forcing transfer of control to the overlay control routine because the V-type address constant was previously modified by the linkage editor to reference the routine. The automatic overlay routine then checks to see whether the required phase is in main storage. If it is, the overlay control routine branches directly to the proper address. If it is not, the automatic load routine:

- determines the phases currently in main storage;
- loads the required phase, and any phase on the path of the required phase, into main storage;
- records the new path as loaded; and
- branches to the proper address.

The overlay control routine executes LOAD macroinstructions as needed to obtain the required phases and records this information in its control tables. The path information is examined by subsequent invocations of the automatic control mechanism and is altered if additional phases are loaded. Attempts made by the problem program to load phases directly do not cause an update of the path information; therefore, you would not issue any LOAD, LOADR, or FETCH macroinstructions in your problem program by using the automatic overlay control routine.

The V-type address constant in register 15 for the automatic loading of overlays has the following characteristics:

- It always occupies four bytes; the entire word is reserved for use by the linkage editor. Byte 0 of this word contains a number (zero or greater) that is an index into the list of NTAB entries. Bytes 1-3 consist of the address of the entry point table; however, a V-CON may have all zeros.

- It is intended for branching only and may not be used for addressing data. Data references do not cause the referenced phase to be loaded automatically because transfer of control is essential for the process to be effective.

- When a V-type address constant addresses a definition that is on the same path as the reference, the constant is treated as though it were a constant of type A. If such a constant is in register 15, the problem program branches directly to the required location, rather than transferring control to the overlay control mechanism.

The data required by the automatic overlay control routine to perform these functions is contained in its associated control tables. Brief descriptions of the components of the automatic overlay mechanism follow.

4.7.4.1. Overlay Control Routine

The overlay control routine is a program used for automatic loading of segmented program structures. It is responsible for loading the program segments (phases) and maintaining the tree structure of the program. It supplies the needed interface and controls to allow you to use segmentation without difficulty. It causes automatic loading of the needed program phases as determined by exclusive V-CON references. An automatic entry point (KL$OCP) always is supplied to allow references to the standard automatic overlay control routine. A second routine (KL$OCPR) is used if the load module structure comprises multiple regions. The control routine is always placed in the root phase of a load module and is entered through the entry point table.

4.7.4.2. Entry Point Table (NTAB)

The NTAB table also is always in the root phase. It front-ends the overlay control routine and contains such control information as: The path being automatically loaded, where the phase is being loaded (or its relative position in main storage), and a list of entry points corresponding of the V-CON definitions in the problem program. The NTAB determines the phase to be loaded when a V-CON reference refers to a phase not in the same path as the reference. An NTAB entry is not created for any V-CON symbol already present in a phase higher in the current path (closer to the root phase). An automatic entry point (KL$NTB) always is supplied to allow references to NTAB.

4.7.4.3. Phase Table (PTAB)

The phase table contains information concerning the relationships between the phases of the load module. Only one phase table is built for any multiphase load module that requires automatic loading, and it is always in the root phase. An automatic entry point (KL$PTB) always is supplied to allow references to PTAB.

4.7.4.4. Region Table (RTAB)

The region table is generated only for load module structures using V-CON references in which multiple regions are involved. This table describes the number of regions making up the load module and the highest phase contained in each region. The RTAB is used in conjunction with NTAB and PTAB, plus the region overlay control routine (KL$OCPR), to manage V-CON references in load modules with two or more regions. An automatic entry point (KL$RTB) always is supplied to allow references to RTAB.

4.7.5. Multiple Definition Resolution Processing

The multiple definition resolution processing performed by the linkage editor depends on the type of definition being referenced.

4.7.5.1. Standard (Non-V-CON) References

If the user references a definition with a standard reference and the definition being referenced is not on the path of the reference, he is required to issue the appropriate supervisor FETCH and LOAD macroinstructions to ensure that the phase containing the proper definition is loaded when the definitions it contains are referenced. (The linkage editor allows standard EXTRN - ENTRY relationships to occur across phases without any special processing when V-CONs are not involved.)

Because of the automatic deletion mechanism of the linkage editor, only one definition will ever be present on each path of a load module. If a definition is present in the root phase, no identical definitions will be in other phases unless the path involved overlays the root phase entirely.

If there is only a single definition for a reference, the linkage editor obtains the appropriate phase number and value for the definition and applies it accordingly. But, when a reference is marked as being multiply defined, each phase number assigned to each definition is, in conjunction with the segment table, used to determine the relationship between definitions and references. The linkage editor then satisfies the reference in accordance with the following logic:

1. If a definition that satisfies a reference is in the same phase as the reference, the linkage editor uses it to satisfy the reference.

2. If a definition that satisfies a reference is on the path of the reference, the linkage editor uses it to satisfy the reference.

3. The linkage editor determines whether the reference is on the path of one or more definitions that can satisfy the reference. If it is, the linkage editor chooses the first definition following the reference to satisfy the reference.

4. The linkage editor chooses the first definition following the reference (higher phase number), regardless of path associations, to satisfy the reference if one exists.

5. The linkage editor uses the first backward definition (lower phase number) to satisfy the reference.

Figure 4-14 illustrates these five reference-definition relationships. Further, wherever the linkage editor must satisfy a reference in accordance with C, D, or E, it is indicated on the link-edit map.

4.7.5.2. V-CON References

The resolution processing for V-CON references is much the same as that for standard references, except that:

- A V-CON reference may be converted to an A-CON reference.
- A V-CON reference forces control to the automatic overlay control mechanism.
- Exclusive V-CON references do not initiate diagnostic messages.
- A V-CON reference resolved to a shared definition is converted to an S-CON (shared constant).

Thus, at resolution time, multiple definitions for any referenced symbol are handled as follows:

1. If a definition is on the path of the reference, that definition is selected by the linkage editor to satisfy the reference. (Only one inclusive definition is possible.)

2. Otherwise (definition is exclusive of the reference), the first forward definition is chosen by the linkage editor if one exists. If none exist, the first backward definition is chosen.

When an object module incorporating V-CONs is being included, V-CON processing will include the following. Whenever a V-CON reference occurs, the linkage editor keeps track of the first such reference since the last definition. For multiply-defined V-CONs, separate NTAB entries may be constructed, as references are assigned to different definitions of the same symbol. This selection of definitions is a function of where the appropriate references occur with respect to path relationsips concerning those definitions.

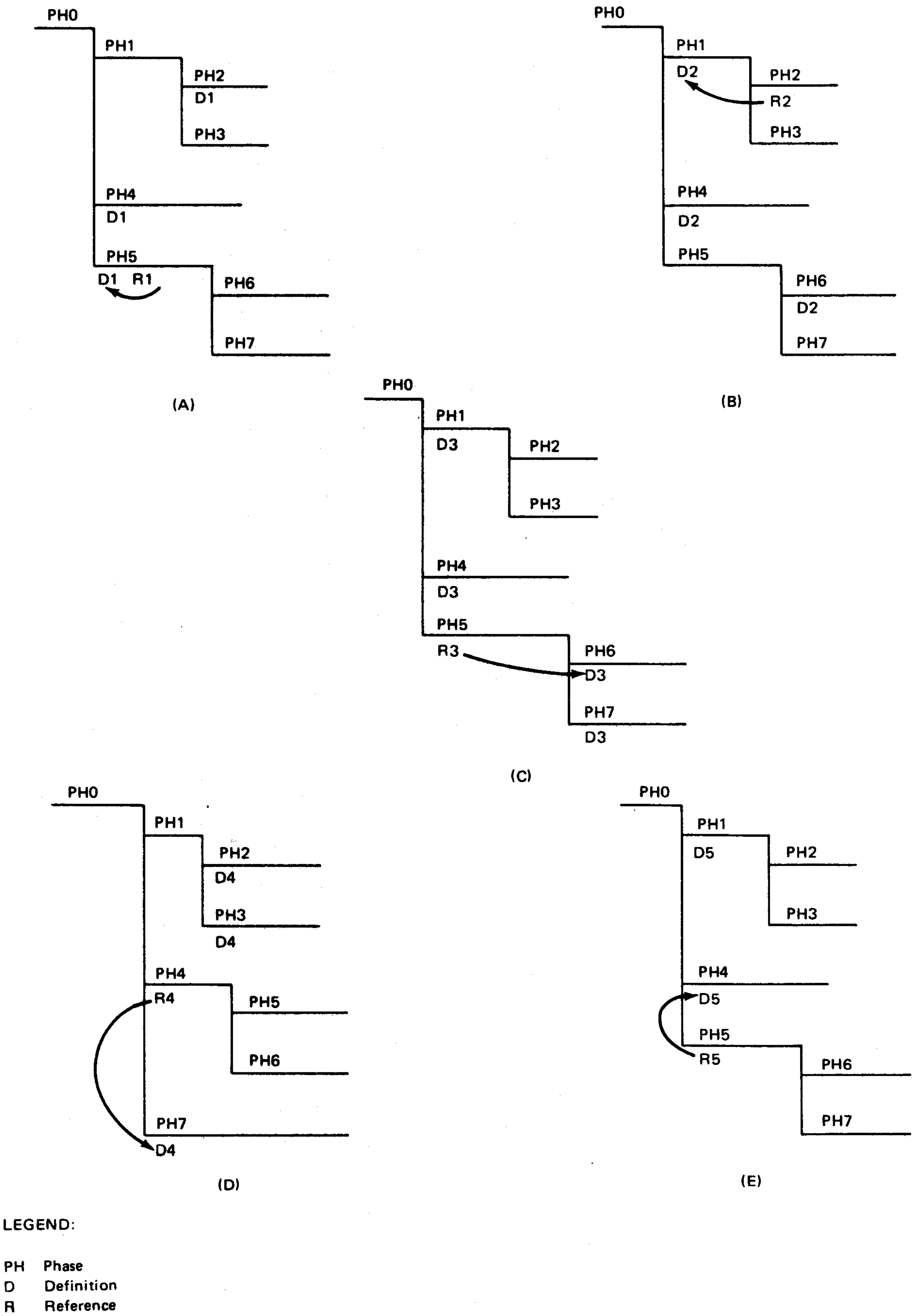


Figure 4-14. Multiple Definition Resolution without V-CON References

Multidefinition V-CON processing occurs as follows:

1. If a V-CON definition is on the path of the V-CON reference (or references), that reference will be treated as a direct A-CON reference and will be flagged accordingly.

2. If a V-CON reference is on the path of a V-CON definition, it is a valid V-CON reference.

3. If a V-CON reference is on the path of multiple definitions, the automatic deletion mechanism will eliminate the subsequent definitions (where such definitions are contained in the same path or phase) and the first definition will be used to satisfy that reference.

4. If V-CON references and multiple definitions exist on separate paths, the first forward definition will be used if there is one; otherwise, the first backward definition is used.

5. If V-CON references and multiple definitions are on separate paths but in the same direction (forward or backward), the first definition encountered is used.

Figure 4-15 illustrates these five reference-definition relationships.

4.7.6. Partial Include Processing

The linkage editor can include only specific CSECTs of an object module in a load module, if you so desire. You specify these CSECTs in INCLUDE control statements that contain a CSECT name list as its second operand, as well as the name of an object module as its first operand. Up to nine control sections can be specified in any one INCLUDE statement.

When CSECTs are specifically referenced in an INCLUDE statement, they are inserted in the load module in the order that they appear in the object module, regardless of the order in which they are specified in the INCLUDE statement. The rearrangement of CSECTs in a load module, however, can be accomplished by multiple INCLUDE statements, each of which references only a particular CSECT. In this way, the INCLUDE statements can be arranged to cause the resulting load module CSECT structure to assume any form desired. The smallest segment of object code that may be included in a load module is a CSECT.

Whenever an object module is accessed for a partial inclusion of its elements, any control statements that may be embedded in the object module are processed by the linkage editor as standard, embedded control statements, and any common sections defined in the object module are included in the load module.

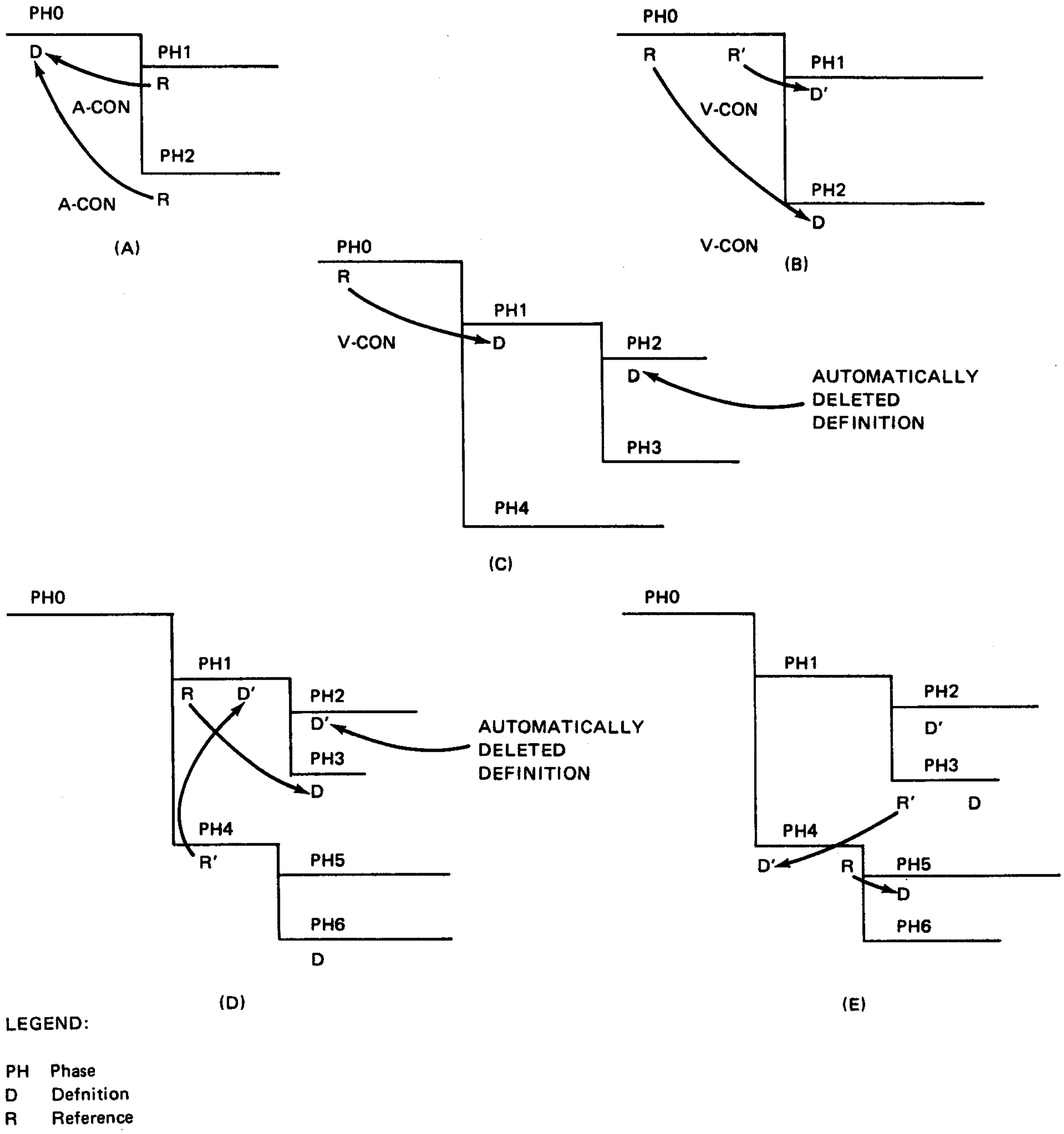


Figure 4-15. Multiple Definition Resolution with V-CON References

4.7.7. Shared Code (Reentrant Code) Processing

Shared or reentrant code is a piece of code that is not self-modifying. Thus, a number of programs can call upon a single copy of this code and run concurrently. Sharing code reduces the main storage requirements of the system. Consider two programs, X and Y, that call upon a reentrant routine, Z. It is obvious from Figure 4-16 that sharing Z allows both programs to be executed in less main storage space than would otherwise be required.

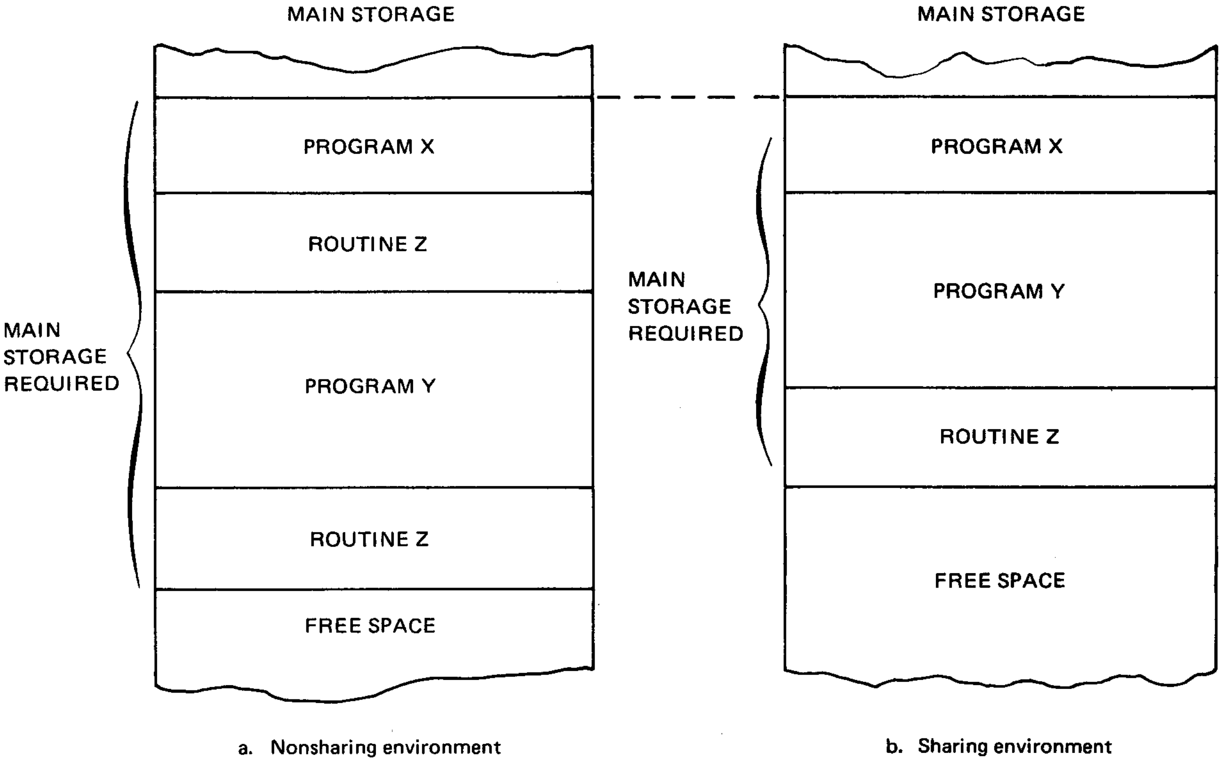


Figure 4-16. Effect of Shared Code on Main Storage Requirements

The librarian is capable of marking object modules reentrant by setting a flag in the object module header record. Unless otherwise marked, object modules are nonreentrant. The flag can be turned on or off by the librarian RENAME control statement.

It is the responsibility of the linkage editor to detect reentrant object modules, delete their text, and create the appropriate interfaces to enable linkages to be established at execution time. This feature can be enabled or disabled through options on // PARAM or LINKOP cards.

Since the reentrant code itself does not appear in the load module produced, it must be link-edited separately. This is done in a special mode of the linkage editor through // PARAM - LINKOP cards.

Thus, there are three processing modes controlled by // PARAM – LINKOP optons:

1. Normal link-edit with no recognition of reentrant object modules (NOSHARE, NORNT).

2. Link-edit with sharing capability enabled (SHARE option).

3. Link-edit of a reentrant module by itself (RNT option).

4.7.7.1. Share Facility

The linkage editor detects reentrant object modules under either of the following conditions:

1. An INCLUDE control statement that was supplied referenced a specific object module, which, when located, was found to be marked reentrant.

2. As a result of an unresolved EXTRN reference in the user module being link-edited, the automatic include mechanism was triggered. When the definition was detected, the object module containing the definition was found to be marked reentrant.

Provided the share processing facility is enabled, the linkage editor, upon encountering the reentrant object module, performs the following:

1. The text from the object module (or part, if partial include) is dropped and does not contribute towards the load module being produced.

2. The CSECT and COM lengths in the object module are not reflected in the load module size.

3. COM sections and ISD records are ignored.

4. All ESD items (except COMs) are entered into the linkage editor's internal symbol table and are processed normally.

5. Shared definitions (definitions encountered in the reentrant object module) are treated as if they occurred in the root phase and are subject to automatic deletion processing rules. This includes block data/common conflicts between reentrant and nonreentrant code.

6. A special record (the resource record) is created in the load module. This record contains the name of the object module. At program execution time, it informs job control of the requirement for a resource, namely the reentrant module.

Satisfying shared resource requirements is a part of the total job scheduling process and is performed before actual program execution begins.

4.7.7.2. Linkage in Shared-Code Environment

In the course of a link-edit with the share facility enabled, the linkage editor distinguishes between the different types of EXTRNs, as shown in Figure 4–17. Also, Figure 4–17 shows that reentrant code cannot call nonreentrant code.

Each SEXTRN causes the creation of a SEXTRN record in the load module. This record contains the SEXTRN name and a unique number assigned to the SEXTRN called the SINDEX number. The SEXTRN record is used to link the load module with the reentrant code at execution time.

EXTRN Found in ↓	Resolved To →	Nonreentrant Object Module	Reentrant Object Module
Nonreentrant object module		Allowed. Normal EXTRN	Allowed. SEXTRN (shared EXTRN)
Reentrant object module		Not allowed, unless the definition is absolute	Allowed provided requirements in 4.7.7.4 are met

Figure 4–17. EXTRN Resolution Processing in Shared-Code Environment

4.7.7.3. Shared Constants

A-type or V-type address constants associated with SEXTRNs are known as shared constants (S-CONs). You can transfer control to shared code by loading such a constant into register 15 and branching to it. Only type 1 linkages are allowed for calls from nonreentrant to reentrant code. Register 13 must be loaded with the address of an 18 full-word save area.

The shared constant in register 15 has the following characteristics:

1. It always occupies four bytes; the entire word is reserved for use by the linkage editor. Byte 0 of this word contains an index - called the SINDEX (shared INDEX) number - which is a unique number for every SEXTRN. Bytes 1-3 consist of the address of the preamble SEXTRN processor. This address is negative since the SEXTRN processor resides in the prologue.

2. It is intended for branching only and may not be used for addressing data. The SEXTRN processor, after receiving control, converts the SINDEX number into an absolute address and branches to it. It uses the save area and user registers for its own housekeeping, and then restores them before branching to reentrant code that must have its own mechanism for saving registers if it needs to.

3. It must be coded as a symbol, not as an expression. For example, V(X) and A(Y) (where Y is declared as an EXTRN) are valid shared references, V(X+4) is not.

4.7.7.4. Link-Editing Reentrant Code

In order for reentrant code to be loaded, it must be link edited. This is done in a special mode of the linkage editor (RNT option on // PARAM – LINKOP cards). The link-edit of reentrant code has the following special characteristics:

1. No overlay structures are permitted.

2. The load module name may be up to eight characters.

3. COM sections are treated as DSECTs. No storage allocation is performed for them.

4. For each relative definition (CSECT, ENTRY, linkage editor EQU), a SENTRY record is created in the load module. This record informs job control at execution time that a definition is available in the module and can be used for establishing linkages. No SENTRY record is produced for absolute ENTRYs and EQUs. The relative definitions are known as SENTRY (shared ENTRY) items.

5. Only specific INCLUDEs are meaningful and the text encountered during the specific include process contributes to the load module. All references in the included object modules must be inclusive; that is, the reentrant load module must be self-contained. Option NOAUTO should be in effect in the course of the link edit.

6. All address constants must be absolute. The code must relocate them if necessary.

7. Normally, only one reentrant object module would be specifically included. The linkage editor will permit more than one reentrant object module to be specifically included, provided the user program directly references only one of the object modules being included. Stated another way, there must be only one object module that is at the highest level of the intercalling sequence within the group of reentrant object modules being included. The name on the LOADM control statement must be the same as the name of the highest level reentrant object module.

8. The reentrant load module must be output on the system load library ($Y$LOD).

9. The ISD records in the object module are ignored.

4.7.7.5. Shared Records

The linkage editor creates special records in load modules that reference reentrant code and also in the reentrant load module. These records provide information on linkages to be established and the reentrant modules that are required.

Figure 4-18 shows the format of a nonreentrant user load module that references reentrant modules. A flag is set in the phase header record, indicating that shared code is referenced. For each reentrant module referenced, a resource record is produced. This record contains the name of the reentrant load module and a unique number called the resource number. For each EXTRN in your module resolved to a definition (CSECT/ENTRY) in the reentrant module, a SEXTRN record is produced containing the CSECT/ENTRY name and number of the resource that contains the definition. If an EXTRN is resolved to an ENTRY which has the same location as its CSECT, the reference is considered being made to the CSECT. Therefore, the SEXTRN record contains the CSECT name rather than the ENTRY name. However, several SEXTRN records can be created for each resource record, depending upon the number of definitions referenced in the resource (reentrant object module). If your module references other reentrant modules, additional sets of resource and SEXTRN records will be created.

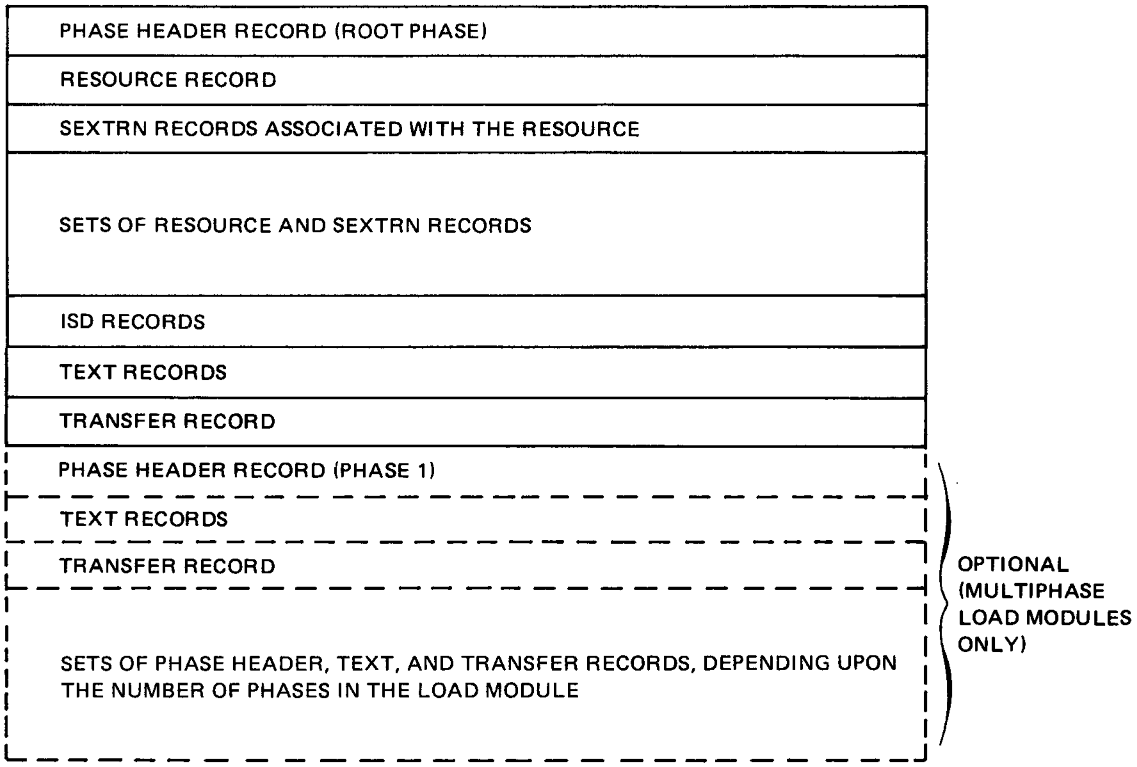

PHASE HEADER RECORD (ROOT PHASE)
RESOURCE RECORD
SEXTRN RECORDS ASSOCIATED WITH THE RESOURCE
SETS OF RESOURCE AND SEXTRN RECORDS
ISD RECORDS
TEXT RECORDS
TRANSFER RECORD
PHASE HEADER RECORD (PHASE 1)
TEXT RECORDS
TRANSFER RECORD
SETS OF PHASE HEADER, TEXT, AND TRANSFER RECORDS, DEPENDING UPON THE NUMBER OF PHASES IN THE LOAD MODULE
OPTIONAL (MULTIPHASE LOAD MODULES ONLY)

Figure 4-18. Format of a Nonreentrant Load Module That References Shared Code

The format of a reentrant load module is shown in Figure 4-19. The phase header is marked with a flag, indicating that it is a reentrant load module. Relative definitions (CSECTs, ENTRYs, and EQUs) detected in your link-edit are called SENTRY (shared ENTRY) items and produce SENTRY records. However, a SENTRY record is not produced for a COMMON, an absolute ENTRY, an absolute EQU, or a relative ENTRY at the same location as its CSECT. A SENTRY item contains the SENTRY name, its link origin, and a unique number called the SENTRY number. No ISD records are produced in a reentrant load module. See Figure 4-19 for the format of a SENTRY record.

PHASE HEADER RECORD (ROOT PHASE)
SENTRY RECORDS
TEXT RECORDS
TRANSFER RECORD

Figure 4-19. Format of a Reentrant Load Module

When your load module is executed, its resource and SEXTRN records are examined to determine reentrant code requirements for the module. An attempt is made to resolve SEXTRNs against reentrant code already loaded. If the required reentrant code is not already loaded, $Y$LOD is searched for the modules named in the resource records. When a required module is found, a match is attempted between the SEXTRNs in your module against the SENTRYs in the reentrant load modules. Linkages are then established based upon the matches, and the appropriate reentrant modules are loaded. Shared constants themselves remain unchanged. Satisfying shared resource requirements is a part of the total scheduling process and is performed before actual program execution begins.

Consider the following example. Your object module USER has external address constants R2 and T1 and a V-type address constant T3. A reentrant object module R1 exists with CSECT R1 and entry points R2 and R3. Another reentrant object module, T1, has CSECTs T1 and T2 and entry point T3. Figures 4-20 through 4-23 depict the outputs of the link-edits of USER, with the NOSHARE and SHARE options specified, and the link-edits of R1 and T1, with the RNT option specified, respectively.

PHASE HEADER RECORD FOR USER
TEXT RECORDS (OBJECT CODE FROM USER, R1 AND T1)
TRANSFER RECORD

Figure 4-20. Link-Edit of USER with NOSHARE Specified

PHASE HEADER RECORD FOR USER
RESOURCE RECORD FOR R1
SEXTRN RECORD FOR R2
RESOURCE RECORD FOR T1
SEXTRN RECORD FOR T1
SEXTRN RECORD FOR T3
TEXT RECORDS (OBJECT CODE FROM USER)
TRANSFER RECORD

Figure 4-21. Link-Edit of USER with SHARE Specified

PHASE HEADER RECORD FOR R1
SENTRY RECORD FOR R1
SENTRY RECORD FOR R2
SENTRY RECORD FOR R3
TEXT RECORDS (OBJECT CODE FROM R1)
TRANSFER RECORD

Figure 4–22. Link-Edit of R1 with RNT Specified

PHASE HEADER RECORD FOR T1
SENTRY RECORD FOR T1
SENTRY RECORD FOR T2
SENTRY RECORD FOR T3
TEXT RECORDS (OBJECT CODE FROM T1)
TEXT RECORDS (OBJECT CODE FROM T2)
TRANSFER RECORD

Figure 4–23. Link-Edit of T1 with RNT Specified

Notice that, in Figure 4–21, no SEXTRN records were created for CSECTs R1 and T2 or ENTRY R3. This is because USER does not require these definitions to satisfy any of its external references.

4.7.8. Internal Symbol Dictionary (ISD) Processing

The internal symbol dictionary (ISD) records describe your program symbols. These records can appear in either object or load modules. When the ISD records appear in object modules, they are generated by certain language processors; while in load modules, the linkage editor generates them.

As just mentioned, the ISD records are used as descriptor records and, therefore, do not increase the size of the object or load module. When they appear in a load module, they are not loaded with the records at execution time, so no additional main storage is required. However, the ISD records do play an important role if your program has an abnormal termination and the JOBDUMP option was specified on the OPTION job control statement. In this case, JOBDUMP reads the ISD records in the load module and:

- prints the dump segmented by the CSECTs of the user program; and

- prints the user-defined symbols and tables. The dump produced is formatted and will include compile and link origins, source line number of the symbol, data type, and value.

You can inhibit the generation of ISD records in your load module by using the linkage editor option NOISD on the // PARAM or LINKOP control statement. If you inhibit the ISD record generation and an abnormal termination occurs in your program, then JOBDUMP will produce an unformatted dump of the load module area.

4.7.8.1. Object ISD Records

The object ISD records are generated in your object module by certain language processors. The linkage editor processes these records so that if an abnormal termination occurs while executing your load module, JOBDUMP can give you a formatted dump displaying user program symbols and their attributes. There are two types of object ISD records.

- Type 3 ISD record

  This record describes user-defined and compiler-generated symbols. The type 3 record contains information such as symbolic name, source line number of the symbol, level number, data type, and compile origin of the symbol.

- Type 4 ISD record

  This record contains English text and is used by JOBDUMP to print titles and headings in the dump.

4.7.8.2. Load ISD Records

The load ISD records are generated in your load module by the linkage editor provided the ISD option on either the // PARAM or LINKOP control statement was specified. The ISD record generation is based on the following:

- The presence of object ISD records in the object module. The linkage editor relocates their addresses and passes them into the load module.

- CSECTs/COMMONs accepted during the link-edit job

- ISD records are not produced for the link-edit of a reentrant load module (RNT parameter).

- If you specified the SHARE parameter, the object ISD records in an included reentrant object module are ignored. Also, CSECTs and COMs in the reentrant object module do not result in the generation of either type 1 or type 2 ISD records.

- Deleted CSECTs or COMs do not contribute toward the generation of load ISD records.

- Object ISD records belonging to deleted CSECTs or COMs are ignored.

There are four types of load ISD records:

- Type 1 ISD record

  This record describes a CSECT. It contains the CSECT name, compile and phase relative origins, size and phase number.

- Type 2 ISD record

  This record describes a COMMON section. It contains the COMMON name, compile and phase relative origins, size and phase number.

- Type 3 ISD record

  This record is the relocated form of the type 3 object ISD record.

- Type 4 ISD record

  This record is the relocated form of the type 4 object ISD record.

These records are not loaded at execution time but are used only if an abnormal termination occurs in your program. JOBDUMP, if specified, uses the type 1 and type 2 ISD records to segment the dump by CSECT and COMMON sections. JOBDUMP uses type 3 and type 4 ISD records to print user program symbols and values converted to their proper data types.

4.7.9. User Program Switch Indicator (UPSI) Setting

The linkage editor prints the UPSI byte settings, along with the error count, at the end of the link-edit map. When external references remain unresolved, the linkage editor sets the UPSI byte to X'20'. Also, when the linkage editor issues an error message, the UPSI byte is set accordingly. The linkage editor error messages are found in the system message programmer/operator reference, UP-8076 (current version).

5. Programming Considerations

5.1. GENERAL

Because the allocation of main storage is a primary concern in a multiprogramming system, you should always consider the advantages of having the linkage editor construct this program either as a multiphase or multiregion load module. Further, you should consider this aspect of your program before you begin to code it because the use of certain object code facilities, such as common storage sections, enhance the ability of a program to be structured.

This section concerns itself with the considerations involved with structuring a load module. To repeat, however, the most important factor in the construction of a load module is how its object code segments are coded, and whether these segments take advantage of the capabilities of the linkage editor.

5.2. OVERLAY STRUCTURES AND DEPENDENCIES

Programs can be overlaid to minimize their main storage requirements. The possibility of constructing an overlay program depends on the relationships between the various control sections and phases to be created. These can overlay each other only if they do not need to be in storage at the same time and do not reference each other, either directly or indirectly.

The phase segments of an overlay program must be organized in an overlay tree structure. The following factors should be considered when organizing the tree structure of a load module:

- phase and control section dependency:

- the length of the multiphase program;

- the frequency of usage of each control section; and

- the possibility of using separate overlay regions.

To begin building an overlay structure, the user should first form a root phase that contains the modules that will receive control from the start of execution and those which should always remain in main storage. The rest of the structure should then be developed in accordance with the information presented in the following paragraphs.

5.2.1. Phase Dependencies

Whenever a phase is in main storage or is being loaded in main storage, all the phases in its path also should be in main storage. Phases may be loaded in any sequence whatsoever and reloaded any number of times, as required by the logic of the program. The assigned location of the phases has no bearing on the order in which the phases are executed. Any part of a phase that is modified during its execution will remain so only until the phase is overlaid.

5.2.2. Control Section Dependencies

A control section can receive control from another control section but both sections must be in main storage before execution can continue beyond a given point in the program. The requirements of a control section for a given routine in another control section determine such dependency. Conversely, that control section is dependent upon any other control section from which it can receive control or where the other section has a need to process data of the former section.

5.2.3. Program Length

You must consider the length of a multiphase program in the construction of your program. The length of the longest path is the minimum storage requirement for a multiphase program. Also, when a program is constructed with the automatic overlay mechanism, the storage requirements of the necessary control routine, entry table, phase table, and possibly region table also must be considered.

5.2.4. Phase Origins and Node Points

The origin of the initial, or root phase, segment is assigned by the linkage editor at zero. The relative node point of each phase is determined as zero plus the length of all phases in the path. The characteristics of the first symbol in the operand field of an OVERLAY statement designate the phase origin and node point.

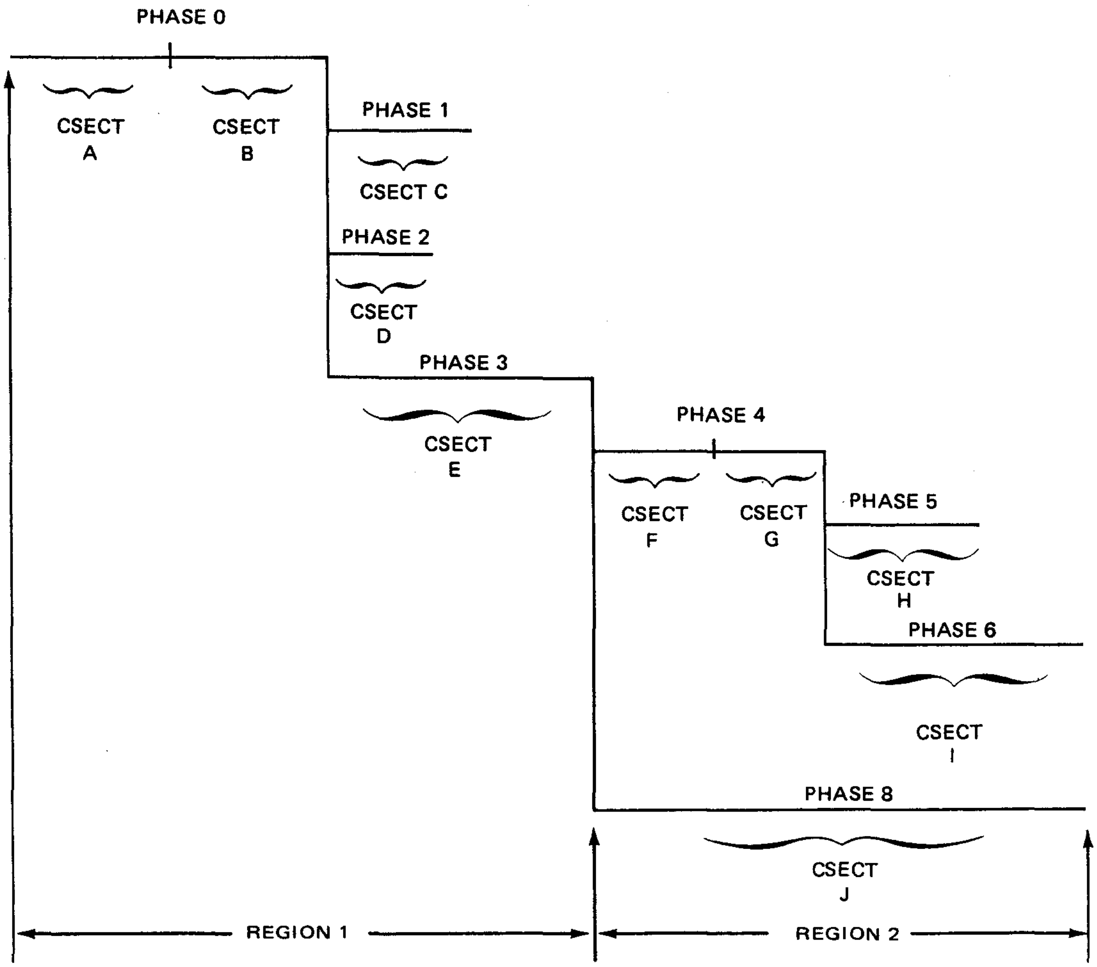
5.2.5. Use of Multiple Regions

Multiple region structures can, in certain instances, capitalize on loading efficiency and realize a saving in main storage space. Phases not on common paths can access each other and, where identical copies of one or more CSECTs are required at different times by different exclusive overlays, they may reside in a separate region not affected by the path loading in the opposite region. Region structures also decrease main storage needs when the creation of a common inclusive phase is not feasible or possible without increasing the length of the longest path of the current region. Such structures can provide a useful and convenient method of load module structuring where such concerns are paramount.

Figure 5-1 illustrates a program whose structure as a multiregion load module results in a saving of main storage space, given the phase and CSECT dependencies noted in the illustration. As shown, if CSECTs F, G, and J were root-phase resident, the load module storage needs would incease because these CSECTs could no longer overlay each other. Creation of two additional phases in region 1, each containing the exclusive CSECTs, would not appear possible because a path conflict would be created. Thus, the region structure shown is the more feasible.

If CSECTs F, G, H, I, and J were required by phases 1, 2, and 3, and if they were required at the same time, so that they could not be overlaid, root-phase residence would be practical.

If CSECTs F, G, H, and I were needed by phases 1 and 2, and CSECT J only needed by phase 3, another common node point and phase could be constructed in region 1. In this case, a second region, as shown, would adversely affect the amount of storage required for the load module.



NOTES:

Assumed program logic dictates the following:

1. Phases 1, 2, and 3 can overlay each other.
2. Phases 1 and 3 require CSECTs F, G, and J.
3. Phase 2 requires CSECT J.

Figure 5-1. Example of a Program Structured as a Multiregion Load Module

6. Control Statements

6.1. GENERAL

The linkage editor control statements described in this section direct the construction of a load module for the SPERRY UNIVAC Operating System/3 (OS/3) from specific or implied object modules and object module elements. Linkage editor control statements normally appear in an OS/3 job control stream, as illustrated in Figure 6-1. However, linkage editor control statements also may be contained in a source module or embedded (nested) in the object modules called by the linkage editor INCLUDE control statements. The rules for coding and embedding linkage editor control statements follow.

```
1        10    16
_________________________________________
// EXEC LNKEDT
// PARAM
/$
            .
            .
            .    LINKAGE EDITOR CONTROL
            .    STATEMENTS COMPRISING
            .    LINKAGE EDITOR CONTROL
            .           STREAM
            .
/*
```

Figure 6-1. Typical Linkage Editor Control Stream

6.2. CODING FORMAT

The general format of a linkage editor control statement is the same as that of an OS/3 assembler statement (Figure 6-2). The label field begins in column 1, is terminated by a blank column, and may contain up to eight alphanumeric characters. The label field is blank for all linkage editor control statements except the equate (EQU) statement. The operation field, which must be preceded and followed by at least one blank column, contains the operation code of the function to be performed. If the label field is blank, the operation field may start in column 2. The operand field begins with the first nonblank character following the operation field, is terminated by a blank column, and cannot extend beyond column 71. The operand field may contain any number of operands, depending on the function to be performed, and operands must be separated by commas. Continuation statements are not allowed.

1 LABEL	△OPERATION△	OPERAND 71	72 8Ø
Blank (1- to 8-character string for EQU only)	Must be delimited by blank columns	Cannot extend beyond column 71 continuation statements are not allowed:	Not used

Figure 6-2. General Linkage Editor Control Statement Format

Comment statements, identified by an asterisk in column 1, may appear anywhere in a linkage editor control stream. Comments do not initiate any processing by the linkage editor but are printed out on the process map portion of the link edit map.

6.3. PLACEMENT OF CONTROL STATEMENTS

In keeping with their functional uses, the following guidelines apply to the placement and sequencing of control statements:

- A LINKOP, LOADM, OVERLAY, or REGION control statement may be followed by an INCLUDE, EQU, MOD, or RES statement.

- An INCLUDE statement must be followed by at least one object module header record and then by any number of embedded control statements, followed by a transfer record and, again, any number of embedded control statements. Any linkage editor control statement may then follow the INCLUDE statements.

- An ENTER statement may not be followed by an INCLUDE statement.

- The LOADM statement normally is the first control statement in a control stream.

- The ENTER statement normally is the last control statement in each phase of a control stream.

- Embedded control statements may be placed before or after the control sections in an object module, but may not be placed within a control section.

- Embedded control statements that affect load module structure (LINKOP, LOADM, OVERLAY, REGION, and ENTER) cannot be embedded in an automatically included object module.

6.4. EMBEDDED CONTROL STATEMENTS

Linkage editor control statements may be embedded in object modules either immediately after the object module header record or immediately after the object module transfer record. They may also be placed before or after the control sections in an object module, but may not be placed within a control section. Embedded control statements must be inserted in object modules prior to their being link edited. The system librarian may be used to perform this function.

Embedded statements are processed as are other control statements, except when a statement that affects the structure of a load module (OVERLAY, REGION, LINKOP, LOADM, and ENTER) is detected in an automatically included module. Because automatically included modules always are included in the root phase of a load module, these statements are not permitted to be embedded in automatically included modules and are flagged as errors in the link-edit map. If one or more object modules contain embedded INCLUDE control statements for a module (including itself) that is already being included, then an inclusion loop may result for the link edit in question.

All the control statements embedded in an object module are processed by the linkage editor, even if the object module is being accessed for a partial inclusion of its object code.

Whenever the control statements LINKOP and LOADM are encountered as embedded control items, they immediately signal the end of the current link edit but are never subsequently processed. Therefore, these control items must not be embedded in object modules with the implied intention of triggering multiple link-edit operations or (via LINKOP) supplying additional parameters to the current link edit.

6.5. BASIC CONTROL STATEMENT PROCESSING

The linkage editor cycles through two control modes for each load module it generates. The exact processing done in each mode is determined by the control statements processed in each mode. For this discussion, the control statements that affect the operation of the linkage editor are divided into two groups. The first group comprises all the control statements that direct the basic operation of the linkage editor and includes:

- all job control statements directed to the linkage editor – start-of-data (/$), parameter specification (// PARAM), and end-of-data (/*);

- all LINKOP statements; and
- the LOADM statement.

These statements are processed in the first control mode.

The second group of control statements consists of those that basically affect the structure and content of the load module, rather than the operation of the linkage editor. All remaining linkage editor control statements comprise this group and are processed in the second control mode.

All the group 1 statements input to the linkage editor for the purpose of constructing any given load module may come from the primary control stream, plus any number of user source libraries. (See // PARAM and LINKOP CLIB keyword description.) All group 2 statements, however, must come from a single source. The first group 2 statement detected initiates mode 2 processing, and mode 2 processing continues until include processing is terminated for the load module. A given load module may thus pick up options and/or its load module name from multiple sources, but its structure must be defined in a single input source (primary control stream input or user source library).

User source libraries for linkage editor control statements are specified by the CLIB keyword of the // PARAM or LINKOP control statements. The processing performed during construction of a load module depends, in some respects, on the source that contains the CLIB specification. In most cases, the statement containing the CLIB specification is the last statement processed for the current load module from the source that contains the CLIB specification. The only exception to this is if the source specified by the CLIB specification contains only group 1 control statements. If the linkage editor control statements are being input from the primary control stream when the CLIB specification is processed, the current location in the control stream is saved and is returned to when the control statements in the specified source are exhausted. In contrast, if a CLIB specification is processed while in a source library, the source library is disconnected and can never be returned to. Thus, it can be seen that multiple CLIB specifications can be meaningfully specified only in the primary control stream because only the first CLIB specification in a source module will ever be processed.

A single link-edit job step that produces multiple load modules proceeds as follows:

1. Enter mode 1 and process group 1 control statements for first load module to be produced.

2. Enter mode 2 and process all group 2 control statements to produce first load module.

3. Reenter mode 1 and process group 1 control statements for second load module.

4. Reenter mode 2 and process group 2 control statements to produce second load module.

5. Repeat steps 3 and 4 until all load modules are produced.

6.6. CONTROL STATEMENT DESCRIPTIONS

The following detailed descriptions of the basic linkage editor control statements are presented in their logical order of appearance in a linkage editor control stream. Each description includes the function of the control statement, illustrates its coding format, describes its parameters, and, when necessary, provides illustrated examples of how it may appear in a control stream.

6.6.1. Specify Linkage Editor Options (// PARAM or LINKOP)

Function:

Specifies the linkage editor options that are to be in effect during construction of a load module. The options are declared as keywords in the operand field of either a // PARAM or LINKOP control statement. Both control statements perform the same function; however, the // PARAM statement cannot appear within a linkage editor control stream and the LINKOP statement cannot appear outside the linkage editor control stream. Thus, the initial linkage editor options may be specified either by a // PARAM or LINKOP control statement, but only the LINKOP statement can be used to change them within a job step. Each time a LINKOP control statement is processed by the linkage editor, it initiates the construction of a new load module.

All option specifications take effect as soon as they are detected, and remain in effect until changed by a succeeding LINKOP control statement or until the job is terminated. This applies to system-supplied (default) parameter specifications, as well as those you specify. Once you override a default specification, your specification remains in effect for the remainder of the job step unless you explicitly respecify the default specification. When conflicting specifications are detected, the linkage editor assumes that the last statement is correct and functions accordingly.

If no // PARAM or LINKOP control statements are present in a control stream, the linkage editor functions under the direction of the default parameter specifications, as follows:

1. Performs automatic inclusion of required object modules, as necessary, and assumes that the standard system object library file ($Y$OBJ) is the only file that might contain the required modules.

2. Processes V-CON references, as required.

3. Stores the output load modules in the system job run library file ($Y$RUN).

4. Uses only the control statements contained in the primary control stream to produce load modules.

5. Generates phase header records that contain blank comment fields and do not require the system loader to clear main storage before the phase is loaded.

6. Processes all the control statements contained in the control stream, even if processing errors that would render a load module useless are detected.

7. Provides for the promotion of common storage areas.

8. Generates a link-edit for each load module generated that lists all the information normally desired in the link-edit map.

9. Produces nonreentrant load modules.

10. Recognizes reentrant object modules and creates the shared records needed to link their load module counterparts with the load module being created.

11. Creates internal symbol dictionary (ISD) records in the load module. At execution time, these records enable JOBDUMP to print a formatted dump of the load module area.

Format:

```
LABEL      △OPERATION△      OPERAND

[//]       {PARAM }         [ALIB=filename]
           {LINKOP}         [,CLIB=modulename/filename]
                            [,CMT={'character-string'}]
                            [     {'△'               }]
                            [,OUT={filename}]
                            [,    {(N)     }]
                            [     {SYSRUN  }]
                            [,RLIB={filename}]
                            [      {SYSOBJ  }]
                            [,{CNL  }] [,{ZRO  }]
                            [ {NOCNL}] [ {NOZRO}]
                            [,{NOAUTO}] [,{NOV}] [,{NOPROM}]
                            [ {AUTO  }] [ {V  }] [ {PROM  }]
                            [,{NOLIST}] [,{NOCNTCD}] [,{NOERR}]
                            [ {LIST  }] [ {CNTCD  }] [ {ERR  }]
                            [,{NODICT}] [,{NODEF}] [,{NOPHS}]
                            [ {DICT  }] [ {DEF  }] [ {PHS  }]
                            [ {DEL  }] [,{RCNTCD  }] [,{REF  }]
                            [ {NODEL}] [ {NORCNTCD}] [ {NOREF}]
                            [,{SHARE  }] [,{RNT  }] [,{ISD  }]
                            [ {NOSHARE}] [ {NORNT}] [ {NOISD}]
```

NOTES:

1. *System-supplied parameters are in effect only until you change them. Once you change them within a job step, you must reset them to their default state before the default option is effective again.*

2. *Keywords may be specified in any order but must be separated by commas with no spaces between the keywords unless they identify a delimited character string.*

3. *Private file names declared are always logical and must be accompanied by the appropriate job control DVC/LFD statements.*

Label:

//

Must be specified if PARAM operation is used.

Keyword Parameter ALIB:

`ALIB=filename`

Specifies the file to be searched during automatic inclusion processing. This keyword essentially provides the capability of naming an additional library for use during automatic inclusion processing. When specified, the ALIB library always is searched first in an attempt to locate a required object module, and if necessary, the file identified by the RLIB keyword is searched.

If omitted, only the RLIB-specified file is searched during the automatic inclusion process.

Keyword Parameter CLIB:

`CLIB=modulename/filename`

Specifies the name of a source module, and the file in which it resides, that contains the linkage editor control statements to be processed for this link-edit job. When this parameter is detected in the primary control stream input, the linkage editor branches to the source module identified in this parameter, processes the control statements contained in the source module, then returns to the primary control stream to complete the processing of the remaining control statements, as applicable. When this parameter is detected in a source module control stream input, the linkage editor branches to the new source module identified in this parameter, processes the control statements contained in the new source module, and returns to the primary control stream. The linkage editor never returns to a source module control stream after processing the first CLIB keyword specification it detects in that control stream source.

If omitted, the current control stream source (primary input or source module input) continues to be accessed for control statements.

Keyword Parameter CMT:

`CMT='character-string'`
Specifies a character string of up to 30 characters that is to be inserted in the comment field of each phase header record produced for the generated load modules. The character string must be enclosed in apostrophes and may contain blanks, commas, and other special symbols.

`CMT='Δ'`
Specifies that the phase header comment fields are to be blank.

If omitted, the phase header comment fields are left blank unless a previous CMT keyword specified otherwise.

Keyword Parameter OUT:

`OUT=filename`
Specifies a library file name in which the output load module is to be stored. If a load module with the same name already exists in the output file specified by this keyword, it is replaced by the new load module. This file name corresponds to the LFD name assigned to the file.

`OUT=(N)`
Indicates that the output load modules produced by the linkage editor are not to be stored in any library file; however, the link-edit map will still be produced if not inhibited by specification of the NOLIST keyword parameter.

`OUT=$Y$RUN`
Specifies that the output load modules produced by the linkage editor are to be stored in the standard system job run library file ($Y$RUN).

If omitted, the load modules produced are output to the system job run library file ($Y$RUN) unless a previous OUT keyword specified otherwise.

NOTE:

The file designated to store the output of the linkage editor is logically extended to accommodate the load modules produced by the linkage editor.

Keyword Parameter RLIB:

`RLIB=filename`
This parameter is used to identify the file to be searched by the linkage editor, under the following conditions:

- during the automatic inclusion process, when no automatic include library file (ALIB) has been specified (no previous ALIB keyword specification present) or when the specified ALIB does not contain the required module;

- during the specific inclusion process, when no file name is specified on an INCLUDE statement and:
  - the module is not found in the standard job run library file ($Y$RUN); or
  - the module is not found in the file last specified on a prior INCLUDE statement or no prior INCLUDE statements exist.

RLIB=$Y$OBJ

Identifies the standard system object library file ($Y$OBJ) as the file to search under the conditions just described.

If omitted, the standard system object library file ($Y$OBJ) is searched under the foregoing conditions, unless a previous RLIB keyword specified otherwise.

Keyword Parameters CNL and NOCNL:

CNL

Specifies that the link-edit job is to be canceled at the end of the link-edit operation for the current load module if any diagnostic processing has been triggered during the generation of the load module; otherwise, processing continues, regardless of the number or type of error diagnostics triggered, until the normal end-of-job function for the link-edit job step is detected.

NOCNL

Specifies that the link-edit job step is to be processed to completion, regardless of the number or type of diagnostic errors detected.

If omitted, NOCNL is assumed unless the CNL keyword was previously specified.

Keyword Parameters ZRO and NOZRO:

ZRO

Specifies that the job region is to be cleared to binary zeros prior to loading the root phase of the load module. This option is effective only if this load module name is specified on the // EXEC job control statement. Specification of this keyword sets a flag in the root phase header record to indicate the clearing requirement to the system loader.

NOZRO

Specifies that main storage is not to be cleared to zeros before the root phase of the load module is loaded.

If omitted, NOZRO is assumed unless the ZRO keyword was previously specified.

Keyword Parameters AUTO and NOAUTO:

AUTO
Specifies that automatic inclusion processing is to be allowed for the load modules being constructed.

NOAUTO
Specifies that automatic inclusion processing is not to be allowed for the load modules being constructed.

If omitted, AUTO is assumed unless the NOAUTO keyword was previously specified.

NOTE:

The NOAUTO specification does not affect the automatic inclusion of the overlay control routine if V-CON processing is not inhibited (NOV keyword not specified) and valid V-CON references exist in the object code being included in the load modules being produced.

Keyword Parameters V and NOV:

V
Specifies that V-CON references are permitted to appear in the object module elements to be included in the load modules and, therefore, that automatic loading of required phases is to be enabled.

NOV
Specifies that V-CON references are to be treated as A-CON references and, therefore, that automatic loading of phases is to be inhibited because it is not required. This specification is significant only when valid V-CON references are present in the object module elements being included.

If omitted, V is assumed unless the NOV keyword was previously specified.

Keyword Parameters PROM and NOPROM:

PROM
Specifies that common storage areas are to be promoted to the most reasonable phase within the load module being constructed.

NOPROM
Specifies that all common storage areas are to be placed in the root phase of the load module being constructed.

If omitted, PROM is assumed unless the NOPROM keyword was previously specified.

Keyword Parameters LIST and NOLIST:

LIST
Specifies that all list options for the link-edit map are to be in effect for the current link-edit job.

NOLIST
Specifies that no link-edit map is to be produced for the current link-edit job.

If omitted, the standard link-edit map will be produced unless LIST or NOLIST was previously specified.

Keyword Parameters CNTCD and NOCNTCD:

CNTCD
Specifies that the process map portion of the link-edit map is to be produced.

NOCNTCD
Specifies that the process map is to be suppressed.

If omitted, CNTCD is assumed unless NOCNTCD was previously specified.

Keyword Parameters ERR and NOERR:

ERR
Specifies that diagnostic messages and unresolved references are to be included in the link-edit map.

NOERR
Specifies that this information is to be suppressed.

If omitted, ERR is assumed unless NOERR was previously specified.

Keyword Parameters DICT and NODICT:

DICT
Specifies that the definitions dictionary and phase structure diagram portions of the link-edit map are to be produced.

NODICT
Specifies that the definitions dictionary and phase structure diagram portions of the link-edit map are to be suppressed.

If omitted, DICT is assumed unless NODICT was previously specified.

Keyword Parameters DEF and NODEF:

DEF
: Specifies that object module headers (including dates and times) and CSECT, COM, and ENTRY items are to be listed in the allocation map portion of the link-edit map, along with their respective object module origin and ESID, length, linked base address, and high limit after linking.

NODEF
: Specifies that listing of ESDs is to be suppressed.

If omitted, DEF is assumed unless NODEF was previously specified.

Keyword Parameters PHS and NOPHS:

PHS
: Specifies that phase data (phase name, alias name, origin, length, transfer address, etc.) is to be listed in the allocation map portion of the link-edit map.

NOPHS
: Specifies that the phase data is to be suppressed.

If omitted, PHS is assumed; unless NOPHS was previously specified.

Keyword Parameters DEL and NODEL:

DEL
: Specifies that all definitions including those automatically deleted due to redundant inclusions on identical paths or items not included because of partial include specifications are to be listed in the allocation map as long as definitions are being listed (NODEF is not specified).

NODEL
: Specifies that automatically deleted or excluded definitions are not to be listed.

If omitted, NODEL is assumed unless DEL was previously specified in the control stream.

Keyword Parameters RCNTCD and NORCNTCD:

RCNTCD
: Specifies that control statements are to be included in the allocation map portion of the link-edit map. You can easily locate a particular control statement and see its effect on the load module. Only action-type control statements are included in the allocation map; // PARAM and LINKOP statements are never listed here.

NORCNTCD
: Specifies that control statements are not to be listed in the allocation map.

If omitted, NORCNTCD is assumed unless RCNTCD was previously specified.

Keyword Parameters REF and NOREF:

REF

Specifies that the allocation map portion of the link-edit map is to list all references (EXTRNs) and transfer records processed, the object module assigned address and ESID, and the resolved value, if appropriate.

NOREF

Specifies that this information is to be suppressed.

If omitted, NOREF is assumed unless REF was previously specified.

Keyword Parameters SHARE and NOSHARE:

SHARE

Specifies that object modules marked reentrant are to be recognized during the inclusion process (specific and automatic). The text from such modules is not to be included. Instead, shared records are to be created for references made to reentrant code. The load module produced is nonreentrant but may contain references to reentrant code. This keyword is ignored if a reentrant load module is being generated (the RNT keyword is specified) or the job control GO option is in effect.

NOSHARE

Specifies that all object modules are to be treated as a nonreentrant and no shared records to be generated. A nonreentrant load module is produced that contains no references to shared code.

If omitted, SHARE is assumed unless the NOSHARE keyword was previously specified.

Keyword Parameters RNT and NORNT:

RNT

Specifies that a reentrant load module is to be produced and SENTRY records are to be generated. Normally, only one object module would be included in the link-edit, in which case, the load module name must be the same as the object module name. Parameter NOAUTO should be specified when using this keyword parameter.

NORNT

Specifies that a nonreentrant load module is to be produced, though references may be made to reentrant code.

If omitted, NORNT is assumed unless RNT keyword was previously specified.

Keyword Parameters ISD and NOISD:

ISD
: Specifies that type 3 and type 4 ISD records from the included object module are relocated and passed to the load module. It also specifies that type 1 and type 2 ISD records are generated based on the undeleted CSECTs and COMMONs detected during the link-edit. If execution of the load module terminates, these records are used by JOBDUMP (if specified) to print a formatted dump, segmenting it by CSECTs and printing user program symbols.

NOISD
: Specifies that ISD record generation is to be suppressed.

If omitted, ISD is assumed unless NOISD keyword was previously specified.

6.6.2. Begin Load Module (LOADM)

Function:

This control statement is used to initiate construction of a load module. Also, it specifies a program name for the load module. The LOADM control statement is normally the first control statement in a control stream, but it may follow a LINKOP or comment statement, or a complete set of previous control statements when used in a control stream that is generating more than one load module.

If the LOADM control statement is omitted from an otherwise valid linkage editor control stream and no LOADM control statement is embedded in an included object module, by default, the load module produced by the linkage editor is assigned the name LNKLOD.

Format:

LABEL	△OPERATION△	OPERAND
	LOADM	[{name LNKLOD}]

Positional Parameter 1:

name
: One to six (eight, if it is the link-edit of a reentrant module) alphanumeric characters, the first of which must be alphabetic, that specifies the name to be assigned to the load module. If the specified name is less than six (eight, if it is the link-edit of a reentrant module) characters, it is padded on the right with EBCDIC zeros. If the specified name is more than six characters (eight, if it is the link-edit of a reentrant module), it is truncated to the maximum allowable limit.

If omitted, the default name LNKLOD is assigned to the load module.

6.6.3. Include Object Code (INCLUDE)

Function:

Requests that a specific object module or selected control sections of a specific object module be included in the current phase of the load module being constructed. The INCLUDE statement may follow any linkage editor statement except the ENTER statement. Also, it may be embedded in an object module. Nesting of embedded INCLUDE statements may continue indefinitely. Nested INCLUDE statements are identical in format and capability to those not nested and may thus specify full or partial inclusion, as well as alternate files. The same applies during the automatic inclusion process, although the initial module located and used to satisfy a specific reference always is included in full.

If no INCLUDE statements are present in an otherwise valid linkage editor control stream, all the object modules currently residing in the system job run library ($Y$RUN) are included in the load module phase being constructed.

Format:

LABEL	△OPERATION△	OPERAND
	INCLUDE	[modulename][(s1,...s9)][,filename]

Positional Parameter 1:

modulename

Is an alphanumeric 1- to 8-character string that identifies the object module to be included.

If omitted, it is assumed that the INCLUDE statement is nested and that the object module being referenced immediately follows the previously referenced object module in the library being accessed. Otherwise, an error message is output on the link-edit map.

Subparameters:

s1,...s9

Is a list of from one to nine control section labels (one to eight characters in length) that identify the control sections to be included in the load module phase being constructed. The control sections referenced in this parameter must be contained in the object module referenced by the INCLUDE statement; otherwise, an error diagnostic is output on the link-edit map. The order in which the control sections appear in the object module is the order in which they will appear in the load module, regardless of the order in which they are listed in this parameter.

When a subparameter list is specified, no unnamed CSECT in the module being scannned is ever included in the load module, but all requests for common storage are honored, as are all embedded control statements.

If omitted, the entire object module referenced is included in the load module except for those control sections that may be automatically deleted by the linkage editor.

Positional Parameter 2:

`filename`
Is an alphanumeric 1- to 8-character string that identifies the symbolic name of the file in which the referenced object module is stored. This name must be specified exactly as it is in the job control logical file definition (LFD) that identified the file; otherwise, an error message is output on the link-edit map. When this parameter is specified, it is the only file searched for the specified object module.

If omitted, the object module is assumed to be in the system job run library ($Y$RUN) or, if not there, in the last library specified in an INCLUDE statement or, if not there and a file name was specified in a previous RLIB keyword in that file; otherwise, if no RLIB library was specified, the object module is assumed to be in the default RLIB library, the system object library file ($Y$OBJ).

6.6.4. Begin Overlay Phase (OVERLAY)

Function:

Identifies the beginning of an overlay phase (a phase other than the initial phase) and defines the relative position of the phase within the load module structure. The object modules included thereafter constitute a single, separate phase until the next OVERLAY, REGION, LOADM, LINKOP, or end-of-data (/*) control statement is detected.

This statement must not be used in the link-edit of a reentrant module.

Format:

LABEL	△OPERATION△	OPERAND
	OVERLAY	symbol[,alias-phasename]

Positional Parameter 1:

`symbol`
Is the name of a logical or relative node point that defines the starting address of the phase. It may consist of one to eight alphanumeric characters. If the symbol is relative (a CSECT, ENTRY, or EQU name), it must be on the path of the phase. However, the relative symbol must not be a shared definition.

The symbol specified may cause the phase to begin at an origin newly established by this node name or may set that origin to an address defined by a previous OVERLAY statement, the LOADM statement, or a relative definition (provided it is on the current path).

The starting address of a phase is not necessarily the entry point to that phase; therefore, use of the symbol in the operand field of an OVERLAY statement does not automatically define the symbol as an entry point. The same symbol, however, may be used to define both the entry point to this phase and the logical (or relative) origin of the phase (if in the current path). If the symbol is on an exclusive path, a logical node is established.

Positional Parameter 2:

alias-phasename

Is a 1- to 6-character, user-supplied phase name that can be used in place of the linkage-editor-supplied phase name, to address the phase being created by the OVERLAY statement. If an alias phase name longer than six characters is supplied, it is truncated to six. Alias phase names are always padded with two trailing blanks.

6.6.5. Begin New Region (REGION)

Function:

Initiates construction of the first phase in a new region, starting at the end of the longest path currently constructed for the load module. Once a region has been started, no prior region structure may be continued. Thus, all parts of a given region should be fully specified and structured before beginning a new region. The only phase common to all regions in a load module is the root phase.

OVERLAY control statements may be interspersed among REGION control statements to structure the phases within each region. The first of any region, including the initial phase, may be overlaid by using an OVERLAY control statement to reference the region node (or LOADM node) or a symbol with that address. Inasmuch as the REGION statement effectively replaces an OVERLAY statement, INCLUDE and other control statements used for the construction of a phase follow immediately. Up to 10 regions may be declared for a single load module.

This statement must not be used in the link-edit of a reentrant module.

Format:

LABEL	△OPERATION△	OPERAND
	REGION	symbol[,alias-phasename]

Positional Parameters 1 and 2:

Refer to parameter descriptions for the OVERLAY control statement. Note that inasmuch as the implied origin of a new region always is at the end of the longest path of the current region, a symbol in a REGION statement may specify only a logical node name, and not a relative definition name as in an OVERLAY statement.

6.6.6. Define Phase Execution Entrance (ENTER)

Function:

Defines the program entry point of the load module phase being constructed. This is the address to which program control is optionally transferred when the phase is loaded by a supervisor FETCH macroinstruction. The ENTER statement, if present, is normally the last control statement issued for a given phase. It may, however, be followed by a MOD, EQU, RES, or comment statement in addition to an OVERLAY statement for the next phase, a REGION statement for a new region, a LOADM or LINKOP statement for a new load module, or an end-of-data (/*) statement, which terminates execution of the linkage editor.

If an INCLUDE statement immediately follows an ENTER statement, a diagnostic warning indicating the sequence error is listed on the link-edit map, but the INCLUDE statement will, nonetheless, be processed for the current phase.

If no ENTER statement is provided for a phase, the phase entry point, or transfer address as it is commonly referred to, is obtained from the first specifically included object module in the phase that has a valid transfer address. If no specifically included object module contains a valid transfer address, the entry point address used by the linkage editor is the relocated address assigned to the first CSECT specifically included in the phase. Automatically included modules are not checked for valid transfer address. If no CSECTs have been included in the phase (zero length phase), then the transfer address is assigned to the node point of the phase.

Format:

LABEL	△OPERATION△	OPERAND
	ENTER	[expression]

Positional Parameter 1:

`expression`

Specifies the transfer address for the phase. This expression, which usually represents a relative phase address, may have one of the following forms:

- A decimal number from 1 to 8 digits long

- A hexadecimal number from 1 to 6 digits long, in the form X'nnnnnn'

- A previously defined symbol (the name of a control section or an entry point in an object module that was previously included or defined by a previous EQU directive). For the link-edit of a nonreentrant module, this symbol must not be a shared definition.

- A previously defined symbol plus or minus a decimal or hexadecimal number as previously described

If omitted, the ENTER statement has no effect and the default criteria previously described apply.

6.6.7. Define Label (EQU)

Function:

The EQU control statement is used to provide the linkage editor with the value of a label that might not otherwise be defined. The definition of a symbol by an EQU statement is subject to the same rules for automatic deletion as entry points.

Format:

LABEL	△OPERATION△	OPERAND
symbol	EQU	expression

Label:

symbol

Is an alphanumeric 1- to 8-character string, which is the label to be defined.

Positional Parameter 1:

expression

Specifies the value to be assigned to the label. The expression may have have one of the following forms:

- A decimal number from 1 to 10 digits long

- A hexadecimal number from 1 to 8 digits long

- A previously defined label (the name of a control section or an entry point in an object module that was previously included or defined by a previous equate statement)

- A previously defined label plus or minus a decimal or hexadecimal number, as previously described. For the link-edit of a nonreentrant module, this label must not be a shared definition.

- An asterisk (*) to indicate a reference to the current value of the location counter

Examples:

```
      1        10     16
   ┌──────────────────────────
1. │ DDSAC     EQU    32
2. │ DDECT1    EQU    X'2Ø'
3. │ DPSCHK1   EQU    DDECT1
4. │ DPSCHK2   EQU    DDECT1+X'2Ø'
5. │ DPSCHK3   EQU    *
```

1. Equates the label DDSAC to the decimal value 32

2. Equates the label DDECT1 to the hexadecimal value 20

3. Equates the label DPSCHK1 to the value of the label DDECT1

4. Equates the label DPSCHK2 to the value of the label DDECT1 plus the hexadecimal value 20

5. Equates the label DPSCHK3 to the current value of the location counter

NOTES:

If the operand of the EQU statement is symbolic (a previously defined symbol is involved) and the label of the EQU statement supplies a multidefinition, then no such earlier definitions may occur between the definition of the operand symbol and the EQU statement itself. If a symbolic operand is multiply defined, the label of the EQU statement is equated to the last such definition.

KE$ALP and KE$RES, which are the addressable entry points to the reserve storage area, cannot be used as operands of the EQU control statement.

6.6.8. Modify Location Counter (MOD)

Function:

The MOD control statement instructs the linkage editor to adjust its location counter to coincide with a specific power of 2 and a specific remainder. Initially, the location counter is set to zero by the LOADM control statement, and then incremented by the length of each control section included in a phase. Also, it is set to the value associated with a node point when an OVERLAY or REGION control statement is detected in the control stream.

Format:

LABEL	△OPERATION△	OPERAND
	MOD	$\text{power}\left[,\begin{Bmatrix}\text{remainder}\\ \text{(Ø)}\end{Bmatrix}\right]$

Positional Parameter 1:

`power`

Is a decimal or hexadecimal (X'n') number that specifies the power of 2 relative to which the location counter is to be adjusted. The only acceptable powers of 2 that may be specified are 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, and 32,768.

Positional Parameter 2:

`remainder`

Is a decimal or hexadecimal (X'n') number that is a multiple of 4, which specifies the desired remainder of the new value of the location counter relative to the specified power of 2. If the decimal number specified for this parameter is not a multiple of 4, it is rounded to the next higher multiple of 4, then truncated to a value less than the specified power of 2.

If omitted, the remainder is assumed to be zero.

Example:

```
1       10   16
-------------------
        MOD  32,8
```

This control statement instructs the linkage editor to adjust its location counter, if necessary, to a value that is 8 more than a multiple of 32. If the current value of the location counter is 16,384, which is an exact multiple of 32 (32 x 512 = 16,384), the location counter would be incremented by 8 to a value of 16,392. If the current value of the location counter were 16,392, no adjustment would be required. If the current value were 16,400, which is 16 greater than a multiple of 32 (32 x 512 + 16 = 16,400), the new value would be adjusted to 16,424 (32 x 513 + 8 = 16,424).

6.6.9. Reserve Storage (RES)

Function:

Instructs the linkage editor to reserve additional load module storage space following the end of the longest path in the highest region of the load module. The additional storage requested by the RES statement adds to the total length of the module and is recorded in each phase header record, but is not included in the size requirements for any particular phase. One or more of these statements, therefore, may be placed anywhere within the control stream for a load module. The sum of all RES values processed during the construction of a load module is used to extend the longest path of the load module.

The linkage editor automatically assigns two addressable entry points to the reserve storage area. These entry points may be addressed by the user to access the reserve storage area by declaring them as EXTRNS in your program. The two addressable entry points assigned are:

- KE$ALP
  Is the effective address of the end of the longest path in the load module or the starting address of the reserve storage area.

- KE$RES
  Is the effective address of the end of the longest path plus the sum of all the reserve storage area size specifications (RES statements).

These two entry points cannot be used in the operand field of the EQU control statement.

Format:

LABEL	△OPERATION△	OPERAND
	RES	value

Positional Parameter 1:

value
Specifies the number of bytes of storage to be reserved in one of the following forms:

- a decimal number from 1 to 10 digits long; or

- a hexadecimal number 1 to 8 digits long, in the form X'nnnnnnnn'.

7. The Link-Edit Map

7.1. GENERAL

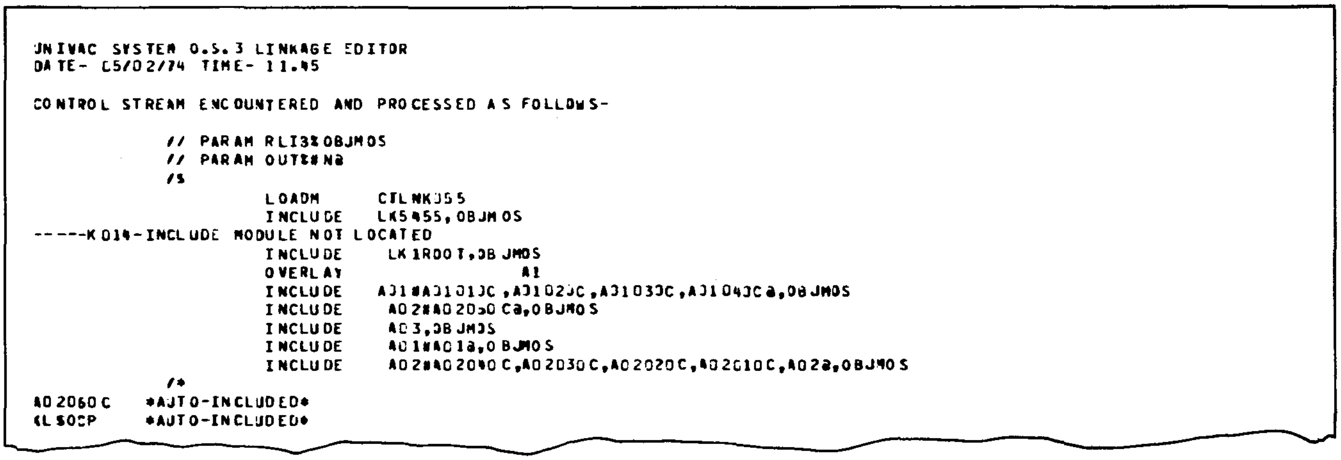
Unless otherwise specified in a // PARAM or LINKOP control statement, the linkage editor produces a link-edit map for each link-edit job it performs. Basically, the map details the load module produced by the linkage editor and is in six parts:

1. process map;
2. unresolved EXTRN reference list;
3. definitions dictionary;
4. phase structure diagram for multiphase load modules;
5. allocation map; and
6. error legend and count list.

However, via // PARAM or LINKOP parameter specifications, all or part of the link-edit map may be suppressed. A detailed description of each map part follows.

7.2. PROCESS MAP

The initial part of the link-edit map is a listing of the linkage editor control stream used to produce the subject load module (Figure 7-1). Both the control statement specifications you listed and those inserted by the linkage editor in the control stream from referenced source and object modules are listed. Special process-map messages also may appear in the process map. These messages, which are always enclosed in asterisks, are used to explain the presence of some of the control statement data included in the process map. Table 7-1 lists and describes the special messages.



```
UNIVAC SYSTEM O.S.3 LINKAGE EDITOR
DATE- 05/02/74 TIME- 11.45

CONTROL STREAM ENCOUNTERED AND PROCESSED AS FOLLOWS-

            // PARAM RLIB%OBJMOS
            // PARAM OUT%#N@
            /$
                        LOADM      CTLNK055
                        INCLUDE    LK5455,OBJMOS
-----K014-INCLUDE MODULE NOT LOCATED
                        INCLUDE     LK1ROOT,OBJMOS
                        OVERLAY               A1
                        INCLUDE    A01#A01010C,A01020C,A01030C,A01040C@,OBJMOS
                        INCLUDE     A02#A02050C@,OBJMOS
                        INCLUDE     A03,OBJMOS
                        INCLUDE     A01#A01@,OBJMOS
                        INCLUDE     A02#A02040C,A02030C,A02020C,A02010C,A02@,OBJMOS
            /*
A02060C    *AUTO-INCLUDED*
KLSOCP     *AUTO-INCLUDED*
```

Figure 7-1. Typical Link-Edit Process-Map Listing

Table 7-1. Special Process-Map Messages

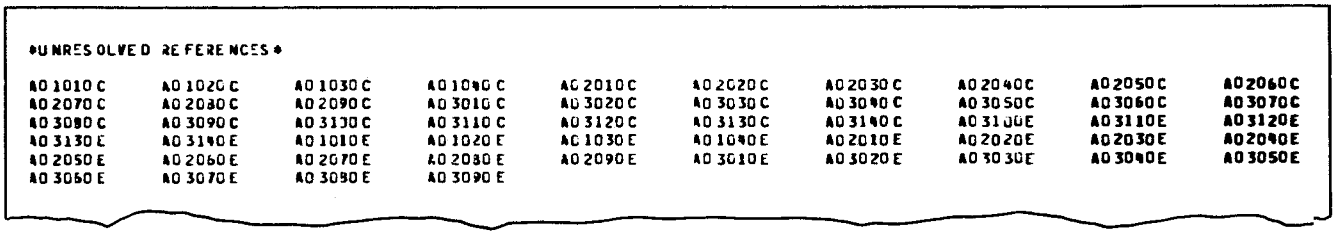
Message	Meaning
AUTO-INCLUDED	Item was, of necessity, automatically included.
EMBEDDED	Control statement, as processed, exists within an included object module.
GENERATED	A necessary control statement was generated by the linkage editor.
SORS FILE	Control statement processed resides within a source module of linkage editor directives.
RUN LIBE MODULE	Module was included by default from the job run library (/* item is not printed on process map).

Diagnostic warnings also may be interspersed among in-error control statements and the processing triggered by their presence. These messages always are delineated by five leading dashes (- - - - -) and an error code. They are listed and described in system messages programmer/operator reference, UP-8076 (current version).

The process map may be suppressed by the declaration of the NOCNTCD keyword specification in the // PARAM or LINKOP control statement.

7.3. UNRESOLVED EXTRN REFERENCE LIST

If, after all include processing terminates, references are made to one or more symbols for which no corresponding definitions exist, a list of the undefined symbols is output on the link-edit map and the UPSI byte is set to X'20'. The symbols are not sorted into any sequence and are listed only for your convenience (Figure 7-2). After this list is generated – and all relocation and phase sizes, CSECT, and address assignments are computed – diagnostic messages may be interspersed among the list of undefined EXTRN references, as applicable.



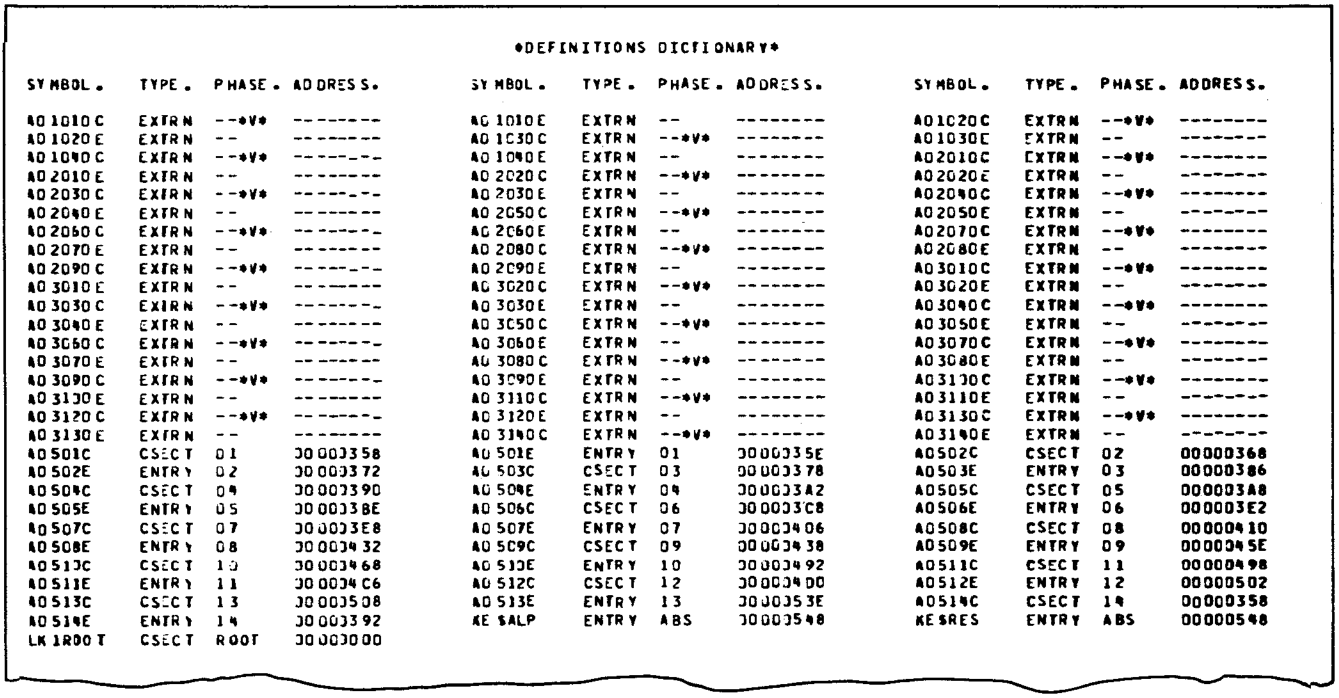
```
*UNRESOLVED REFERENCES*

AO1010C    AO1020C    AO1030C    AO1040C    AO2010C    AO2020C    AO2030C    AO2040C    AO2050C    AO2060C
AO2070C    AO2080C    AO2090C    AO3010C    AO3020C    AO3030C    AO3040C    AO3050C    AO3060C    AO3070C
AO3080C    AO3090C    AO3100C    AO3110C    AO3120C    AO3130C    AO3140C    AO3100E    AO3110E    AO3120E
AO3130E    AO3140E    AO1010E    AO1020E    AO1030E    AO1040E    AO2010E    AO2020E    AO2030E    AO2040E
AO2050E    AO2060E    AO2070E    AO2080E    AO2090E    AO3010E    AO3020E    AO3030E    AO3040E    AO3050E
AO3060E    AO3070E    AO3080E    AO3090E
```

Figure 7-2. Typical Unresolved EXTRN Reference List

7.4. DEFINITIONS DICTIONARY

The definitions dictionary part of the link-edit map lists and describes all the symbols referenced in the link-edit job. These symbols are listed alphabetically, in three horizontal columns (Figure 7-3). Each symbol definition includes:

- a type identification;
- a phase assignment identification for nonshared items;
- a linkage-editor-assigned address when applicable; and
- an optional information character.



```
                                               *DEFINITIONS DICTIONARY*

SYMBOL.   TYPE.   PHASE.  ADDRESS.          SYMBOL.   TYPE.   PHASE.  ADDRESS.          SYMBOL.   TYPE.   PHASE.  ADDRESS.

AO1010C   EXTRN   --*V*   --------          AO1010E   EXTRN   --      --------          AO1020C   EXTRN   --*V*   --------
AO1020E   EXTRN   --      --------          AO1030C   EXTRN   --*V*   --------          AO1030E   EXTRN   --      --------
AO1040C   EXTRN   --*V*   --------          AO1040E   EXTRN   --      --------          AO2010C   EXTRN   --*V*   --------
AO2010E   EXTRN   --      --------          AO2020C   EXTRN   --*V*   --------          AO2020E   EXTRN   --      --------
AO2030C   EXTRN   --*V*   --------          AO2030E   EXTRN   --      --------          AO2040C   EXTRN   --*V*   --------
AO2040E   EXTRN   --      --------          AO2050C   EXTRN   --*V*   --------          AO2050E   EXTRN   --      --------
AO2060C   EXTRN   --*V*   --------          AO2060E   EXTRN   --      --------          AO2070C   EXTRN   --*V*   --------
AO2070E   EXTRN   --      --------          AO2080C   EXTRN   --*V*   --------          AO2080E   EXTRN   --      --------
AO2090C   EXTRN   --*V*   --------          AO2090E   EXTRN   --      --------          AO3010C   EXTRN   --*V*   --------
AO3010E   EXTRN   --      --------          AO3020C   EXTRN   --*V*   --------          AO3020E   EXTRN   --      --------
AO3030C   EXTRN   --*V*   --------          AO3030E   EXTRN   --      --------          AO3040C   EXTRN   --*V*   --------
AO3040E   EXTRN   --      --------          AO3050C   EXTRN   --*V*   --------          AO3050E   EXTRN   --      --------
AO3060C   EXTRN   --*V*   --------          AO3060E   EXTRN   --      --------          AO3070C   EXTRN   --*V*   --------
AO3070E   EXTRN   --      --------          AO3080C   EXTRN   --*V*   --------          AO3080E   EXTRN   --      --------
AO3090C   EXTRN   --*V*   --------          AO3090E   EXTRN   --      --------          AO3100C   EXTRN   --*V*   --------
AO3100E   EXTRN   --      --------          AO3110C   EXTRN   --*V*   --------          AO3110E   EXTRN   --      --------
AO3120C   EXTRN   --*V*   --------          AO3120E   EXTRN   --      --------          AO3130C   EXTRN   --*V*   --------
AO3130E   EXTRN   --      --------          AO3140C   EXTRN   --*V*   --------          AO3140E   EXTRN   --      --------
AO501C    CSECT   01      00000358          AO501E    ENTRY   01      0000035E          AO502C    CSECT   02      00000368
AO502E    ENTRY   02      00000372          AO503C    CSECT   03      00000378          AO503E    ENTRY   03      00000386
AO504C    CSECT   04      00000390          AO504E    ENTRY   04      000003A2          AO505C    CSECT   05      000003A8
AO505E    ENTRY   05      000003BE          AO506C    CSECT   06      000003C8          AO506E    ENTRY   06      000003E2
AO507C    CSECT   07      000003E8          AO507E    ENTRY   07      00000406          AO508C    CSECT   08      00000410
AO508E    ENTRY   08      00000432          AO509C    CSECT   09      00000438          AO509E    ENTRY   09      0000045E
AO510C    CSECT   10      00000468          AO510E    ENTRY   10      00000492          AO511C    CSECT   11      00000498
AO511E    ENTRY   11      000004C6          AO512C    CSECT   12      000004D0          AO512E    ENTRY   12      00000502
AO513C    CSECT   13      00000508          AO513E    ENTRY   13      0000053E          AO514C    CSECT   14      00000358
AO514E    ENTRY   14      00000392          KE$ALP    ENTRY   ABS     00000548          KE$RES    ENTRY   ABS     00000548
LK1ROOT   CSECT   ROOT    00000000
```

Figure 7-3. Typical Link-Edit Definitions Dictionary List

Table 7-2 lists the type identifications that may be specified for a symbol and their meanings.

Table 7-2. Definitions Dictionary Type Identifications

Type	Meaning
COM①	Identifies a common section name.
ENTRY	Identifies an absolute or relative entry point definition.
CSECT①	Identifies a control section name.
EQU	Identifies a link-time-defined symbol for an entry point.
EQU*	Identifies a link-time reference to the location counter for an entry point.
EXTRN	Identifies a symbol referenced but never defined (also listed in the unresolved EXTRN references list).

① If a COM or CSECT symbol name is blank, the item listed represents a blank common or unnamed control section, respectively.

The phase assignment identification identifies the load module in which the symbol appears. Table 7-3 lists and describes the phase identifications that may appear in this field.

Table 7-3. Definitions Dictionary Phase Field

Phase	Meaning
1—99	Load module phase number
ROOT	Root phase
ABS	Absolute ENTRY symbol (no phase assignment applicable)
— —	EXTRN symbol (no phase assignment)
SHR	Shared definition

The address field is expressed in hexadecimal and may be absolute or relocatable, depending on the definition type. Dashes appear in the address field of EXTRN symbols. Blanks appear in the address field of shared definitions since they do not have an address.

Table 7-4 lists and describes the information characters that also may appear in a symbol definition. When included, these characters are listed between the PHASE and ADDRESS fields of a svmbol definition.

Table 7-4. Definitions Dictionary Information Characters

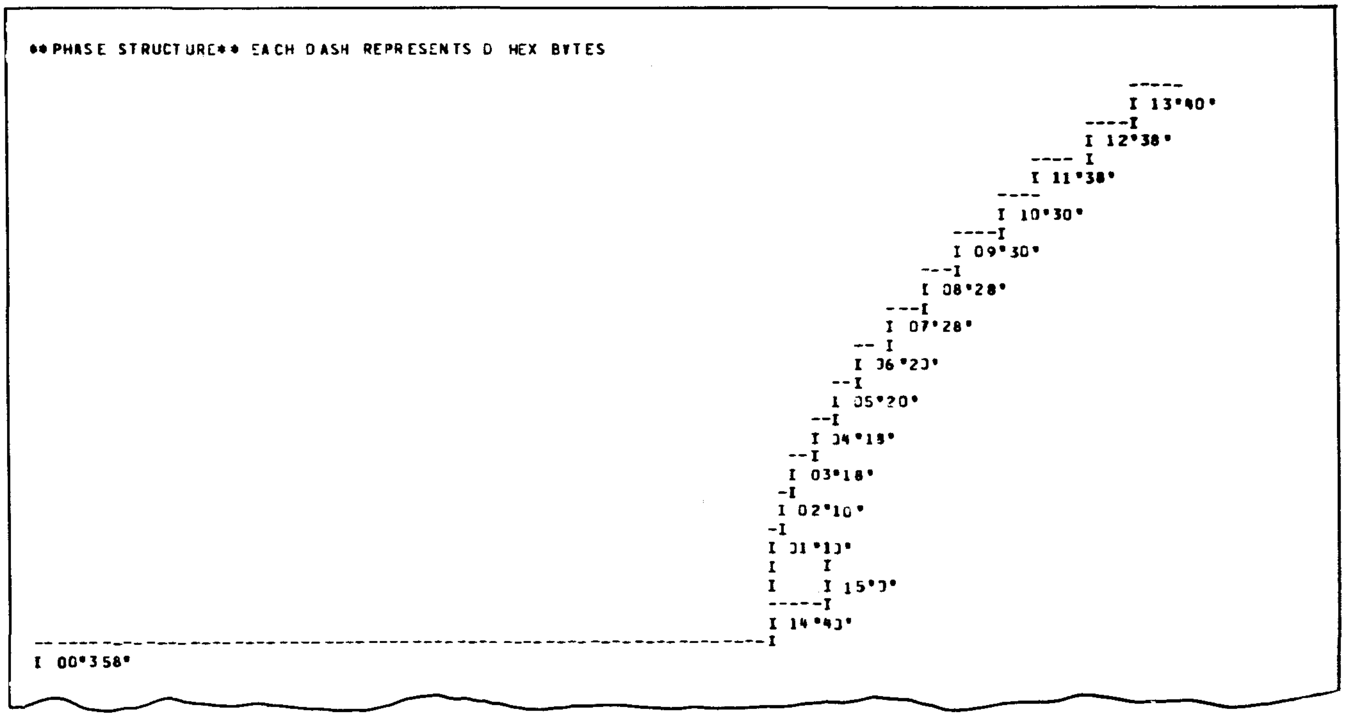
Character	Meaning
G	Used to identify a dictionary item referenced by text and included in the load module during a partial include sequence, and defined in another CSECT of the same object module that was not to be included in the load module. Whenever such a symbol is detected, the linkage editor generates an EXTRN label for the symbol in hopes of satisfying the reference. No automatic include processing ever is triggered for such references and, therefore, the symbol remains undefined unless specifically obtained by the user through an INCLUDE or EQU statement.
M	Used to identify a symbol that is multiply defined and, therefore, listed at least twice
V	Used to indicate that a symbol is a valid V-type definition and is a candidate for residence in the entry point table (KL$NTB) for V-references

Printing of the definitions dictionary can be suppressed by a // PARAM or LINKOP control statement that specifies the keyword NODICT; however, suppression of the definitions dictionary also causes suppression of the phase structure map.

7.5. PHASE STRUCTURE DIAGRAM

A scaled pictorial drawing for multiphase load modules is provided next in the link-edit map (Figure 7-4). This figure depicts the phase and overlay structure of the load module being generated. Each phase is shown with its linked load origin in association with other phases. Each phase is identified by a decimal phase number and its size is listed in hexadecimal. Vertical bars (Is) represent established node origins; horizontal bars (dashes) represent the phase lengths. Each printer bar is intended to depict a specific number of bytes with regard to phase storage needs and is scaled to a predetermined factor that is specified in the heading of the listing. In Figure 7-4, each horizontal bar represents 0D hexadecimal (13 decimal) bytes.

The phase structure diagram can be suppressed by a // PARAM or LINKOP control statement that specifies the keyword NODICT; however, suppression of the phase structure diagram also causes suppression of the definitions dictionary. In either event, no phase structure diagram is produced for single-phase load modules.



```
**PHASE STRUCTURE** EACH DASH REPRESENTS D HEX BYTES

                                                                                               -----
                                                                                               I 13*40*
                                                                                           ----I
                                                                                           I 12*38*
                                                                                      ---- I
                                                                                      I 11*38*
                                                                                   ----
                                                                                   I 10*30*
                                                                               ----I
                                                                               I 09*30*
                                                                            ---I
                                                                            I 08*28*
                                                                         ---I
                                                                         I 07*28*
                                                                      -- I
                                                                      I 06*20*
                                                                    --I
                                                                    I 05*20*
                                                                  --I
                                                                  I 04*18*
                                                                --I
                                                                I 03*18*
                                                               -I
                                                               I 02*10*
                                                              -I
                                                              I 01*10*
                                                              I    I
                                                              I    I 15*30*
                                                              -----I
                                                              I 14*40*
--------------------------------------------------------------I
I 00*358*
```

Figure 7-4. Typical Phase Structure Diagram

7.6. ALLOCATION MAP

The allocation map provides a detailed description of the items processed during the link-edit operation. The following information, depending on the // PARAM or LINKOP specified parameters, may include:

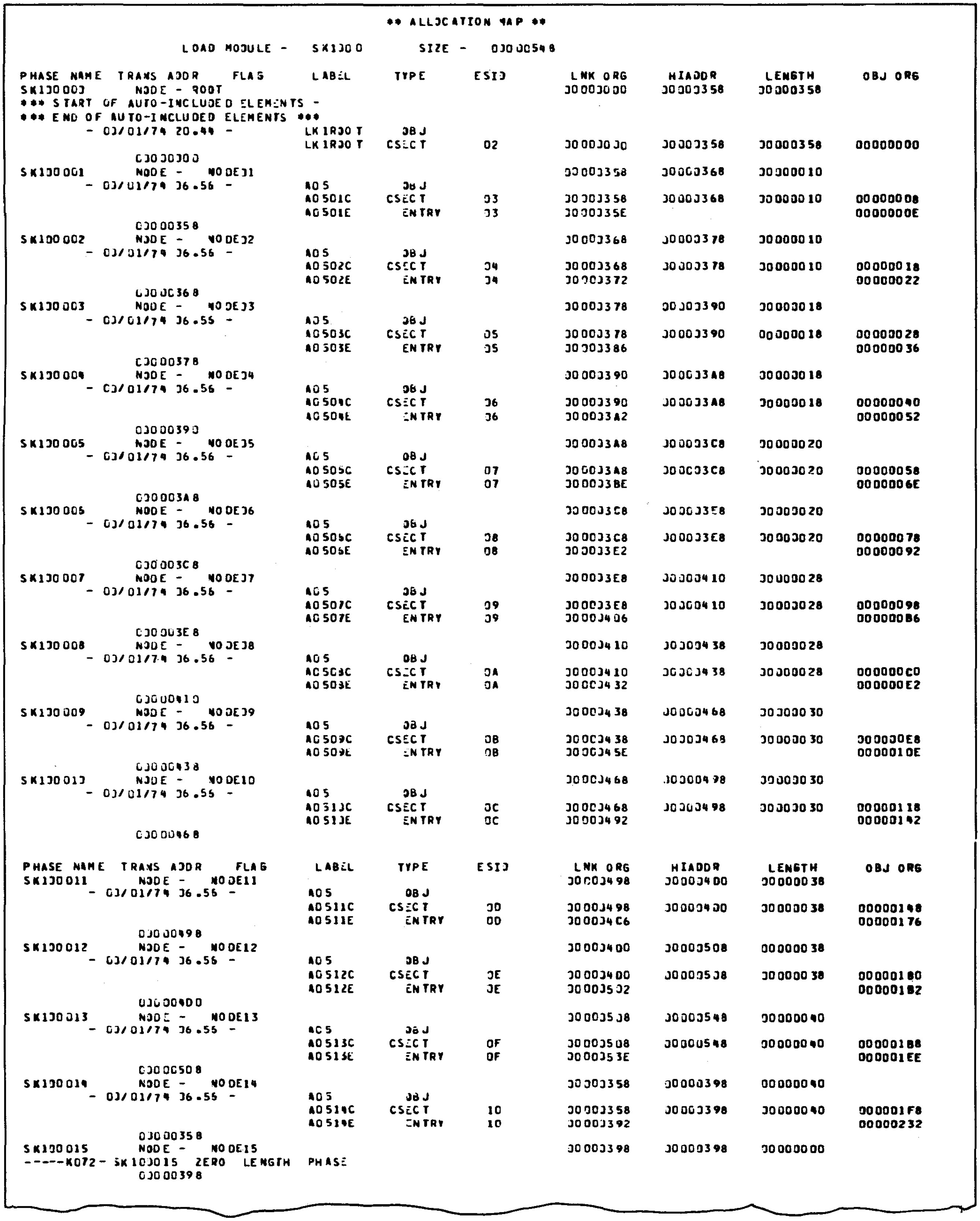
- the name and size of the load module;
- the link-edit assigned name of each phase (and possible alias name) and an indication of any common storage areas assigned to the phase (COMMON indicates residence in current phase);
- the origin, length, and high address of each phase;
- the transfer address assigned to each phase;
- the name, type, and ESID of each definition or reference processed;
- the linked origin, high address, length, and object module origin (if applicable) of each definition or reference processed;

  NOTE:

  Certain language processors omit the length field in the CSECT record but indicate it in the transfer record. When the linkage editor lists such a CSECT, the high address is omitted, the word DEFERRED appears under the LENGTH heading, and the code L appears under the FLAG heading. The actual CSECT length is printed when the transfer record is listed. Blanks appear in the link origin high address and length field for shared definitions.

- a set of flag codes for items representing special conditions;
- a relist of interspersed control statements that triggered the results shown;
- an indication of the accessed object modules and whether they were automatically included; and
- object module transfer records processed. If the deferred length is present, it is printed under LENGTH and the code L appears under FLAG.

Figure 7-5 illustrates a typical allocation map.



```
                                         ** ALLOCATION MAP **

                  LOAD MODULE -   SK1000        SIZE -   00000548

PHASE NAME  TRANS ADDR    FLAG      LABEL      TYPE       ESID          LNK ORG       HIADDR      LENGTH       OBJ ORG
SK100000       NODE - ROOT                                             00000000      00000358    00000358
*** START OF AUTO-INCLUDED ELEMENTS -
*** END OF AUTO-INCLUDED ELEMENTS ***
       - 03/01/74 20.44 -           LK1ROOT    OBJ
                                    LK1ROOT    CSECT       02          00000000      00000358    00000358     00000000
            00000000
SK100001       NODE -   NODE01                                         00003358      00003368    00000010
       - 03/01/74 06.56 -           AD5        OBJ
                                    AD501C     CSECT       03          00003358      00003368    00000010     00000008
                                    AD501E       ENTRY     03          0000335E                               0000000E
            00000358
SK100002       NODE -   NODE02                                         00003368      00003378    00000010
       - 03/01/74 06.56 -           AD5        OBJ
                                    AD502C     CSECT       04          00003368      00003378    00000010     00000018
                                    AD502E       ENTRY     04          00003372                               00000022
            00000368
SK100003       NODE -   NODE03                                         00003378      00003390    00000018
       - 03/01/74 06.55 -           AD5        OBJ
                                    AD503C     CSECT       05          00003378      00003390    00000018     00000028
                                    AD503E       ENTRY     05          00003386                               00000036
            00000378
SK100004       NODE -   NODE04                                         00003390      000033A8    00000018
       - 03/01/74 06.56 -           AD5        OBJ
                                    AD504C     CSECT       06          00003390      000033A8    00000018     00000040
                                    AD504E       ENTRY     06          000033A2                               00000052
            00000390
SK100005       NODE -   NODE05                                         000033A8      000033C8    00000020
       - 03/01/74 06.56 -           AD5        OBJ
                                    AD505C     CSECT       07          000033A8      000033C8    00000020     00000058
                                    AD505E       ENTRY     07          000033BE                               0000006E
            000003A8
SK100006       NODE -   NODE06                                         000033C8      000033E8    00000020
       - 03/01/74 06.56 -           AD5        OBJ
                                    AD506C     CSECT       08          000033C8      000033E8    00000020     00000078
                                    AD506E       ENTRY     08          000033E2                               00000092
            000003C8
SK100007       NODE -   NODE07                                         000033E8      00003410    00000028
       - 03/01/74 06.56 -           AD5        OBJ
                                    AD507C     CSECT       09          000033E8      00003410    00000028     00000098
                                    AD507E       ENTRY     09          00003406                               000000B6
            000003E8
SK100008       NODE -   NODE08                                         00003410      00003438    00000028
       - 03/01/74 06.56 -           AD5        OBJ
                                    AD508C     CSECT       0A          00003410      00003438    00000028     000000C0
                                    AD508E       ENTRY     0A          00003432                               000000E2
            00000410
SK100009       NODE -   NODE09                                         00003438      00003468    00000030
       - 03/01/74 06.56 -           AD5        OBJ
                                    AD509C     CSECT       0B          00003438      00003468    00000030     000000E8
                                    AD509E       ENTRY     0B          0000345E                               0000010E
            00000438
SK100010       NODE -   NODE10                                         00003468      00003498    00000030
       - 03/01/74 06.55 -           AD5        OBJ
                                    AD510C     CSECT       0C          00003468      00003498    00000030     00000118
                                    AD510E       ENTRY     0C          00003492                               00000142
            00000468

PHASE NAME  TRANS ADDR    FLAG      LABEL      TYPE       ESID          LNK ORG       HIADDR      LENGTH       OBJ ORG
SK100011       NODE -   NODE11                                         00003498      000034D0    00000038
       - 03/01/74 06.56 -           AD5        OBJ
                                    AD511C     CSECT       0D          00003498      000034D0    00000038     00000148
                                    AD511E       ENTRY     0D          000034C6                               00000176
            00000498
SK100012       NODE -   NODE12                                         000034D0      00003508    00000038
       - 03/01/74 06.56 -           AD5        OBJ
                                    AD512C     CSECT       0E          000034D0      00003508    00000038     00000180
                                    AD512E       ENTRY     0E          00003502                               000001B2
            000004D0
SK100013       NODE -   NODE13                                         00003508      00003548    00000040
       - 03/01/74 06.55 -           AD5        OBJ
                                    AD513C     CSECT       0F          00003508      00003548    00000040     000001B8
                                    AD513E       ENTRY     0F          0000353E                               000001EE
            00000508
SK100014       NODE -   NODE14                                         00003358      00003398    00000040
       - 03/01/74 06.55 -           AD5        OBJ
                                    AD514C     CSECT       10          00003358      00003398    00000040     000001F8
                                    AD514E       ENTRY     10          00003392                               00000232
            00000358
SK100015       NODE -   NODE15                                         00003398      00003398    00000000
-----K072- SK100015  ZERO  LENGTH  PHASE
            00000398
```

Figure 7-5. Typical Allocation Map

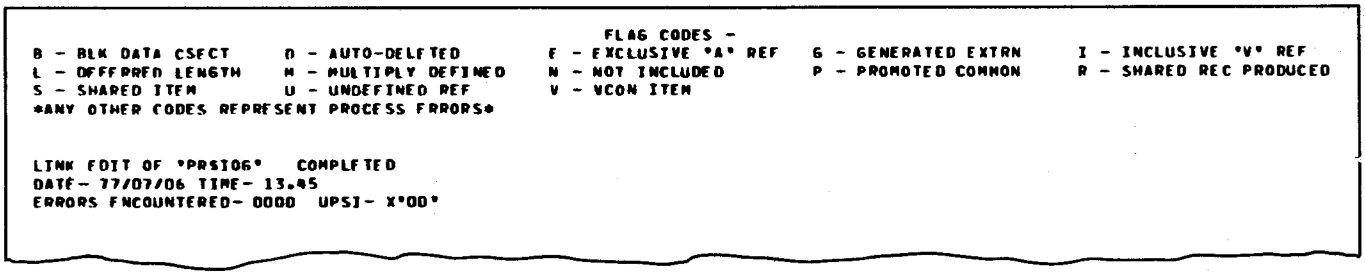
7.7. ERROR LEGEND, ERROR COUNT LIST, AND UPSI SETTING

The final part of the link-edit map concerns the termination of the link-edit job itself. An error code legend is printed, indicating the meanings of flags that may have been detected during the link-edit job. The error legend is followed by a printout showing a count of the number of errors appearing in the link-edit map and an UPSI byte setting at the end of the map. Table 7–5 summarizes the flag code legend.

Table 7-5. Error Legend and Count List Flag Code Descriptions

Flag Code	Descriptions
B	The control section is considered a block data CSECT and is to be used to load a common storage area.
D	The item has been automatically deleted because another relative definition for the same symbol is in the same path or one absolute definition already exists. If a transfer record is involved, the item has not been accepted for processing because no valid address is in the record.
E	A direct A-type reference has been made, which is exclusive and whose definition may not be resident with the reference.
G	An EXTRN has been generated because of the partial inclusion of a control section whose text references a control section not included.
I	A V-type reference has been made to a definition that is inclusive and, therefore, has been converted to a direct A-type reference.
L	A deferred CSECT or COMMON length is in effect and the actual length is listed with the object module transfer record.
M	The item is validly multiply defined in this link edit.
N	The item has not been included because of a partial include, which purposely omitted it.
P	The item, which is a COM storage area, has been promoted to another phase based on the phases in which it was declared, the references pointing to it, and the block data CSECTS, which load it.
R	A shared record has been produced for this item.
S	This is a shared item.
U	The item was unresolved and remains undefined in the load module.
V	The item is a V-CON item and will be loaded automatically when referenced by a V-type constant.

If the link-edit job has been successful, a completion message and the date time will so indicate. Figure 7–6 illustrates a typical error legend and count listing.



```
                                              FLAG CODES -
B - BLK DATA CSECT     D - AUTO-DELFTED       E - EXCLUSIVE 'A' REF   G - GENERATED EXTRN     I - INCLUSIVE 'V' REF
L - DFFFRRFD LENGTH    M - MULTIPLY DEFINED   N - NOT INCLUDED        P - PROMOTED COMMON     R - SHARED REC PRODUCED
S - SHARED ITEM        U - UNDEFINED REF      V - VCON ITEM
*ANY OTHER CODES REPRFSENT PROCESS FRRORS*


LINK FDIT OF 'PRSI06'   COMPLFTED
DATE- 77/07/06 TIME- 13.45
ERRORS FNCOUNTERED- 0000  UPSI- X'00'
```

Figure 7-6. Typical Error Legend and Count List

8. Program Examples

A representative set of link-edit jobs is illustrated in this section. Figure 8-1 illustrates a typical job control stream that could be used to execute the linkage editor, while Figures 8-2 through 8-6 illustrate the link-edit maps produced for each job. Figure 8-2 illustrates a rather simple single phase load module. Figures 8-3 through 8-5 illustrate some typical multiphase load modules, and Figure 8-6 illustrates a rather sophisticated multiphase load module.

```
1          10     16
// JOB LINKSTA
// DVC 2Ø // LFD PRNTR
// WORK1
// EXEC LNKEDT
/$
** LINKAGE EDITOR CONTROL STATEMENTS GO HERE **
/*
/&
// FIN
```

NOTES:

1. When storage is available, the job card should specify a maximum storage size of X'8000' for optimum performance. If you omit the maximum main storage parameter, job control allocates the minimum and maximum main storage requirements to execute the linkage editor. The speed of the linkage editor is directly related to the amount of main storage allocated to the job.

2. The printer must always be allocated regardless of the fact that the list options selected may cause only minimal printing.

3. If the keyword parameter EXTSP is coded on WORK1 jproc, a minimum value of 5 must be specified.

Figure 8-1. Typical Linkage Editor Job Control Stream (Part 1 of 2)

4. In addition to using the // EXEC LNKEDT statement to call the linkage editor, a job procedure call statement (jproc call) is also available. The jproc call is an easier method for executing the linkage editor. For example, if all the defaults associated with either the LINKOPT or // PARAM statement were utilized, and the only other linkage editor control statement required was the LOADM statement, the following jproc call can be used:

```
//LINK
```

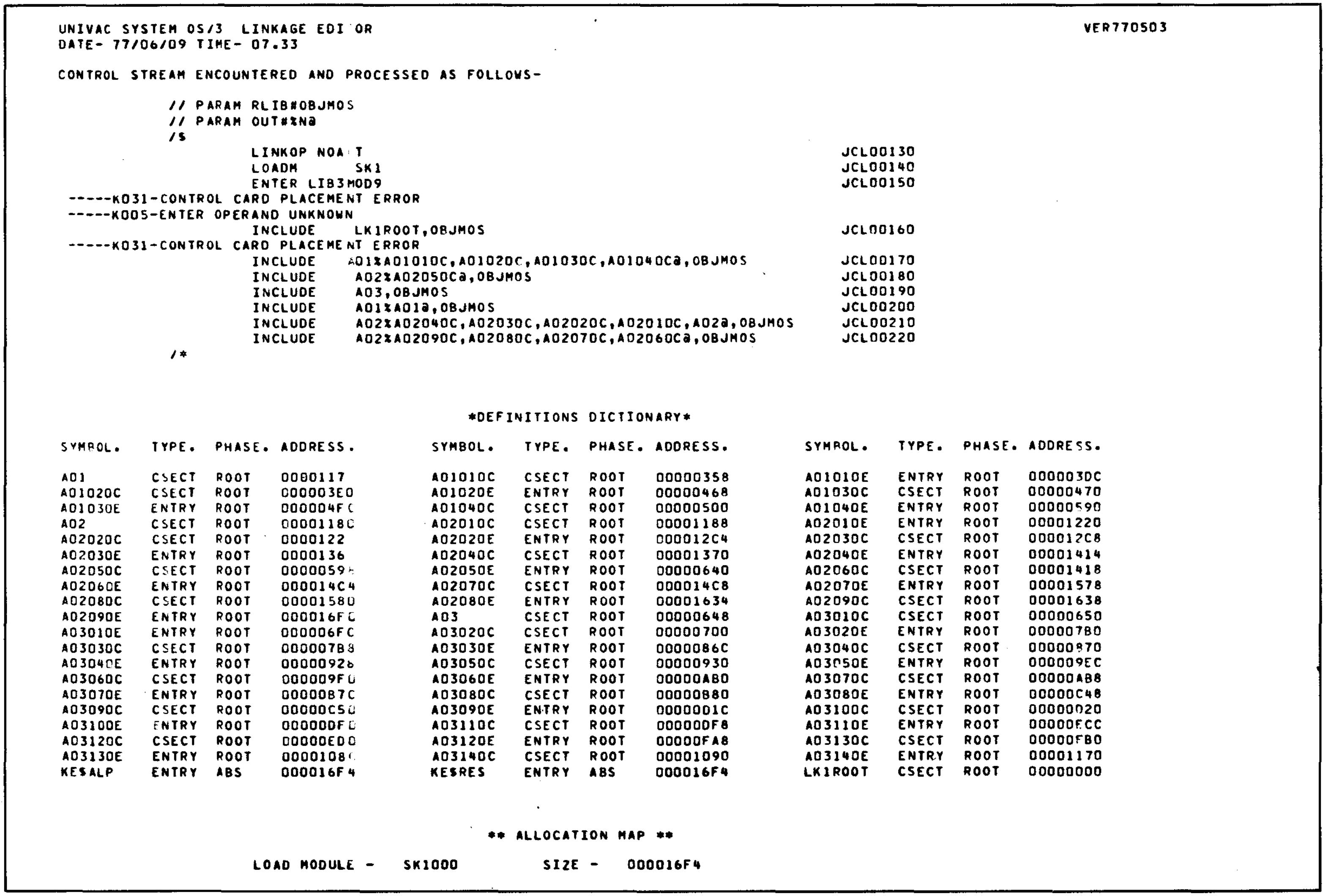
When you use this one statement, the following is automatically generated:

- printer assignment:
- work file;
- EXEC LNKEDT statement;
- // LOADM statement; and
- data delimiters (/$ and /*) job control statements.

You can alter the default conditions by using the optional parameters. By choosing the optional parameters, you can use a specific printer, a certain scratch file, a specific file containing the load module, etc.

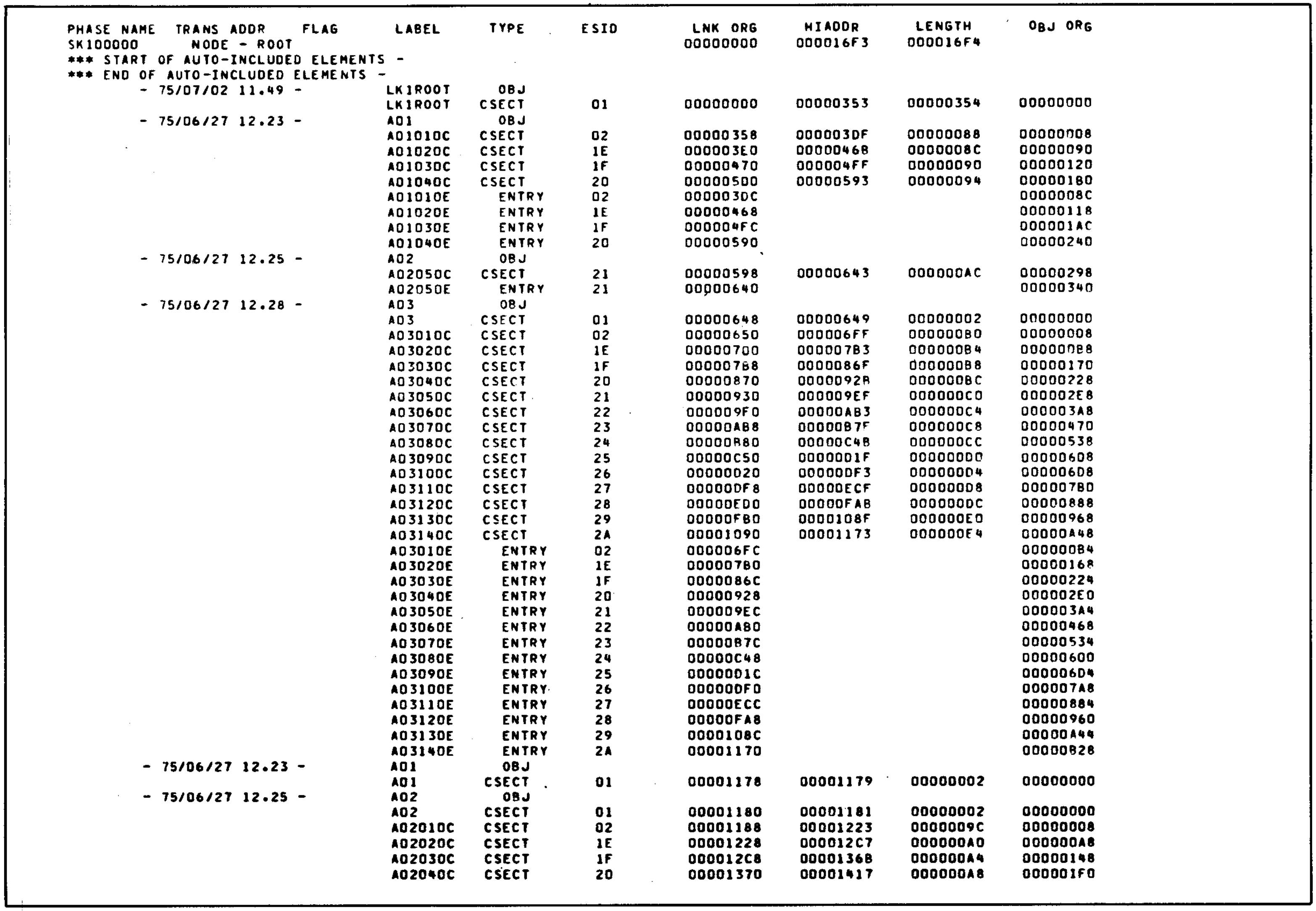
For more information about the jproc call, refer to the job control user guide, UP-8065 (current version).

Figure 8–1. Typical Linkage Editor Job Control Stream (Part 2 of 2)



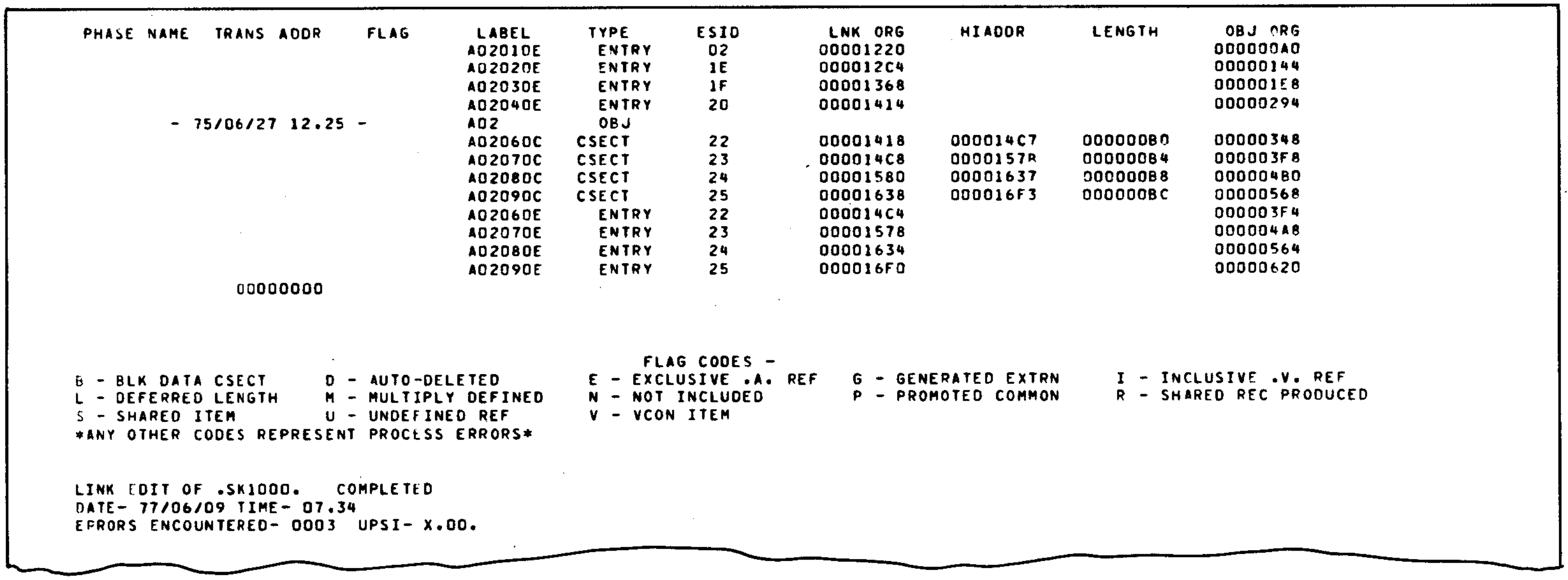
```
UNIVAC SYSTEM OS/3  LINKAGE EDI OR                                                                          VER770503
DATE- 77/06/09 TIME- 07.33

CONTROL STREAM ENCOUNTERED AND PROCESSED AS FOLLOWS-

            // PARAM RLIB#OBJMOS
            // PARAM OUT#%N@
            /$
                     LINKOP NOA T                                                        JCL00130
                     LOADM      SK1                                                      JCL00140
                     ENTER LIB3MOD9                                                      JCL00150
 -----K031-CONTROL CARD PLACEMENT ERROR
 -----K005-ENTER OPERAND UNKNOWN
                     INCLUDE    LK1ROOT,OBJMOS                                           JCL00160
 -----K031-CONTROL CARD PLACEMENT ERROR
                     INCLUDE    A01%A01010C,A01020C,A01030C,A01040C@,OBJMOS              JCL00170
                     INCLUDE    A02%A02050C@,OBJMOS                                      JCL00180
                     INCLUDE    A03,OBJMOS                                               JCL00190
                     INCLUDE    A01%A01@,OBJMOS                                          JCL00200
                     INCLUDE    A02%A02040C,A02030C,A02020C,A02010C,A02@,OBJMOS          JCL00210
                     INCLUDE    A02%A02090C,A02080C,A02070C,A02060C@,OBJMOS              JCL00220
            /*


                                        *DEFINITIONS DICTIONARY*

SYMROL.   TYPE.  PHASE. ADDRESS.        SYMBOL.   TYPE.  PHASE. ADDRESS.        SYMROL.   TYPE.  PHASE. ADDRESS.

A01       CSECT  ROOT   0080117         A01010C   CSECT  ROOT   00000358        A01010E   ENTRY  ROOT   000003DC
A01020C   CSECT  ROOT   000003E0        A01020E   ENTRY  ROOT   00000468        A01030C   CSECT  ROOT   00000470
A01030E   ENTRY  ROOT   000004FC        A01040C   CSECT  ROOT   00000500        A01040E   ENTRY  ROOT   00000590
A02       CSECT  ROOT   0000118C        A02010C   CSECT  ROOT   00001188        A02010E   ENTRY  ROOT   00001220
A02020C   CSECT  ROOT   0000122         A02020E   ENTRY  ROOT   000012C4        A02030C   CSECT  ROOT   000012C8
A02030E   ENTRY  ROOT   0000136         A02040C   CSECT  ROOT   00001370        A02040E   ENTRY  ROOT   00001414
A02050C   CSECT  ROOT   0000059B        A02050E   ENTRY  ROOT   00000640        A02060C   CSECT  ROOT   00001418
A02060E   ENTRY  ROOT   000014C4        A02070C   CSECT  ROOT   000014C8        A02070E   ENTRY  ROOT   00001578
A02080C   CSECT  ROOT   00001580        A02080E   ENTRY  ROOT   00001634        A02090C   CSECT  ROOT   00001638
A02090E   ENTRY  ROOT   000016FC        A03       CSECT  ROOT   00000648        A03010C   CSECT  ROOT   00000650
A03010E   ENTRY  ROOT   000006FC        A03020C   CSECT  ROOT   00000700        A03020E   ENTRY  ROOT   00000780
A03030C   CSECT  ROOT   000007B8        A03030E   ENTRY  ROOT   0000086C        A03040C   CSECT  ROOT   00000870
A03040E   ENTRY  ROOT   0000092B        A03050C   CSECT  ROOT   00000930        A03050E   ENTRY  ROOT   000009EC
A03060C   CSECT  ROOT   000009F0        A03060E   ENTRY  ROOT   00000A80        A03070C   CSECT  ROOT   00000A88
A03070E   ENTRY  ROOT   00000B7C        A03080C   CSECT  ROOT   00000B80        A03080E   ENTRY  ROOT   00000C48
A03090C   CSECT  ROOT   00000C50        A03090E   ENTRY  ROOT   0000001C        A03100C   CSECT  ROOT   00000020
A03100E   ENTRY  ROOT   00000DF0        A03110C   CSECT  ROOT   00000DF8        A03110E   ENTRY  ROOT   00000ECC
A03120C   CSECT  ROOT   00000ED0        A03120E   ENTRY  ROOT   00000FA8        A03130C   CSECT  ROOT   00000FB0
A03130E   ENTRY  ROOT   0000108C        A03140C   CSECT  ROOT   00001090        A03140E   ENTRY  ROOT   00001170
KE$ALP    ENTRY  ABS    000016F4        KE$RES    ENTRY  ABS    000016F4        LK1ROOT   CSECT  ROOT   00000000


                                          ** ALLOCATION MAP **

                     LOAD MODULE -    SK1000         SIZE -    000016F4
```

Figure 8-2. Link-Edit Example 1 (Part 1 of 3)



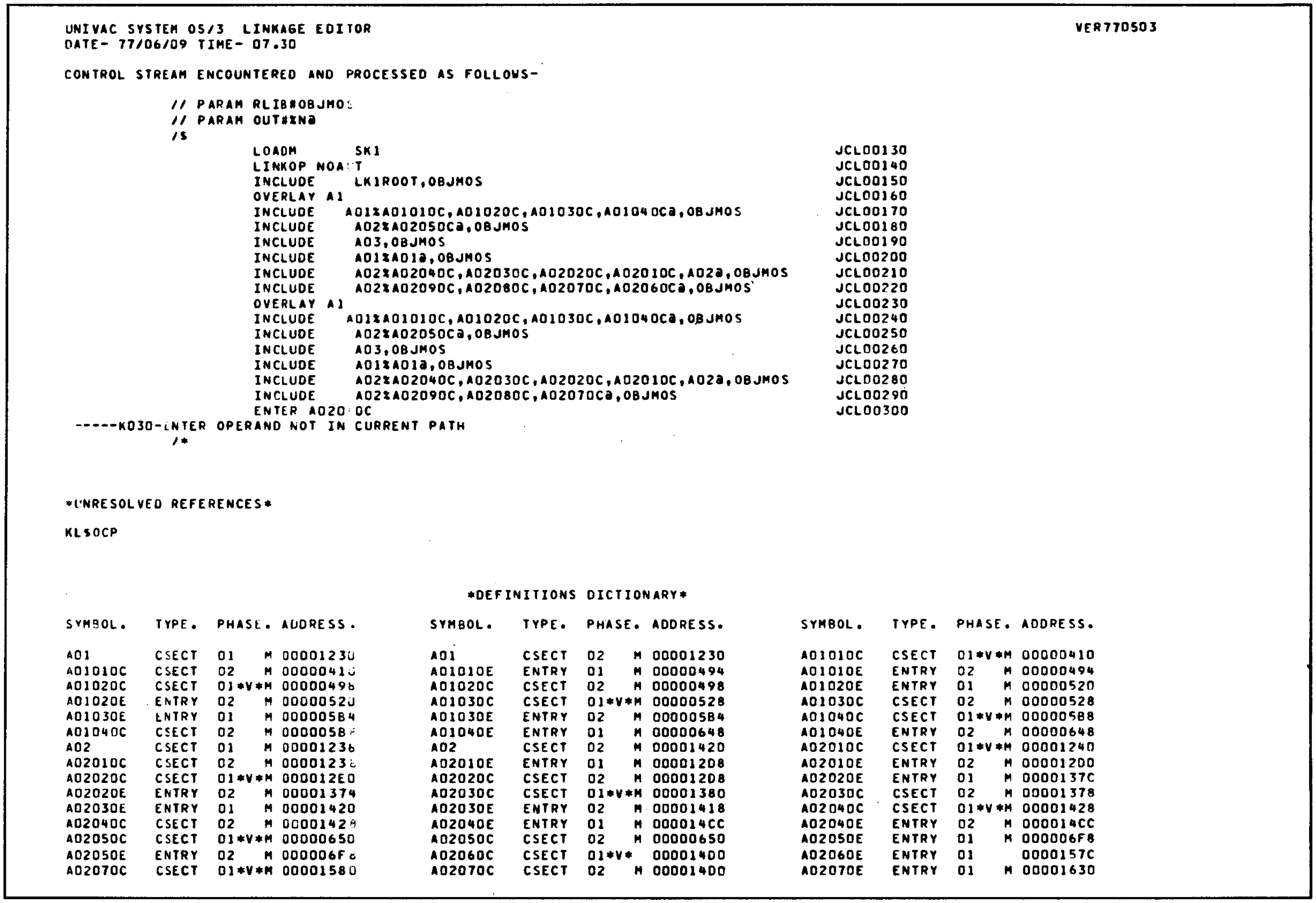
```
PHASE NAME  TRANS ADDR    FLAG       LABEL      TYPE        ESID         LNK ORG     HIADDR      LENGTH      OBJ ORG
SK100000       NODE - ROOT                                               00000000    000016F3    000016F4
*** START OF AUTO-INCLUDED ELEMENTS -
*** END OF AUTO-INCLUDED ELEMENTS -
        - 75/07/02 11.49 -           LK1ROOT      OBJ
                                     LK1ROOT    CSECT        01          00000000    00000353    00000354    00000000
        - 75/06/27 12.23 -           A01          OBJ
                                     A01010C    CSECT        02          00000358    000003DF    00000088    00000008
                                     A01020C    CSECT        1E          000003E0    00000468    0000008C    00000090
                                     A01030C    CSECT        1F          00000470    000004FF    00000090    00000120
                                     A01040C    CSECT        20          00000500    00000593    00000094    00000180
                                     A01010E      ENTRY      02          000003DC                            0000008C
                                     A01020E      ENTRY      1E          00000468                            00000118
                                     A01030E      ENTRY      1F          000004FC                            000001AC
                                     A01040E      ENTRY      20          00000590                            00000240
        - 75/06/27 12.25 -           A02          OBJ
                                     A02050C    CSECT        21          00000598    00000643    000000AC    00000298
                                     A02050E      ENTRY      21          00000640                            00000340
        - 75/06/27 12.28 -           A03          OBJ
                                     A03        CSECT        01          00000648    00000649    00000002    00000000
                                     A03010C    CSECT        02          00000650    000006FF    000000B0    00000008
                                     A03020C    CSECT        1E          00000700    000007B3    000000B4    000000B8
                                     A03030C    CSECT        1F          000007B8    0000086F    000000B8    00000170
                                     A03040C    CSECT        20          00000870    0000092B    000000BC    00000228
                                     A03050C    CSECT        21          00000930    000009EF    000000C0    000002E8
                                     A03060C    CSECT        22          000009F0    00000AB3    000000C4    000003A8
                                     A03070C    CSECT        23          00000AB8    00000B7F    000000C8    00000470
                                     A03080C    CSECT        24          00000B80    00000C4B    000000CC    00000538
                                     A03090C    CSECT        25          00000C50    0000001F    000000D0    00000608
                                     A03100C    CSECT        26          00000020    00000DF3    000000D4    000006D8
                                     A03110C    CSECT        27          00000DF8    00000ECF    000000D8    000007B0
                                     A03120C    CSECT        28          00000ED0    00000FAB    000000DC    00000888
                                     A03130C    CSECT        29          00000FB0    0000108F    000000E0    00000968
                                     A03140C    CSECT        2A          00001090    00001173    000000E4    00000A48
                                     A03010E      ENTRY      02          000006FC                            00000084
                                     A03020E      ENTRY      1E          000007B0                            00000168
                                     A03030E      ENTRY      1F          0000086C                            00000224
                                     A03040E      ENTRY      20          00000928                            000002E0
                                     A03050E      ENTRY      21          000009EC                            000003A4
                                     A03060E      ENTRY      22          00000AB0                            00000468
                                     A03070E      ENTRY      23          00000B7C                            00000534
                                     A03080E      ENTRY      24          00000C48                            00000600
                                     A03090E      ENTRY      25          0000001C                            000006D4
                                     A03100E      ENTRY      26          000000F0                            000007A8
                                     A03110E      ENTRY      27          00000ECC                            00000884
                                     A03120E      ENTRY      28          00000FA8                            00000960
                                     A03130E      ENTRY      29          0000108C                            00000A44
                                     A03140E      ENTRY      2A          00001170                            00000B28
        - 75/06/27 12.23 -           A01          OBJ
                                     A01        CSECT        01          00001178    00001179    00000002    00000000
        - 75/06/27 12.25 -           A02          OBJ
                                     A02        CSECT        01          00001180    00001181    00000002    00000000
                                     A02010C    CSECT        02          00001188    00001223    0000009C    00000008
                                     A02020C    CSECT        1E          00001228    000012C7    000000A0    000000A8
                                     A02030C    CSECT        1F          000012C8    0000136B    000000A4    00000148
                                     A02040C    CSECT        20          00001370    00001417    000000A8    000001F0
```

Figure 8-2. Link-Edit Example 1 (Part 2 of 3)



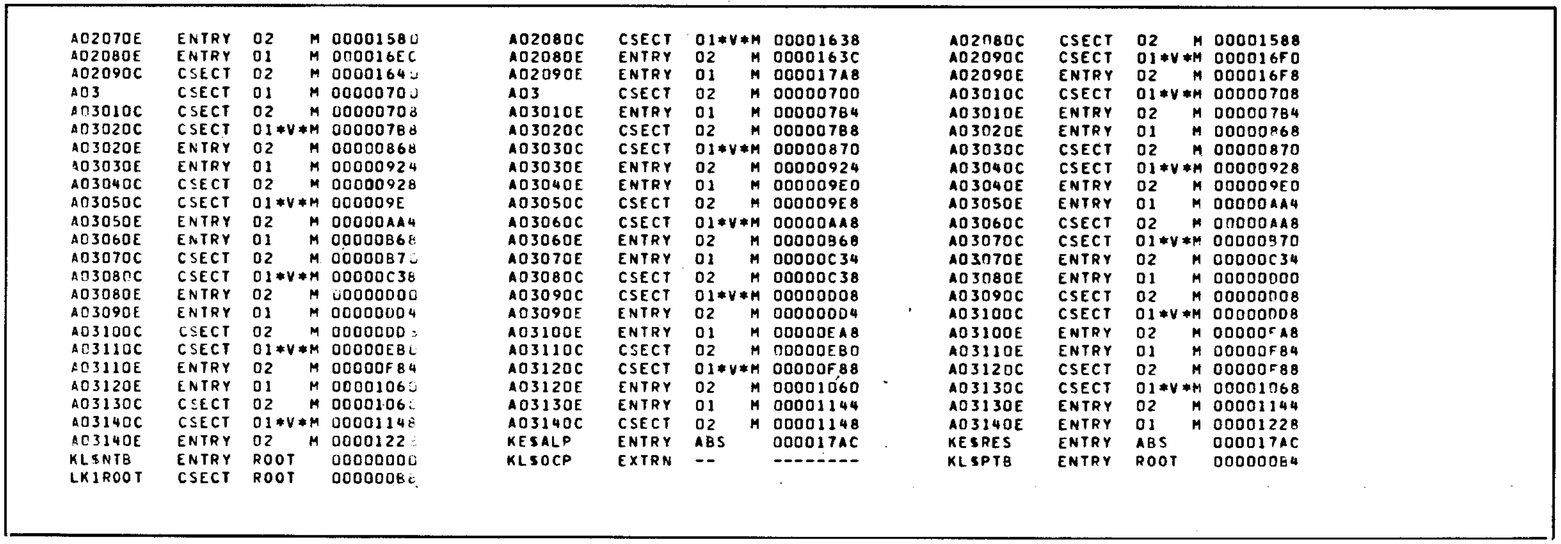
```
  PHASE NAME  TRANS ADDR    FLAG       LABEL      TYPE      ESID        LNK ORG     HIADDR      LENGTH      OBJ ORG
                                       A02010E     ENTRY     02         00001220                            000000A0
                                       A02020E     ENTRY     1E         000012C4                            00000144
                                       A02030E     ENTRY     1F         00001368                            000001E8
                                       A02040E     ENTRY     20         00001414                            00000294
           - 75/06/27 12.25 -          A02         OBJ
                                       A02060C    CSECT      22         00001418    000014C7    000000B0    00000348
                                       A02070C    CSECT      23         000014C8    0000157B    000000B4    000003F8
                                       A02080C    CSECT      24         00001580    00001637    000000B8    000004B0
                                       A02090C    CSECT      25         00001638    000016F3    000000BC    00000568
                                       A02060E     ENTRY     22         000014C4                            000003F4
                                       A02070E     ENTRY     23         00001578                            000004A8
                                       A02080E     ENTRY     24         00001634                            00000564
                                       A02090E     ENTRY     25         000016F0                            00000620
               00000000


                                                         FLAG CODES -
 B - BLK DATA CSECT      D - AUTO-DELETED        E - EXCLUSIVE .A. REF   G - GENERATED EXTRN     I - INCLUSIVE .V. REF
 L - DEFERRED LENGTH     M - MULTIPLY DEFINED    N - NOT INCLUDED        P - PROMOTED COMMON     R - SHARED REC PRODUCED
 S - SHARED ITEM         U - UNDEFINED REF       V - VCON ITEM
 *ANY OTHER CODES REPRESENT PROCESS ERRORS*


 LINK EDIT OF .SK1000.    COMPLETED
 DATE- 77/06/09 TIME- 07.34
 ERRORS ENCOUNTERED- 0003  UPSI- X.00.
```

Figure 8–2. Link-Edit Example 1 (Part 3 of 3)



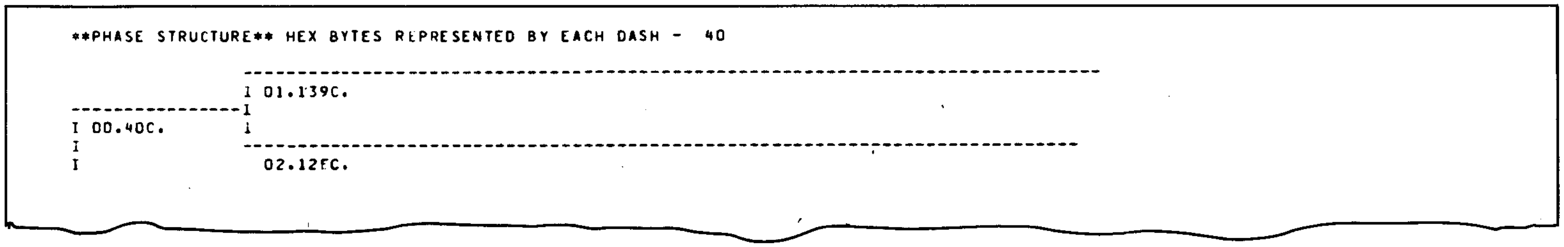
```
UNIVAC SYSTEM OS/3  LINKAGE EDITOR                                                                          VER770503
DATE- 77/06/09 TIME- 07.30

CONTROL STREAM ENCOUNTERED AND PROCESSED AS FOLLOWS-

            // PARAM RLIB#OBJMOS
            // PARAM OUT#%N@
            /$
                         LOADM      SK1                                                    JCL00130
                         LINKOP NOAUT                                                      JCL00140
                         INCLUDE    LK1ROOT,OBJMOS                                         JCL00150
                         OVERLAY A1                                                        JCL00160
                         INCLUDE   A01%A01010C,A01020C,A01030C,A01040C@,OBJMOS             JCL00170
                         INCLUDE    A02%A02050C@,OBJMOS                                    JCL00180
                         INCLUDE    A03,OBJMOS                                             JCL00190
                         INCLUDE    A01%A01@,OBJMOS                                        JCL00200
                         INCLUDE    A02%A02040C,A02030C,A02020C,A02010C,A02@,OBJMOS        JCL00210
                         INCLUDE    A02%A02090C,A02080C,A02070C,A02060C@,OBJMOS            JCL00220
                         OVERLAY A1                                                        JCL00230
                         INCLUDE   A01%A01010C,A01020C,A01030C,A01040C@,OBJMOS             JCL00240
                         INCLUDE    A02%A02050C@,OBJMOS                                    JCL00250
                         INCLUDE    A03,OBJMOS                                             JCL00260
                         INCLUDE    A01%A01@,OBJMOS                                        JCL00270
                         INCLUDE    A02%A02040C,A02030C,A02020C,A02010C,A02@,OBJMOS        JCL00280
                         INCLUDE    A02%A02090C,A02080C,A02070C@,OBJMOS                    JCL00290
                         ENTER A02010C                                                     JCL00300
 -----K030-ENTER OPERAND NOT IN CURRENT PATH
            /*


*UNRESOLVED REFERENCES*

KLSOCP


                                          *DEFINITIONS DICTIONARY*

SYMBOL.    TYPE.  PHASE. ADDRESS.          SYMBOL.    TYPE.  PHASE. ADDRESS.          SYMBOL.    TYPE.  PHASE. ADDRESS.

A01        CSECT  01    M 00001230         A01        CSECT  02    M 00001230         A01010C    CSECT  01*V*M 00000410
A01010C    CSECT  02    M 00000410         A01010E    ENTRY  01    M 00000494         A01010E    ENTRY  02    M 00000494
A01020C    CSECT  01*V*M 00000498          A01020C    CSECT  02    M 00000498         A01020E    ENTRY  01    M 00000520
A01020E    ENTRY  02    M 00000520         A01030C    CSECT  01*V*M 00000528          A01030C    CSECT  02    M 00000528
A01030E    ENTRY  01    M 000005B4         A01030E    ENTRY  02    M 000005B4         A01040C    CSECT  01*V*M 000005B8
A01040C    CSECT  02    M 000005B8         A01040E    ENTRY  01    M 00000648         A01040E    ENTRY  02    M 00000648
A02        CSECT  01    M 00001236         A02        CSECT  02    M 00001420         A02010C    CSECT  01*V*M 00001240
A02010C    CSECT  02    M 00001238         A02010E    ENTRY  01    M 00001208         A02010E    ENTRY  02    M 00001200
A02020C    CSECT  01*V*M 000012E0          A02020C    CSECT  02    M 00001208         A02020E    ENTRY  01    M 0000137C
A02020E    ENTRY  02    M 00001374         A02030C    CSECT  01*V*M 00001380          A02030C    CSECT  02    M 00001378
A02030E    ENTRY  01    M 00001420         A02030E    ENTRY  02    M 00001418         A02040C    CSECT  01*V*M 00001428
A02040C    CSECT  02    M 00001428         A02040E    ENTRY  01    M 000014CC         A02040E    ENTRY  02    M 000014CC
A02050C    CSECT  01*V*M 00000650          A02050C    CSECT  02    M 00000650         A02050E    ENTRY  01    M 000006F8
A02050E    ENTRY  02    M 000006F8         A02060C    CSECT  01*V*    00001400        A02060E    ENTRY  01      0000157C
A02070C    CSECT  01*V*M 00001580          A02070C    CSECT  02    M 00001400         A02070E    ENTRY  01    M 00001630
```

Figure 8-3. Link-Edit Example 2 (Part 1 of 6)



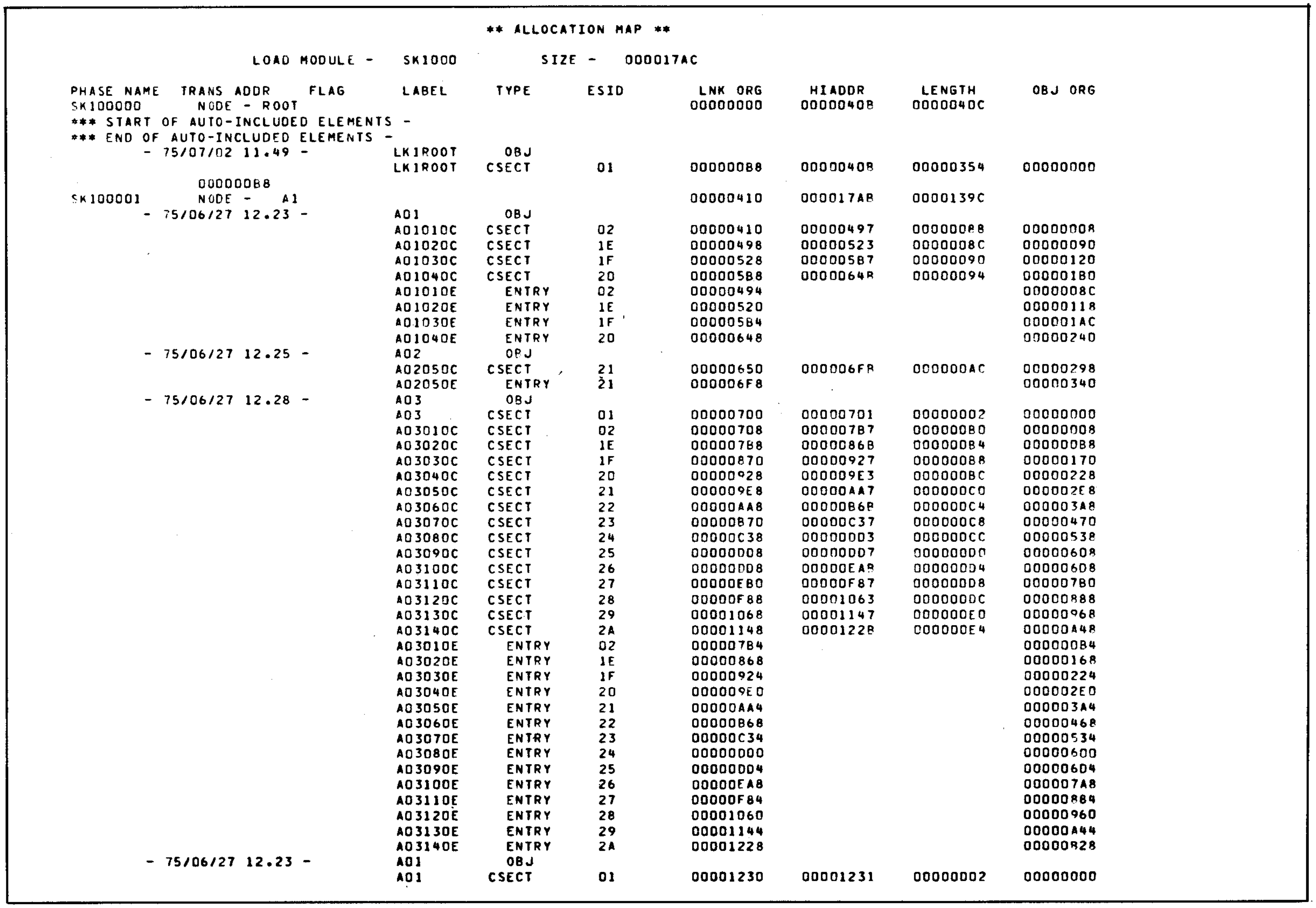
```
A02070E   ENTRY  02     M 00001580     A02080C   CSECT  01*V*M 00001638     A02080C   CSECT  02     M 00001588
A02080E   ENTRY  01     M 000016EC     A02080E   ENTRY  02     M 0000163C     A02090C   CSECT  01*V*M 000016F0
A02090C   CSECT  02     M 00001640     A02090E   ENTRY  01     M 000017A8     A02090E   ENTRY  02     M 000016F8
A03       CSECT  01     M 00000700     A03       CSECT  02     M 00000700     A03010C   CSECT  01*V*M 00000708
A03010C   CSECT  02     M 00000708     A03010E   ENTRY  01     M 000007B4     A03010E   ENTRY  02     M 000007B4
A03020C   CSECT  01*V*M 000007B8     A03020C   CSECT  02     M 000007B8     A03020E   ENTRY  01     M 00000868
A03020E   ENTRY  02     M 00000868     A03030C   CSECT  01*V*M 00000870     A03030C   CSECT  02     M 00000870
A03030E   ENTRY  01     M 00000924     A03030E   ENTRY  02     M 00000924     A03040C   CSECT  01*V*M 00000928
A03040C   CSECT  02     M 00000928     A03040E   ENTRY  01     M 000009E0     A03040E   ENTRY  02     M 000009E0
A03050C   CSECT  01*V*M 000009E        A03050C   CSECT  02     M 000009E8     A03050E   ENTRY  01     M 00000AA4
A03050E   ENTRY  02     M 00000AA4     A03060C   CSECT  01*V*M 00000AA8     A03060C   CSECT  02     M 00000AA8
A03060E   ENTRY  01     M 00000B68     A03060E   ENTRY  02     M 00000B68     A03070C   CSECT  01*V*M 00000B70
A03070C   CSECT  02     M 00000B70     A03070E   ENTRY  01     M 00000C34     A03070E   ENTRY  02     M 00000C34
A03080C   CSECT  01*V*M 00000C38     A03080C   CSECT  02     M 00000C38     A03080E   ENTRY  01     M 00000000
A03080E   ENTRY  02     M 00000000     A03090C   CSECT  01*V*M 00000008     A03090C   CSECT  02     M 00000008
A03090E   ENTRY  01     M 00000004     A03090E   ENTRY  02     M 000000D4     A03100C   CSECT  01*V*M 000000D8
A03100C   CSECT  02     M 000000D8     A03100E   ENTRY  01     M 00000EA8     A03100E   ENTRY  02     M 00000FA8
A03110C   CSECT  01*V*M 00000EB0     A03110C   CSECT  02     M 00000EB0     A03110E   ENTRY  01     M 00000F84
A03110E   ENTRY  02     M 00000F84     A03120C   CSECT  01*V*M 00000F88     A03120C   CSECT  02     M 00000F88
A03120E   ENTRY  01     M 00001060     A03120E   ENTRY  02     M 00001060     A03130C   CSECT  01*V*M 00001068
A03130C   CSECT  02     M 00001068     A03130E   ENTRY  01     M 00001144     A03130E   ENTRY  02     M 00001144
A03140C   CSECT  01*V*M 00001148     A03140C   CSECT  02     M 00001148     A03140E   ENTRY  01     M 00001228
A03140E   ENTRY  02     M 00001228     KE$ALP    ENTRY  ABS      000017AC     KE$RES    ENTRY  ABS      000017AC
KL$NTB    ENTRY  ROOT     00000000     KL$OCP    EXTRN  --       --------     KL$PTB    ENTRY  ROOT     000000B4
LK1ROOT   CSECT  ROOT     000000B8
```

Figure 8–3. Link-Edit Example 2 (Part 2 of 6)



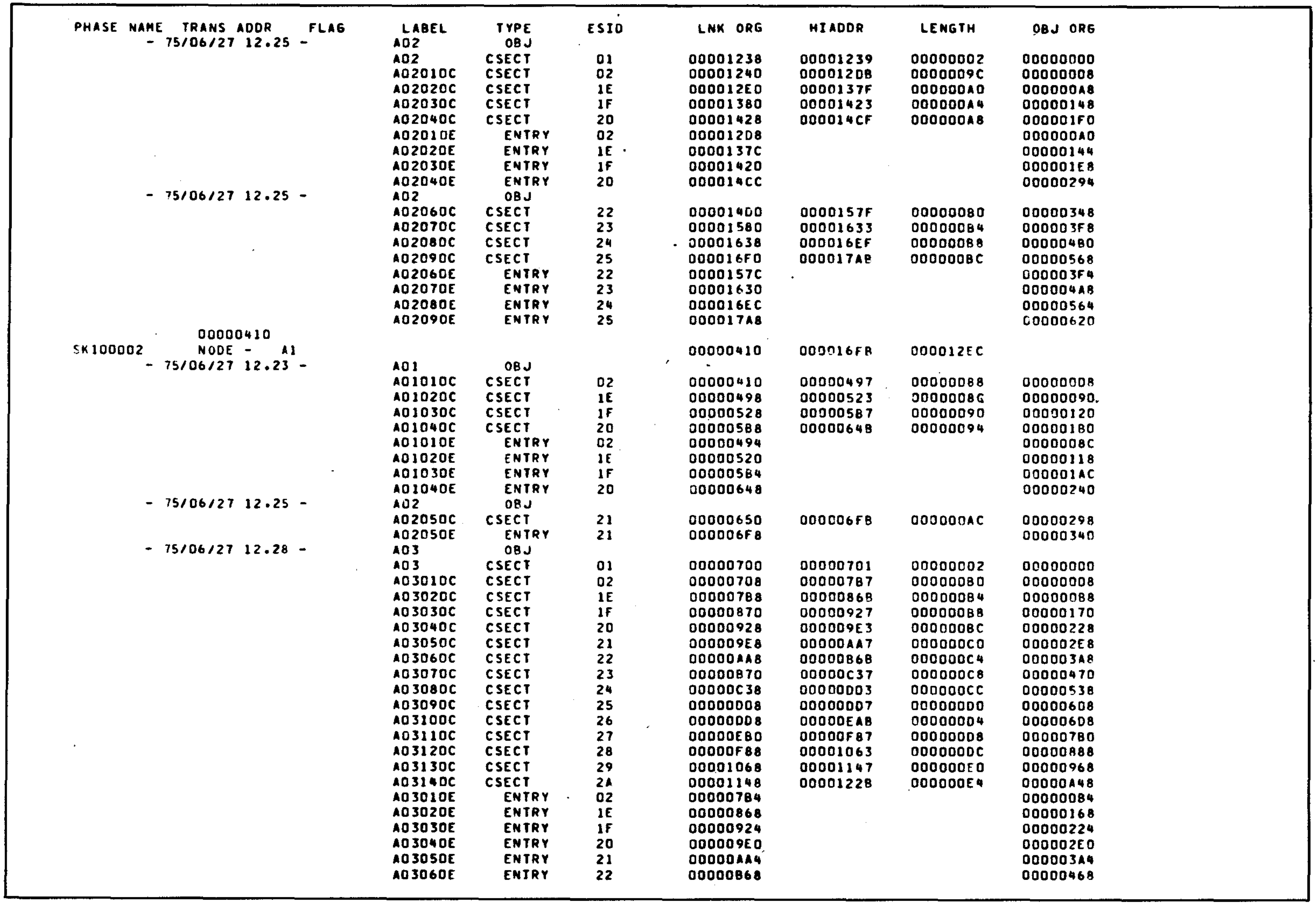
```
**PHASE STRUCTURE** HEX BYTES REPRESENTED BY EACH DASH -  40

                ------------------------------------------------------------------------------
                I 01.139C.
----------------I
I 00.40C.       I
I               ----------------------------------------------------------------------------
I                 02.12FC.
```

Figure 8-3. Link-Edit Example 2 (Part 3 of 6)



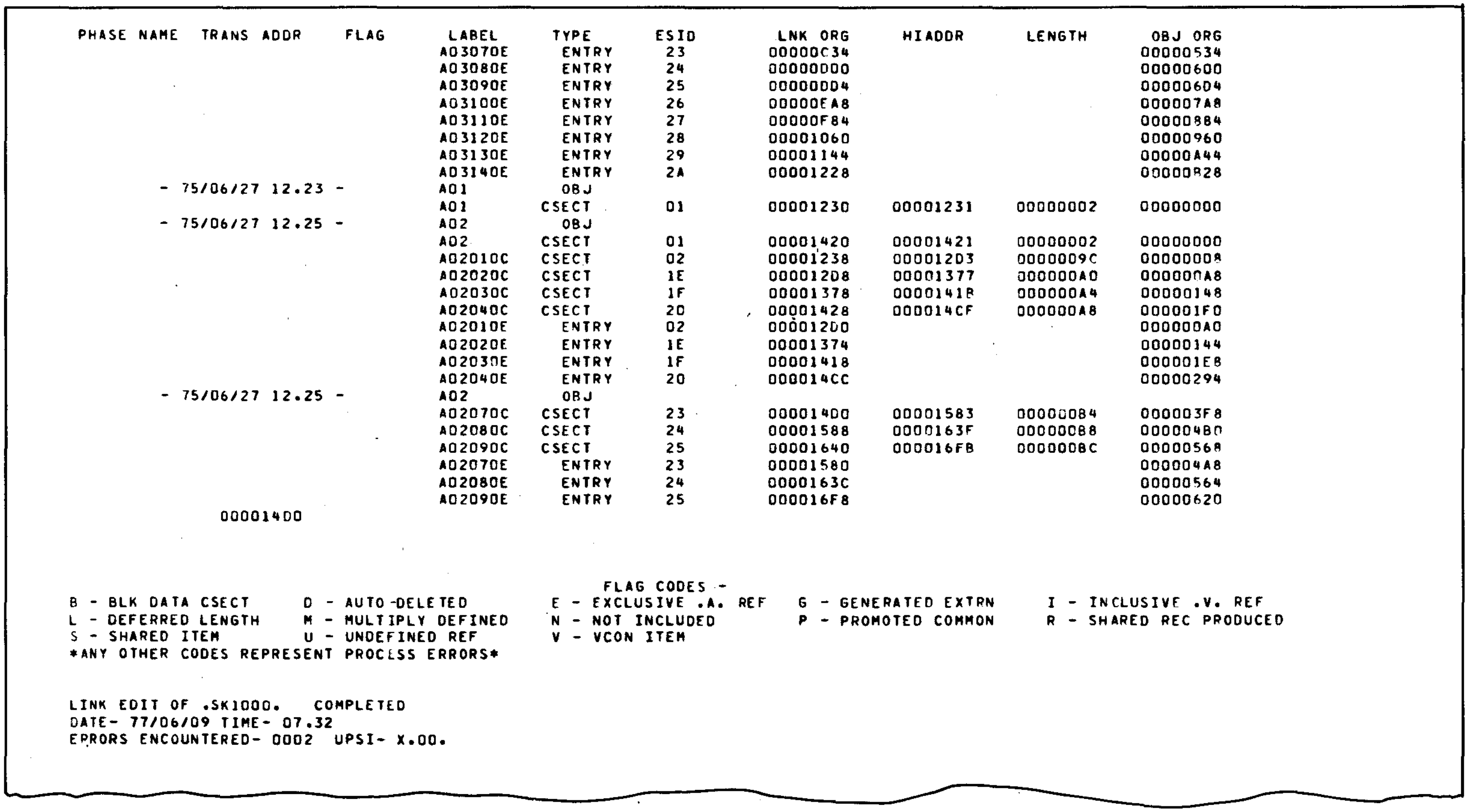
```
                                              ** ALLOCATION MAP **

                         LOAD MODULE -   SK1000           SIZE -   000017AC

PHASE NAME  TRANS ADDR    FLAG       LABEL      TYPE       ESID         LNK ORG      HIADDR       LENGTH       OBJ ORG
SK100000       NODE - ROOT                                              00000000     0000040B     0000040C
*** START OF AUTO-INCLUDED ELEMENTS -
*** END OF AUTO-INCLUDED ELEMENTS -
       - 75/07/02 11.49 -           LK1ROOT     OBJ
                                    LK1ROOT    CSECT        01          000000B8     0000040B     00000354     00000000
               000000B8
SK100001       NODE -   A1                                              00000410     000017AB     0000139C
       - 75/06/27 12.23 -           A01         OBJ
                                    A01010C    CSECT        02          00000410     00000497     00000088     00000008
                                    A01020C    CSECT        1E          00000498     00000523     0000008C     00000090
                                    A01030C    CSECT        1F          00000528     000005B7     00000090     00000120
                                    A01040C    CSECT        20          000005B8     0000064B     00000094     000001B0
                                    A01010E      ENTRY      02          00000494                               0000008C
                                    A01020E      ENTRY      1E          00000520                               00000118
                                    A01030E      ENTRY      1F          000005B4                               000001AC
                                    A01040E      ENTRY      20          00000648                               00000240
       - 75/06/27 12.25 -           A02         OBJ
                                    A02050C    CSECT        21          00000650     000006FB     000000AC     00000298
                                    A02050E      ENTRY      21          000006F8                               00000340
       - 75/06/27 12.28 -           A03         OBJ
                                    A03        CSECT        01          00000700     00000701     00000002     00000000
                                    A03010C    CSECT        02          00000708     000007B7     000000B0     00000008
                                    A03020C    CSECT        1E          000007B8     0000086B     000000B4     000000B8
                                    A03030C    CSECT        1F          00000870     00000927     000000B8     00000170
                                    A03040C    CSECT        20          00000928     000009E3     000000BC     00000228
                                    A03050C    CSECT        21          000009E8     00000AA7     000000C0     000002E8
                                    A03060C    CSECT        22          00000AA8     00000B6B     000000C4     000003A8
                                    A03070C    CSECT        23          00000B70     00000C37     000000C8     00000470
                                    A03080C    CSECT        24          00000C38     00000D03     000000CC     00000538
                                    A03090C    CSECT        25          00000D08     00000DD7     000000D0     00000608
                                    A03100C    CSECT        26          00000DD8     00000EAB     000000D4     000006D8
                                    A03110C    CSECT        27          00000EB0     00000F87     000000D8     000007B0
                                    A03120C    CSECT        28          00000F88     00001063     000000DC     00000888
                                    A03130C    CSECT        29          00001068     00001147     000000E0     00000968
                                    A03140C    CSECT        2A          00001148     0000122B     000000E4     00000A48
                                    A03010E      ENTRY      02          000007B4                               000000B4
                                    A03020E      ENTRY      1E          00000868                               00000168
                                    A03030E      ENTRY      1F          00000924                               00000224
                                    A03040E      ENTRY      20          000009E0                               000002E0
                                    A03050E      ENTRY      21          00000AA4                               000003A4
                                    A03060E      ENTRY      22          00000B68                               00000468
                                    A03070E      ENTRY      23          00000C34                               00000534
                                    A03080E      ENTRY      24          00000D00                               00000600
                                    A03090E      ENTRY      25          00000DD4                               000006D4
                                    A03100E      ENTRY      26          00000EA8                               000007A8
                                    A03110E      ENTRY      27          00000F84                               00000884
                                    A03120E      ENTRY      28          00001060                               00000960
                                    A03130E      ENTRY      29          00001144                               00000A44
                                    A03140E      ENTRY      2A          00001228                               00000B28
       - 75/06/27 12.23 -           A01         OBJ
                                    A01        CSECT        01          00001230     00001231     00000002     00000000
```

Figure 8-3. Link-Edit Example 2 (Part 4 of 6)



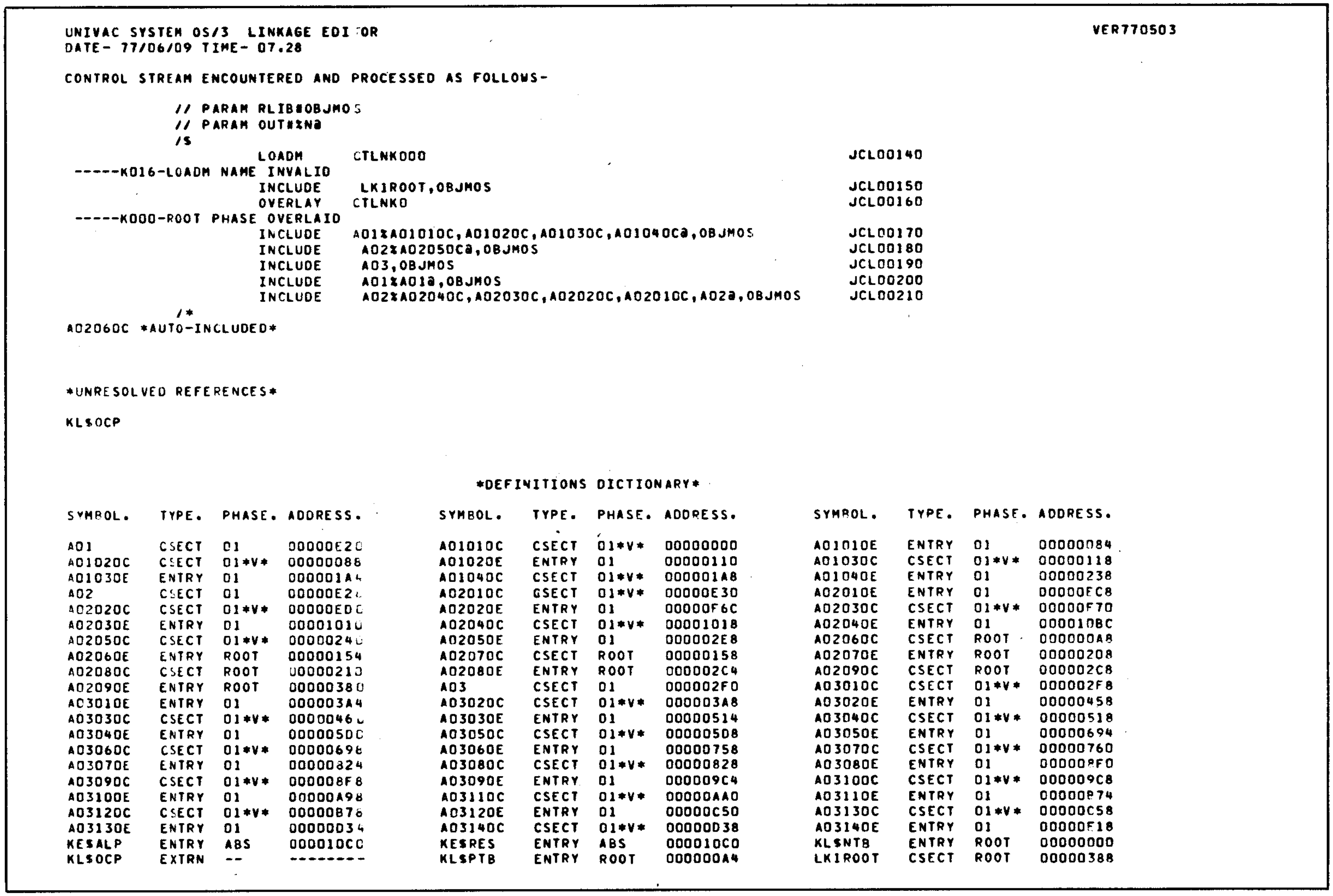
```
PHASE NAME  TRANS ADDR    FLAG       LABEL      TYPE       ESID         LNK ORG      HIADDR      LENGTH       OBJ ORG
      - 75/06/27 12.25 -             A02          OBJ
                                     A02        CSECT       01          00001238    00001239    00000002     00000000
                                     A02010C    CSECT       02          00001240    000012D8    0000009C     00000008
                                     A02020C    CSECT       1E          000012E0    0000137F    000000A0     000000A8
                                     A02030C    CSECT       1F          00001380    00001423    000000A4     00000148
                                     A02040C    CSECT       20          00001428    000014CF    000000A8     000001F0
                                     A02010E      ENTRY     02          000012D8                             000000A0
                                     A02020E      ENTRY     1E          0000137C                             00000144
                                     A02030E      ENTRY     1F          00001420                             000001E8
                                     A02040E      ENTRY     20          000014CC                             00000294
      - 75/06/27 12.25 -             A02          OBJ
                                     A02060C    CSECT       22          000014D0    0000157F    00000080     00000348
                                     A02070C    CSECT       23          00001580    00001633    000000B4     000003F8
                                     A02080C    CSECT       24          00001638    000016EF    000000B8     000004B0
                                     A02090C    CSECT       25          000016F0    000017AB    000000BC     00000568
                                     A02060E      ENTRY     22          0000157C                             000003F4
                                     A02070E      ENTRY     23          00001630                             000004A8
                                     A02080E      ENTRY     24          000016EC                             00000564
                                     A02090E      ENTRY     25          000017A8                             00000620
            00000410
SK100002    NODE -   A1                                                 00000410    000016FB    000012EC
      - 75/06/27 12.23 -             A01          OBJ
                                     A01010C    CSECT       02          00000410    00000497    00000088     00000008
                                     A01020C    CSECT       1E          00000498    00000523    0000008C     00000090
                                     A01030C    CSECT       1F          00000528    000005B7    00000090     00000120
                                     A01040C    CSECT       20          000005B8    0000064B    00000094     000001B0
                                     A01010E      ENTRY     02          00000494                             0000008C
                                     A01020E      ENTRY     1E          00000520                             00000118
                                     A01030E      ENTRY     1F          000005B4                             000001AC
                                     A01040E      ENTRY     20          00000648                             00000240
      - 75/06/27 12.25 -             A02          OBJ
                                     A02050C    CSECT       21          00000650    000006FB    000000AC     00000298
                                     A02050E      ENTRY     21          000006F8                             00000340
      - 75/06/27 12.28 -             A03          OBJ
                                     A03        CSECT       01          00000700    00000701    00000002     00000000
                                     A03010C    CSECT       02          00000708    000007B7    000000B0     00000008
                                     A03020C    CSECT       1E          000007B8    0000086B    000000B4     000000B8
                                     A03030C    CSECT       1F          00000870    00000927    000000B8     00000170
                                     A03040C    CSECT       20          00000928    000009E3    000000BC     00000228
                                     A03050C    CSECT       21          000009E8    00000AA7    000000C0     000002E8
                                     A03060C    CSECT       22          00000AA8    00000B6B    000000C4     000003A8
                                     A03070C    CSECT       23          00000B70    00000C37    000000C8     00000470
                                     A03080C    CSECT       24          00000C38    00000D03    000000CC     00000538
                                     A03090C    CSECT       25          00000D08    00000DD7    000000D0     00000608
                                     A03100C    CSECT       26          00000DD8    00000EAB    000000D4     000006D8
                                     A03110C    CSECT       27          00000EB0    00000F87    000000D8     000007B0
                                     A03120C    CSECT       28          00000F88    00001063    000000DC     00000888
                                     A03130C    CSECT       29          00001068    00001147    000000E0     00000968
                                     A03140C    CSECT       2A          00001148    0000122B    000000E4     00000A48
                                     A03010E      ENTRY     02          000007B4                             00000084
                                     A03020E      ENTRY     1E          00000868                             00000168
                                     A03030E      ENTRY     1F          00000924                             00000224
                                     A03040E      ENTRY     20          000009E0                             000002E0
                                     A03050E      ENTRY     21          00000AA4                             000003A4
                                     A03060E      ENTRY     22          00000B68                             00000468
```

Figure 8-3. Link-Edit Example 2 (Part 5 of 6)



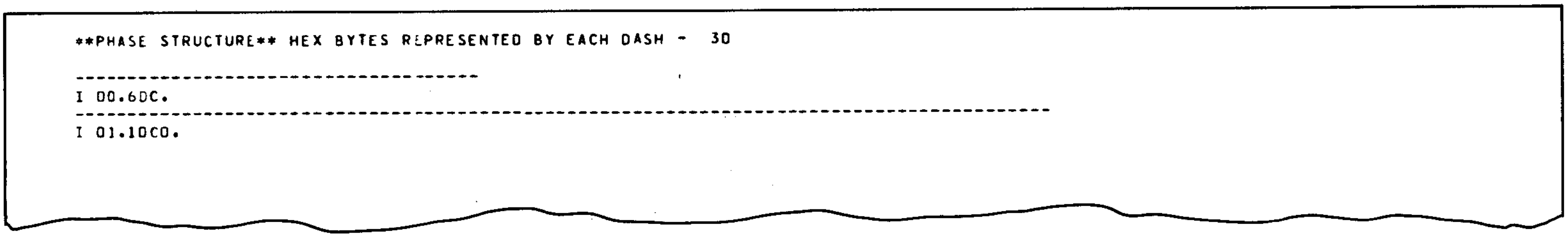
```
  PHASE NAME  TRANS ADDR    FLAG      LABEL     TYPE      ESID       LNK ORG      HIADDR      LENGTH       OBJ ORG
                                      A03070E    ENTRY     23        00000C34                              00000534
                                      A03080E    ENTRY     24        00000D00                              00000600
                                      A03090E    ENTRY     25        00000DD4                              000006D4
                                      A03100E    ENTRY     26        00000EA8                              000007A8
                                      A03110E    ENTRY     27        00000F84                              00000884
                                      A03120E    ENTRY     28        00001060                              00000960
                                      A03130E    ENTRY     29        00001144                              00000A44
                                      AD3140E    ENTRY     2A        00001228                              00000B28
         - 75/06/27 12.23 -           A01        OBJ
                                      A01       CSECT      01        00001230    00001231    00000002    00000000
         - 75/06/27 12.25 -           A02        OBJ
                                      A02       CSECT      01        00001420    00001421    00000002    00000000
                                      A02010C   CSECT      02        00001238    000012D3    0000009C    00000008
                                      A02020C   CSECT      1E        000012D8    00001377    000000A0    000000A8
                                      A02030C   CSECT      1F        00001378    0000141B    000000A4    00000148
                                      A02040C   CSECT      20        00001428    000014CF    000000A8    000001F0
                                      A02010E    ENTRY     02        000012D0                              000000A0
                                      A02020E    ENTRY     1E        00001374                              00000144
                                      A02030E    ENTRY     1F        00001418                              000001E8
                                      A02040E    ENTRY     20        000014CC                              00000294
         - 75/06/27 12.25 -           A02        OBJ
                                      A02070C   CSECT      23        000014D0    00001583    00000084    000003F8
                                      A02080C   CSECT      24        00001588    0000163F    00000088    000004B0
                                      A02090C   CSECT      25        00001640    000016FB    0000008C    00000568
                                      A02070E    ENTRY     23        00001580                              000004A8
                                      A02080E    ENTRY     24        0000163C                              00000564
                                      A02090E    ENTRY     25        000016F8                              00000620
              00001400


                                                    FLAG CODES -
B - BLK DATA CSECT     D - AUTO-DELETED        E - EXCLUSIVE .A. REF   G - GENERATED EXTRN     I - INCLUSIVE .V. REF
L - DEFERRED LENGTH    M - MULTIPLY DEFINED    N - NOT INCLUDED        P - PROMOTED COMMON     R - SHARED REC PRODUCED
S - SHARED ITEM        U - UNDEFINED REF       V - VCON ITEM
*ANY OTHER CODES REPRESENT PROCESS ERRORS*


LINK EDIT OF .SK1000.   COMPLETED
DATE- 77/06/09 TIME- 07.32
ERRORS ENCOUNTERED- 0002  UPSI- X.00.
```

Figure 8-3. Link-Edit Example 2 (Part 6 of 6)



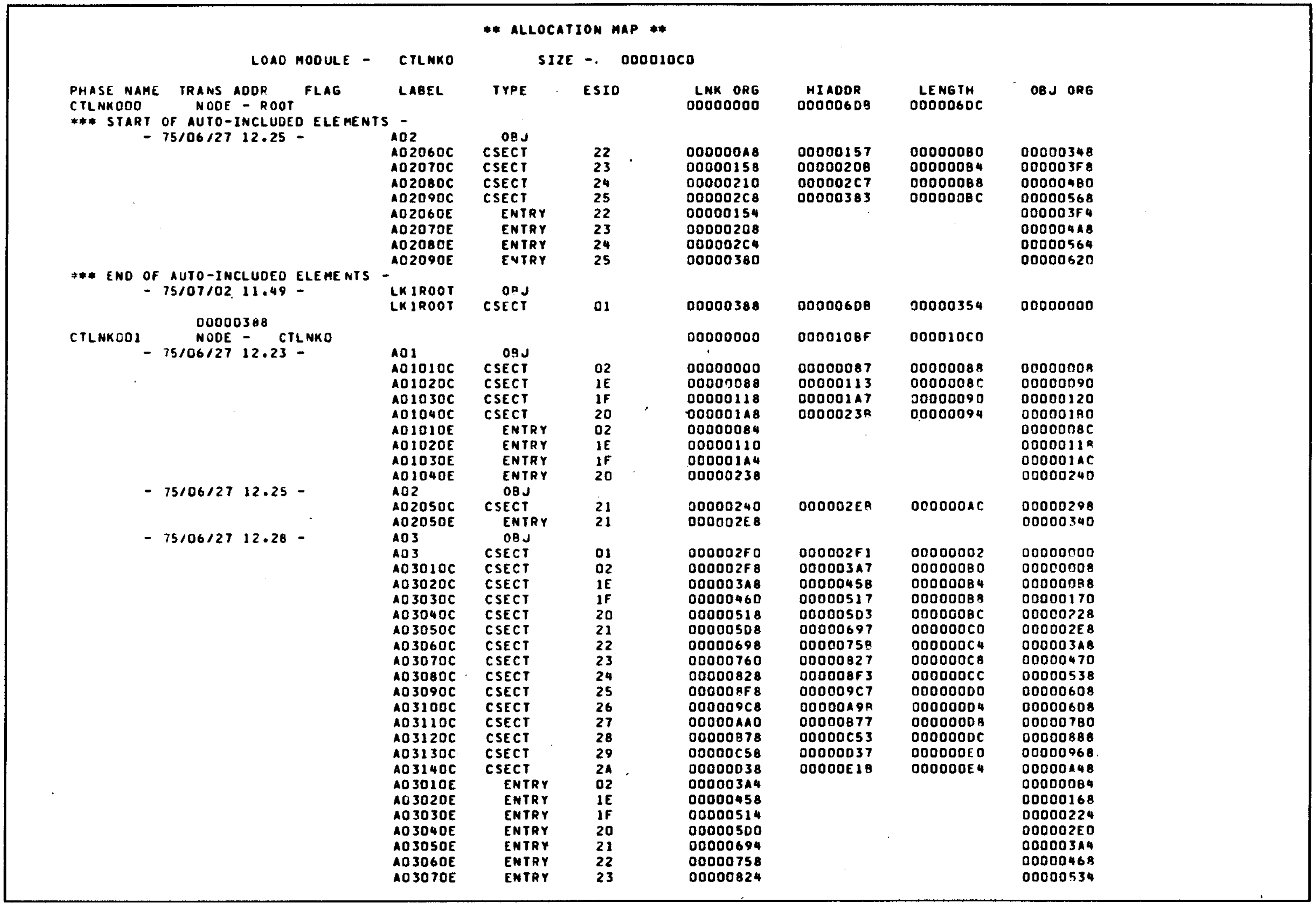
```
UNIVAC SYSTEM OS/3  LINKAGE EDITOR                                                                           VER770503
DATE- 77/06/09 TIME- 07.28

CONTROL STREAM ENCOUNTERED AND PROCESSED AS FOLLOWS-

           // PARAM RLIB#OBJMOS
           // PARAM OUT#%N@
           /$
                   LOADM      CTLNK000                                                      JCL00140
 -----K016-LOADM NAME INVALID
                   INCLUDE     LK1ROOT,OBJMOS                                               JCL00150
                   OVERLAY    CTLNK0                                                        JCL00160
 -----K000-ROOT PHASE OVERLAID
                   INCLUDE    A01%A01010C,A01020C,A01030C,A01040C@,OBJMOS                   JCL00170
                   INCLUDE     A02%A02050C@,OBJMOS                                          JCL00180
                   INCLUDE     A03,OBJMOS                                                   JCL00190
                   INCLUDE     A01%A01@,OBJMOS                                              JCL00200
                   INCLUDE     A02%A02040C,A02030C,A02020C,A02010C,A02@,OBJMOS              JCL00210
           /*
A02060C *AUTO-INCLUDED*


*UNRESOLVED REFERENCES*

KL$OCP


                                              *DEFINITIONS DICTIONARY*

SYMBOL.   TYPE.  PHASE. ADDRESS.          SYMBOL.   TYPE.  PHASE. ADDRESS.          SYMBOL.   TYPE.  PHASE. ADDRESS.

A01       CSECT  01     00000E20          A01010C   CSECT  01*V*  00000000          A01010E   ENTRY  01     00000084
A01020C   CSECT  01*V*  00000088          A01020E   ENTRY  01     00000110          A01030C   CSECT  01*V*  00000118
A01030E   ENTRY  01     000001A4          A01040C   CSECT  01*V*  000001A8          A01040E   ENTRY  01     00000238
A02       CSECT  01     00000E28          A02010C   GSECT  01*V*  00000E30          A02010E   ENTRY  01     00000EC8
A02020C   CSECT  01*V*  00000EDC          A02020E   ENTRY  01     00000F6C          A02030C   CSECT  01*V*  00000F70
A02030E   ENTRY  01     00001010          A02040C   CSECT  01*V*  00001018          A02040E   ENTRY  01     000010BC
A02050C   CSECT  01*V*  00000240          A02050E   ENTRY  01     000002E8          A02060C   CSECT  ROOT   000000A8
A02060E   ENTRY  ROOT   00000154          A02070C   CSECT  ROOT   00000158          A02070E   ENTRY  ROOT   00000208
A02080C   CSECT  ROOT   00000210          A02080E   ENTRY  ROOT   000002C4          A02090C   CSECT  ROOT   000002C8
A02090E   ENTRY  ROOT   00000380          A03       CSECT  01     000002F0          A03010C   CSECT  01*V*  000002F8
A03010E   ENTRY  01     000003A4          A03020C   CSECT  01*V*  000003A8          A03020E   ENTRY  01     00000458
A03030C   CSECT  01*V*  00000460          A03030E   ENTRY  01     00000514          A03040C   CSECT  01*V*  00000518
A03040E   ENTRY  01     000005DC          A03050C   CSECT  01*V*  00000508          A03050E   ENTRY  01     00000694
A03060C   CSECT  01*V*  00000698          A03060E   ENTRY  01     00000758          A03070C   CSECT  01*V*  00000760
A03070E   ENTRY  01     00000824          A03080C   CSECT  01*V*  00000828          A03080E   ENTRY  01     000008F0
A03090C   CSECT  01*V*  000008F8          A03090E   ENTRY  01     000009C4          A03100C   CSECT  01*V*  000009C8
A03100E   ENTRY  01     00000A98          A03110C   CSECT  01*V*  00000AA0          A03110E   ENTRY  01     00000B74
A03120C   CSECT  01*V*  00000B78          A03120E   ENTRY  01     00000C50          A03130C   CSECT  01*V*  00000C58
A03130E   ENTRY  01     00000D34          A03140C   CSECT  01*V*  00000D38          A03140E   ENTRY  01     00000E18
KE$ALP    ENTRY  ABS    000010C0          KE$RES    ENTRY  ABS    000010C0          KL$NTB    ENTRY  ROOT   00000000
KL$OCP    EXTRN  --     --------          KL$PTB    ENTRY  ROOT   000000A4          LK1ROOT   CSECT  ROOT   00000388
```

Figure 8–4. Link-Edit Example 3 (Part 1 of 4)



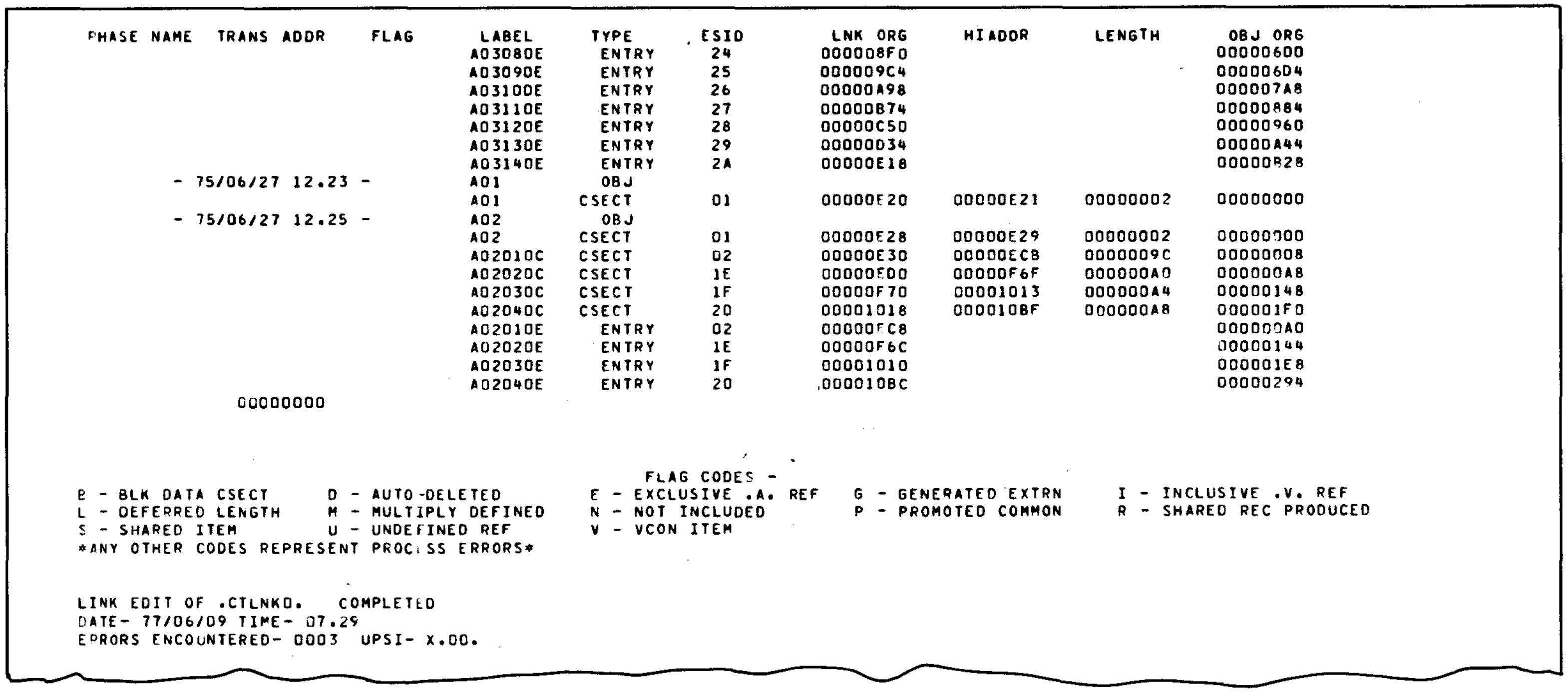
```
**PHASE STRUCTURE** HEX BYTES REPRESENTED BY EACH DASH -  30

-------------------------------------
I 00.6DC.
------------------------------------------------------------------------------------------
I 01.10C0.
```

Figure 8-4. Link-Edit Example 3 (Part 2 of 4)



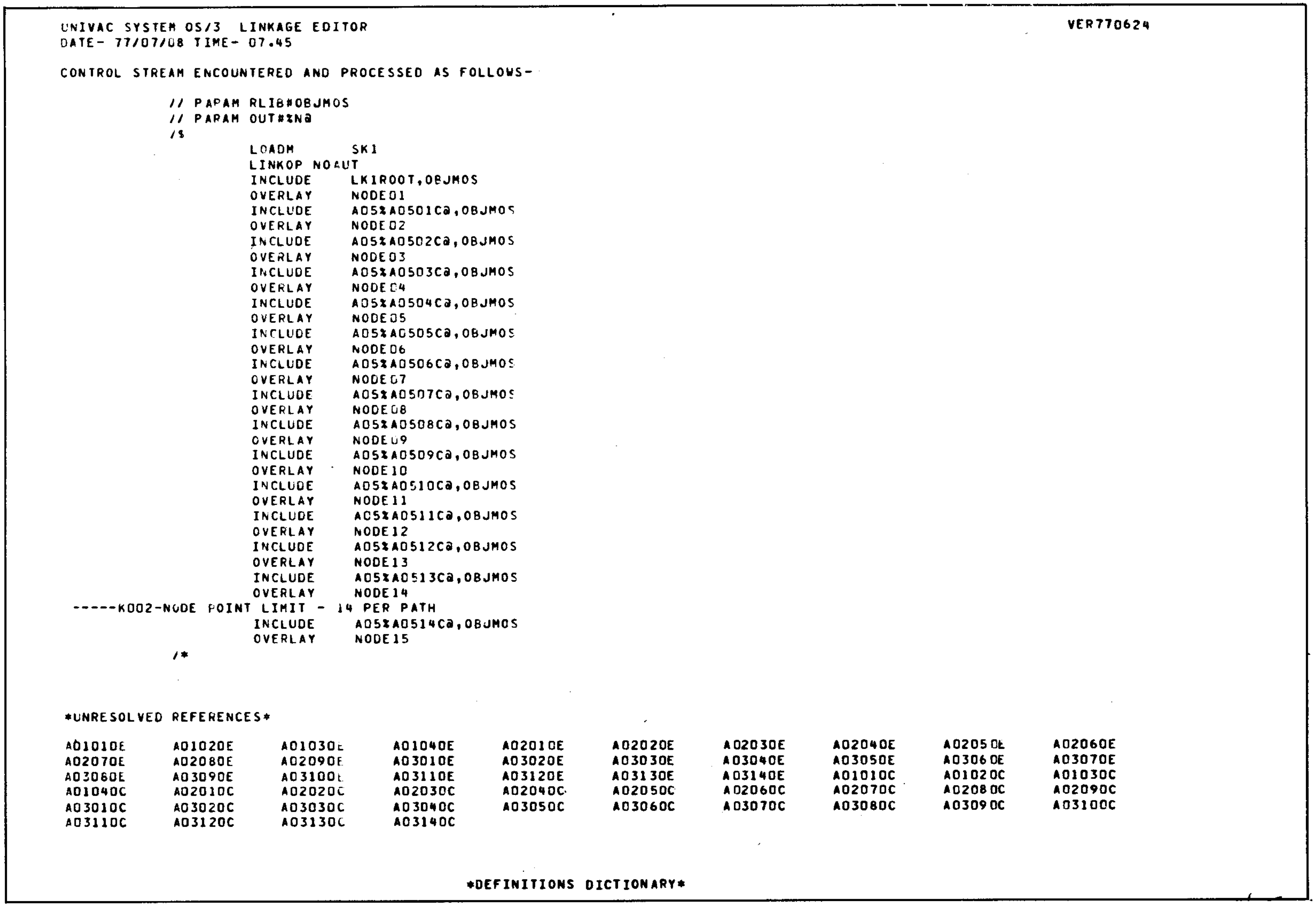
```
                                                    ** ALLOCATION MAP **

                         LOAD MODULE -   CTLNKO          SIZE -.  000010C0

PHASE NAME  TRANS ADDR    FLAG          LABEL      TYPE        ESID         LNK ORG       HIADDR        LENGTH        OBJ ORG
CTLNK000      NODE - ROOT                                                   00000000      000006D8      000006DC
*** START OF AUTO-INCLUDED ELEMENTS -
      - 75/06/27 12.25 -               A02           OBJ
                                       A02060C     CSECT       22          000000A8      00000157      00000080      00000348
                                       A02070C     CSECT       23          00000158      00000208      00000084      000003F8
                                       A02080C     CSECT       24          00000210      000002C7      00000088      000004B0
                                       A02090C     CSECT       25          000002C8      00000383      000000BC      00000568
                                       A02060E       ENTRY     22          00000154                                  000003F4
                                       A02070E       ENTRY     23          00000208                                  000004A8
                                       A02080E       ENTRY     24          000002C4                                  00000564
                                       A02090E       ENTRY     25          00000380                                  00000620
*** END OF AUTO-INCLUDED ELEMENTS -
      - 75/07/02 11.49 -               LK1ROOT       OBJ
                                       LK1ROOT     CSECT       01          00000388      000006D8      00000354      00000000
          00000388
CTLNK001      NODE -   CTLNKO                                               00000000      0000108F      000010C0
      - 75/06/27 12.23 -               A01           OBJ
                                       A01010C     CSECT       02          00000000      00000087      00000088      00000008
                                       A01020C     CSECT       1E          00000088      00000113      0000008C      00000090
                                       A01030C     CSECT       1F          00000118      000001A7      00000090      00000120
                                       A01040C     CSECT       20          000001A8      00000238      00000094      000001B0
                                       A01010E       ENTRY     02          00000084                                  0000008C
                                       A01020E       ENTRY     1E          00000110                                  00000118
                                       A01030E       ENTRY     1F          000001A4                                  000001AC
                                       A01040E       ENTRY     20          00000238                                  00000240
      - 75/06/27 12.25 -               A02           OBJ
                                       A02050C     CSECT       21          00000240      000002E8      000000AC      00000298
                                       A02050E       ENTRY     21          000002E8                                  00000340
      - 75/06/27 12.28 -               A03           OBJ
                                       A03         CSECT       01          000002F0      000002F1      00000002      00000000
                                       A03010C     CSECT       02          000002F8      000003A7      000000B0      00000008
                                       A03020C     CSECT       1E          000003A8      0000045B      000000B4      000000B8
                                       A03030C     CSECT       1F          00000460      00000517      000000B8      00000170
                                       A03040C     CSECT       20          00000518      000005D3      000000BC      00000228
                                       A03050C     CSECT       21          000005D8      00000697      000000C0      000002E8
                                       A03060C     CSECT       22          00000698      0000075B      000000C4      000003A8
                                       A03070C     CSECT       23          00000760      00000827      000000C8      00000470
                                       A03080C     CSECT       24          00000828      000008F3      000000CC      00000538
                                       A03090C     CSECT       25          000008F8      000009C7      000000D0      00000608
                                       A03100C     CSECT       26          000009C8      00000A9B      000000D4      00000608
                                       A03110C     CSECT       27          00000AA0      00000B77      000000D8      000007B0
                                       A03120C     CSECT       28          00000B78      00000C53      000000DC      00000888
                                       A03130C     CSECT       29          00000C58      00000D37      000000E0      00000968
                                       A03140C     CSECT       2A          00000D38      00000E1B      000000E4      00000A48
                                       A03010E       ENTRY     02          000003A4                                  000000B4
                                       A03020E       ENTRY     1E          00000458                                  00000168
                                       A03030E       ENTRY     1F          00000514                                  00000224
                                       A03040E       ENTRY     20          000005D0                                  000002E0
                                       A03050E       ENTRY     21          00000694                                  000003A4
                                       A03060E       ENTRY     22          00000758                                  00000468
                                       A03070E       ENTRY     23          00000824                                  00000534
```

Figure 8-4. Link-Edit Example 3 (Part 3 of 4)



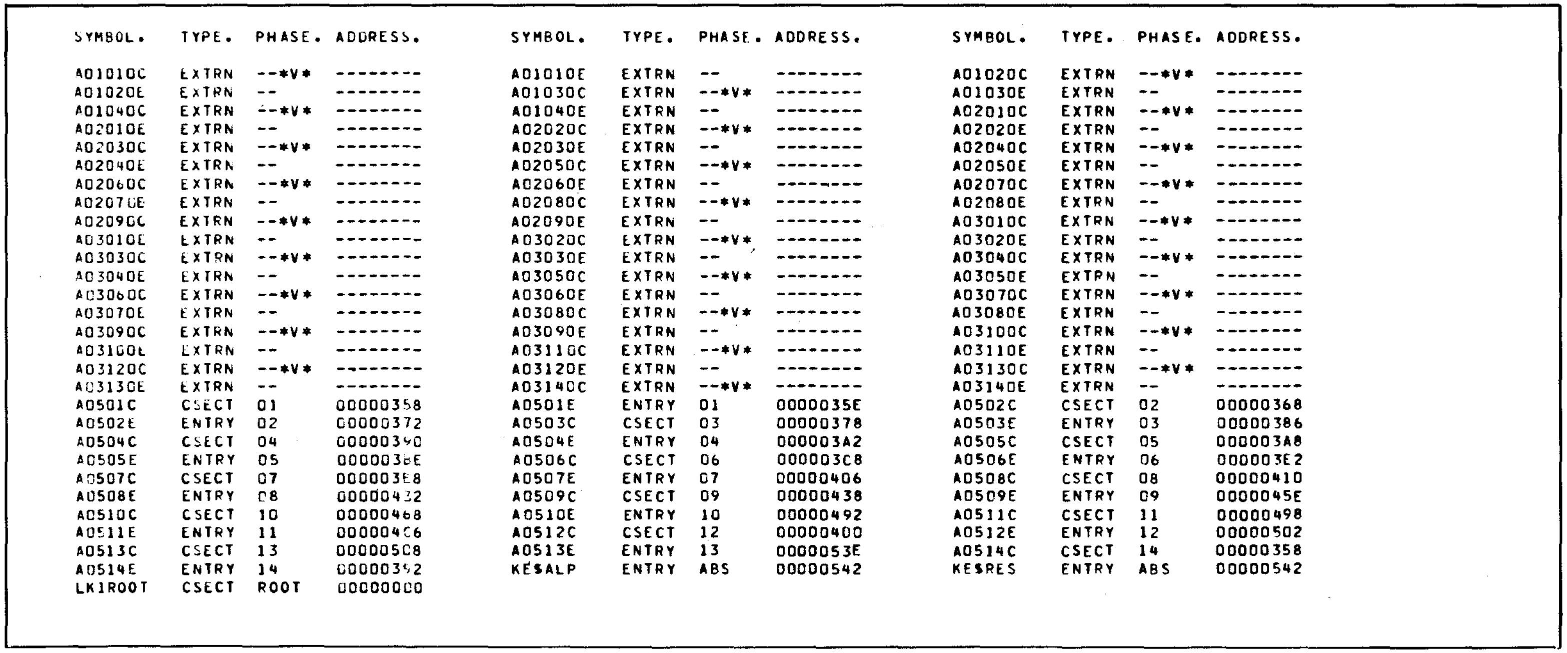
```
PHASE NAME  TRANS ADDR    FLAG      LABEL      TYPE     ESID       LNK ORG     HIADDR      LENGTH       OBJ ORG
                                   A03080E     ENTRY     24       000008F0                             00000600
                                   A03090E     ENTRY     25       000009C4                             000006D4
                                   A03100E     ENTRY     26       00000A98                             000007A8
                                   A03110E     ENTRY     27       00000B74                             00000884
                                   A03120E     ENTRY     28       00000C50                             00000960
                                   A03130E     ENTRY     29       00000D34                             00000A44
                                   A03140E     ENTRY     2A       00000E18                             00000B28
        - 75/06/27 12.23 -         A01         OBJ
                                   A01       CSECT       01       00000E20    00000E21    00000002     00000000
        - 75/06/27 12.25 -         A02         OBJ
                                   A02       CSECT       01       00000E28    00000E29    00000002     00000000
                                   A02010C   CSECT       02       00000E30    00000ECB    0000009C     00000008
                                   A02020C   CSECT       1E       00000ED0    00000F6F    000000A0     000000A8
                                   A02030C   CSECT       1F       00000F70    00001013    000000A4     00000148
                                   A02040C   CSECT       20       00001018    000010BF    000000A8     000001F0
                                   A02010E     ENTRY     02       00000FC8                             000000A0
                                   A02020E     ENTRY     1E       00000F6C                             00000144
                                   A02030E     ENTRY     1F       00001010                             000001E8
                                   A02040E     ENTRY     20       000010BC                             00000294
              00000000


                                                    FLAG CODES -
B - BLK DATA CSECT      D - AUTO-DELETED        E - EXCLUSIVE .A. REF   G - GENERATED EXTRN     I - INCLUSIVE .V. REF
L - DEFERRED LENGTH     M - MULTIPLY DEFINED    N - NOT INCLUDED        P - PROMOTED COMMON     R - SHARED REC PRODUCED
S - SHARED ITEM         U - UNDEFINED REF       V - VCON ITEM
*ANY OTHER CODES REPRESENT PROCESS ERRORS*


LINK EDIT OF .CTLNKD.   COMPLETED
DATE- 77/06/09 TIME- 07.29
ERRORS ENCOUNTERED- 0003  UPSI- X.00.
```

Figure 8-4. Link-Edit Example 3 (Part 4 of 4)



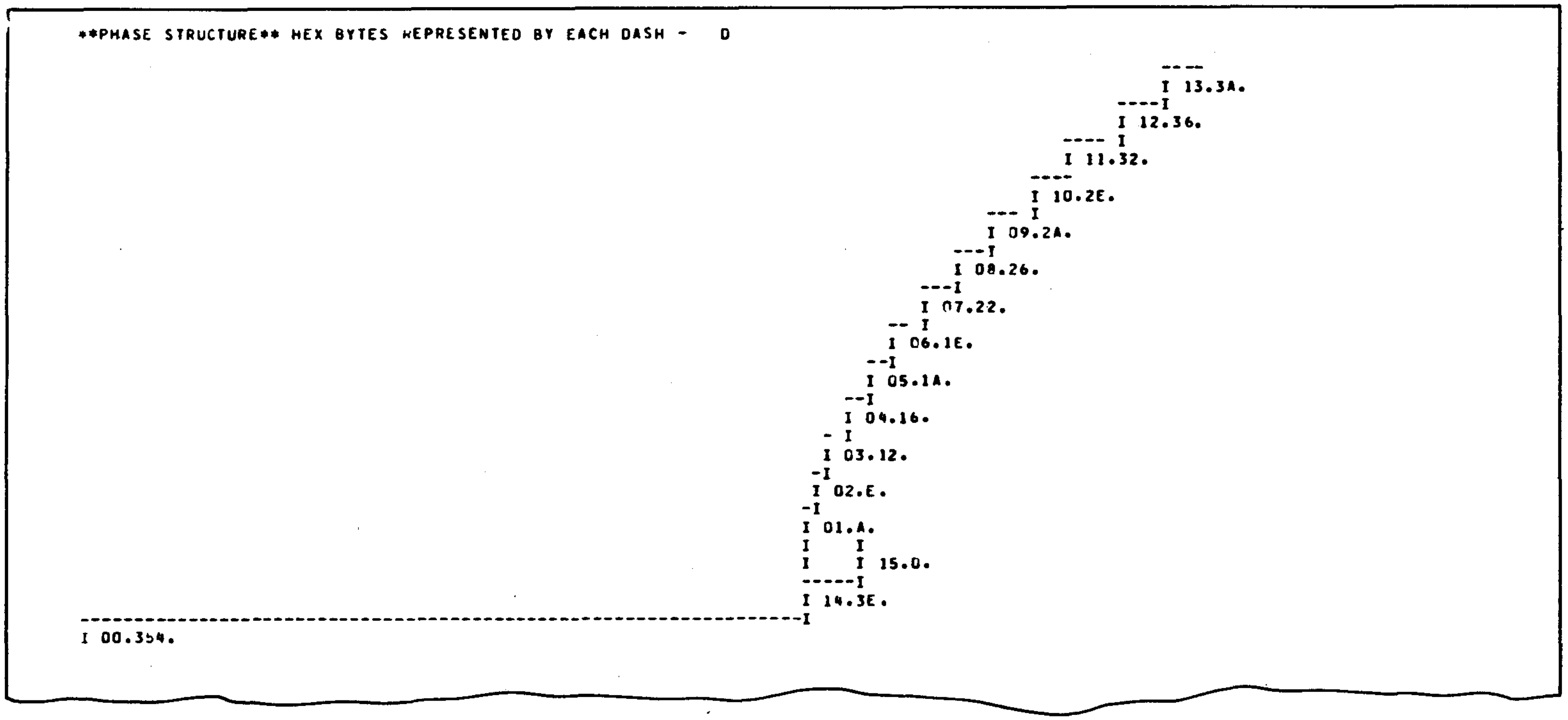
```
UNIVAC SYSTEM OS/3  LINKAGE EDITOR                                              VER770624
DATE- 77/07/08 TIME- 07.45

CONTROL STREAM ENCOUNTERED AND PROCESSED AS FOLLOWS-

            // PAPAM RLIB#OBJMOS
            // PARAM OUT#%N@
            /$
                     LOADM       SK1
                     LINKOP NOAUT
                     INCLUDE     LK1ROOT,OBJMOS
                     OVERLAY     NODE01
                     INCLUDE     A05%A0501C@,OBJMOS
                     OVERLAY     NODE02
                     INCLUDE     A05%A0502C@,OBJMOS
                     OVERLAY     NODE03
                     INCLUDE     A05%A0503C@,OBJMOS
                     OVERLAY     NODE04
                     INCLUDE     A05%A0504C@,OBJMOS
                     OVERLAY     NODE05
                     INCLUDE     A05%A0505C@,OBJMOS
                     OVERLAY     NODE06
                     INCLUDE     A05%A0506C@,OBJMOS
                     OVERLAY     NODE07
                     INCLUDE     A05%A0507C@,OBJMOS
                     OVERLAY     NODE08
                     INCLUDE     A05%A0508C@,OBJMOS
                     OVERLAY     NODE09
                     INCLUDE     A05%A0509C@,OBJMOS
                     OVERLAY     NODE10
                     INCLUDE     A05%A0510C@,OBJMOS
                     OVERLAY     NODE11
                     INCLUDE     A05%A0511C@,OBJMOS
                     OVERLAY     NODE12
                     INCLUDE     A05%A0512C@,OBJMOS
                     OVERLAY     NODE13
                     INCLUDE     A05%A0513C@,OBJMOS
                     OVERLAY     NODE14
 -----K002-NODE POINT LIMIT - 14 PER PATH
                     INCLUDE     A05%A0514C@,OBJMOS
                     OVERLAY     NODE15
            /*


*UNRESOLVED REFERENCES*

A01010E    A01020E    A01030E    A01040E    A02010E    A02020E    A02030E    A02040E    A02050E    A02060E
A02070E    A02080E    A02090E    A03010E    A03020E    A03030E    A03040E    A03050E    A03060E    A03070E
A03060E    A03090E    A03100E    A03110E    A03120E    A03130E    A03140E    A01010C    A01020C    A01030C
A01040C    A02010C    A02020C    A02030C    A02040C    A02050C    A02060C    A02070C    A02080C    A02090C
A03010C    A03020C    A03030C    A03040C    A03050C    A03060C    A03070C    A03080C    A03090C    A03100C
A03110C    A03120C    A03130C    A03140C


                              *DEFINITIONS DICTIONARY*
```

Figure 8-5. Link-Edit Example 4 (Part 1 of 5)



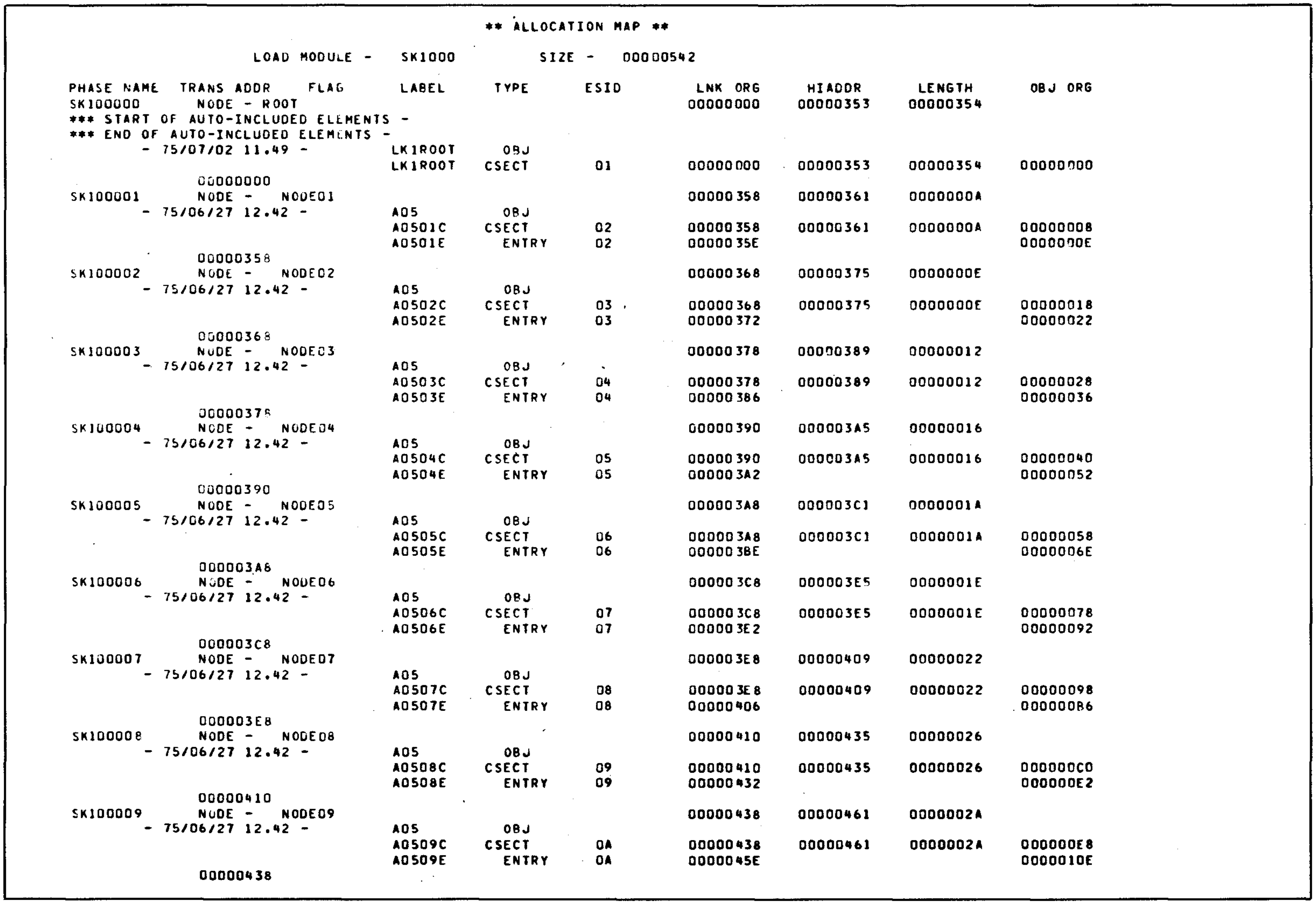
```
SYMBOL.   TYPE.  PHASE. ADDRESS.        SYMBOL.   TYPE.  PHASE. ADDRESS.        SYMBOL.   TYPE.  PHASE. ADDRESS.

A01010C   EXTRN  --*V*  --------        A01010E   EXTRN  --     --------        A01020C   EXTRN  --*V*  --------
A01020E   EXTRN  --     --------        A01030C   EXTRN  --*V*  --------        A01030E   EXTRN  --     --------
A01040C   EXTRN  --*V*  --------        A01040E   EXTRN  --     --------        A02010C   EXTRN  --*V*  --------
A02010E   EXTRN  --     --------        A02020C   EXTRN  --*V*  --------        A02020E   EXTRN  --     --------
A02030C   EXTRN  --*V*  --------        A02030E   EXTRN  --     --------        A02040C   EXTRN  --*V*  --------
A02040E   EXTRN  --     --------        A02050C   EXTRN  --*V*  --------        A02050E   EXTRN  --     --------
A02060C   EXTRN  --*V*  --------        A02060E   EXTRN  --     --------        A02070C   EXTRN  --*V*  --------
A02070E   EXTRN  --     --------        A02080C   EXTRN  --*V*  --------        A02080E   EXTRN  --     --------
A02090C   EXTRN  --*V*  --------        A02090E   EXTRN  --     --------        A03010C   EXTRN  --*V*  --------
A03010E   EXTRN  --     --------        A03020C   EXTRN  --*V*  --------        A03020E   EXTRN  --     --------
A03030C   EXTRN  --*V*  --------        A03030E   EXTRN  --     --------        A03040C   EXTRN  --*V*  --------
A03040E   EXTRN  --     --------        A03050C   EXTRN  --*V*  --------        A03050E   EXTRN  --     --------
A03060C   EXTRN  --*V*  --------        A03060E   EXTRN  --     --------        A03070C   EXTRN  --*V*  --------
A03070E   EXTRN  --     --------        A03080C   EXTRN  --*V*  --------        A03080E   EXTRN  --     --------
A03090C   EXTRN  --*V*  --------        A03090E   EXTRN  --     --------        A03100C   EXTRN  --*V*  --------
A03100E   EXTRN  --     --------        A03110C   EXTRN  --*V*  --------        A03110E   EXTRN  --     --------
A03120C   EXTRN  --*V*  --------        A03120E   EXTRN  --     --------        A03130C   EXTRN  --*V*  --------
A03130E   EXTRN  --     --------        A03140C   EXTRN  --*V*  --------        A03140E   EXTRN  --     --------
A0501C    CSECT  01     00000358        A0501E    ENTRY  01     0000035E        A0502C    CSECT  02     00000368
A0502E    ENTRY  02     00000372        A0503C    CSECT  03     00000378        A0503E    ENTRY  03     00000386
A0504C    CSECT  04     00000390        A0504E    ENTRY  04     000003A2        A0505C    CSECT  05     000003A8
A0505E    ENTRY  05     000003BE        A0506C    CSECT  06     000003C8        A0506E    ENTRY  06     000003E2
A0507C    CSECT  07     000003E8        A0507E    ENTRY  07     00000406        A0508C    CSECT  08     00000410
A0508E    ENTRY  08     00000432        A0509C    CSECT  09     00000438        A0509E    ENTRY  09     0000045E
A0510C    CSECT  10     00000468        A0510E    ENTRY  10     00000492        A0511C    CSECT  11     00000498
A0511E    ENTRY  11     000004C6        A0512C    CSECT  12     00000400        A0512E    ENTRY  12     00000502
A0513C    CSECT  13     000005C8        A0513E    ENTRY  13     0000053E        A0514C    CSECT  14     00000358
A0514E    ENTRY  14     00000392        KE$ALP    ENTRY  ABS    00000542        KE$RES    ENTRY  ABS    00000542
LK1ROOT   CSECT  ROOT   00000000
```

Figure 8-5. Link-Edit Example 4 (Part 2 of 5)



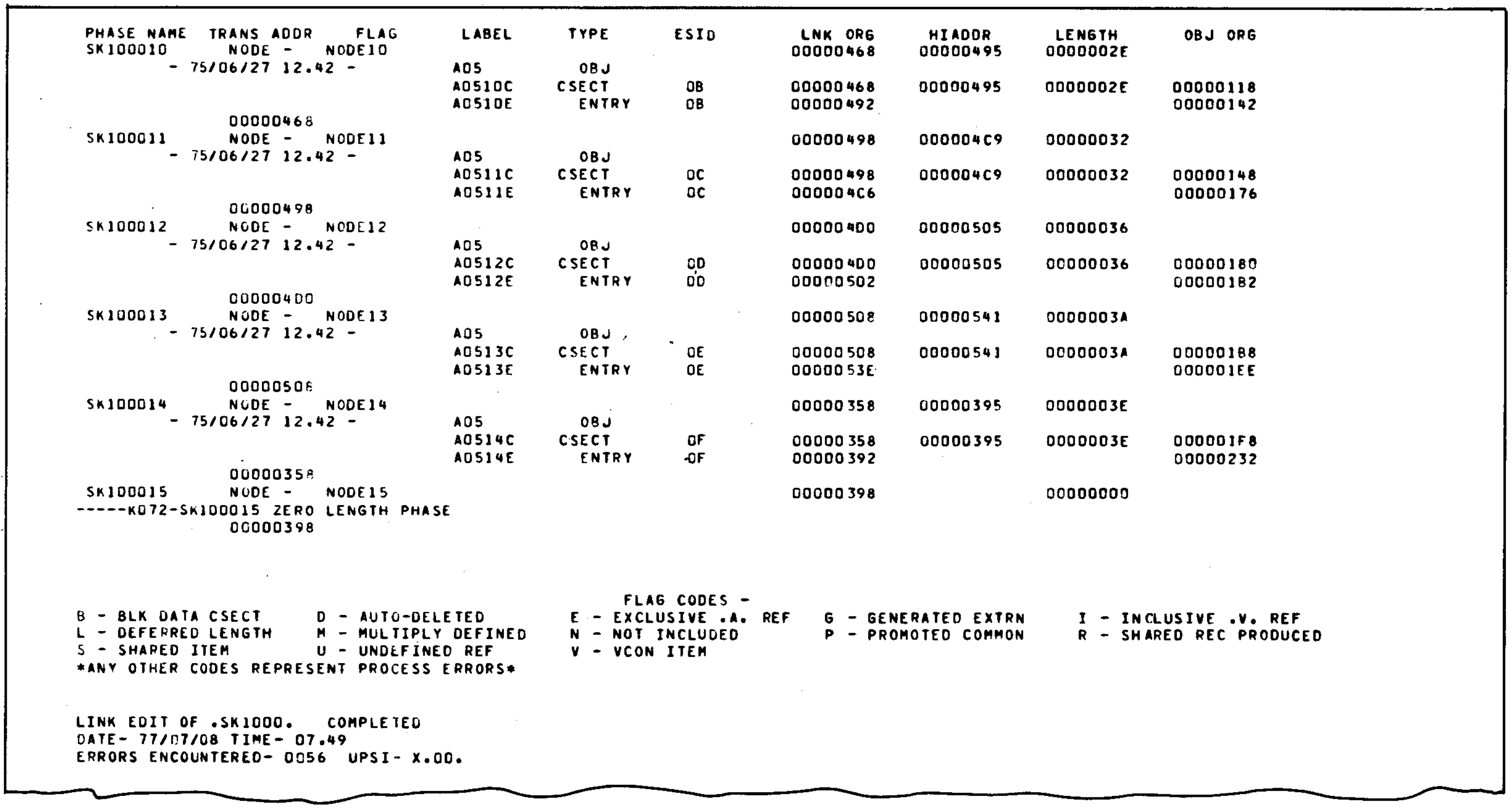
```
**PHASE STRUCTURE** HEX BYTES REPRESENTED BY EACH DASH -   D

                                                                                                  ----
                                                                                                  I 13.3A.
                                                                                              ----I
                                                                                              I 12.36.
                                                                                         ---- I
                                                                                         I 11.32.
                                                                                      ----
                                                                                      I 10.2E.
                                                                                  --- I
                                                                                  I 09.2A.
                                                                               ---I
                                                                               I 08.26.
                                                                            ---I
                                                                            I 07.22.
                                                                         -- I
                                                                         I 06.1E.
                                                                       --I
                                                                       I 05.1A.
                                                                     --I
                                                                     I 04.16.
                                                                   - I
                                                                   I 03.12.
                                                                  -I
                                                                  I 02.E.
                                                                 -I
                                                                 I 01.A.
                                                                 I    I
                                                                 I    I 15.0.
                                                                 -----I
                                                                 I 14.3E.
-----------------------------------------------------------------I
I 00.354.
```

Figure 8-5. Link-Edit Example 4 (Part 3 of 5)



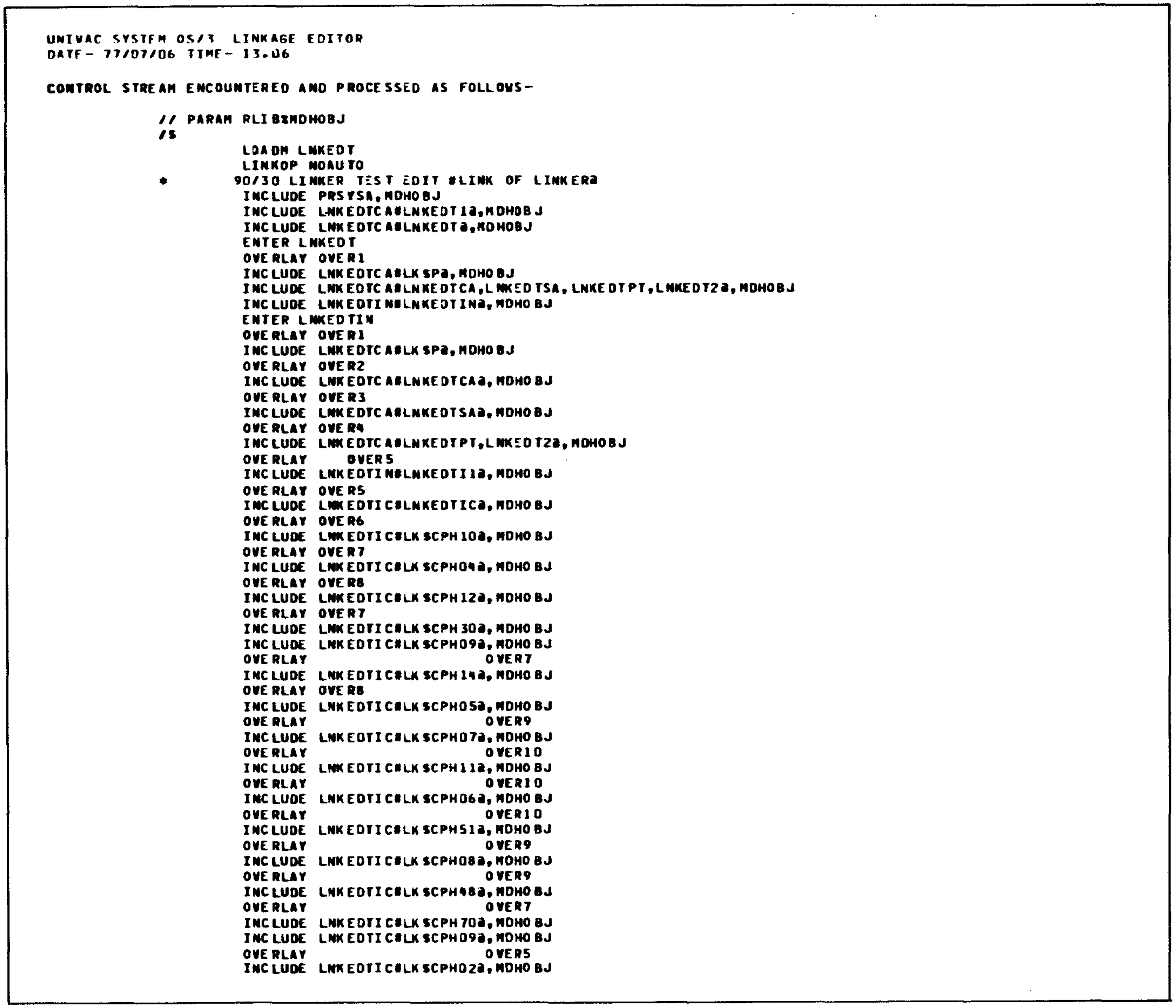
```
                                             ** ALLOCATION MAP **

                    LOAD MODULE -   SK1000         SIZE -   00000542

PHASE NAME  TRANS ADDR    FLAG      LABEL      TYPE       ESID        LNK ORG     HIADDR      LENGTH      OBJ ORG
SK100000      NODE - ROOT                                            00000000    00000353    00000354
*** START OF AUTO-INCLUDED ELEMENTS -
*** END OF AUTO-INCLUDED ELEMENTS -
        - 75/07/02 11.49 -         LK1ROOT     OBJ
                                   LK1ROOT   CSECT       01          00000000    00000353    00000354    00000000
              00000000
SK100001      NODE -   NODE01                                        00000358    00000361    0000000A
        - 75/06/27 12.42 -         A05         OBJ
                                   A0501C    CSECT       02          00000358    00000361    0000000A    00000008
                                   A0501E      ENTRY     02          0000035E                            0000000E
              00000358
SK100002      NODE -   NODE02                                        00000368    00000375    0000000E
        - 75/06/27 12.42 -         A05         OBJ
                                   A0502C    CSECT       03          00000368    00000375    0000000E    00000018
                                   A0502E      ENTRY     03          00000372                            00000022
              00000368
SK100003      NODE -   NODE03                                        00000378    00000389    00000012
        - 75/06/27 12.42 -         A05         OBJ
                                   A0503C    CSECT       04          00000378    00000389    00000012    00000028
                                   A0503E      ENTRY     04          00000386                            00000036
              00000378
SK100004      NODE -   NODE04                                        00000390    000003A5    00000016
        - 75/06/27 12.42 -         A05         OBJ
                                   A0504C    CSECT       05          00000390    000003A5    00000016    00000040
                                   A0504E      ENTRY     05          000003A2                            00000052
              00000390
SK100005      NODE -   NODE05                                        000003A8    000003C1    0000001A
        - 75/06/27 12.42 -         A05         OBJ
                                   A0505C    CSECT       06          000003A8    000003C1    0000001A    00000058
                                   A0505E      ENTRY     06          000003BE                            0000006E
              000003A8
SK100006      NODE -   NODE06                                        000003C8    000003E5    0000001E
        - 75/06/27 12.42 -         A05         OBJ
                                   A0506C    CSECT       07          000003C8    000003E5    0000001E    00000078
                                   A0506E      ENTRY     07          000003E2                            00000092
              000003C8
SK100007      NODE -   NODE07                                        000003E8    00000409    00000022
        - 75/06/27 12.42 -         A05         OBJ
                                   A0507C    CSECT       08          000003E8    00000409    00000022    00000098
                                   A0507E      ENTRY     08          00000406                            000000B6
              000003E8
SK100008      NODE -   NODE08                                        00000410    00000435    00000026
        - 75/06/27 12.42 -         A05         OBJ
                                   A0508C    CSECT       09          00000410    00000435    00000026    000000C0
                                   A0508E      ENTRY     09          00000432                            000000E2
              00000410
SK100009      NODE -   NODE09                                        00000438    00000461    0000002A
        - 75/06/27 12.42 -         A05         OBJ
                                   A0509C    CSECT       0A          00000438    00000461    0000002A    000000E8
                                   A0509E      ENTRY     0A          0000045E                            0000010E
              00000438
```

Figure 8-5. Link-Edit Example 4 (Part 4 of 5)



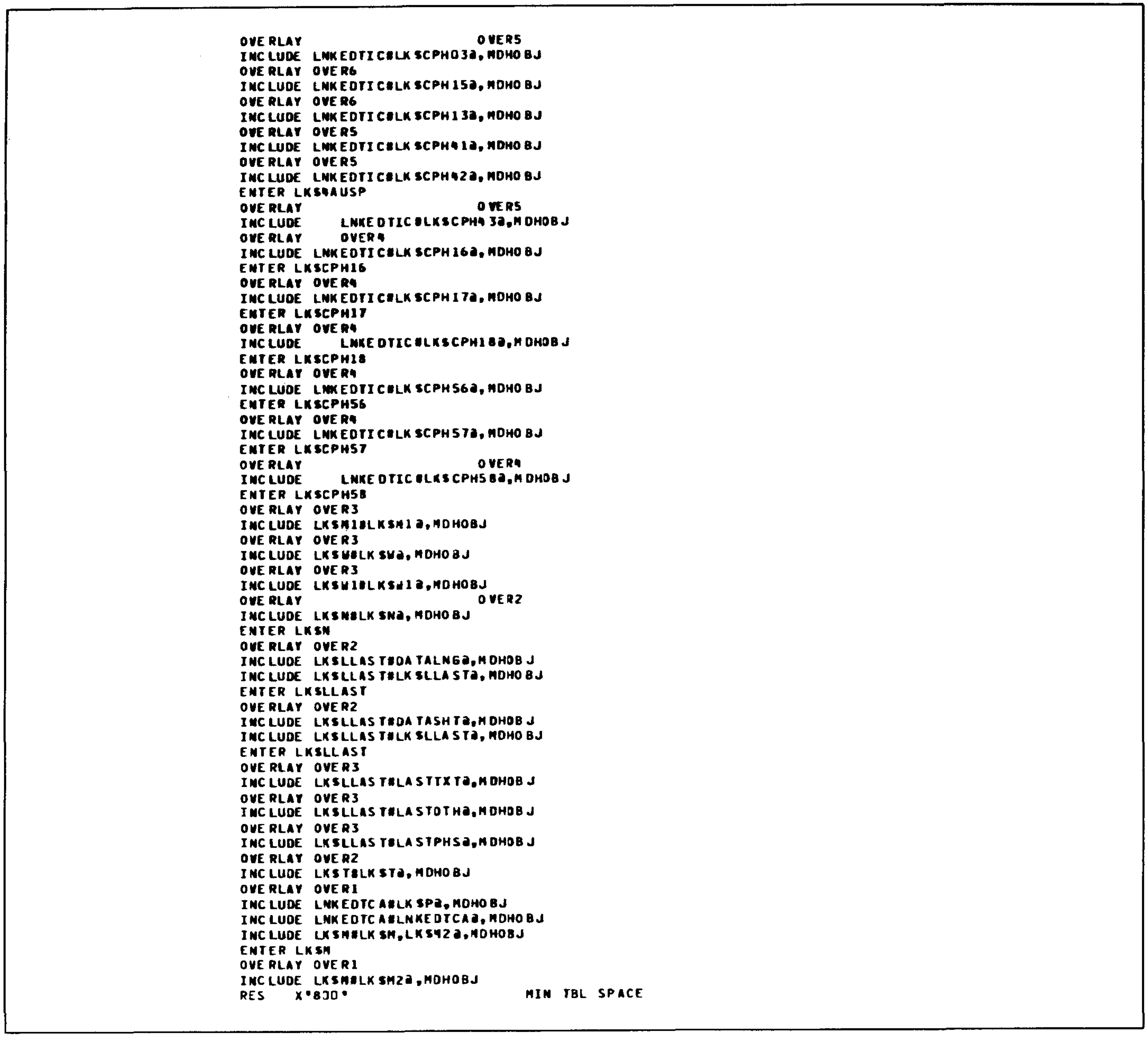
```
 PHASE NAME  TRANS ADDR    FLAG         LABEL      TYPE       ESID        LNK ORG     HIADDR      LENGTH       OBJ ORG
 SK100010      NODE -   NODE10                                             00000468    00000495    0000002E
         - 75/06/27 12.42 -             A05          OBJ
                                        A0510C     CSECT        0B         00000468    00000495    0000002E    00000118
                                        A0510E       ENTRY      0B         00000492                            00000142
               00000468
 SK100011      NODE -   NODE11                                             00000498    000004C9    00000032
         - 75/06/27 12.42 -             A05          OBJ
                                        A0511C     CSECT        0C         00000498    000004C9    00000032    00000148
                                        A0511E       ENTRY      0C         000004C6                            00000176
               00000498
 SK100012      NODE -   NODE12                                             000004D0    00000505    00000036
         - 75/06/27 12.42 -             A05          OBJ
                                        A0512C     CSECT        0D         000004D0    00000505    00000036    00000180
                                        A0512E       ENTRY      0D         00000502                            00000182
               000004D0
 SK100013      NODE -   NODE13                                             00000508    00000541    0000003A
         - 75/06/27 12.42 -             A05          OBJ ,
                                        A0513C     CSECT        0E         00000508    00000541    0000003A    00000188
                                        A0513E       ENTRY      0E         0000053E                            000001EE
               00000508
 SK100014      NODE -   NODE14                                             00000358    00000395    0000003E
         - 75/06/27 12.42 -             A05          OBJ
                                        A0514C     CSECT        0F         00000358    00000395    0000003E    000001F8
                                        A0514E       ENTRY      -0F        00000392                            00000232
               00000358
 SK100015      NODE -   NODE15                                             00000398                0000000
-----K072-SK100015 ZERO LENGTH PHASE
               00000398


                                                         FLAG CODES -
B - BLK DATA CSECT      D - AUTO-DELETED        E - EXCLUSIVE .A. REF   G - GENERATED EXTRN      I - INCLUSIVE .V. REF
L - DEFERRED LENGTH     M - MULTIPLY DEFINED    N - NOT INCLUDED        P - PROMOTED COMMON      R - SHARED REC PRODUCED
S - SHARED ITEM         U - UNDEFINED REF       V - VCON ITEM
*ANY OTHER CODES REPRESENT PROCESS ERRORS*


LINK EDIT OF .SK1000.   COMPLETED
DATE- 77/07/08 TIME- 07.49
ERRORS ENCOUNTERED- 0056  UPSI- X.00.
```

Figure 8-5. Link-Edit Example 4 (Part 5 of 5)



```
UNIVAC SYSTFM OS/3  LINKAGE EDITOR
DATF- 77/07/06 TIME- 13.06

CONTROL STREAM ENCOUNTERED AND PROCESSED AS FOLLOWS-

           // PARAM RLIB%MDHOBJ
           /S
                   LOADM LNKEDT
                   LINKOP NOAUTO
           *      90/30 LINKER TEST EDIT #LINK OF LINKER@
                   INCLUDE PRSYSA,MDHOBJ
                   INCLUDE LNKEDTCA#LNKEDT1@,MDHOBJ
                   INCLUDE LNKEDTCA#LNKEDT@,MDHOBJ
                   ENTER LNKEDT
                   OVERLAY OVER1
                   INCLUDE LNKEDTCA#LKSP@,MDHOBJ
                   INCLUDE LNKEDTCA#LNKEDTCA,LNKEDTSA,LNKEDTPT,LNKEDT2@,MDHOBJ
                   INCLUDE LNKEDTIN#LNKEDTIN@,MDHOBJ
                   ENTER LNKEDTIN
                   OVERLAY OVER1
                   INCLUDE LNKEDTCA#LKSP@,MDHOBJ
                   OVERLAY OVER2
                   INCLUDE LNKEDTCA#LNKEDTCA@,MDHOBJ
                   OVERLAY OVER3
                   INCLUDE LNKEDTCA#LNKEDTSA@,MDHOBJ
                   OVERLAY OVER4
                   INCLUDE LNKEDTCA#LNKEDTPT,LNKEDT2@,MDHOBJ
                   OVERLAY    OVER5
                   INCLUDE LNKEDTIN#LNKEDTI1@,MDHOBJ
                   OVERLAY OVER5
                   INCLUDE LNKEDTIC#LNKEDTIC@,MDHOBJ
                   OVERLAY OVER6
                   INCLUDE LNKEDTIC#LKSCPH10@,MDHOBJ
                   OVERLAY OVER7
                   INCLUDE LNKEDTIC#LKSCPH04@,MDHOBJ
                   OVERLAY OVER8
                   INCLUDE LNKEDTIC#LKSCPH12@,MDHOBJ
                   OVERLAY OVER7
                   INCLUDE LNKEDTIC#LKSCPH30@,MDHOBJ
                   INCLUDE LNKEDTIC#LKSCPH09@,MDHOBJ
                   OVERLAY                 OVER7
                   INCLUDE LNKEDTIC#LKSCPH14@,MDHOBJ
                   OVERLAY OVER8
                   INCLUDE LNKEDTIC#LKSCPH05@,MDHOBJ
                   OVERLAY                 OVER9
                   INCLUDE LNKEDTIC#LKSCPH07@,MDHOBJ
                   OVERLAY                 OVER10
                   INCLUDE LNKEDTIC#LKSCPH11@,MDHOBJ
                   OVERLAY                 OVER10
                   INCLUDE LNKEDTIC#LKSCPH06@,MDHOBJ
                   OVERLAY                 OVER10
                   INCLUDE LNKEDTIC#LKSCPH51@,MDHOBJ
                   OVERLAY                 OVER9
                   INCLUDE LNKEDTIC#LKSCPH08@,MDHOBJ
                   OVERLAY                 OVER9
                   INCLUDE LNKEDTIC#LKSCPH48@,MDHOBJ
                   OVERLAY                 OVER7
                   INCLUDE LNKEDTIC#LKSCPH70@,MDHOBJ
                   INCLUDE LNKEDTIC#LKSCPH09@,MDHOBJ
                   OVERLAY                 OVER5
                   INCLUDE LNKEDTIC#LKSCPH02@,MDHOBJ
```

Figure 8-6. Link-Edit Example 5 (Part 1 of 14)

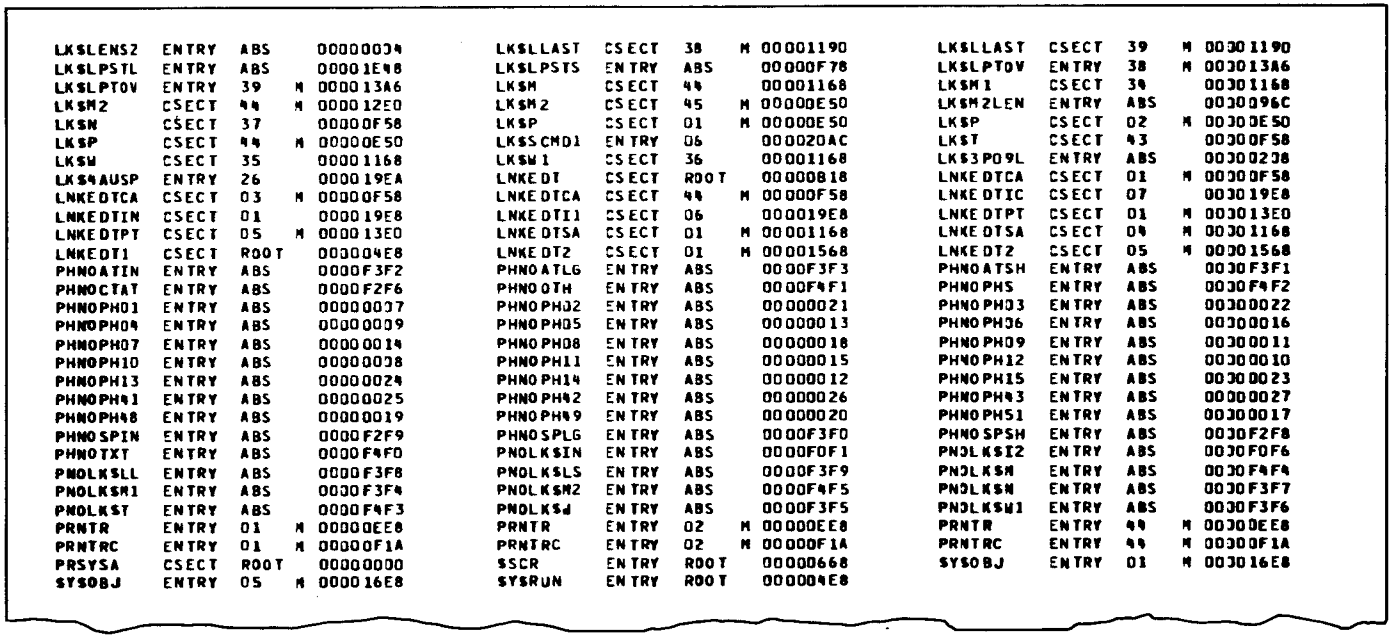


```
OVERLAY                        OVER5
INCLUDE  LNKEDTIC#LK$CPH03@,MDHOBJ
OVERLAY  OVER6
INCLUDE  LNKEDTIC#LK$CPH15@,MDHOBJ
OVERLAY  OVER6
INCLUDE  LNKEDTIC#LK$CPH13@,MDHOBJ
OVERLAY  OVER5
INCLUDE  LNKEDTIC#LK$CPH41@,MDHOBJ
OVERLAY  OVER5
INCLUDE  LNKEDTIC#LK$CPH42@,MDHOBJ
ENTER LK$4AUSP
OVERLAY                        OVER5
INCLUDE     LNKEDTIC#LK$CPH43@,MDHOBJ
OVERLAY     OVER4
INCLUDE  LNKEDTIC#LK$CPH16@,MDHOBJ
ENTER LK$CPH16
OVERLAY  OVER4
INCLUDE  LNKEDTIC#LK$CPH17@,MDHOBJ
ENTER LK$CPH17
OVERLAY  OVER4
INCLUDE     LNKEDTIC#LK$CPH18@,MDHOBJ
ENTER LK$CPH18
OVERLAY  OVER4
INCLUDE  LNKEDTIC#LK$CPH56@,MDHOBJ
ENTER LK$CPH56
OVERLAY  OVER4
INCLUDE  LNKEDTIC#LK$CPH57@,MDHOBJ
ENTER LK$CPH57
OVERLAY                        OVER4
INCLUDE     LNKEDTIC#LK$CPH58@,MDHOBJ
ENTER LK$CPH58
OVERLAY  OVER3
INCLUDE  LK$M1#LK$M1@,MDHOBJ
OVERLAY  OVER3
INCLUDE  LK$W#LK$W@,MDHOBJ
OVERLAY  OVER3
INCLUDE  LK$W1#LK$W1@,MDHOBJ
OVERLAY                        OVER2
INCLUDE  LK$N#LK$N@,MDHOBJ
ENTER LK$N
OVERLAY  OVER2
INCLUDE  LK$LLAST#DATALNG@,MDHOBJ
INCLUDE  LK$LLAST#LK$LLAST@,MDHOBJ
ENTER LK$LLAST
OVERLAY  OVER2
INCLUDE  LK$LLAST#DATASHT@,MDHOBJ
INCLUDE  LK$LLAST#LK$LLAST@,MDHOBJ
ENTER LK$LLAST
OVERLAY  OVER3
INCLUDE  LK$LLAST#LASTTXT@,MDHOBJ
OVERLAY  OVER3
INCLUDE  LK$LLAST#LASTOTH@,MDHOBJ
OVERLAY  OVER3
INCLUDE  LK$LLAST#LASTPHS@,MDHOBJ
OVERLAY  OVER2
INCLUDE  LK$T#LK$T@,MDHOBJ
OVERLAY  OVER1
INCLUDE  LNKEDTCA#LK$P@,MDHOBJ
INCLUDE  LNKEDTCA#LNKEDTCA@,MDHOBJ
INCLUDE  LK$M#LK$M,LK$42@,MDHOBJ
ENTER LK$M
OVERLAY  OVER1
INCLUDE  LK$M#LK$M2@,MDHOBJ
RES    X'830'                       MIN TBL SPACE
```

Figure 8-6. Link-Edit Example 5 (Part 2 of 14)

```
                                         *DEFINITIONS DICTIONARY*

SYMBOL.   TYPE.  PHASE. ADDRESS.        SYMBOL.   TYPE.  PHASE. ADDRESS.        SYMBOL.   TYPE.  PHASE. ADDRESS.

DATALNG   CSECT  38       00000F58      DATASHT   CSECT  39       00000F58      DPSCOM0   ENTRY  ROOT     00000000
DPSCOM1   ENTRY  ROOT     00000000      DPSCOM2   ENTRY  ROOT     00000000      DPSCOM3   ENTRY  ROOT     00000000
DPSCOM4   ENTRY  ROOT     00000000      DPSCOM5   ENTRY  ROOT     00000000      DPSCOM6   ENTRY  ROOT     00000000
DPSCOM7   ENTRY  ROOT     00000000      KESALP    ENTRY  ABS      00002590      KESRES    ENTRY  ABS      00002D90
LASTOTH   CSECT  41       00001620      LASTPHS   CSECT  42       00001620      LASTTXT   CSECT  40       00001620
LB93USR   ENTRY  01     M 00001568      LB93USR   ENTRY  05     M 00001568      LB93USS   ENTRY  01     M 00001868
LB93USS   ENTRY  05     M 00001868      LKSCDIF1  ENTRY  ABS      00000998      LKSCDIF2  ENTRY  ABS      000007C8
LKSCOBPC  ENTRY  01     M 00001818      LKSCOBPC  ENTRY  05     M 00001818      LKSCOFPC  ENTRY  01     M 00001840
LKSCOFPC  ENTRY  05     M 00001840      LKSCPH01  ENTRY  07       000019E8      LKSCPH02  CSECT  21       000019E8
LKSCPH03  CSECT  22       000019E8      LKSCPH04  CSECT  09       00001010      LKSCPH05  CSECT  13       00001F28
LKSCPH06  CSECT  16       00002250      LKSCPH07  CSECT  14       00002168      LKSCPH08  CSECT  18       00002168
LKSCPH09  CSECT  11     M 00001D70      LKSCPH09  CSECT  20     M 00001D40      LKSCPH10  CSECT  08       00001C50
LKSCPH11  CSECT  15       00002250      LKSCPH12  CSECT  10       00002008      LKSCPH13  CSECT  24       00001B40
LKSCPH14  CSECT  12       00001D10      LKSCPH15  CSECT  23       00001B40      LKSCPH16  CSECT  28       000013E0
LKSCPH17  CSECT  29       000013E0      LKSCPH18  CSECT  30       000013E0      LKSCPH30  CSECT  11       00001D10
LKSCPH41  CSECT  25       000019E8      LKSCPH42  CSECT  26       000019E8      LKSCPH43  CSECT  27       000019E8
LKSCPH48  CSECT  19       00002168      LKSCPH51  CSECT  17       00002250      LKSCPH56  CSECT  31       000013E0
LKSCPH57  CSECT  32       000013E0      LKSCPH58  CSECT  33       000013E0      LKSCPH70  CSECT  20       00001D10
LKSCPOR6  ENTRY  ABS      00000C50      LKSCPPCA  ENTRY  ROOT     000007E8      LKSCREPC  ENTRY  ROOT     000007C0
LKSCRFPC  ENTRY  ROOT     00000640      LKSCRPCA  ENTRY  ROOT     00000618      LKSCRTPC  ENTRY  ROOT     00000798
LKSCUMFP  ENTRY  01     M 000019C0      LKSCUMFP  ENTRY  05     M 000019C0      LKSCUMPC  ENTRY  01     M 00001998
LKSCUMPC  ENTRY  05     M 00001998      LKSCUPCA  ENTRY  01     M 00001698      LKSCUPCA  ENTRY  05     M 00001698
LKSCUPCF  ENTRY  01     M 000016C0      LKSCUPCF  ENTRY  05     M 000016C0      LKSDBEXT  ENTRY  ABS      00000006
LKSDBINC  ENTRY  ABS      00000000      LKSDBTRN  ENTRY  ABS      00000000      LKSDCALP  ENTRY  ABS      00000000
LKSDCHAL  ENTRY  ABS      00000002      LKSDCMDF  ENTRY  ABS      00000000      LKSDCOMP  ENTRY  ABS      00000000
LKSDDLN   ENTRY  ABS      00000002      LKSDESDA  ENTRY  ABS      00000006      LKSDGTCT  ENTRY  ABS      00000000
LKSDIBCD  ENTRY  ABS      00000000      LKSDLIBS  ENTRY  ABS      00000000      LKSDMHDR  ENTRY  ABS      00000004
LKSDMODP  ENTRY  ABS      00000000      LKSDNESP  ENTRY  ABS      00000002      LKSDNSTK  ENTRY  ABS      00000000
LKSDOPSC  ENTRY  ABS      00000006      LKSDRDB1  ENTRY  ABS      00000008      LKSDSARC  ENTRY  ABS      00000000
LKSDSCNA  ENTRY  ABS      00000004      LKSDSCNC  ENTRY  ABS      00000002      LKSDSCNM  ENTRY  ABS      00000002
LKSDSCNI  ENTRY  ABS      00000002      LKSDSTSC  ENTRY  ABS      00000004      LKSDWRRC  ENTRY  ABS      0000000A
LKSEALDF  ENTRY  ABS      00000686      LKSEBCMT  ENTRY  ABS      0000065C      LKSEBDCI  ENTRY  ABS      00000570
LKSEBDLB  ENTRY  ABS      000003F7      LKSEBDLD  ENTRY  ABS      00000631      LKSEBDLI  ENTRY  ABS      00000420
LKSEBDRF  ENTRY  ABS      00000008      LKSEBDSN  ENTRY  ABS      000003CA      LKSEBPAR  ENTRY  ABS      000004A7
LKSECSNM  ENTRY  ABS      000000F7      LKSECSNO  ENTRY  ABS      00000130      LKSECSNP  ENTRY  ABS      000005B4
LKSEDNOP  ENTRY  ABS      0000003C      LKSEENCM  ENTRY  ABS      000002E8      LKSEENEX  ENTRY  ABS      00000724
LKSEENLA  ENTRY  ABS      000001CE      LKSEENLM  ENTRY  ABS      00000083      LKSEENPC  ENTRY  ABS      000003A8
LKSEEOPE  ENTRY  ABS      000001DE      LKSEEQPH  ENTRY  ABS      000001F0      LKSEILNF  ENTRY  ABS      0000050F
LKSEINVL  ENTRY  ABS      000005F4      LKSEINVO  ENTRY  ABS      0000027C      LKSEIVDE  ENTRY  ABS      00000112
LKSEIVJC  ENTRY  ABS      000006E1      LKSEIVPM  ENTRY  ABS      000004C4      LKSELABN  ENTRY  ABS      0000009C
LKSELBDE  ENTRY  ABS      00000158      LKSELMLB  ENTRY  ABS      000001B8      LKSELMNS  ENTRY  ABS      00000468
LKSELNNF  ENTRY  ABS      00000443      LKSEMIVD  ENTRY  ABS      0000023E      LKSEMNDE  ENTRY  ABS      00000255
LKSEMNDF  ENTRY  ABS      0000021E      LKSEMOPN  ENTRY  ABS      000002C7      LKSEMPDL  ENTRY  ABS      0000048C
LKSEMPSL  ENTRY  ABS      000006FF      LKSEMTDL  ENTRY  ABS      0000030F      LKSEMTNA  ENTRY  ABS      00000291
LKSENCUP  ENTRY  ABS      0000034C      LKSENON   ENTRY  ABS      0000002F      LKSENDSZ  ENTRY  ABS      00000391
LKSENODL  ENTRY  ABS      0000019D      LKSENOOP  ENTRY  ABS      00000051      LKSENOPH  ENTRY  ABS      00000014
LKSENORN  ENTRY  ABS      00000552      LKSENORS  ENTRY  ABS      0000052F      LKSENOSC  ENTRY  ABS      00000018
LKSENOSR  ENTRY  ABS      000004E3      LKSENOTE  ENTRY  ABS      0000032C      LKSENOTF  ENTRY  ABS      0000017F
LKSENOUS  ENTRY  ABS      00000512      LKSEOPAB  ENTRY  ABS      000000AE      LKSEOPRN  ENTRY  ABS      000000C4
LKSEORNF  ENTRY  ABS      00000207      LKSEOVAB  ENTRY  ABS      00000748      LKSEOVEX  ENTRY  ABS      0000076D
LKSEOVOP  ENTRY  ABS      000007B3      LKSEOVRG  ENTRY  ABS      0000078C      LKSEOVRT  ENTRY  ABS      00000003
LKSEPNO   ENTRY  ABS      0000001A      LKSEREGN  ENTRY  ABS      0000006E      LKSERNAL  ENTRY  ABS      0000059B
LKSERTIO  ENTRY  ABS      00000010      LKSESLIB  ENTRY  ABS      000007D4      LKSESOBJ  ENTRY  ABS      00000702
LKSESTBD  ENTRY  ABS      00000004      LKSETBOF  ENTRY  ABS      0000068C      LKSEUOBJ  ENTRY  ABS      000007D3
LKSEWTYP  ENTRY  ABS      00000371      LKSEXNO   ENTRY  ABS      000000E1      LKSFAIM   ENTRY  ABS      0000063C
LKSFRLM   ENTRY  ABS      0000061C      LKSILDND  ENTRY  06       00002240      LKSI0300  ENTRY  06       00001F7A
LKSI0320  ENTRY  06       00001F82      LKSI0340  ENTRY  01       00001F66      LKSI0350  ENTRY  01       00001F72
```

Figure 8-6. Link-Edit Example 5 (Part 3 of 14)

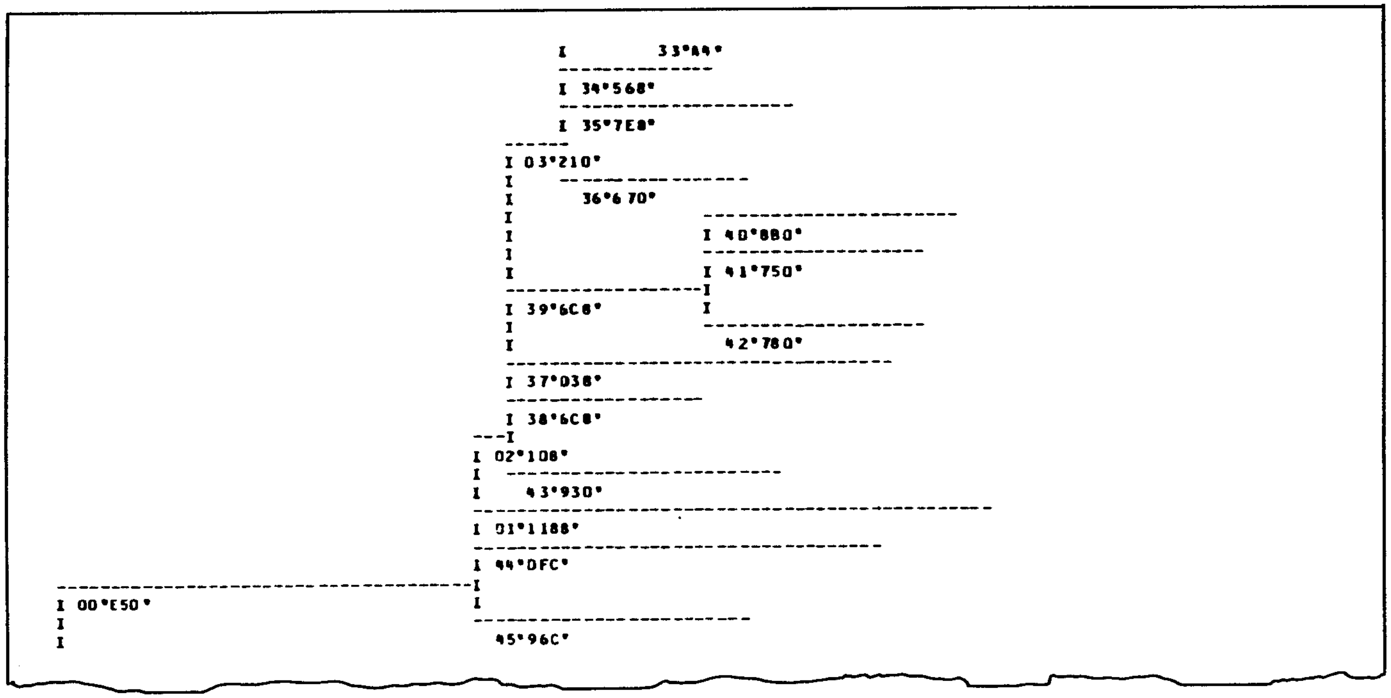


```
LK$LENS2   ENTRY  ABS      00000034     LK$LLAST   CSECT  38    M 00001190     LK$LLAST   CSECT  39    M 00001190
LK$LPSTL   ENTRY  ABS      00001E48     LK$LPSTS   ENTRY  ABS     00000F78     LK$LPTOV   ENTRY  38    M 000013A6
LK$LPTOV   ENTRY  39     M 000013A6     LK$M       CSECT  44      00001168     LK$M1      CSECT  34      00001168
LK$M2      CSECT  44     M 000012E0     LK$M2      CSECT  45    M 00000E50     LK$M2LEN   ENTRY  ABS     0000096C
LK$N       CSECT  37       00000F58     LK$P       CSECT  01    M 00000E50     LK$P       CSECT  02    M 00000E50
LK$P       CSECT  44     M 00000E50     LK$SCMD1   ENTRY  06      000020AC     LK$T       CSECT  43      00000F58
LK$W       CSECT  35       00001168     LK$W1      CSECT  36      00001168     LK$3PO9L   ENTRY  ABS     00000208
LK$4AUSP   ENTRY  26       000019EA     LNKEDT     CSECT  ROOT    00000818     LNKEDTCA   CSECT  01    M 00000F58
LNKEDTCA   CSECT  03     M 00000F58     LNKEDTCA   CSECT  44    M 00000F58     LNKEDTIC   CSECT  07      000019E8
LNKEDTIN   CSECT  01       000019E8     LNKEDTI1   CSECT  06      000019E8     LNKEDTPT   CSECT  01    M 000013E0
LNKEDTPT   CSECT  05     M 000013E0     LNKEDTSA   CSECT  01    M 00001168     LNKEDTSA   CSECT  04    M 00001168
LNKEDT1    CSECT  ROOT     000004E8     LNKEDT2    CSECT  01    M 00001568     LNKEDT2    CSECT  05    M 00001568
PHNOATIN   ENTRY  ABS      0000F3F2     PHNOATLG   ENTRY  ABS     0000F3F3     PHNOATSH   ENTRY  ABS     0000F3F1
PHNOCTAT   ENTRY  ABS      0000F2F6     PHNOOTH    ENTRY  ABS     0000F4F1     PHNOPHS    ENTRY  ABS     0000F4F2
PHNOPH01   ENTRY  ABS      00000007     PHNOPH02   ENTRY  ABS     00000021     PHNOPH03   ENTRY  ABS     00000022
PHNOPH04   ENTRY  ABS      00000009     PHNOPH05   ENTRY  ABS     00000013     PHNOPH06   ENTRY  ABS     00000016
PHNOPH07   ENTRY  ABS      00000014     PHNOPH08   ENTRY  ABS     00000018     PHNOPH09   ENTRY  ABS     00000011
PHNOPH10   ENTRY  ABS      00000008     PHNOPH11   ENTRY  ABS     00000015     PHNOPH12   ENTRY  ABS     00000010
PHNOPH13   ENTRY  ABS      00000024     PHNOPH14   ENTRY  ABS     00000012     PHNOPH15   ENTRY  ABS     00000023
PHNOPH41   ENTRY  ABS      00000025     PHNOPH42   ENTRY  ABS     00000026     PHNOPH43   ENTRY  ABS     00000027
PHNOPH48   ENTRY  ABS      00000019     PHNOPH49   ENTRY  ABS     00000020     PHNOPH51   ENTRY  ABS     00000017
PHNOSPIN   ENTRY  ABS      0000F2F9     PHNOSPLG   ENTRY  ABS     0000F3F0     PHNOSPSH   ENTRY  ABS     0000F2F8
PHNOTXT    ENTRY  ABS      0000F4F0     PNOLK$IN   ENTRY  ABS     0000F0F1     PNOLK$I2   ENTRY  ABS     0000F0F6
PNOLK$LL   ENTRY  ABS      0000F3F8     PNOLK$LS   ENTRY  ABS     0000F3F9     PNOLK$M    ENTRY  ABS     0000F4F4
PNOLK$M1   ENTRY  ABS      0000F3F4     PNOLK$M2   ENTRY  ABS     0000F4F5     PNOLK$N    ENTRY  ABS     0000F3F7
PNOLK$T    ENTRY  ABS      0000F4F3     PNOLK$J    ENTRY  ABS     0000F3F5     PNOLK$W1   ENTRY  ABS     0000F3F6
PRNTR      ENTRY  01     M 00000EE8     PRNTR      ENTRY  02    M 00000EE8     PRNTR      ENTRY  44    M 00000EE8
PRNTRC     ENTRY  01     M 00000F1A     PRNTRC     ENTRY  02    M 00000F1A     PRNTRC     ENTRY  44    M 00000F1A
PRSYSA     CSECT  ROOT     00000000     $SCR       ENTRY  ROOT    00000668     $Y$OBJ     ENTRY  01    M 000016E8
$Y$OBJ     ENTRY  05     M 000016E8     $Y$RUN     ENTRY  ROOT    000004E8
```

Figure 8-6. Link-Edit Example 5 (Part 4 of 14)

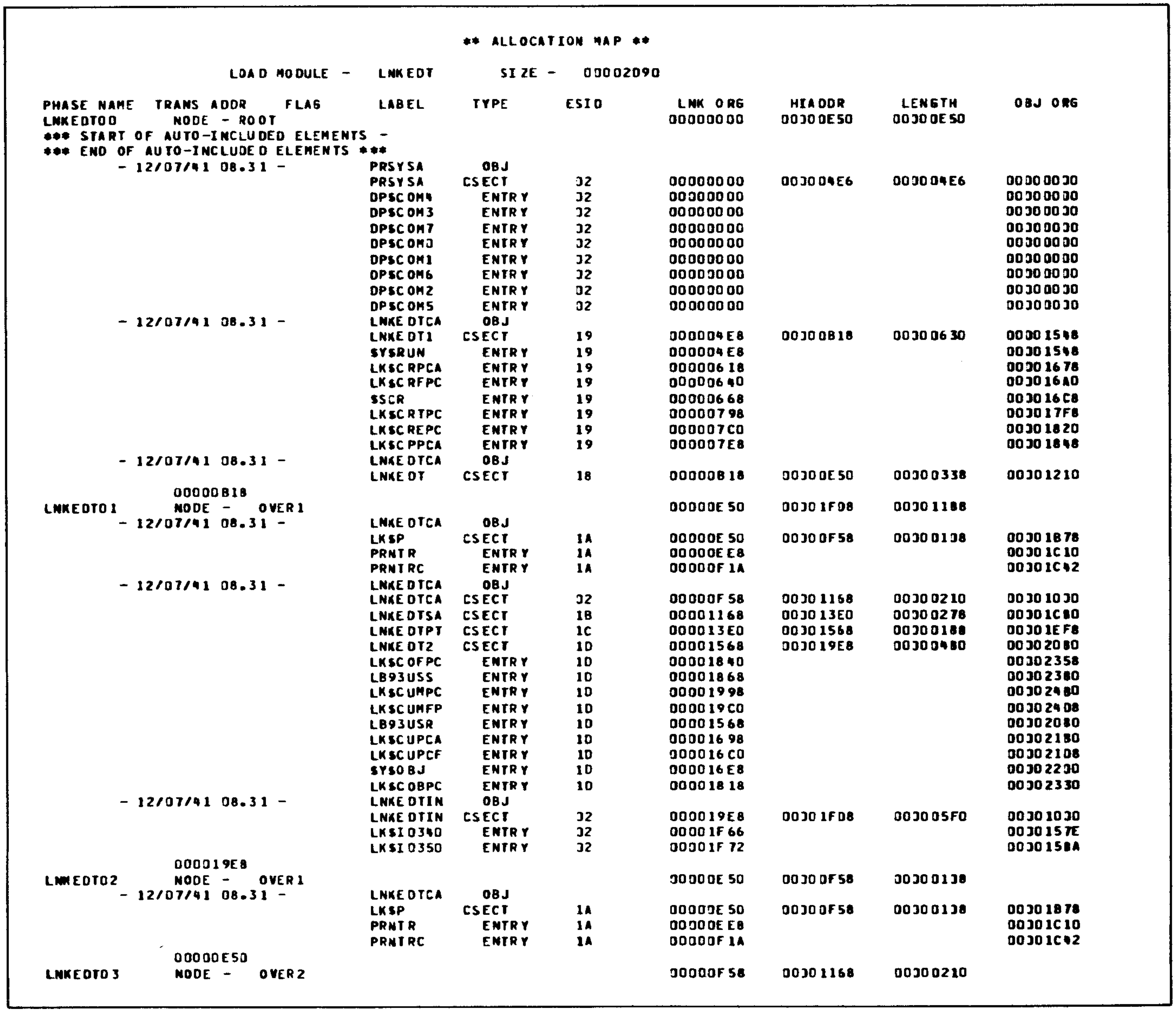
```
**PHASE STRUCTURE** EACH DASH REPRESENTS 60 HEX BYTES
                                             ------
                                             I 15'268'
                                             ---------
                                             I 16'340'
                                           --I
                                           I 14'E8'
                                           I ---
                                           I   17'F8'
                                           --
                                           I 18'E8'
                                     ------I
                                     I 13'243'
                                     I     ---
                                     I       19'138'
                               ------I
                               I 12'218'
                               I       ----
                               I       I 10'160'
                               --------I
                               I 09'2F8'
                               -----
                               I 11'208'
                             --I
                             I 08'C0'
                             I -----
                             I   20'108'
                       ------I
                       I 07'268'
                       I  ----
                       I  I 23'190'
                       ----
                       I 22'158'
                       I  -----
                       I     24'108'
                       -----------------------
                       I 06'858'
                       ----
                       I 21'160'
                       -----
                       I 25'1C8'
       ----------------I
       I 05'608'       I
       I               -------
       I               I 26'2C0'
       I               -----
       I                 27'1C0'
       --
       I 28'A8'
       --
       I 29'A8'
       --
       I 30'A8'
-------I
I 04'278'
I      --
I      I 31'A8'
I      --
I      I 32'A8'
I      --
```

Figure 8-6. Link-Edit Example 5 (Part 5 of 14)



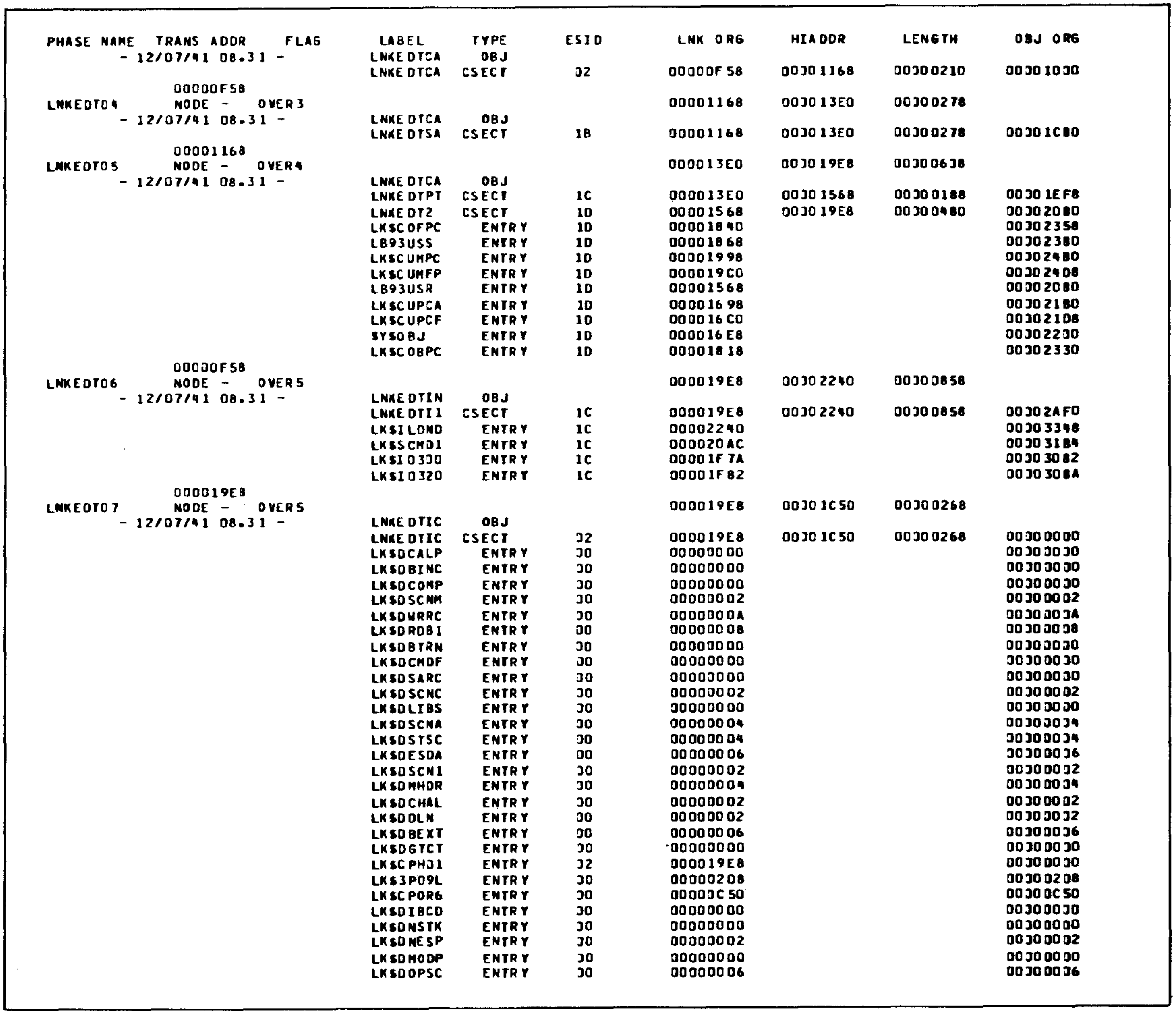
```
                                                 I         33'A4'
                                                 ---------------
                                                 I 34'568'
                                                 -----------------------
                                                 I 35'7E8'
                                            ------
                                            I 03'210'
                                            I    ------------------
                                            I       36'6 70'
                                            I                   -------------------------
                                            I                   I 40'880'
                                            I                   ---------------------
                                            I                   I 41'750'
                                            --------------------I
                                            I 39'6C8'           I
                                            I                   ---------------------
                                            I                     42'780'
                                            --------------------------------------
                                            I 37'038'
                                            -------------------
                                            I 38'6C8'
                                         ---I
                                         I 02'108'
                                         I  ---------------------------
                                         I    43'930'
                                         ---------------------------------------------------
                                         I 01'1188'
                                         ----------------------------------------
                                         I 44'DFC'
-----------------------------------------I
I 00'E50'                                I
I                                        ---------------------------
I                                          45'96C'
```

Figure 8-6. Link-Edit Example 5 (Part 6 of 14)



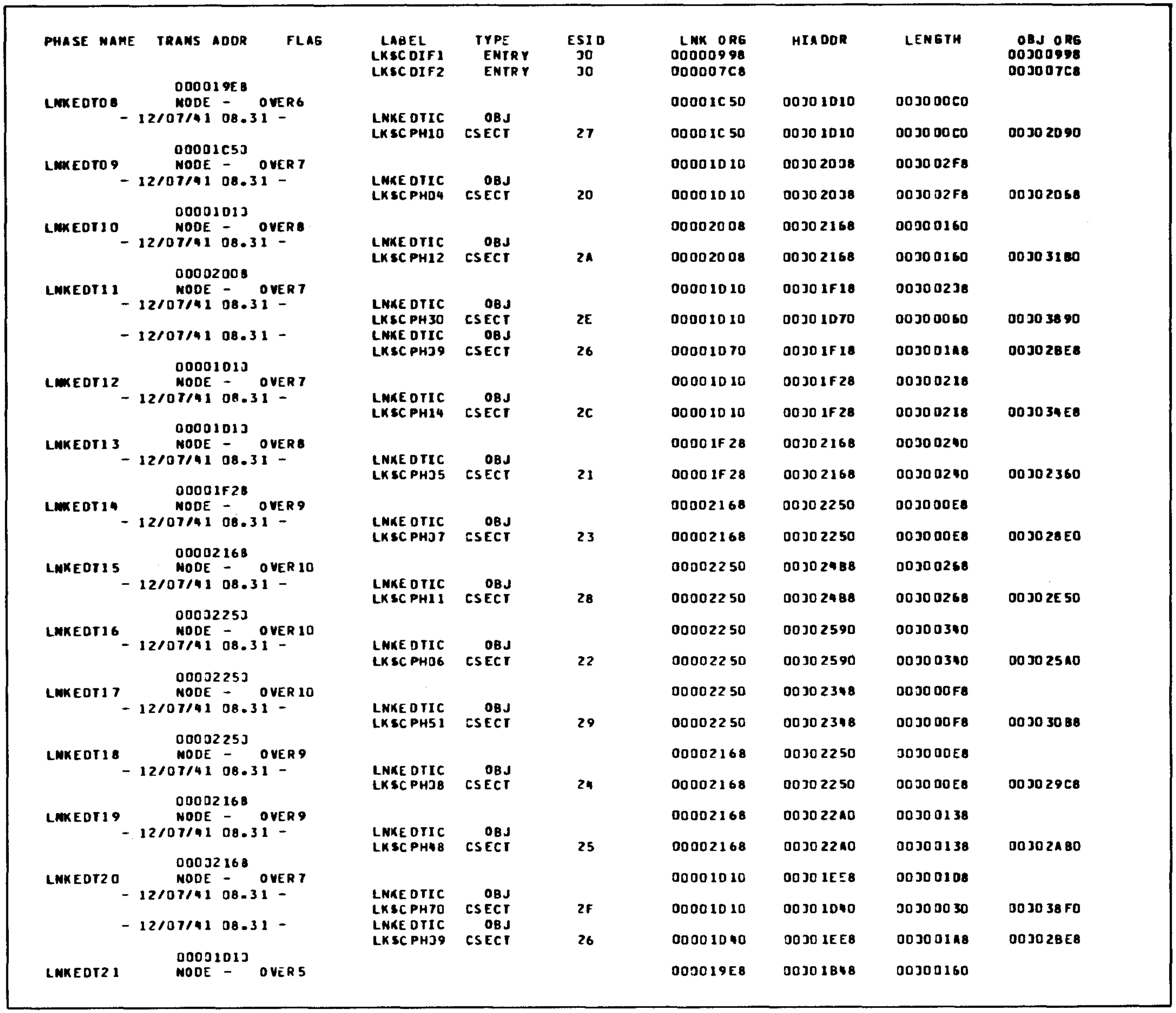
```
                                                ** ALLOCATION MAP **

                         LOAD MODULE -   LNKEDT        SIZE -   00002090

PHASE NAME  TRANS ADDR    FLAG        LABEL       TYPE        ESID         LNK ORG       HIADDR        LENGTH        OBJ ORG
LNKEDT00      NODE - ROOT                                                  00000000      00000E50      00000E50
*** START OF AUTO-INCLUDED ELEMENTS -
*** END OF AUTO-INCLUDED ELEMENTS ***
        - 12/07/41 08.31 -           PRSYSA        OBJ
                                     PRSYSA      CSECT         02          00000000      000004E6      000004E6      00000000
                                     DPSCOM4       ENTRY       02          00000000                                  00000000
                                     DPSCOM3       ENTRY       02          00000000                                  00000000
                                     DPSCOM7       ENTRY       02          00000000                                  00000000
                                     DPSCOM0       ENTRY       02          00000000                                  00000000
                                     DPSCOM1       ENTRY       02          00000000                                  00000000
                                     DPSCOM6       ENTRY       02          00000000                                  00000000
                                     DPSCOM2       ENTRY       02          00000000                                  00000000
                                     DPSCOM5       ENTRY       02          00000000                                  00000000
        - 12/07/41 08.31 -           LNKEDTCA      OBJ
                                     LNKEDT1     CSECT         19          000004E8      00000B18      00000630      00001548
                                     $YSRUN        ENTRY       19          000004E8                                  00001548
                                     LK$CRPCA      ENTRY       19          00000618                                  00001678
                                     LK$CRFPC      ENTRY       19          00000640                                  000016A0
                                     $SCR          ENTRY       19          00000668                                  000016C8
                                     LK$CRTPC      ENTRY       19          00000798                                  000017F8
                                     LK$CREPC      ENTRY       19          000007C0                                  00001820
                                     LK$CPPCA      ENTRY       19          000007E8                                  00001848
        - 12/07/41 08.31 -           LNKEDTCA      OBJ
                                     LNKEDT      CSECT         18          00000818      00000E50      00000338      00001210
              00000818
LNKEDT01      NODE -   OVER1                                               00000E50      00001F08      00001188
        - 12/07/41 08.31 -           LNKEDTCA      OBJ
                                     LK$P        CSECT         1A          00000E50      00000F58      00000108      00001878
                                     PRNTR         ENTRY       1A          00000EE8                                  00001C10
                                     PRNTRC        ENTRY       1A          00000F1A                                  00001C42
        - 12/07/41 08.31 -           LNKEDTCA      OBJ
                                     LNKEDTCA    CSECT         02          00000F58      00001168      00000210      00001030
                                     LNKEDTSA    CSECT         1B          00001168      000013E0      00000278      00001C80
                                     LNKEDTPT    CSECT         1C          000013E0      00001568      00000188      00001EF8
                                     LNKEDT2     CSECT         1D          00001568      000019E8      00000480      00002080
                                     LK$COFPC      ENTRY       1D          00001840                                  00002358
                                     LB93USS       ENTRY       1D          00001868                                  00002380
                                     LK$CUMPC      ENTRY       1D          00001998                                  00002480
                                     LK$CUMFP      ENTRY       1D          000019C0                                  00002408
                                     LB93USR       ENTRY       1D          00001568                                  00002080
                                     LK$CUPCA      ENTRY       1D          00001698                                  00002180
                                     LK$CUPCF      ENTRY       1D          000016C0                                  00002108
                                     $YSOBJ        ENTRY       1D          000016E8                                  00002200
                                     LK$COBPC      ENTRY       1D          00001818                                  00002330
        - 12/07/41 08.31 -           LNKEDTIN      OBJ
                                     LNKEDTIN    CSECT         02          000019E8      00001FD8      000005F0      00001030
                                     LK$I0340      ENTRY       02          00001F66                                  0000157E
                                     LK$I0350      ENTRY       02          00001F72                                  0000158A
              000019E8
LNKEDT02      NODE -   OVER1                                               00000E50      00000F58      00000108
        - 12/07/41 08.31 -           LNKEDTCA      OBJ
                                     LK$P        CSECT         1A          00000E50      00000F58      00000108      00001878
                                     PRNTR         ENTRY       1A          00000EE8                                  00001C10
                                     PRNTRC        ENTRY       1A          00000F1A                                  00001C42
              00000E50
LNKEDT03      NODE -   OVER2                                               00000F58      00001168      00000210
```

Figure 8-6. Link-Edit Example 5 (Part 7 of 14)



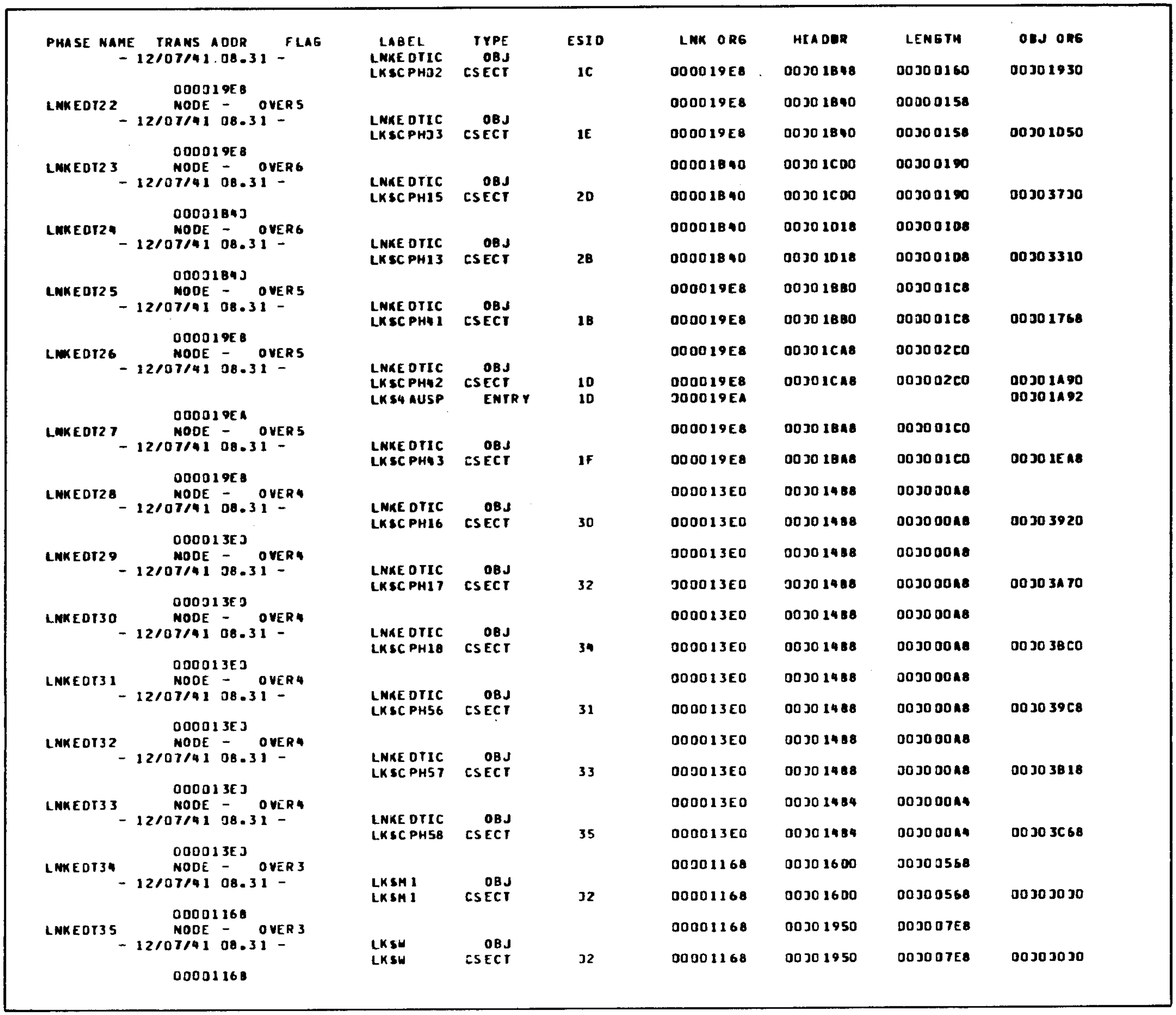
```
PHASE NAME  TRANS ADDR     FLAG          LABEL       TYPE       ESID       LNK ORG     HIADDR      LENGTH      OBJ ORG
          - 12/07/41 08.31 -            LNKEDTCA      OBJ
                                        LNKEDTCA    CSECT      02         00000F58    00001168    00000210    00001000

                    00000F58
LNKEDT04            NODE  -    OVER3                                      00001168    000013E0    00000278
          - 12/07/41 08.31 -            LNKEDTCA      OBJ
                                        LNKEDTSA    CSECT      1B         00001168    000013E0    00000278    00001C80

                    00001168
LNKEDT05            NODE  -    OVER4                                      000013E0    000019E8    00000638
          - 12/07/41 08.31 -            LNKEDTCA      OBJ
                                        LNKEDTPT    CSECT      1C         000013E0    00001568    00000188    00001EF8
                                        LNKEDT2     CSECT      1D         00001568    000019E8    00000480    00002080
                                        LK$COFPC      ENTRY    1D         00001840                            00002358
                                        LB93USS       ENTRY    1D         00001868                            00002380
                                        LK$CUMPC      ENTRY    1D         00001998                            00002480
                                        LK$CUMFP      ENTRY    1D         000019C0                            00002408
                                        LB93USR       ENTRY    1D         00001568                            00002080
                                        LK$CUPCA      ENTRY    1D         00001698                            00002180
                                        LK$CUPCF      ENTRY    1D         000016C0                            00002108
                                        $YSOBJ        ENTRY    1D         000016E8                            00002200
                                        LK$COBPC      ENTRY    1D         00001818                            00002330

                    00000F58
LNKEDT06            NODE  -    OVER5                                      000019E8    00002240    00000858
          - 12/07/41 08.31 -            LNKEDTIN      OBJ
                                        LNKEDTI1    CSECT      1C         000019E8    00002240    00000858    00002AF0
                                        LK$ILDND      ENTRY    1C         00002240                            00003348
                                        LK$SCMDI      ENTRY    1C         000020AC                            000031B4
                                        LK$I0300      ENTRY    1C         00001F7A                            00003082
                                        LK$I0320      ENTRY    1C         00001F82                            0000308A

                    000019E8
LNKEDT07            NODE  -    OVER5                                      000019E8    00001C50    00000268
          - 12/07/41 08.31 -            LNKEDTIC      OBJ
                                        LNKEDTIC    CSECT      02         000019E8    00001C50    00000268    00000000
                                        LK$DCALP      ENTRY    00         00000000                            00000000
                                        LK$DBINC      ENTRY    00         00000000                            00000000
                                        LK$DCOMP      ENTRY    00         00000000                            00000000
                                        LK$DSCNM      ENTRY    00         00000002                            00000002
                                        LK$DWRRC      ENTRY    00         0000000A                            0000000A
                                        LK$DRDBI      ENTRY    00         00000008                            00000008
                                        LK$DBTRN      ENTRY    00         00000000                            00000000
                                        LK$DCMDF      ENTRY    00         00000000                            00000000
                                        LK$DSARC      ENTRY    00         00000000                            00000000
                                        LK$DSCNC      ENTRY    00         00000002                            00000002
                                        LK$DLIBS      ENTRY    00         00000000                            00000000
                                        LK$DSCNA      ENTRY    00         00000004                            00000004
                                        LK$DSTSC      ENTRY    00         00000004                            00000004
                                        LK$DESDA      ENTRY    00         00000006                            00000006
                                        LK$DSCN1      ENTRY    00         00000002                            00000002
                                        LK$DMHDR      ENTRY    00         00000004                            00000004
                                        LK$DCHAL      ENTRY    00         00000002                            00000002
                                        LK$DOLN       ENTRY    00         00000002                            00000002
                                        LK$DBEXT      ENTRY    00         00000006                            00000006
                                        LK$DGTCT      ENTRY    00         00000000                            00000000
                                        LK$CPH01      ENTRY    02         000019E8                            00000000
                                        LK$3P09L      ENTRY    00         00000208                            00000208
                                        LK$CPOR6      ENTRY    00         00000C50                            00000C50
                                        LK$DIBCD      ENTRY    00         00000000                            00000000
                                        LK$DNSTK      ENTRY    00         00000000                            00000000
                                        LK$DNESP      ENTRY    00         00000002                            00000002
                                        LK$DMODP      ENTRY    00         00000000                            00000000
                                        LK$DOPSC      ENTRY    00         00000006                            00000006
```

Figure 8-6. Link-Edit Example 5 (Part 8 of 14)



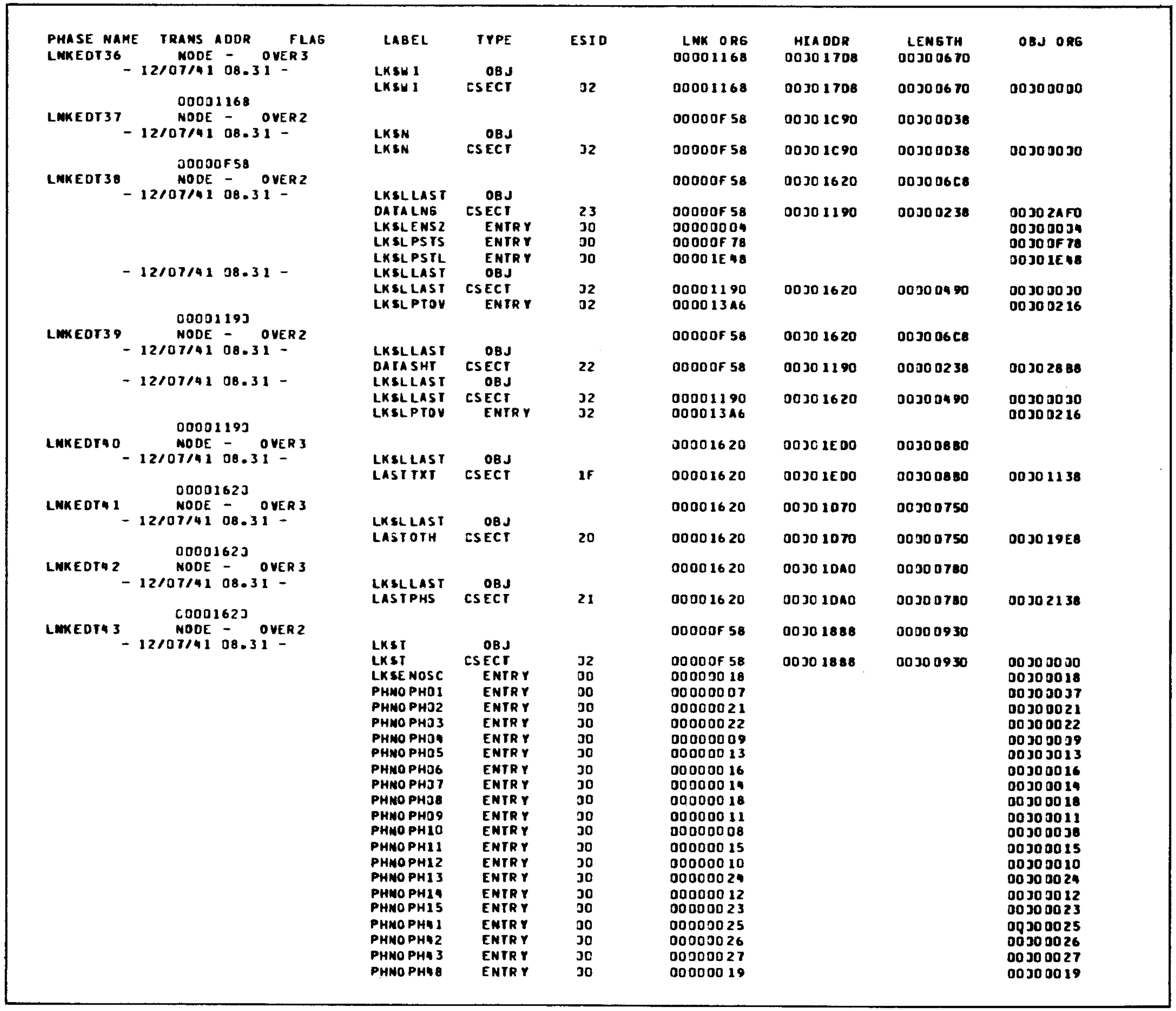
```
PHASE NAME  TRANS ADDR    FLAG     LABEL      TYPE      ESID       LNK ORG      HIADDR      LENGTH      OBJ ORG
                                   LKSCDIF1    ENTRY     00        00000998                             00000998
                                   LKSCDIF2    ENTRY     00        000007C8                             000007C8
              000019E8
LNKEDT08      NODE -   OVER6                                       00001C50     00001010    000000C0
      - 12/07/41 08.31 -           LNKEDTIC    OBJ
                                   LKSCPH10   CSECT      27        00001C50     00001010    000000C0    00002D90
              00001C50
LNKEDT09      NODE -   OVER7                                       00001D10     00002008    000002F8
      - 12/07/41 08.31 -           LNKEDTIC    OBJ
                                   LKSCPH04   CSECT      20        00001D10     00002038    000002F8    00002068
              00001010
LNKEDT10      NODE -   OVER8                                       00002008     00002168    00000160
      - 12/07/41 08.31 -           LNKEDTIC    OBJ
                                   LKSCPH12   CSECT      2A        00002008     00002168    00000160    00003180
              00002008
LNKEDT11      NODE -   OVER7                                       00001D10     00001F18    00000238
      - 12/07/41 08.31 -           LNKEDTIC    OBJ
                                   LKSCPH30   CSECT      2E        00001010     00001D70    00000060    00003890
      - 12/07/41 08.31 -           LNKEDTIC    OBJ
                                   LKSCPH39   CSECT      26        00001D70     00001F18    000001A8    00002BE8
              00001010
LNKEDT12      NODE -   OVER7                                       00001D10     00001F28    00000218
      - 12/07/41 08.31 -           LNKEDTIC    OBJ
                                   LKSCPH14   CSECT      2C        00001010     00001F28    00000218    000034E8
              00001010
LNKEDT13      NODE -   OVER8                                       00001F28     00002168    00000240
      - 12/07/41 08.31 -           LNKEDTIC    OBJ
                                   LKSCPH05   CSECT      21        00001F28     00002168    00000240    00002360
              00001F28
LNKEDT14      NODE -   OVER9                                       00002168     00002250    000000E8
      - 12/07/41 08.31 -           LNKEDTIC    OBJ
                                   LKSCPH07   CSECT      23        00002168     00002250    000000E8    000028E0
              00002168
LNKEDT15      NODE -   OVER10                                      00002250     000024B8    00000268
      - 12/07/41 08.31 -           LNKEDTIC    OBJ
                                   LKSCPH11   CSECT      28        00002250     000024B8    00000268    00002E50
              00002250
LNKEDT16      NODE -   OVER10                                      00002250     00002590    00000340
      - 12/07/41 08.31 -           LNKEDTIC    OBJ
                                   LKSCPH06   CSECT      22        00002250     00002590    00000340    000025A0
              00002250
LNKEDT17      NODE -   OVER10                                      00002250     00002348    000000F8
      - 12/07/41 08.31 -           LNKEDTIC    OBJ
                                   LKSCPH51   CSECT      29        00002250     00002348    000000F8    000030B8
              00002250
LNKEDT18      NODE -   OVER9                                       00002168     00002250    000000E8
      - 12/07/41 08.31 -           LNKEDTIC    OBJ
                                   LKSCPH08   CSECT      24        00002168     00002250    000000E8    000029C8
              00002168
LNKEDT19      NODE -   OVER9                                       00002168     000022A0    00000138
      - 12/07/41 08.31 -           LNKEDTIC    OBJ
                                   LKSCPH48   CSECT      25        00002168     000022A0    00000138    00002A80
              00002168
LNKEDT20      NODE -   OVER7                                       00001D10     00001EE8    000001D8
      - 12/07/41 08.31 -           LNKEDTIC    OBJ
                                   LKSCPH70   CSECT      2F        00001D10     00001D40    00000030    000038F0
      - 12/07/41 08.31 -           LNKEDTIC    OBJ
                                   LKSCPH39   CSECT      26        00001D40     00001EE8    000001A8    00002BE8
              00001010
LNKEDT21      NODE -   OVER5                                       000019E8     00001B48    00000160
```

Figure 8–6. Link-Edit Example 5 (Part 9 of 14)



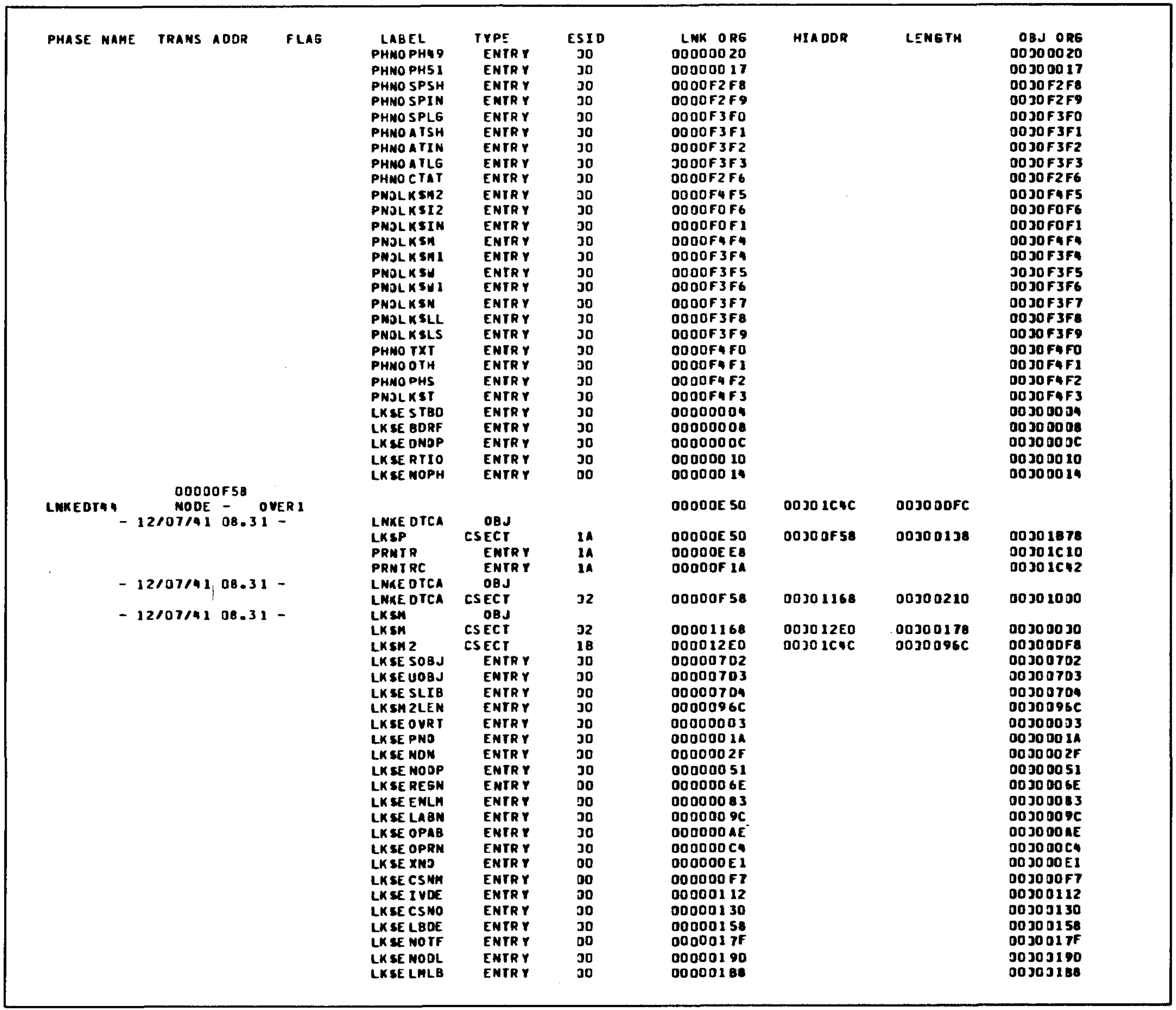
```
PHASE NAME  TRANS ADDR     FLAG          LABEL      TYPE       ESID       LNK ORG      HIADDR      LENGTH       OBJ ORG
         - 12/07/41.08.31 -             LNKEDTIC    OBJ
                                        LKSCPH02   CSECT       1C        000019E8     00001B48     00000160     00001930
              000019E8
LNKEDT22      NODE -    OVER5                                            000019E8     00001B40     00000158
         - 12/07/41 08.31 -             LNKEDTIC    OBJ
                                        LKSCPH03   CSECT       1E        000019E8     00001B40     00000158     00001D50
              000019E8
LNKEDT23      NODE -    OVER6                                            00001B40     00001C00     00000190
         - 12/07/41 08.31 -             LNKEDTIC    OBJ
                                        LKSCPH15   CSECT       2D        00001B40     00001C00     00000190     00003700
              00001B40
LNKEDT24      NODE -    OVER6                                            00001B40     00001D18     00000108
         - 12/07/41 08.31 -             LNKEDTIC    OBJ
                                        LKSCPH13   CSECT       2B        00001B40     00001D18     00000108     00003310
              00001B40
LNKEDT25      NODE -    OVER5                                            000019E8     00001B80     000001C8
         - 12/07/41 08.31 -             LNKEDTIC    OBJ
                                        LKSCPH41   CSECT       1B        000019E8     00001B80     000001C8     00001768
              000019E8
LNKEDT26      NODE -    OVER5                                            000019E8     00001CA8     000002C0
         - 12/07/41 08.31 -             LNKEDTIC    OBJ
                                        LKSCPH42   CSECT       1D        000019E8     00001CA8     000002C0     00001A90
                                        LKS4AUSP    ENTRY      1D        000019EA                               00001A92
              000019EA
LNKEDT27      NODE -    OVER5                                            000019E8     00001BA8     000001C0
         - 12/07/41 08.31 -             LNKEDTIC    OBJ
                                        LKSCPH43   CSECT       1F        000019E8     00001BA8     000001C0     00001EA8
              000019E8
LNKEDT28      NODE -    OVER4                                            000013E0     00001488     000000A8
         - 12/07/41 08.31 -             LNKEDTIC    OBJ
                                        LKSCPH16   CSECT       30        000013E0     00001488     000000A8     00003920
              000013E0
LNKEDT29      NODE -    OVER4                                            000013E0     00001488     000000A8
         - 12/07/41 08.31 -             LNKEDTIC    OBJ
                                        LKSCPH17   CSECT       32        000013E0     00001488     000000A8     00003A70
              000013E0
LNKEDT30      NODE -    OVER4                                            000013E0     00001488     000000A8
         - 12/07/41 08.31 -             LNKEDTIC    OBJ
                                        LKSCPH18   CSECT       34        000013E0     00001488     000000A8     00003BC0
              000013E0
LNKEDT31      NODE -    OVER4                                            000013E0     00001488     000000A8
         - 12/07/41 08.31 -             LNKEDTIC    OBJ
                                        LKSCPH56   CSECT       31        000013E0     00001488     000000A8     000039C8
              000013E0
LNKEDT32      NODE -    OVER4                                            000013E0     00001488     000000A8
         - 12/07/41 08.31 -             LNKEDTIC    OBJ
                                        LKSCPH57   CSECT       33        000013E0     00001488     000000A8     00003B18
              000013E0
LNKEDT33      NODE -    OVER4                                            000013E0     00001484     000000A4
         - 12/07/41 08.31 -             LNKEDTIC    OBJ
                                        LKSCPH58   CSECT       35        000013E0     00001484     000000A4     00003C68
              000013E0
LNKEDT34      NODE -    OVER3                                            00001168     00001600     00000568
         - 12/07/41 08.31 -             LKSM1       OBJ
                                        LKSM1      CSECT       02        00001168     00001600     00000568     00000000
              00001168
LNKEDT35      NODE -    OVER3                                            00001168     00001950     000007E8
         - 12/07/41 08.31 -             LKSW        OBJ
                                        LKSW       CSECT       02        00001168     00001950     000007E8     00000000
              00001168
```

Figure 8–6. Link-Edit Example 5 (Part 10 of 14)



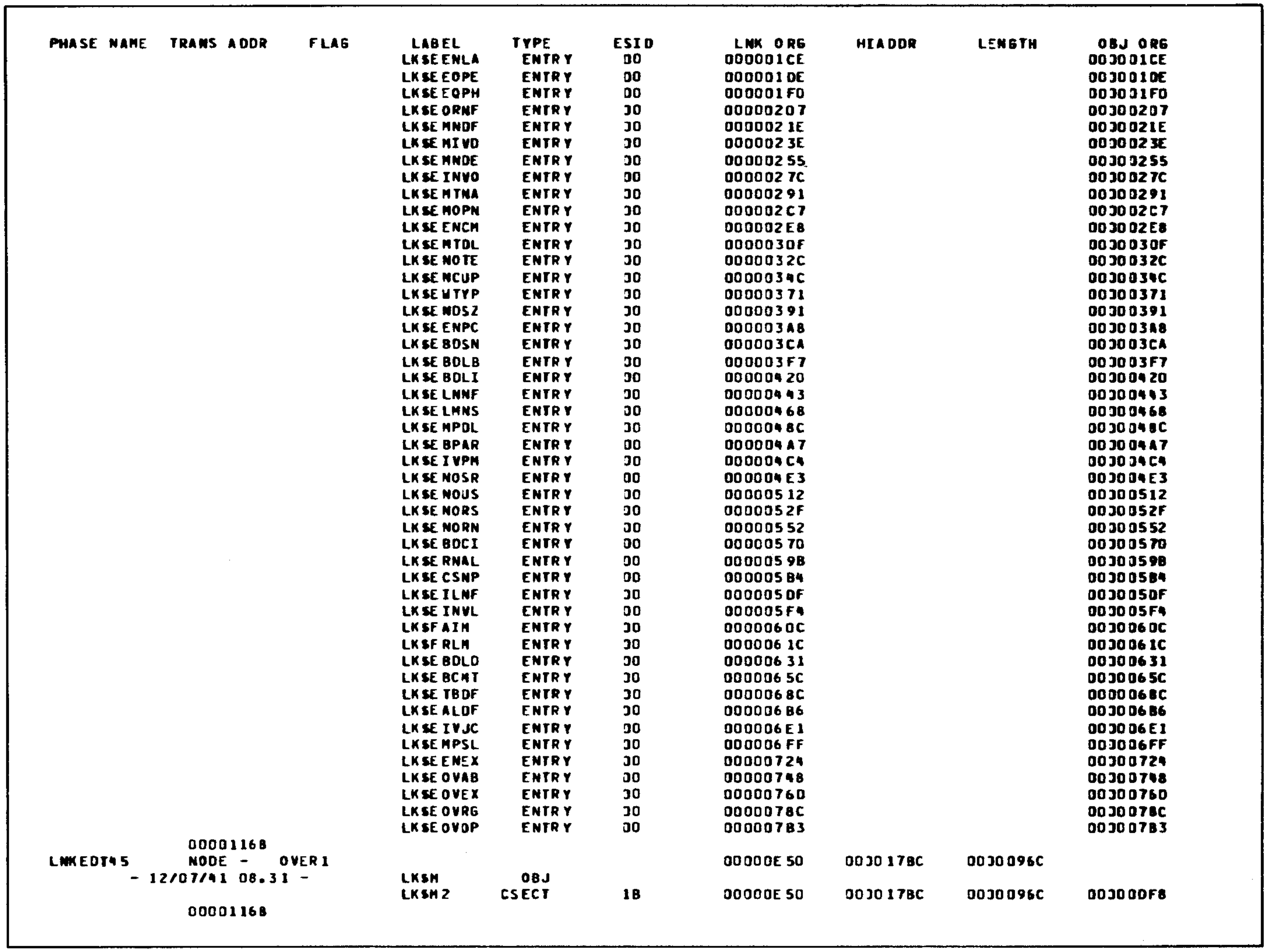
```
PHASE NAME  TRANS ADDR    FLAG        LABEL      TYPE        ESID          LNK ORG     HIADDR     LENGTH       OBJ ORG
LNKEDT36        NODE -     OVER3                                          00001168    00001708   00000670
        - 12/07/41 08.31 -            LKSW1        OBJ
                                      LKSW1      CSECT        02          00001168    00001708   00000670    00000000
               00001168
LNKEDT37        NODE -     OVER2                                          00000F58    00001C90   00000038
        - 12/07/41 08.31 -            LKSN         OBJ
                                      LKSN       CSECT        02          00000F58    00001C90   00000038    00000000
               00000F58
LNKEDT38        NODE -     OVER2                                          00000F58    00001620   000006C8
        - 12/07/41 08.31 -            LKSLLAST     OBJ
                                      DATALNG    CSECT        23          00000F58    00001190   00000238    00002AF0
                                      LKSLENSZ     ENTRY      00          00000004                           00000034
                                      LKSLPSTS     ENTRY      00          00000F78                           00000F78
                                      LKSLPSTL     ENTRY      00          00001E48                           00001E48
        - 12/07/41 08.31 -            LKSLLAST     OBJ
                                      LKSLLAST   CSECT        02          00001190    00001620   00000490    00000000
                                      LKSLPTOV     ENTRY      02          000013A6                           00000216
               00001190
LNKEDT39        NODE -     OVER2                                          00000F58    00001620   000006C8
        - 12/07/41 08.31 -            LKSLLAST     OBJ
                                      DATASHT    CSECT        22          00000F58    00001190   00000238    00002888
        - 12/07/41 08.31 -            LKSLLAST     OBJ
                                      LKSLLAST   CSECT        02          00001190    00001620   00000490    00000000
                                      LKSLPTOV     ENTRY      02          000013A6                           00000216
               00001190
LNKEDT40        NODE -     OVER3                                          00001620    00001ED0   00000880
        - 12/07/41 08.31 -            LKSLLAST     OBJ
                                      LASTTXT    CSECT        1F          00001620    00001ED0   00000880    00001138
               00001620
LNKEDT41        NODE -     OVER3                                          00001620    00001070   00000750
        - 12/07/41 08.31 -            LKSLLAST     OBJ
                                      LASTOTH    CSECT        20          00001620    00001070   00000750    000019E8
               00001620
LNKEDT42        NODE -     OVER3                                          00001620    00001DA0   00000780
        - 12/07/41 08.31 -            LKSLLAST     OBJ
                                      LASTPHS    CSECT        21          00001620    00001DA0   00000780    00002138
               C0001620
LNKEDT43        NODE -     OVER2                                          00000F58    00001888   00000930
        - 12/07/41 08.31 -            LKST         OBJ
                                      LKST       CSECT        02          00000F58    00001888   00000930    00000000
                                      LKSENOSC     ENTRY      00          00000018                           00000018
                                      PHNOPH01     ENTRY      00          00000007                           00000037
                                      PHNOPH02     ENTRY      00          00000021                           00000021
                                      PHNOPH03     ENTRY      00          00000022                           00000022
                                      PHNOPH04     ENTRY      00          00000009                           00000039
                                      PHNOPH05     ENTRY      00          00000013                           00000013
                                      PHNOPH06     ENTRY      00          00000016                           00000016
                                      PHNOPH07     ENTRY      00          00000014                           00000014
                                      PHNOPH08     ENTRY      00          00000018                           00000018
                                      PHNOPH09     ENTRY      00          00000011                           00000011
                                      PHNOPH10     ENTRY      00          00000008                           00000038
                                      PHNOPH11     ENTRY      00          00000015                           00000015
                                      PHNOPH12     ENTRY      00          00000010                           00000010
                                      PHNOPH13     ENTRY      00          00000024                           00000024
                                      PHNOPH14     ENTRY      00          00000012                           00000012
                                      PHNOPH15     ENTRY      00          00000023                           00000023
                                      PHNOPH41     ENTRY      00          00000025                           0Q000025
                                      PHNOPH42     ENTRY      00          00000026                           00000026
                                      PHNOPH43     ENTRY      0C          00000027                           00000027
                                      PHNOPH48     ENTRY      00          00000019                           00000019
```

Figure 8-6. Link-Edit Example 5 (Part 11 of 14)



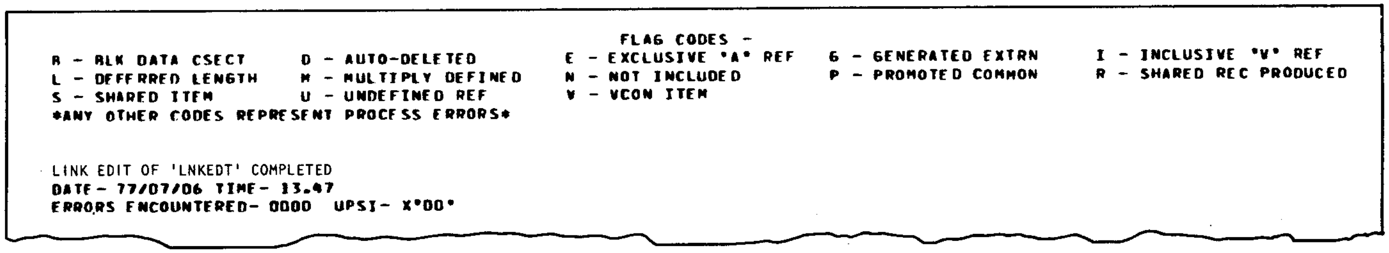
```
PHASE NAME  TRANS ADDR    FLAG      LABEL      TYPE    ESID     LNK ORG   HIADDR    LENGTH    OBJ ORG
                                    PHNOPH49   ENTRY   00      00000020                      00000020
                                    PHNOPH51   ENTRY   00      00000017                      00000017
                                    PHNOSPSH   ENTRY   00      0000F2F8                      0000F2F8
                                    PHNOSPIN   ENTRY   00      0000F2F9                      0000F2F9
                                    PHNOSPLG   ENTRY   00      0000F3F0                      0000F3F0
                                    PHNOATSH   ENTRY   00      0000F3F1                      0000F3F1
                                    PHNOATIN   ENTRY   00      0000F3F2                      0000F3F2
                                    PHNOATLG   ENTRY   00      0000F3F3                      0000F3F3
                                    PHNOCTAT   ENTRY   00      0000F2F6                      0000F2F6
                                    PNOLK$M2   ENTRY   00      0000F4F5                      0000F4F5
                                    PNOLK$I2   ENTRY   00      0000F0F6                      0000F0F6
                                    PNOLK$IN   ENTRY   00      0000F0F1                      0000F0F1
                                    PNOLK$M    ENTRY   00      0000F4F4                      0000F4F4
                                    PNOLK$M1   ENTRY   00      0000F3F4                      0000F3F4
                                    PNOLK$W    ENTRY   00      0000F3F5                      0000F3F5
                                    PNOLK$W1   ENTRY   00      0000F3F6                      0000F3F6
                                    PNOLK$N    ENTRY   00      0000F3F7                      0000F3F7
                                    PNOLK$LL   ENTRY   00      0000F3F8                      0000F3F8
                                    PNOLK$LS   ENTRY   00      0000F3F9                      0000F3F9
                                    PHNOTXT    ENTRY   00      0000F4F0                      0000F4F0
                                    PHNOOTH    ENTRY   00      0000F4F1                      0000F4F1
                                    PHNOPHS    ENTRY   00      0000F4F2                      0000F4F2
                                    PNOLK$T    ENTRY   00      0000F4F3                      0000F4F3
                                    LK$ESTBO   ENTRY   00      00000004                      00000004
                                    LK$EBDRF   ENTRY   00      00000008                      00000008
                                    LK$EDNOP   ENTRY   00      0000000C                      0000000C
                                    LK$ERTIO   ENTRY   00      00000010                      00000010
                                    LK$ENOPH   ENTRY   00      00000014                      00000014
                  00000F58
LNKEDT44          NODE -    OVER1                              00000E50  00001C4C  00000DFC
      - 12/07/41 08.31 -            LNKEDTCA  OBJ
                                    LK$P      CSECT    1A      00000E50  00000F58  00000138  00001B78
                                    PRNTR      ENTRY   1A      00000EE8                      00001C10
                                    PRNTRC     ENTRY   1A      00000F1A                      00001C42
      - 12/07/41 08.31 -            LNKEDTCA  OBJ
                                    LNKEDTCA  CSECT    02      00000F58  00001168  00000210  00001000
      - 12/07/41 08.31 -            LK$M      OBJ
                                    LK$M      CSECT    02      00001168  000012E0  00000178  00000030
                                    LK$M2     CSECT    18      000012E0  00001C4C  0000096C  00000DF8
                                    LK$ESOBJ   ENTRY   00      000007D2                      000007D2
                                    LK$EUOBJ   ENTRY   00      000007D3                      000007D3
                                    LK$ESLIB   ENTRY   00      000007D4                      000007D4
                                    LK$M2LEN   ENTRY   00      0000096C                      0000096C
                                    LK$EOVRT   ENTRY   00      00000003                      00000003
                                    LK$EPNO    ENTRY   00      0000001A                      0000001A
                                    LK$ENON    ENTRY   00      0000002F                      0000002F
                                    LK$ENODP   ENTRY   00      00000051                      00000051
                                    LK$EREGN   ENTRY   00      0000006E                      0000006E
                                    LK$EENLM   ENTRY   00      00000083                      00000083
                                    LK$ELABN   ENTRY   00      0000009C                      0000009C
                                    LK$EOPAB   ENTRY   00      000000AE                      000000AE
                                    LK$EOPRN   ENTRY   00      000000C4                      000000C4
                                    LK$EXNO    ENTRY   00      000000E1                      000000E1
                                    LK$ECSNM   ENTRY   00      000000F7                      000000F7
                                    LK$EIVDE   ENTRY   00      00000112                      00000112
                                    LK$ECSNO   ENTRY   00      00000130                      00000130
                                    LK$ELBDE   ENTRY   00      00000158                      00000158
                                    LK$ENOTF   ENTRY   00      0000017F                      0000017F
                                    LK$ENODL   ENTRY   00      00000190                      00000190
                                    LK$ELMLB   ENTRY   00      00000188                      00000188
```

Figure 8-6. Link-Edit Example 5 (Part 12 of 14)



```
PHASE NAME  TRANS ADDR    FLAG       LABEL      TYPE      ESID        LNK ORG     HIADDR      LENGTH       OBJ ORG
                                    LKSEENLA    ENTRY      00        000001CE                             000001CE
                                    LKSEEOPE    ENTRY      00        000001DE                             000001DE
                                    LKSEEQPH    ENTRY      00        000001F0                             000001F0
                                    LKSEORNF    ENTRY      00        00000207                             00000207
                                    LKSEMNDF    ENTRY      00        0000021E                             0000021E
                                    LKSEMIVD    ENTRY      00        0000023E                             0000023E
                                    LKSEMNDE    ENTRY      00        00000255                             00000255
                                    LKSEINVO    ENTRY      00        0000027C                             0000027C
                                    LKSEMTNA    ENTRY      00        00000291                             00000291
                                    LKSEMOPN    ENTRY      00        000002C7                             000002C7
                                    LKSEENCM    ENTRY      00        000002E8                             000002E8
                                    LKSEMTDL    ENTRY      00        0000030F                             0000030F
                                    LKSENOTE    ENTRY      00        0000032C                             0000032C
                                    LKSENCUP    ENTRY      00        0000034C                             0000034C
                                    LKSEWTYP    ENTRY      00        00000371                             00000371
                                    LKSENDSZ    ENTRY      00        00000391                             00000391
                                    LKSEENPC    ENTRY      00        000003A8                             000003A8
                                    LKSEBDSN    ENTRY      00        000003CA                             000003CA
                                    LKSEBDLB    ENTRY      00        000003F7                             000003F7
                                    LKSEBDLI    ENTRY      00        00000420                             00000420
                                    LKSELNNF    ENTRY      00        00000443                             00000443
                                    LKSELMNS    ENTRY      00        00000468                             00000468
                                    LKSEMPDL    ENTRY      00        0000048C                             0000048C
                                    LKSEBPAR    ENTRY      00        000004A7                             000004A7
                                    LKSEIVPM    ENTRY      00        000004C4                             000004C4
                                    LKSENOSR    ENTRY      00        000004E3                             000004E3
                                    LKSENOUS    ENTRY      00        00000512                             00000512
                                    LKSENORS    ENTRY      00        0000052F                             0000052F
                                    LKSENORN    ENTRY      00        00000552                             00000552
                                    LKSEBDCI    ENTRY      00        00000570                             00000570
                                    LKSERNAL    ENTRY      00        0000059B                             0000059B
                                    LKSECSNP    ENTRY      00        000005B4                             000005B4
                                    LKSEILNF    ENTRY      00        000005DF                             000005DF
                                    LKSEINVL    ENTRY      00        000005F4                             000005F4
                                    LKSFAIM     ENTRY      00        0000060C                             0000060C
                                    LKSFRLM     ENTRY      00        0000061C                             0000061C
                                    LKSEBDLD    ENTRY      00        00000631                             00000631
                                    LKSEBCNT    ENTRY      00        0000065C                             0000065C
                                    LKSETBDF    ENTRY      00        0000068C                             0000068C
                                    LKSEALDF    ENTRY      00        000006B6                             000006B6
                                    LKSEIVJC    ENTRY      00        000006E1                             000006E1
                                    LKSEMPSL    ENTRY      00        000006FF                             000006FF
                                    LKSEENEX    ENTRY      00        00000724                             00000724
                                    LKSEOVAB    ENTRY      00        00000748                             00000748
                                    LKSEOVEX    ENTRY      00        0000076D                             0000076D
                                    LKSEOVRG    ENTRY      00        0000078C                             0000078C
                                    LKSEOVDP    ENTRY      00        000007B3                             000007B3
            00001168
LNKEDT45    NODE -    OVER1                                      00000E50    0000178C    0000096C
      - 12/07/41 08.31 -            LKSM        OBJ
                                    LKSM2      CSECT       1B        00000E50    0000178C    0000096C    00000DF8
            00001168
```

Figure 8-6. Link-Edit Example 5 (Part 13 of 14)



```
                                                        FLAG CODES -
B - BLK DATA CSECT      D - AUTO-DELETED        E - EXCLUSIVE 'A' REF   G - GENERATED EXTRN     I - INCLUSIVE 'V' REF
L - DEFFRRED LENGTH     M - MULTIPLY DEFINED    N - NOT INCLUDED        P - PROMOTED COMMON     R - SHARED REC PRODUCED
S - SHARED ITEM         U - UNDEFINED REF       V - VCON ITEM
*ANY OTHER CODES REPRESENT PROCESS ERRORS*


LINK EDIT OF 'LNKEDT' COMPLETED
DATE- 77/07/06 TIME- 13.47
ERRORS ENCOUNTERED- 0000  UPSI- X'00'
```

Figure 8-6. Link-Edit Example 5 (Part 14 of 14)

PART 4. SYSTEM UTILITIES

9. INITIALIZE Disk Routine (DSKPRP)

9.1. GENERAL

Before any of your disk packs or diskettes may be used, the first thing you must do is initialize the medium, a process commonly referred to as prepping. Because of the difference in design of the disk packs and diskettes, the prepping functions and options for each are different. In this section, we will discuss each type of prep separately.

In addition to the disk prep routine, there is another prep facility in the form of a canned job control stream called SETREL. SETREL can prep and allocate SYSRES files. There are other canned job control streams to copy, print, and pack SYSRES files. Appendix A explains all of the SYSRES canned job control streams.

9.2. PREPPING YOUR DISK PACK

Prepping a disk pack is made up of two functions. The first function is analyzing the surface of your disk pack for possible defective tracks, keeping a history of these defective tracks, and assigning alternate tracks. The result of this analysis is the track condition table (TCT), which is printed out and used to determine the condition of your disk pack. We will talk more about the TCT a little later.

The second function is building the initial records that must be written on the tracks of your disk pack before it can receive any data or programs. The initial records include the disk volume label records, known as the standard volume label (VOL1) records, and the volume table of contents (VTOC) records.

In addition to the two functions just discussed, you can also do the following operations:

- Prepping specific tracks or groups of tracks
- Choosing options to specify prepping only, fast surface checking (fast prep), or an extreme accuracy surface analysis with or without prepping your disk pack
- Replacing the volume serial number only
- Adding or replacing IL (COS) only
- Creating a new volume serial number and a VTOC without generating a new TCT

- Checking the expiration date for all files

The name of the disk prep routine is called DSKPRP and it can prep any disk supported by OS/3, IDA sectored, selector channel nonsectored, or IPC diskette. The IDA sectored disk packs are 8415, 8416, and 8418; the selector channel nonsectored disk packs are the 8411, 8414, 8424/8425, 8430, and 8433. The IPC diskette is the 8413.

9.3. SPECIFYING THE PREP OPTIONS

You choose the prep options by selecting one or more of the following keywords. The only keyword you must select is the SERNR keyword identifying the volume serial number of your disk pack. As many keywords as can be contained on one card are permitted and as many cards as needed can be present in your control stream. Remember also, keywords may be specified in any order.

The format of the keywords is:

```
[,ALTRK={N}] [,ILOPT=(C)] [,INSRT={N}]
[       {Y}] [       {N}] [       {X}]
             [       {1}] [       {Y}]
             [       {2}]
             [       {3}]

[,IPLDK={N}] [,RPVOL={N}] [,PARTL={N}]
[       {Y}] [       {Y}] [       {S}]
                          [       {V}]

[,PREPT=(1)] [,RETRY={nn}] [,PTBEG={ccchh }]
[       {2}] [       {ØA}] [       {ØØØØØØ}]
[       {3}]
[       {C}]
[       {F}]

[,PTEND=cccchh] ,SERNR=volume serial number (six characters)

[,TRCON=(C)] [,TRKCT=(C)] [,VERFY={N}] [,UNXFC={N}]
[       {D}] [       {D}] [       {Y}] [       {Y}]
[       {N}] [       {L}]
[       {S}] [       {P}]
             [       {X}]
             [       {Z}]

[,VTOCB=(cccchh                                                       )]
[       {ØØØØØ1 for all non-IPL volumes                               }]
[       {ØØ64ØØ for 8411,8414 IPL volumes                             }]
[       {ØØCAØØ for 8415,8416,8418,8424/25,843Ø,8433 IPL volumes      }]

[,VTOCE=(cccchh                                  )]
[       {ØØØØØ9 for 8411 non-IPL volume          }]
[       {ØØØØ13 for 8414,8424/25 non-IPL volume  }]
[       {ØØØØØ2 for 8415 fixed, non-IPL volume   }]
[       {ØØØØØ1 for 8415 removable,non-IPL volume}]
[       {ØØØØØ6 for 8416,8418 non-IPL volumes    }]
[       {ØØØØ12 for 843Ø,8433 non-IPL volumes    }]
[       {ØØ64Ø9 for 8411 IPL volume              }]
[       {ØØ6413 for 8414 IPL volume              }]
[       {ØØCAØ2 for 8415 fixed, IPL volume       }]
[       {ØØCAØ1 for 8415 removable, IPL volume   }]
[       {ØØCAØ6 for 8416,8418 IPL volumes        }]
[       {ØØCA13 for 8424/25 IPL volumes          }]
[       {ØØCA12 for 843Ø,8433 IPL volume         }]
```

9.3.1. Testing Alternate Track Areas (ALTRK)

By either specifying ALTRK=Y or omitting the ALTRK keyword, the alternate track areas of your disk pack are tested. As stated earlier, one of the prep functions is performing a surface analysis of your disk pack. During this process, any defective tracks are flagged and an alternate track is assigned. The ALTRK keyword performs a surface analysis of these alternate tracks ensuring that the track is usable as an alternate track.

You can suppress the testing of the alternate tracks by specifying ALTRK=N.

9.3.2. Indicating the Type of Initial Load Control Storage (ILOPT)

By specifying one of the options associated with the ILOPT keyword, you indicate the type of initial load (IL) control storage (COS) to be written on your disk pack.

You select the C option signifying that the IL (COS) is card input. In this case, the COS card deck must be the second embedded data set passed to DSKPRP, as shown in 9.9, example 3.

You select the 1 option, signifying that the IL (COS) is disk input and is 1K capacity.

You select the 2 option, signifying that the IL (COS) is disk input but is 2K capacity.

You select the 3 option, signifying that the IL (COS) is disk input but is 2K fast capacity.

When you specify ILOPT=N or omit the keyword, no IL (COS) is written.

In addition to the C option, there is also a canned job control stream (ADDnnCOS) that performs the same function. However, when ADDnnCOS is run, immediately afterwards a prep canned job control stream PRPnnCOS must be run to activate COS. See 9.11 for a description of both the prep and the add COS canned job control streams.

9.3.3. Automatically Recording Defective Tracks (INSRT)

The SPERRY UNIVAC 8415, 8416, and 8418 sectored disk packs go through an extensive surface analysis during their manufacturing process. Any defective tracks detected during this process are listed decimally and hexadecimally in cylinder and head format (ccccchh) in a defective track table. This table is located on the bottom of the plastic cover on a label that also lists your pack's volume serial number. Regardless of where the defective tracks are listed, the addresses of these defective tracks must be specified *hexadecimally* on the INSERT control statement; however, you must also specify the X or Y option in the INSRT keyword, telling the prep routine there are INSERT control statements present in your control stream.

You use INSRT=X, indicating only INSERT control statements are used to designate defective tracks, while the INSRT=Y indicates the INSERT control statements are used together with the normal surface analysis function to flag any defective tracks on your disk pack. The Y option is automatically set whenever the keyword TRCON=N is used and a sectorized disk pack is being prepped.

If you specify INSRT=N or omit the keyword, you indicate that no INSERT control statements are present in your control stream.

9.3.4. Indicating Your Disk Pack Is an IPL Volume (IPLDK)

If you specify IPLDK=Y or omit the keyword, you are indicating your disk pack is to be used as an IPL volume. Your SYSRES pack is always an IPL volume.

If this pack is not to be used as an IPL volume, then you must specify IPLDK=N.

9.3.5. Renumbering Your Volume Serial Number or Replacing Initial Load Control Storage (RPVOL)

When you want to renumber your volume serial number, you specify RPVOL=Y and then insert your new volume serial number in the SERNR keyword. If you are renumbering in a multiple volume environment, then you cannot use the VCHECK parameter in the // LBL job control statement for volume checking. The volume check function checks the sequence between the volume serial number and the file serial number. Your volume serial number and file serial number are part of the VTOC. The RPVOL keyword does not change the file serial number; therefore, in multiple volume files, data management does not know the sequence of volumes being processed.

In addition to renumbering your volume serial number, you can also replace the existing IL (COS) without destroying the existing information on your disk pack. You use the ILOPT keyword in conjunction with RPVOL=Y keyword for this operation. However, you cannot renumber your volume serial number and replace IL (COS) at the same time. Whenever you use the RPVOL keyword, all other keywords except ILOPT and the SERNR are ignored. As with any prep operation, the VOL1 control statement must be specified.

You may also replace the user address specified for this volume by specifying RPVOL=Y and including the new user address on the VOL1 card. If you do not wish to change the user address, you must leave that field blank on the VOL1 card.

If you need to renumber your volume serial number and the disk pack does not need to be prepped, you can use the canned job control stream called CHGVSN.

NOTE:

If changing the volume serial number of a SYSRES, SYSRUN, or spooling disk pack, extreme care should be used not to specify a previously assigned vsn. Such duplication may not cause a system duplication warning message to be issued but will severely impact the system. If a duplicate vsn is inadvertently assigned, the entire system must be reinitialized (IPL).

9.3.6. Specifying a Partial Prep or Changing Your Volume Serial Number and VTOC (PARTL)

If you only want to prep a portion of your disk pack, regardless of your pack being allocated or not, then you would specify the PARTL=S keyword.

When you use PARTL=S, the keywords PTBEG and PTEND must be specified since they indicate the area of your disk pack being prepped. The keywords TRCON=N and TRKCT=P must not be specified since a partial prep does not include generating a new track condition table. The testing of alternate tracks is automatically suppressed via the keyword ALTRK=N. You may specify the INSRT and PREPT keywords; however, the VTOCB, VTOCE, ILOPT, and IPLDK keywords are ignored if they are present in your control stream. As with any prep, the keyword SERNR, which indicates the volume serial number, and the VOL1 card, which creates the standard VOL1 labels, must be specified.

When you specify this option, it is possible to destroy any and all information residing in the area being prepped via the PTBEG and PTEND keywords. This area can also include the volume serial number and the VTOC. All areas outside the prep area remain unchanged.

If you want to specify a new volume serial number, then you would specify the PARTL=V keyword. Your new volume serial number, which is specified on the SERNR keyword, is written in the VOL1 label. This keyword removes all the entries from the VTOC, thus making your disk pack unusable unless you know the specific address of your files. This addressing could have been done by using the ADDR parameter in the EXT job control statement when you allocated your files. If you want to renumber your volume serial number and still have your disk pack usable, and you did not use the ADDR parameter in the // EXT job control statement, you must use the RPVOL keyword.

9.3.7. How Accurate a Prep Do You Need? (PREPT and RETRY)

The first time a disk pack is prepped, you should perform an extreme accuracy prep. You select the extreme accuracy prep by specifying PREPT=3 or PREPT=C (complete prep). In the case of unrecoverable data errors, a second complete prep (PREPT=C) may be desirable. Once a disk pack has been prepped, all subsequent preps should be specified with PREPT=F (fast prep). If you are prepping an 8416 disk pack, you automatically receive the extreme accuracy prep, unless PREPT=F is specified. This keyword indicates that only a minimum surface analysis is to be performed. There are two other preps available, PREPT=1 and PREPT=2. These two preps are more accurate than the fast prep but less accurate than the extreme accuracy prep.

Regardless of the type of prep requested, DSKPRP will test a given track up to the number of times specified by the RETRY keyword, in an attempt to have it pass the surface analysis test. If a track is found defective the first time it is tested, it is retested up to the number of times specified by the RETRY keyword before it is declared bad, and an alternate track is substituted for it. Thus, lowering the RETRY specification has the effect of making the disk prep routine perform a more critical surface analysis by giving each track fewer chances to pass the surface analysis test. Conversely, increasing the RETRY specification lowers the effectiveness of the surface analysis test by giving each track more chances to pass the test.

The number of retries is specified as a hexadecimal number from 00 to FF. The default number of retries is 10 ($0A_{16}$).

9.3.8. Specifying Where Prepping Starts (PTBEG) and Ends (PTEND)

When you omit both the PTBEG and PTEND keywords, disk prep starts at the primary track address of 000000 and ends at the highest primary track address of your disk pack.

If you want prepping to start other than at the beginning of the disk pack and end other than at the highest primary track address, you would specify these addresses (hexadecimal) in cylinder/head format (cccchh).

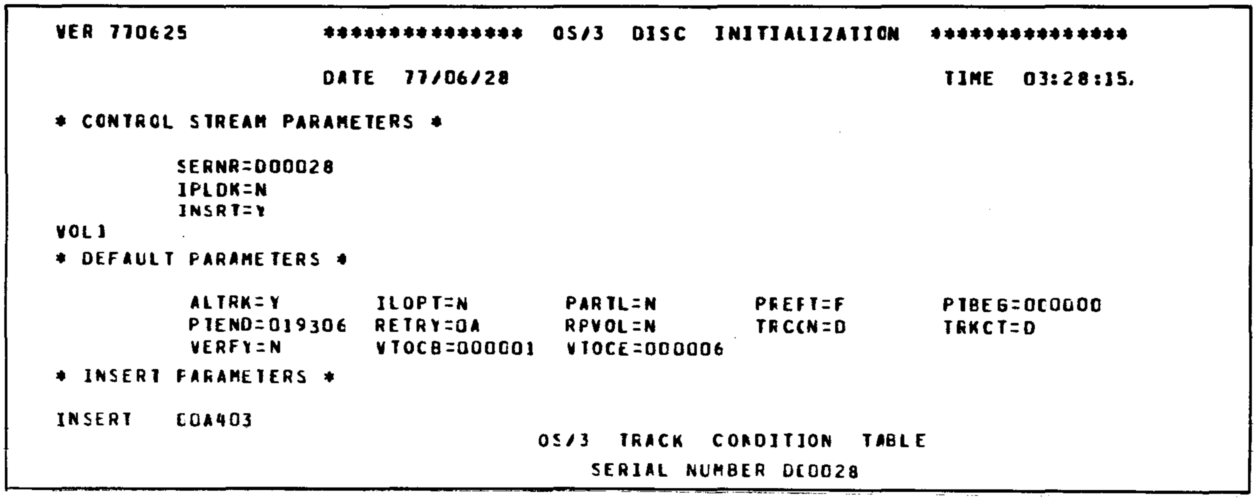
The use of PTBEG and PTEND does not prevent the initialization of the VOL1 record, the boot/IPL record, COS, or the VTOC. To suppress the initialization of these records, you must specify the PARTL=S parameter.

9.3.9. Specifying Your Volume Serial Number (SERNR)

The volume serial number is six alphanumeric characters, which make up the serial number of the disk volume being prepped, and may not contain any blank characters. Your volume serial number may already have been assigned to the disk volume through a previous prep or it may specify a new serial number. In either case, the SERNR keyword must always be present in your disk prep control stream. If a new serial number is being specified on a previously prepped disk pack, you should specify either the RPVOL parameter or the NOV option on the VOL statement.

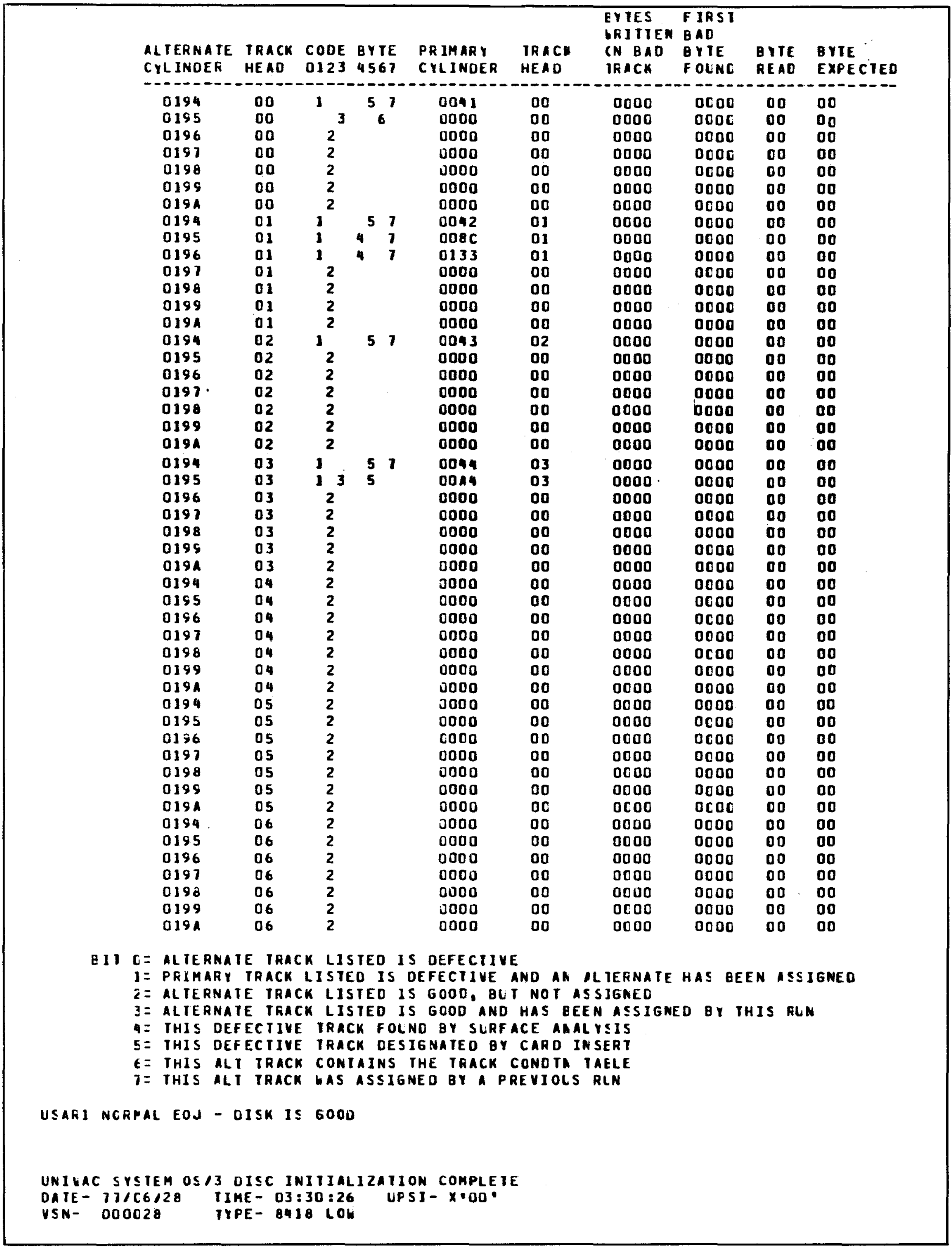
9.3.10. Specifying a Track Condition Table (TRCON and TRKCT)

The track condition table tells you the general condition of your disk pack and provides subsequent preps with a history of the alternate track assignments. Figure 9-1 shows a typical track condition table printout generated by DSKPRP.



```
VER 770625          ***************  OS/3  DISC  INITIALIZATION  ***************

                    DATE  77/06/28                                     TIME  03:28:15.

* CONTROL STREAM PARAMETERS *

         SERNR=000028
         IPLDK=N
         INSRT=Y
VOL1
* DEFAULT PARAMETERS *

         ALTRK=Y       ILOPT=N       PARTL=N       PREFT=F       PTBEG=000000
         PTEND=019306  RETRY=0A      RPVOL=N       TRCON=D       TRKCT=D
         VERFY=N       VTOCB=000001  VTOCE=000006
* INSERT PARAMETERS *

INSERT   C0A403
                                  OS/3  TRACK  CONDITION  TABLE
                                       SERIAL NUMBER 000028
```

Figure 9-1. Track Condition Table (Part 1 of 2)



```
                                               BYTES    FIRST
                                               WRITTEN  BAD
ALTERNATE  TRACK  CODE BYTE  PRIMARY   TRACK   ON BAD   BYTE     BYTE  BYTE
CYLINDER   HEAD   0123 4567  CYLINDER  HEAD    TRACK    FOUND    READ  EXPECTED
-------------------------------------------------------------------------------
  0194      00     1    5 7   0041      00      0000     0000     00    00
  0195      00       3   6    0000      00      0000     0000     00    00
  0196      00      2         0000      00      0000     0000     00    00
  0197      00      2         0000      00      0000     0000     00    00
  0198      00      2         0000      00      0000     0000     00    00
  0199      00      2         0000      00      0000     0000     00    00
  019A      00      2         0000      00      0000     0000     00    00
  0194      01     1    5 7   0042      01      0000     0000     00    00
  0195      01     1   4  7   008C      01      0000     0000     00    00
  0196      01     1   4  7   0133      01      0000     0000     00    00
  0197      01      2         0000      00      0000     0000     00    00
  0198      01      2         0000      00      0000     0000     00    00
  0199      01      2         0000      00      0000     0000     00    00
  019A      01      2         0000      00      0000     0000     00    00
  0194      02     1    5 7   0043      02      0000     0000     00    00
  0195      02      2         0000      00      0000     0000     00    00
  0196      02      2         0000      00      0000     0000     00    00
  0197      02      2         0000      00      0000     0000     00    00
  0198      02      2         0000      00      0000     0000     00    00
  0199      02      2         0000      00      0000     0000     00    00
  019A      02      2         0000      00      0000     0000     00    00
  0194      03     1    5 7   0044      03      0000     0000     00    00
  0195      03     1 3  5     00A4      03      0000     0000     00    00
  0196      03      2         0000      00      0000     0000     00    00
  0197      03      2         0000      00      0000     0000     00    00
  0198      03      2         0000      00      0000     0000     00    00
  0199      03      2         0000      00      0000     0000     00    00
  019A      03      2         0000      00      0000     0000     00    00
  0194      04      2         0000      00      0000     0000     00    00
  0195      04      2         0000      00      0000     0000     00    00
  0196      04      2         0000      00      0000     0000     00    00
  0197      04      2         0000      00      0000     0000     00    00
  0198      04      2         0000      00      0000     0000     00    00
  0199      04      2         0000      00      0000     0000     00    00
  019A      04      2         0000      00      0000     0000     00    00
  0194      05      2         0000      00      0000     0000     00    00
  0195      05      2         0000      00      0000     0000     00    00
  0196      05      2         0000      00      0000     0000     00    00
  0197      05      2         0000      00      0000     0000     00    00
  0198      05      2         0000      00      0000     0000     00    00
  0199      05      2         0000      00      0000     0000     00    00
  019A      05      2         0000      00      0000     0000     00    00
  0194      06      2         0000      00      0000     0000     00    00
  0195      06      2         0000      00      0000     0000     00    00
  0196      06      2         0000      00      0000     0000     00    00
  0197      06      2         0000      00      0000     0000     00    00
  0198      06      2         0000      00      0000     0000     00    00
  0199      06      2         0000      00      0000     0000     00    00
  019A      06      2         0000      00      0000     0000     00    00

     BIT 0= ALTERNATE TRACK LISTED IS DEFECTIVE
         1= PRIMARY TRACK LISTED IS DEFECTIVE AND AN ALTERNATE HAS BEEN ASSIGNED
         2= ALTERNATE TRACK LISTED IS GOOD, BUT NOT ASSIGNED
         3= ALTERNATE TRACK LISTED IS GOOD AND HAS BEEN ASSIGNED BY THIS RUN
         4= THIS DEFECTIVE TRACK FOUND BY SURFACE ANALYSIS
         5= THIS DEFECTIVE TRACK DESIGNATED BY CARD INSERT
         6= THIS ALT TRACK CONTAINS THE TRACK CONDTN TABLE
         7= THIS ALT TRACK WAS ASSIGNED BY A PREVIOUS RUN

USAR1 NORMAL EOJ - DISK IS GOOD


UNIVAC SYSTEM OS/3 DISC INITIALIZATION COMPLETE
DATE- 77/06/28    TIME- 03:30:26    UPSI- X'00'
VSN-  000028      TYPE- 8418 LOW
```

Figure 9-1. Track Condition Table (Part 2 of 2)

Looking at our track condition table, we see that there are seven defective tracks designated by a 1 bit in the code byte. Five of the tracks are designated by card inserts (bit 5 in code byte) and two were found by surface analysis. When a defective track is found, an alternate track is automatically assigned. The tracks reserved as alternates are listed in the first two columns on the table. As shown, the last tracks (highest addresses) on the disk pack are used as alternate tracks. Further, the first good alternate track is used to store the track condition table. Bits 3 and 6 of the code byte indicate the condition.

Looking at our track condition table, we see that there were no defective tracks found through the surface analysis as indicated by zeros being in both the primary cylinder and track head columns. The only track for which an alternate was assigned was the track identified as bad on the INSERT statement (track 4 of cylinder 69). The tracks reserved as alternates are listed in the first two columns on the table. As shown, the last tracks (highest addresses) on the disk pack are used as alternate tracks. Further, the first good alternate track is used to store the track condition table. Bits 3 and 6 of the code byte indicate the condition. The only other alternate track being used is track 4 of cylinder 194. This track is being used in place of track 4 of cylinder 69. All other alternate tracks are listed as good and are assigned.

There are two keywords associated with your track condition table: TRCON and TRKCT. When you specify TRCON, the following options are available:

- You use TRCON=C, indicating the track condition is input from cards. Normally, this option is not used, since the track condition table is in binary form when punched in cards.

- You use TRCON=D, indicating the track condition table is input from disk. You use this method when your disk pack has been in use and the current information is to be retained.

- You use TRCON=N or TRCON=S, indicating a new track condition table is to be generated. You must use this method when prepping the disk pack for the first time or if you cannot recover the existing track condition table by using one of the first methods.

It is worth noting that, when you prep an IDA disk pack using TRCON=N or TRCON=S, INSERT processing is forced and INSRT=Y is automatically set.

When you specify TRKCT, the following options are available.

- You use TRKCT=C to indicate your track condition table is to be punched on cards.

- You use TRKCT=D to indicate your track condition table is written to an alternate track.

- You use TRKCT=L to indicate your track condition table is listed on the printer and no other prepping functions are to take place. Your disk must already be prepped with TRCON set to D. This option is useful for future references about the track condition of your disk pack.

- You use TRKCT=P to indicate your track condition table is listed on the printer. When you prep an IDA disk pack, this option is not permitted.

- You use TRKCT=X to indicate your track condition table is punched on cards and listed on the printer.

- You use TRKCT=Z to indicate your track condition table is written to an alternate track and listed on the printer.

It is recommended that either TRKCT=D or TRKCT=Z be selected so subsequent preps may reap the benefits of previous surface analysis results and INSERT specifications.

9.3.11. Testing an Area before Prepping (VERFY)

If you are doing a partial prep, it is a good rule to test the area to be prepped first to make sure that area is free of data. You use the VERFY=Y keyword to do this testing. Whenever you specify VERFY=Y, the following keywords must also be specified: PTBEG, PTEND, and SERNR; in addition, TRCON cannot be an N and TRKCT cannot be a P. The keyword ALTRK=N is assumed while the keywords VTOCB, VTOCE, ILOPT, and IPLOK are ignored if they are present in your control stream. The INSRT and PREPT keywords may be used, if needed. As with all preps, the VOL1 card must be specified.

When you specify VERFY=N or omit the keyword, no testing is performed.

NOTE:

If the area defined by the PTBEG and PTEND keywords is occupied, the prep will automatically stop, a message will be posted, and no further action will take place. Also, the VOL1 label or the VTOC area is not rebuilt.

9.3.12. Checking the File Expiration Date (UNXFC)

You use the UNXFC keyword to check the expiration date for all files on your volume. When you use this keyword, you prevent any file from being scratched if the expiration date has not expired.

DSKPRP compares the system date to the expiration date for each file. Whenever the expiration date is greater than the system date, a message is displayed that indicates the expiration date has not expired (up to 10 files at a time can be displayed). Associated with this message, DSKPRP gives you the option to ignore the message and continue with the prep or cancel the prep. Normally, you would cancel the prep when the expiration date has not expired.

If you want expiration date validation to be performed, you must override the default by specifying UNXFC=Y.

9.3.13. Specifying the VTOC Address (VTOCB and VTOCE)

The VTOC is a directory containing the addresses of the files in your volume. You can indicate where you want the VTOC to reside by specifying six hexadecimal numbers representing the starting primary track address in cylinder/head format (cccchh) on the VTOCB keyword. The starting address must be less than the end VTOC address and must be at least one track in length; however, we recommend that one cylinder be allocated for the VTOC. When you omit the VTOCB keyword, DSKPRP automatically assigns the starting VTOC address for you. Table 9-1 shows the default VTOC starting addresses for both IPL and non-IPL disk volumes.

Table 9-1. Default Starting VTOC Addresses

SPERRY UNIVAC Disk	Non-IPL Volume	IPL Volume
8411	000001	006400
8414	000001	006400
8415	000001	00CA00
8416	000001	00CA00
8418	000001	00CA00
8424/25	000001	00CA00
8430	000001	00CA00
8433	000001	00CA00

NOTE:

On IPL volumes, the VTOC must start on a cylinder boundary.

After you have indicated the starting address of your VTOC, the next step is specifying the ending address. You can specify the hexadecimal numbers to indicate the cylinder and head address of the ending track on the VTOCE keyword. When you omit the VTOCE keyword, DSKPRP automatically assigns the ending VTOC address for you. Table 9-2 shows the VTOC ending addresses for both IPL and non-IPL disk volumes.

If you are prepping an 8415 with removable disk as an IPL volume, the disk prep routine assumes there will be a small VTOC. If you suspect the need for a large VTOC, due to amount of data or the number of files being stored, then you should specify the VTOCB and VTOCE keywords.

Table 9-2. Default Ending VTOC Addresses

SPERRY UNIVAC Disk	Non-IPL Volume	IPL Volume
8411	000009	006409
8414	000013	006413
8415 fixed	000002	00CA02
8415 removable	000001	00CA01
8416	000006	00CA06
8418	000006	00CA06
8414/25	000013	00CA13
8430	000012	00CA12
8433	000012	00CA12

9.4. FLAGGING DEFECTIVE TRACKS AUTOMATICALLY

The INSERT control statement is used to flag any known defective tracks. An alternate track is automatically assigned for each flagged track. As stated earlier, during the manufacturing process of 8415, 8416, and 8418 disk packs, any tracks detected as being defective are listed on the bottom of the plastic cover in *decimal* numbers. You must identify these defective tracks on INSERT control statements by using their *hexadecimal* equivalents. Only one defective track may be specified on each INSERT control statement. Whenever INSERT control statements containing defective track addresses are present, you must specify either the keyword INSRT=X or INSRT=Y.

The format of the INSERT control statement is:

```
1        10
____________________
INSERT   {cccchh}
         {NONE  }
```

where:

cccchh

Is the *hexadecimal* address of the track to be flagged in cylinder/head format. Note also, this specification must begin in column 10.

NONE

Indicates that there are no defective tracks.

If you are prepping an IDA disk pack and TRCON=N or TRCON=S is specified, INSRT=Y processing is automatically set, and your control stream must contain at least one INSERT control statement in it or else it will be considered invalid. This is true even if no defective tracks are listed on the plastic cover. Thus, if the pack you are prepping contains no defective tracks, you must use an INSERT control statement coded as follows:

```
1        10    16
____________________
INSERT   NONE
```

If you are prepping an 8411, 8414, 8424/25, 8430, or 8433 disk pack and there are no defective tracks listed on the plastic cover, INSRT=N is valid and the INSERT control statement is not required.

9.5. CREATING THE STANDARD VOLUME LABELS

The VOL1 label is the standard volume label in OS/3. It is used to identify your volume by a unique serial number and is also used to locate the address of the VTOC. You must specify a VOL1 statement everytime you do a prep.

The format of the VOL1 statement is:

1	11	42	51
VOL1	[r]	[aaaaaaaaaa]	

When you code the VOL1 statement, VOL1 must start in column 1. The next entry to specify is a volume security byte in column 11 (r). A 1 in this column implements the volume security check, while a 0 assumes that there is no security check. The default is zero. The last entry starts in column 42 and continues to column 51 for a name or address of the disk pack. Normally, this is an installation-supplied identifier. Figure 9–2 shows the format of a VOL1 label as it appears on a disk pack.

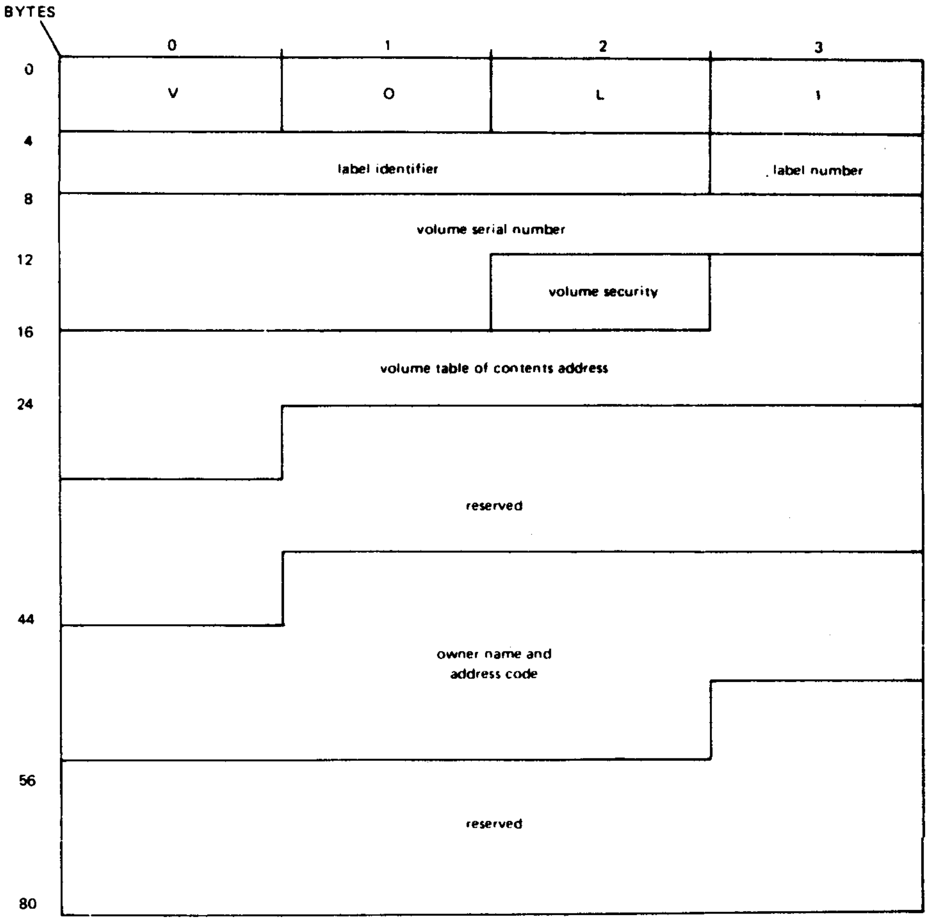


Figure 9–2. VOL1 Format

The VOL1 label is identified by the word VOL1 in the label identification field, which is the first field in the label. The VOL1 label is always written on cylinder 0, head 0, record 3.

9.6. PREPPING YOUR DISKETTE

Since a diskette is different in design from a disk pack, there are only four prep functions available:

- initializing the data records;
- initializing the data set labels (VTOC);
- changing the volume serial number; and
- checking the file expiration date.

Because the diskettes are factory prepped, the formatting of the diskettes is neither supported nor required by the prep routine.

Although no track condition table can be generated, an extensive surface analysis is performed during the manufacturing process. If any tracks are found to be defective, the manufacturer will automatically assign the defective tracks to alternate tracks.

9.7. SPECIFYING THE PREP OPTIONS FOR A DISKETTE

As was the case when prepping a disk pack, you choose the prep options by selecting one or more of the following keywords. The only keyword you must select is the SERNR keyword identifying the volume serial number of your diskette.

The format of the keywords is:

```
[,RPVOL={N}] [,FDATA={Y}] [,PARTL={N}] ,SERNR=volume serial number [,UNXFC={Y}]
[       {Y}] [       {N}] [       {V}]                              [       {N}]
```

9.7.1. Renumbering Your Diskette Volume Serial Number (RPVOL)

When you want to renumber your volume serial number, you specify RPVOL=Y and then insert your new volume serial number in the SERNR keyword.

9.7.2. Specifying File Allocation for DSL Diskettes (FDATA)

The disk prep routine automatically allocates the entire diskette (FDATA=Y) as being one file and names that file DATA. Therefore, if you are using the diskette as a work file, there is no need to allocate file space (using the // EXT statement). However, if you later decide to use the diskette in a multifile environment or use a different name, you must scratch the file before allocating a new one. You can scratch files by using either the PARTL=V keyword (9.7.3.) or via the SCR job control statement.

You also can specify the diskette to be made available for any file allocation (just as a disk) by specifying FDATA=N. Once prepped, you then allocate the required file space by using the // EXT job control statement with the BLK parameter or the interactive services ALLOCATE command.

NOTE:

You also can scratch files interactively via the interactive services ERASE command. See the interactive services commands and facilities, UP-8845 (current version) for details.

9.7.3. Changing Your Diskette Volume Serial Number and VTOC (PARTL)

If you want to specify a new volume serial number and VTOC, then you would specify the PARTL=V keyword. Your new volume serial number, which is specified on the SERNR keyword, is written in the VOL1 label. This keyword removes all the entries from the VTOC, thus making your diskette unusable. The use of the PARTL=V keyword also provides for a faster prep of your diskette since no surface analysis is performed.

9.7.4. Specifying Your Diskette Volume Serial Number (SERNR)

The volume serial number is six alphanumeric characters, which make up the serial number of the diskette being prepped, and may not contain any blank characters. Your volume serial number may already have been assigned to the diskette through a previous prep or it may specify a new serial number. In either case, the SERNR keyword must always be present in your prep control stream.

If SERNR is the only keyword used, the prep will automatically perform the following:

1. Reinitialize the data set labels (VTOC).

2. Change the volume serial number of the value of SERNR.

3. Write a prep pattern to all data records of the diskette, ensuring their availability.

9.7.5. Checking the File Expiration Date (UNXFC)

Use the UNXFC keyword to check the expiration date for all files on your volume. When you use this keyword, you prevent any file from being scratched if the expiration date has not expired.

DSKPRP compares the system date to the expiration date for each file. Whenever the expiration date is greater than the system date, a message is displayed, indicating the expiration date has not expired (up to 10 files at a time can be displayed). Associated with this message, DSKPRP gives you the option to ignore the message and continue with the prep or cancel the prep. Normally, you would cancel the prep when the expiration date has not expired.

If you omit the UNXFC keyword, no expiration date validation is performed.

9.8. INITIALIZING THE DATA SET LABELS

The data set labels written on track 00 of your diskette are similar to the VTOC of a disk pack. The data set labels are reserved for defining the name of a set of records and the addresses associated with the maximum space the set of records can occupy. You must specify a VOL1 statement every time you do a prep.

The format of the VOL1 statement is:

```
1
__________
VOL1
```

Notice that VOL1 is the only entry required.

9.9. EXECUTING THE DISK PREP (DSKPRP)

When executing DSKPRP to prep any kind of disk, there are two file names that must always be specified on the // LFD job statements. The file name for the printer must be PRNTR, while the file name of your disk pack must be DISKIN. No minimum or maximum main storage sizes should be specified on the // JOB card, as the minimum amount of main storage needed by DKSPRP is automatically allocated for the job by job control, provided any more than the minimum only wastes valuable main storage space. DSKPRP is not a program that uses main storage dynamically. Also, because DKSPRP only requires approximately 24K bytes of main storage to run, it is capable of running in a minimum system configuration.

NOTE:

You can prevent the inadvertent accessing of a disk that is being prepped by specifying the no-share (NS) parameter on the VOL statement in the prep job control stream. You can use the NOV parameter in place of NS, but the name specified in the VOL statement must be unique. The duplication of the name in another job stream or interactive request allows access to the disk.

In the following control streams, you will see some typical DSKPRP examples:

Example 1: Basic 8416 Prep

```
      1         10     16
     ------------------------------------------
 1.  | // JOB D16PREP
 2.  | // DVC 2Ø // LFD PRNTR
 3.  | // DVC 6Ø // VOL DS8416(NOV) // LFD DISKIN
 4.  | // EXEC DSKPRP
 5.  | /$
 6.  |    SERNR=DS8416,TRCON=N
 7.  | VOL1
 8.  | INSERT   NONE
 9.  | /*
10.  | /&
11.  | // FIN
```

Line 1 is your job control statement identifying your job D16PREP. Next, you specified the file name PRNTR for the printer; remember, this is the required file name for the printer. On line 3, you specified your device assignment set for the disk pack to be prepped. According to this assignment set, your disk pack is mounted on device number 60 (which is an IDA type device), your volume serial number is DS8416, and you are using the NOV option, indicating that no check is required for your volume serial number. The file name DISKIN must be specified on the // LFD job control statement since this is the required file name for the disk. The EXEC job control statement (line 4) calls the disk prep routine (DSKPRP) from $Y$LOD. The /$ job control statement (line 5) indicates the start of the prep data.

All your prep keywords and control statements must immmediately follow the /$ job control statement. In executing a basic IDA prep, there are only two keywords that must be specified: SERNR, indicating your volume serial number (DS8416), and TRCON=N, indicating a new track condition table is to be generated. Also, at least one INSERT control statement must be specified. When you specify only these two keywords, the following is assumed:

- The entire disk pack is being prepped.
- The alternate tracks are being tested.
- The disk pack is an IPL volume.
- A fast surface analysis is being performed.
- A track condition table is being generated and written to an alternate track and listed on the printer.
- At least one INSERT control statement is present in the control stream.
- No IL (COS) is written on the disk pack.
- The VTOC is written on cylinder 00CA.
- No verification is being performed.

The VOL1 card (line 7) must appear in most disk prep control streams and always follows the last specified keyword. The only exception to this rule is when you are using the assign alternate track (AAT) feature of the disk prep routine. The INSERT control statement (line 8) flags any known defective tracks. In this case, since there are no known defective tracks, you coded the word NONE starting in column 10. Whenever you prep an IDA disk pack with the keyword TRCON=N, the INSERT control statement must appear in the control stream. The /*, /&, and // FIN job control statements (line 9-11) terminate your job.

Example 2: Basic 8414 Prep

```
1        10    16
----------------------------
// JOB D14PREP
// DVC 2Ø // LFD PRNTR
// DVC 82 // VOL DS8414(NOV) // LFD DISKIN
// EXEC DSKPRP
/$
  SERNR=DS8414
VOL1
/*
/&
// FIN
```

When you execute a basic 8414 disk prep, the only keyword required is SERNR, indicating your volume serial number (DS8414). By specifying only this keyword, you assume:

- the entire disk pack is prepped;
- the alternate tracks are tested;
- the disk pack is an IPL volume;
- a fast surface analysis is performed;
- no INSERT control statements are present in the control stream;
- no IL (COS) is written on the disk pack;
- the VTOC is written on cylinder 0064;
- track condition table is input from disk and is written to an alternate track and listed on the printer; and
- no verification is performed.

Example 3: 8414 Prep with Selected Options

```
1        10     16
______________________________________________________
// JOB D14PREP
// DVC 2Ø // LFD PRNTR
// DVC 4Ø // LFD PUNCH
// DVC 82 // VOL DS8414 // LFD DISKIN
// EXEC DSKPRP
/$
    SERNR=DS8414,TRCON=C,TRKCT=X,INSRT=Y
   ILOPT=C
VOL1                                     DEPT 6944
  (Track condition table cards go here)
INSERT   ØØ72Ø5
INSERT   ØØ2111
/*
/$
   (IL(COS) cards go here)
/*
/&
// FIN
```

Here, you are prepping an 8414 disk pack whose serial number is DS8414. You specified the keyword TRCON=C, indicating that your track condition table is input from cards, while TRKCT=X indicates your new track condition table is output both to cards and the printer. The INSRT=Y keyword is specified, indicating that both INSERT control statements and a surface analysis is used to flag all defective tracks. Whenever you use the Y option with the INSERT keyword, there must be an INSERT control statement present in the control stream. The ILOPT=C keyword indicates that IL (COS) is input from cards rather than disk. As with any prep, the VOL1 statement must be present.

Example 4: Prepping a Select Portion of a Disk Pack

```
1       10   16
________________________________________
// JOB D3ØPREP
// DVC 2Ø // LFD PRNTR
// DVC 71 // VOL DS843Ø // LFD DISKIN
// EXEC DSKPRP
/$
   SERNR=DS843Ø,PTBEG=Ø114ØØ,PTEND=Ø11F12
   PREPT=2,INSRT=Y,VERFY=Y
VOL1
INSERT   Ø11EØØ
/*
/&
// FIN
```

Here, you are prepping only a portion of your disk pack (DS8430), namely, from cylinder/head 011400 to 011F12 as specified on the PTBEG and PTEND keywords. The PREPT=2 keyword indicates the degree of accuracy to be performed. The INSRT=Y keyword indicates that there are INSERT control statements present in the control stream to flag known defective tracks. The VERFY=Y keyword checks to make sure the area being prepped has not been allocated and is, therefore, free of any data. When you use this keyword, the PTBEG, PTEND, and SERNR keywords must also be specified, TRCON cannot be an N, TRKCT cannot be a P, and the alternate track area is automatically not tested. As mentioned in previous examples, the VOL1 must appear following the last specified keyword. The INSERT control statement follows the VOL1 card and flags the specified track. Remember, this is used in conjunction with the INSRT=Y keyword.

Example 5: Basic 8413 Prep

```
// JOB D13PREP
// DVC 2Ø // LFD PRNTR
// DVC 13Ø // VOL DS8413(NOV) // LFD DISKIN
// EXEC DSKPRP
/$
   SERNR=DS8413
VOL1
/*
/&
// FIN
```

When you execute the basic 8413 diskette prep, the only keyword required is SERNR, indicating your volume serial number (DS8413).

Example 6: Changing Volume Serial Number without Prepping

```
1        1Ø    16
------------------------------------------
// JOB D13CHNG
// DVC 2Ø // LFD PRNTR
// DVC 13Ø // VOL DS8413(NOV) // LFD DISKIN
// EXEC DSKPRP
/$
   SERNR=DUMMY1,RPVOL=Y
VOL1
/*
/&
// FIN
```

Here, you are changing the volume serial number from DS8413 to DUMMY1 while leaving the rest of the diskette unchanged.

9.10. ERROR PROCESSING

The prep routine always terminates normally, even if errors are detected during the execution of your prep. Error messages, however, are listed on the printer. At prep completion, the hexadecimal value of your program switch indicator (UPSI) byte is listed on the printer. The UPSI byte indicates the severity of the errors detected.

9.11. PREP CANNED JOB CONTROL STREAMS

The following canned job control streams provide you with a more convenient method of performing certain prep functions without specifying the parameters and job control statements normally required to run them. They are:

- ADD1KCOS

  Adds 1K COS to $Y$SRC on SYSRES.

- ADD2KCOS

  Adds 2K COS to $Y$SRC on SYSRES.

- ADD3KCOS

  Adds 2K of fast COS to $Y$SRC on SYSRES.

- CGV/CHGVSN

  Changes the volume serial number on a previously prepped disk.

- PRP1KCOS

  Repositions 1K COS from $Y$SRC on SYSRES for COS–IPL.

- PREP2KCOS

  Repositions 2K COS from $Y$SRC on SYSRES for COS–IPL.

- PRP3KCOS

  Repositions 2K of fast COS from $Y$SRC on SYSRES for COS–IPL.

- SETREL

  Preps and allocates RELEASE/SYSRES files.

- COPYREL

  Copies selected RELEASE/SYSRES files.

These functions are initiated from the system console by keying in their associated job control stream name.

9.11.1. Add COS to $Y$SRC on SYSRES (ADDnnCOS)

The ADDnnCOS routine copies COS microcode from cards into the $Y$SRC file on your SYSRES pack. The value you give for *nn* (1K, 2K, 3K) indicates which COS module is to be filed. OS/3 disk intialization routines use the COS module to prepare an initial program load (IPL) disk.

To execute ADDnnCOS, place the COS card deck, ended by a // FIN job control statement, in the card reader. Then key in from the system console the ADDnnCOS command that corresponds to the COS module you're loading in:

```
RU ⎧ADD1KCOS⎫
   ⎨ADD2KCOS⎬
   ⎩ADD3KCOS⎭
```

The COS card deck in the reader must correspond to the command entered.

The COS module placed in $Y$SRC cannot be loaded into the system until PRPnnCOS has been run.

9.11.2. Change a Volume Serial Number

The CGV/CHGVSN system utility routine changes the volume serial number on a previously prepped disk pack. The new volume serial number replaces the old volume serial number in the VOL1 record. The required parameters are supplied either on cards in the card reader or as operands with a command keyed in at the system console.

NOTE:

Extreme care should be used if changing the volume serial number of a SYSRUN, SYSRES, or spooling disk pack. Specification of a duplicate vsn may not cause a duplication error message to be issued but will severely impact the system. If a duplicate vsn is inadvertently assigned, the entire system must be reinitialized (IPL).

9.11.2.1. System Console Keyin (CGV)

When the parameters are entered as operands with a command keyed in at the system console, the command has the following format:

```
RV CGV,,O=old-vsn,N=new-vsn,T=disc-type
```

Keyword Parameters:

`O=old-vsn`
Specifies the old volume serial number of the previously prepped disk pack.

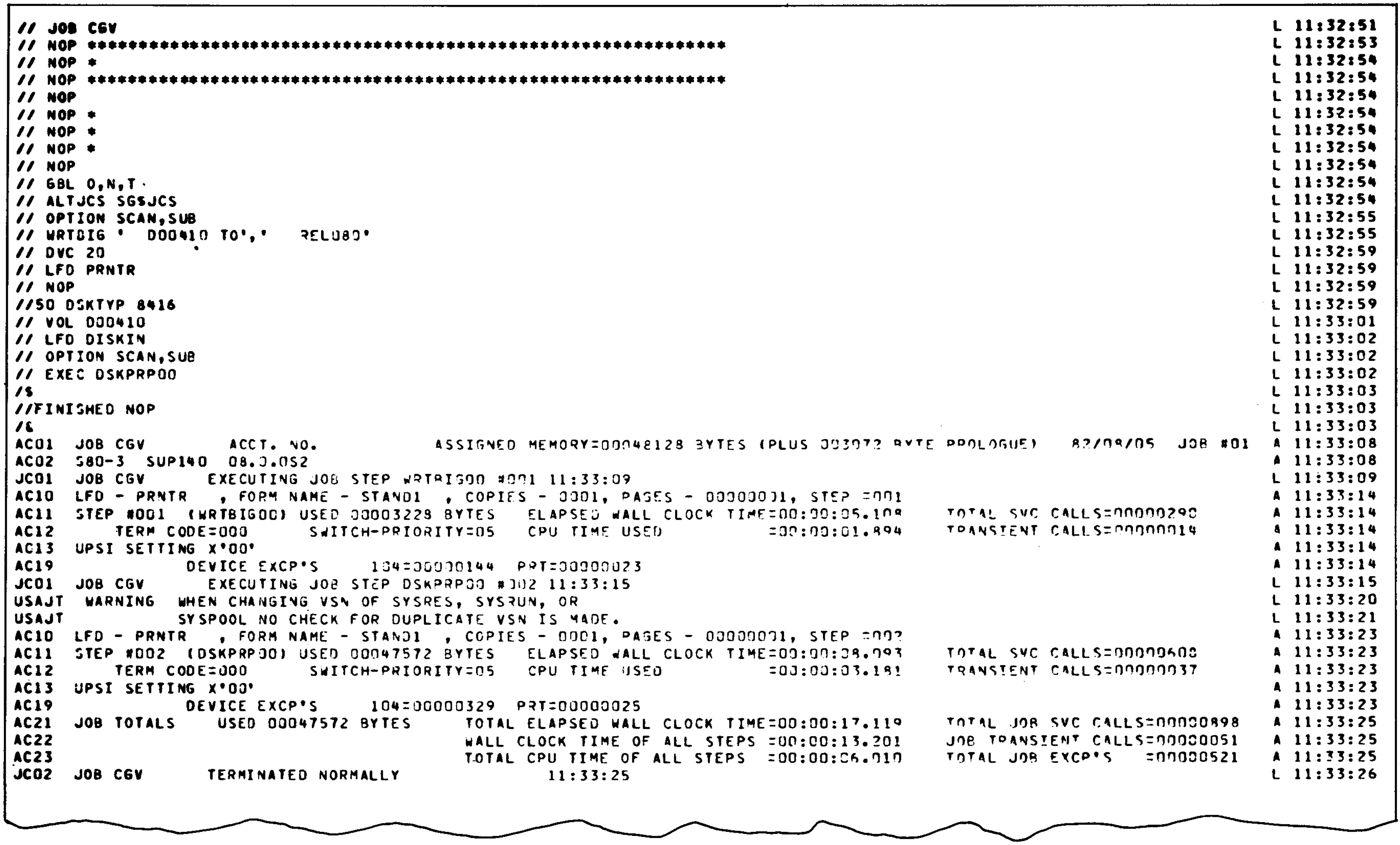
`N=new-vsn`
Specifies the new volume serial number to be assigned to the prepped disk pack.

`T=disc-type`
Specifies the type of disk subsystem being used. The values may be:

Value	Disk Type
11	8411
13	8413
14	8414
15F	8415 fixed
15R	8415 removable
16	8416
18A	8418 low density
18B	8418 high density
24	8424
25	8425
30	8430
33	8433

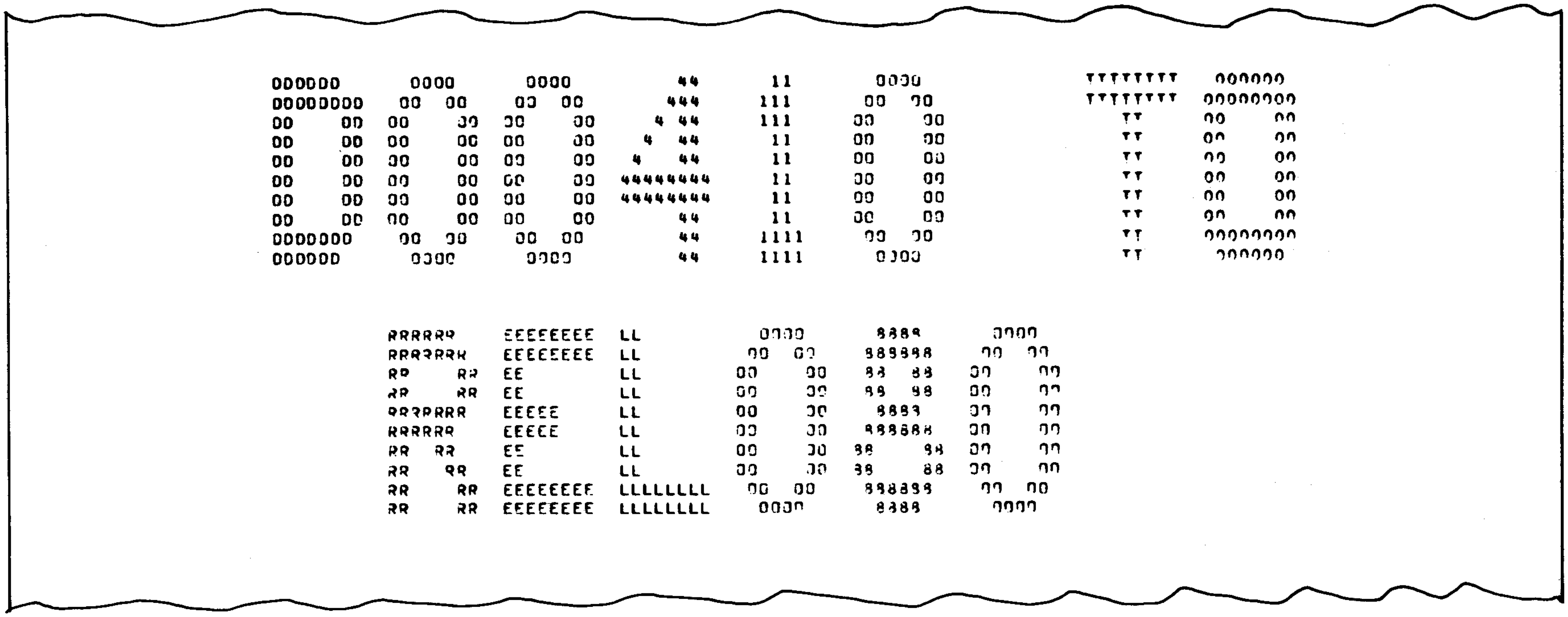
After you enter the RV CGV command, the CGV job executes the DSKPRP routine to change your volume serial number. When the DSKPRP routine has completed, you receive a printed listing like the one in Figure 9–3. It shows the job control for and information about the CGV job, the old and new volume serial numbers in large print, and the parameters that CGV includes for your DSKPRP routine. You can keep this listing as a record of the way your disk is prepped. After you change a volume serial number, make sure you physically change the label on the outside of the disk so that the label always shows the correct volume serial number. For this sample listing, your command would have been:

```
RV CGV,,O=DØØ41Ø,N=RELØ8Ø,T=16
```



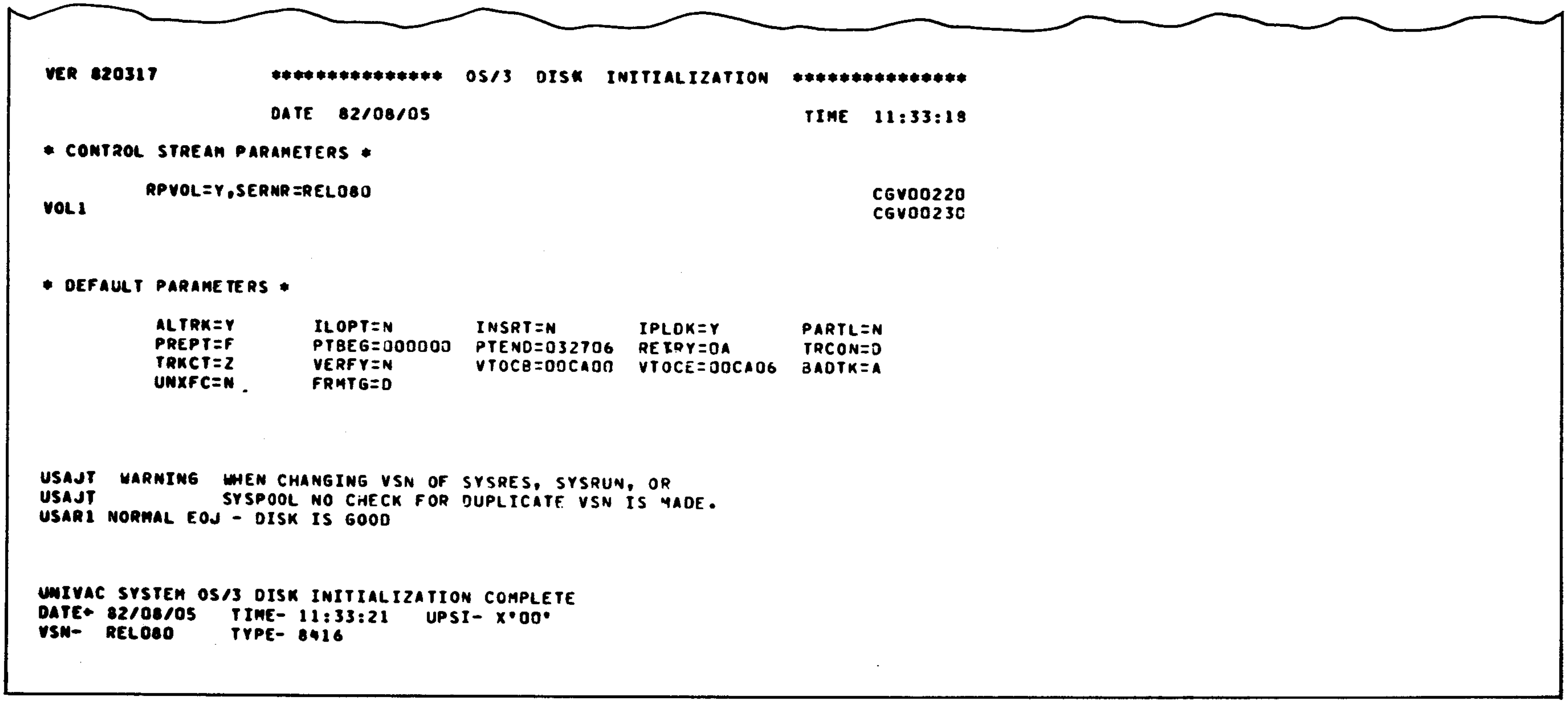
```
// JOB CGV                                                                                                              L 11:32:51
// NOP ****************************************************                                                              L 11:32:53
// NOP *                                                                                                                 L 11:32:54
// NOP ****************************************************                                                              L 11:32:54
// NOP                                                                                                                   L 11:32:54
// NOP *                                                                                                                 L 11:32:54
// NOP *                                                                                                                 L 11:32:54
// NOP *                                                                                                                 L 11:32:54
// NOP                                                                                                                   L 11:32:54
// GBL 0,N,T                                                                                                             L 11:32:54
// ALTJCS SG$JCS                                                                                                         L 11:32:54
// OPTION SCAN,SUB                                                                                                       L 11:32:55
// WRTBIG '  D00410 TO','   REL080'                                                                                      L 11:32:55
// DVC 20                                                                                                                L 11:32:59
// LFD PRNTR                                                                                                             L 11:32:59
// NOP                                                                                                                   L 11:32:59
//SO DSKTYP 8416                                                                                                         L 11:32:59
// VOL D00410                                                                                                            L 11:33:01
// LFD DISKIN                                                                                                            L 11:33:02
// OPTION SCAN,SUB                                                                                                       L 11:33:02
// EXEC DSKPRP00                                                                                                         L 11:33:02
/$                                                                                                                       L 11:33:03
//FINISHED NOP                                                                                                           L 11:33:03
/&                                                                                                                       L 11:33:03
AC01  JOB CGV       ACCT. NO.           ASSIGNED MEMORY=00048128 BYTES (PLUS 003072 BYTE PROLOGUE)   82/09/05  JOB #01  A 11:33:08
AC02  S80-3  SUP140  08.0.052                                                                                            A 11:33:08
JC01  JOB CGV     EXECUTING JOB STEP WRTBIG00 #001 11:33:09                                                              L 11:33:09
AC10  LFD - PRNTR   , FORM NAME - STAND1  , COPIES - 0001, PAGES - 00000001, STEP =001                                   A 11:33:14
AC11  STEP #001 (WRTBIG00) USED 00003228 BYTES   ELAPSED WALL CLOCK TIME=00:00:05.108     TOTAL SVC CALLS=00000290       A 11:33:14
AC12      TERM CODE=000     SWITCH-PRIORITY=05   CPU TIME USED          =00:00:01.894     TRANSIENT CALLS=00000014       A 11:33:14
AC13  UPSI SETTING X'00'                                                                                                 A 11:33:14
AC19            DEVICE EXCP'S     104=00000144  PRT=00000023                                                             A 11:33:14
JC01  JOB CGV     EXECUTING JOB STEP DSKPRP00 #002 11:33:15                                                              L 11:33:15
USAJT  WARNING  WHEN CHANGING VSN OF SYSRES, SYSRUN, OR                                                                  L 11:33:20
USAJT           SYSPOOL NO CHECK FOR DUPLICATE VSN IS MADE.                                                              L 11:33:21
AC10  LFD - PRNTR   , FORM NAME - STAND1  , COPIES - 0001, PAGES - 00000001, STEP =002                                   A 11:33:23
AC11  STEP #002 (DSKPRP00) USED 00047572 BYTES   ELAPSED WALL CLOCK TIME=00:00:08.093     TOTAL SVC CALLS=00000600       A 11:33:23
AC12      TERM CODE=000     SWITCH-PRIORITY=05   CPU TIME USED          =00:00:03.181     TRANSIENT CALLS=00000037       A 11:33:23
AC13  UPSI SETTING X'00'                                                                                                 A 11:33:23
AC19            DEVICE EXCP'S     104=00000329  PRT=00000025                                                             A 11:33:23
AC21  JOB TOTALS   USED 00047572 BYTES     TOTAL ELAPSED WALL CLOCK TIME=00:00:17.119     TOTAL JOB SVC CALLS=00000898   A 11:33:25
AC22                                       WALL CLOCK TIME OF ALL STEPS =00:00:13.201     JOB TRANSIENT CALLS=00000051   A 11:33:25
AC23                                       TOTAL CPU TIME OF ALL STEPS  =00:00:06.010     TOTAL JOB EXCP'S   =00000521   A 11:33:25
JC02  JOB CGV     TERMINATED NORMALLY            11:33:25                                                                L 11:33:26
```

Figure 9–3. Sample Listing for CGV Job (Part 1 of 3)



```
DDDDDD      0000      0000        44     11      0000          TTTTTTTT   OOOOOO
DDDDDDDD   00  00    00  00      444    111     00  00         TTTTTTTT  OOOOOOOO
DD    DD  00    00  00    00    4 44    111    00    00           TT     OO    OO
DD    DD  00    00  00    00   4  44     11    00    00           TT     OO    OO
DD    DD  00    00  00    00  4   44     11    00    00           TT     OO    OO
DD    DD  00    00  00    00  44444444   11    00    00           TT     OO    OO
DD    DD  00    00  00    00  44444444   11    00    00           TT     OO    OO
DD    DD  00    00  00    00      44     11    00    00           TT     OO    OO
DDDDDDDD   00  00    00  00       44    1111    00  00            TT     OOOOOOOO
DDDDDD      0000      0000        44    1111     0000             TT      OOOOOO

          RRRRRRR   EEEEEEEE  LL         0000     8888     0000
          RRRRRRRR  EEEEEEEE  LL        00  00   888888   00  00
          RR    RR  EE        LL       00    00  88  88  00    00
          RR    RR  EE        LL       00    00  88  88  00    00
          RRRRRRRR  EEEEE     LL       00    00   8888   00    00
          RRRRRRR   EEEEE     LL       00    00  888888  00    00
          RR  RR    EE        LL       00    00 88    88 00    00
          RR   RR   EE        LL       00    00 88    88 00    00
          RR    RR  EEEEEEEE  LLLLLLLL  00  00   888888   00  00
          RR    RR  EEEEEEEE  LLLLLLLL   0000     8888     0000
```

Figure 9–3. Sample Listing for CGV Job (Part 2 of 3)



```
VER 820317          ***************  OS/3  DISK  INITIALIZATION  ***************

                    DATE  82/08/05                                   TIME  11:33:18

* CONTROL STREAM PARAMETERS *

        RPVOL=Y,SERNR=REL080                                         CGV00220
VOL1                                                                 CGV00230


* DEFAULT PARAMETERS *

         ALTRK=Y       ILOPT=N        INSRT=N        IPLOK=Y        PARTL=N
         PREPT=F       PTBEG=000000   PTEND=032706   RETRY=0A       TRCON=D
         TRKCT=Z       VERFY=N        VTOCB=00CA00   VTOCE=00CA06   BADTK=A
         UNXFC=N       FRMTG=D



USAJT  WARNING  WHEN CHANGING VSN OF SYSRES, SYSRUN, OR
USAJT           SYSPOOL NO CHECK FOR DUPLICATE VSN IS MADE.
USAR1 NORMAL EOJ - DISK IS GOOD


UNIVAC SYSTEM OS/3 DISK INITIALIZATION COMPLETE
DATE- 82/08/05   TIME- 11:33:21   UPSI- X'00'
VSN-  REL080     TYPE- 8416
```

Figure 9–3. Sample Listing for CGV Job (Part 3 of 3)

9.11.2.2. Card Input (CHGVSN)

When the parameters are input via cards in the card reader, the following command is keyed in at the system console:

```
RU CHGVSN
```

In addition, the following cards must be in the card reader.

```
1        10    16
//OLDVSN JSET  'old-vsn'
//NEWVSN JSET  'new-vsn'
//TYPE   JSET  'disc-type'
// FIN
```

where:

'old-vsn'
Specifies the old volume serial number of the previously prepped disk pack.

'new-vsn'
Specifies the new volume serial number to be assigned to the prepped disk pack.

'disc-type'
Specifies the type of disk subsystem being used. The values may be:

Value	Disk Type
8411	8411
8413	8413
8414	8414
8415F	8415 fixed
8415R	8415 removable
8416	8416
8418A	8418 low density
8418B	8418 high density
8424	8424
8425	8425
8430	8430
8433	8433

9.11.3. Reposition COS from $Y$SRC on SYSRES for COS–IPL (PRPnnCOS)

The PRPnnCOS routine repositions the COS module from the $Y$SRC file on SYSRES to the COS–IPL area on a disk you specify. The value you give for *nn* (1K, 2K, 3K) indicates the type of COS module to be repositioned. PRPnnCOS repositions the COS module from $Y$SRC to an area immediately following the VOL1 label on the disk; from this position, it can then be loaded into the system. This routine must be run before you try to load your COS module into the system.

The required parameters are supplied either on cards in the card reader or as operands with a command keyed in at the system console.

9.11.3.1. Card Input

When the parameters are input via cards in the card reader, one of the following commands is keyed in at the system console:

```
RU (PRP1KCOS)
   {PRP2KCOS}
   (PRP3KCOS)
```

For either command, the following cards must be in the card reader:

```
1        10    16
_________________________
//VSN    JSET  'vsn'
//TYPE   JSET  'disc-type'
// FIN
```

where:

'vsn'
Specifies the volume serial number of the disk being prepared with COS.

'disk-type'
Specifies the type of disk subsystem being used. The values may be:

Value	Disk Type
8411	8411
8414	8414
8415F	8415 fixed
8415R	8415 removable
8416	8416
8418A	8418 low density
8418B	8418 high density
8424	8424
8425	8425
8430	8430
8433	8433

9.11.3.2. System Console Keyin

When the parameters are entered as operands with either command keyed in at the system console, the command has the following format:

```
RV (PRP1KCOS),,V=vsn,T=disc-type
   {PRP2KCOS}
   (PRP3KCOS)
```

Keyword Parameters:

```
V=vsn
```

Specifies the volume serial number of the disk being prepared with COS.

```
T=disc-type
```

Specifies the type of disk subsystem being used. The values are as follows:

Value	Disk Type
11	8411
14	8414
15F	8415 fixed
15R	8415 removable
16	8416
18A	8418 low density
18B	8418 high density
24	8424
25	8425
30	8430
33	8433

9.11.4. Prep and Allocate RELEASE/SYSRES Files (SETREL)

The SETREL system utility prepares a disk volume for use as a RELEASE or SYSRES volume by executing a disk initialization (prep) run and allocating the standard SYSRES files. The library size is allocated according to the type of volume required. Disk may be initialized with or without full surface analysis. The required parameters are supplied either as operands with a command keyed in at the system console or on cards in the card reader. For the 8415 removable disk pack, SETREL must be run by using card input.

NOTE:

SETREL preps the disk pack with 1K COS. If 2K COS is required, you must run the canned job control stream PRP2KCOS immediately after running SETREL. This also applies to installations requiring 2K fast COS, in which case PRP3KCOS would be run.

9.11.4.1. System Console Keyin

When the disk being prepared is to be a copy of the current SYSRES volume, a keyin at the system console may be used to initiate a SETREL run.

The console keyin as the following format.

```
RV SETREL,,V=vsn,T=disc-type,P=prep-type
```

Keyword Parameters:

V=vsn

Specifies the volume serial number of the output disk pack being prepped. For a full prep, this may be a new volume serial number to be assigned to the disk. Otherwise, it must be the same as the current volume serial number. If omitted, the card reader is activated to read the parameters from cards.

T=disc-type

Specifies the type of disk subsystem being used. The values are as follows:

Value	Disk Type
11	8411
14	8414
15F	8415 fixed
16	8416
18	8418 high and low density
24	8424
25	8425
30	8430
33	8433

P=prep-type

Specifies the type of prep to be performed. The codes may be:

Code	Meaning
N	No prep performed, assign files only
F	Full prep, with surface analysis
P	Partial prep, without surface analysis

If omitted, P is assumed.

When P=F is specified, the INSERT△△△NONE or INSERT△△△cccchh card must be present in the card reader. This card must be followed by a // FIN card.

If you selected the no prep option (N), it is assumed that your disk pack was previously prepped by using SETREL.

9.11.4.2. Card Input

An optional way of preparing a SYSRES volume is to run SETREL and specify the required parameters on parameter cards. Note that, using this method, all files are release volume size files.

To run SETREL in this mode, the following command is keyed in at the system console:

```
RU SETREL
```

The following cards must be in the card reader, depending on the type of prep being requested and the type of disk subsystem being used. The values of the parameters are given following the listings of the card input.

- Partial prep without surface analysis

```
1         10     16
_______________________________
//PREP    JSET   'Ø'
//VSNO    JSET   'vsn'
//TYPE    JSET   'disc-type'
// FIN
```

- Full prep with surface analysis

```
//PREP    JSET   '1'
//VSNO    JSET   'vsn'
//TYPE    JSET   'disc-type'
// FIN
INSERT    ⎧NONE  ⎫
          ⎩cccchh⎭
// FIN
```

The values that may be assigned to the various parameters are as follows:

'vsn'
Specifies the volume serial number of the output disk pack.

'disk-type'
Specifies the type of disk subsystem being used. The values are:

Value	Disk Type
8411	8411
8414	8414
8415F	8415 fixed
8415R	8415 removable
8416	8416
8418	8418 high and low density
8424	8424
8425	8425
8430	8430
8433	8433

NONE
Indicates that there are no defective cylinders and tracks on the disk pack. To prep a selector channel device disk pack with no defective tracks, omit the INSERT△△△NONE statement but still use both // FIN statements.

cccchh

Specifies the hexadecimal address of the defective cylinder and track as listed on the disk pack.

If you are using SETREL to prepare an 8415 removable device, you must run the control stream twice, first for the primary disk and then again for the secondary disk. On the first run, //VSNO (as shown in the examples) is replaced by //VSN1 and with //VSN2 in the second run. The respective volume serial numbers for the primary and secondary disks are entered on the appropriate cards.

9.11.4.3. Diagnostics and Error Messages

If unrecoverable errors occur during the prepping of the volume, a message is displayed on the console and the job is terminated immediately. If other errors are encountered, a warning message is displayed and the job continues processing.

NOTES:

1. *The console message ''PREP TERMINATED WITH ERRORS'' is displayed if the UPSI byte is set (40_{16} or 80_{16}).*

2. *The U response to PIOCS console error messages during the execution of DSKPRP does not necessarily cause immediate termination. In several cases, especially during track analysis, the U response will result in the assignment of an alternate track.*

9.11.5. Copy System/Release Files (COPYREL)

After successfully using SETREL to prep and allocate your disk volume containing the system and release libraries, copy those libraries with COPYREL. Only the files listed in Table 9–3 can be copied. Execute COPYREL by using the following console keyin:

```
RV△COPYREL,,V=vsn,T=disk-type,[,S=first-file][,E=last-file]
```

where:

V=vsn

Specifies the volume serial number of the output disk being copied.

T=disk-type

Specifies the type of disk subsystem being used. The values are as follows:

Value	Disk Type
17	8417
19	8419

`S=first-file`

Specifies the code identifying the first file that is to be copied. Table 9–3 shows the order that COPYREL copies the system files and shows the codes for each system file. If you omit the S keyword, COPYREL starts copying at $Y$SRC.

`E=last file`

Specifies the code for the last file to be copied. (See Table 9–3.) If you omit the E keyword, copying ends at $Y$TRANA.

Table 9–3. COPYREL Copy Order

Copy Order	Code	File Name
1	S	$Y$SRC
2	O	$Y$OBJ
3	L	$Y$LOD
4	M	$Y$MAC
5	J	$Y$JCS
6	G	SG$JCS
7	SGMAC	SG$MAC
8	SGOBJ	SG$OBJ
9	SGLOD	SG$LOD
10	SCLOD	$Y$SCLOD
11	MIC	$Y$MIC
12	IVP	IVPLIB
13	SMCFILE	SMCFILE
14	FMT	$Y$FMT
15	SAVE	$Y$SAVE
16	DIALOG	$Y$DIALOG
17	SDF	$Y$SDF
18	HELP	$Y$HELP
19	T	$Y$TRAN
20	A	$Y$TRANA

NOTE:

→ *More detailed discussion of both SETREL and COPYREL are found in the current version of the system installation user guide/programmer reference, UP-8074.*

10. Assign Alternate Track (AAT)

10.1. AAT CAPABILITY

The AAT routine is used to conditionally or unconditionally assign an alternate track for a suspected defective track. The suspected defective track may be either a primary data track or another alternate track. Assigned conditionally means that an alternate track is assigned only after a surface analysis is performed and the analysis shows defects in the track. Assigned unconditionally means that no surface analysis is done and the track is assigned regardless of its condition.

For conditional assignment, an available alternate track is assigned by AAT and the contents of the suspected defective track are copied to it, one record at a time. During the copy procedure, any errors detected are listed on the printer. These errors can be corrected in subsequent runs of AAT by using the record updating facility, ASUPD. After the records are copied to the alternate track, a surface analysis is performed on the primary track. If the track is found to be defective, it is marked as unusable and the alternate track is permanently assigned. However, if the primary track is found to be usable, then all the records on the alternate track are copied back to the primary track and the alternate track is again made available for use.

For an absolute or unconditional assignment, the process is similar. The suspected defective track is copied to an available alternate track on a record-by-record basis. Any defective records are printed. At this point, the reassignment is made permanent and no surface analysis is performed on the suspected track. Remember that a suspected defective track can be an alternate track as well as a primary track. If it is an alternate track, another alternate track is reassigned as the alternate track for the original primary data track. The suspected alternate track is then assigned to itself so that it cannot be used again.

NOTE:

For all IDA disks, the maximum number of defective primary tracks supported per head address is 7. Defective tracks in excess of 7 will not have alternate tracks assigned to them and the disk will be unusable.

10.2. INTERFACING WITH DSKPRP

The AAT function is a feature built into DSKPRP; therefore, you must execute DSKPRP (specify DSKPRP on the EXEC job control statement) to use the AAT capability. Before you assign any alternate tracks, your disk must have been previously prepped.

10.3. SPECIFYING AAT OPTIONS

There are five keywords associated with the assigning of any alternate tracks. You must specify both the ASGTK and SERNR keywords; the remaining keywords are optional and have default values. The format of the AAT keywords is:

$$\mathrm{ASGTK{=}ccccch h}\ \left[,\mathrm{ASGPR}{=}\begin{Bmatrix}\mathrm{A}\\ \colorbox{lightgray}{E}\end{Bmatrix}\right]$$

$$\left[,\mathrm{ASURF}{=}\begin{Bmatrix}\mathrm{N}\\ \colorbox{lightgray}{S}\end{Bmatrix}\right]\ \left[,\mathrm{ASUPD}{=}\begin{Bmatrix}\colorbox{lightgray}{N}\\ \mathrm{Y}\end{Bmatrix}\right]$$

$$,\mathrm{SERNR{=}volume\ serial\ number}$$

10.3.1. Specifying Any Suspected Defective Tracks (ASGTK)

Since the AAT capability is a function of the disk prep routine, you must specify ASGTK to indicate the function to be performed. You specify the suspected defective track, in hexadecimal, in cylinder/head format (cccchh). Remember, you cannot assign an alternate track to an active SYSRES pack. If you need to assign alternate tracks to your SYSRES, another SYSRES or the current release volume must be used as the operating system. If you omit ASGTK, the disk prep routine is executed; however, the disk prep routine will be terminated because there is no VOL1 statement present in the control system. VOL1 cards cannot be used in an AAT run.

10.3.2. Printing Your Records (ASGPR)

When you read the records from the primary track to the alternate track, any records detected in error are listed on the printer automatically by assuming the ASGPR keyword with the E option. However, if you need to print all the records being read, you specify the A option. If you use the A option in conjunction with the ASUPD=Y keyword, no records will be printed. In other words, ASGPR=A has no effect when you specify ASUPD=Y.

10.3.3. Testing the Alternate Track (ASURF)

If you recall, you can assign the alternate track conditionally, that is, only after ensuring that the primary track is defective, or unconditionally, meaning no surface analysis is performed. When you omit ASURF or specify the S option, alternate tracks are assigned conditionally. If you are certain the primary track is unusable, specify ASURF=N, and no surface analysis will be performed.

10.3.4. Patching or Modifying Existing Records (ASUPD)

If any errors were detected in the reading of your records from the primary track for writing on the alternate track, they will be listed on the printer. From this listing, you keypunch the missing information into update records. These update records then patch the incomplete record. However, if the missing information cannot be determined, you must recreate your file.

Whenever update records are present in your control stream, you must specify the keyword parameter ASUPD=Y. If there are no update records present, the default ASUPD=N is assumed.

At least two cards are required for each update record; one to identify the record and field to be updated and another to contain the correction data. The number of the record to be updated must always be specified as a hexadecimal number and must always begin in column 1. The record numbers on a given track appear on the listing of errors found during the recovery of data during a previous AAT run. The location and type of field being corrected must also be identified by specifying either KEY=([d],l) or DATA=([d],l), where d represents a displacement value and l, a length value. The displacement value, relative to 0, may be up to four hexadecimal characters long, or may be omitted, to indicate that the patch is to be made beginning with the first character in the identified record field. The length value must be specified and can be from one to four hexadecimal characters long. Both of these parameters cannot be coded on the same card, as the correction data for a key field must immediately follow the KEY keyword parameter and the correction data for a data field must immediately follow the DATA keyword. Any one of them, however, can be coded on the card containing the number of the record (rn parameter) to be patched. If coded with the rn parameter, a comma must separate the two specifications. If coded alone, the keyword must begin in column 1. Thus, the format of an identifier update record could be illustrated as:

$$\text{rn} \quad \textit{or} \quad \text{rn,}\left\{\begin{array}{l}\text{KEY=}\left(\left[\left\{\begin{array}{c}\text{d}\\ \text{Ø}\end{array}\right\}\right]\text{,}l\right)\\ \text{DATA=}\left(\left[\left\{\begin{array}{c}\text{d}\\ \text{Ø}\end{array}\right\}\right]\text{,}l\right)\end{array}\right\} \quad \textit{or} \quad \left\{\begin{array}{l}\text{KEY=}\left(\left[\left\{\begin{array}{c}\text{d}\\ \text{Ø}\end{array}\right\}\right]\text{,}l\right)\\ \text{DATA=}\left(\left[\left\{\begin{array}{c}\text{d}\\ \text{Ø}\end{array}\right\}\right]\text{,}l\right)\end{array}\right\}$$

The actual data that is to be written must be submitted on a separate card apart from the update record. The actual data must be in hexadecimal format (0–9, A–F) and start in column 1. The length of the data must equal the length value specified in the KEY or DATA parameters. No embedded blanks are permitted and as many cards as needed may be used. The first blank character found in the data cards indicates the end of the record.

It should also be noted that only one KEY or one DATA keyword may appear in any one set of update records; when updating records on an IDA unit disk pack, the KEY parameter cannot be specified; and when updating records on a selector unit disk pack, only existing key and data fields may be updated, the count field cannot be altered, and the length of key and data fields must remain unchanged. When ASUPD=Y is specified, the ASGPR and ASURF keywords are ignored.

A typical example of an update control stream could look like:

```
1        10    16
_________________________________________
              .
              .  job control statements
/$
  ASGTK=ØØ1AØ6,SERNR=DSPØØ1
  ASUPD=Y
/*
/$
1B,KEY=(3,2)
FØF1
DATA=(3C,A)
C1C2C3C4C5C6C7C8C9D1
/*
/&
```

Notice the KEY parameter was specified on the same line as the record number (1B), while the DATA keyword is on a separate line. The length of your key field (2) equals the length of the actual data (FØF1).

In this example, record 1B is being updated. There is a displacement value of 60 bytes ($3C_{16}$) and a length of 10 bytes (A_{16}). The new data being written is specified on a separate card and must be in hexadecimal format. If more than one record is to be updated, an additional data set is needed for each record. See 10.4 for more typical examples of using update records.

10.3.5. Specifying Your Volume Serial Number (SERNR)

The volume serial number is six alphanumeric characters which make up the serial number of the disk volume being used. The SERNR keyword must always be present in your assign alternate track control stream.

10.4. EXECUTING AAT

When executing the assign alternate track routine, the same file names that were required in the disk prep routine must be specified here. Namely, PRNTR must be specified on the LFD job control statement for assigning the printer while DISKIN must be specified for assigning the disk pack. In the following control streams, you will see some typical examples of assigning alternate tracks.

Example 1:

```
1        10    16
________________________________________
// JOB AATRACK
// DVC 20 // LFD PRNTR
// DVC 51 // VOL DSP028  // LFD DISKIN
// EXEC DSKPRP
/$
   ASGTK=00C106,SERNR=DSP028
/*
/&
// FIN
```

Here is the basic AAT control stream. Your suspected defective track is on cylinder 00C1, head 06, on disk pack DSP028. You are omitting the ASGPR, ASURF, and ASUPD keywords, thus using the default options. Any records in error are listed on the printer (ASGPR=E), a surface analysis is performed on the track (ASURF=S), and there are no update records present in your control stream.

Example 2:

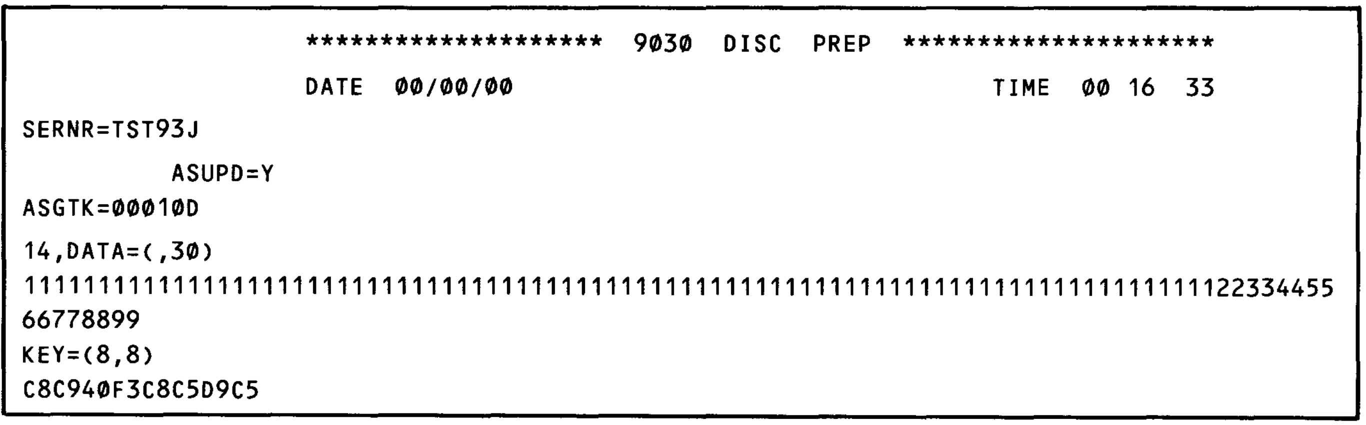
```
// JOB UPDATE
// DVC 20 // LFD PRNTR
// DVC 51 // VOL TST93J // LFD DISKIN
// EXEC DSKPRP
/$
   SERNR=TST93J
   ASUPD=Y
   ASGTK=00010D
/*
/$
14,DATA=(,32)
111111111111111111111111111111111111111111111111111111111111111111111111122334455
66778899
KEY=(8,8)
C8C940E3C8C5D9C5
/*
/&
// FIN
```

Here, you are using update records to patch specific key and data fields on your disk pack. The volume serial number of your disk pack is TST93J, which is equated with the VOL job control statement. Since your control stream contains update records, you specified ASUPD=Y. As you can see, all the keyword parameters are part of the first data set. The record data set contains the update records themselves. The record number in error is 14_{16} (decimal 20). The data being corrected has a length of 32_{16} (decimal 50) bytes and is shown on the next two cards. You are also patching the key field, which has a displacement and length of 8_{16} bytes.

NOTE:

If more than one data card is required, the previous card must have all 80 columns used (example 2).

Figure 10-1 shows the output generated from the control stream.



```
                    ********************  9030  DISC  PREP  ********************
                    DATE  00/00/00                                 TIME  00 16  33
SERNR=TST93J
          ASUPD=Y
ASGTK=00010D
14,DATA=(,30)
111111111111111111111111111111111111111111111111111111111111111111111111111111111122334455
66778899
KEY=(8,8)
C8C940F3C8C5D9C5
```

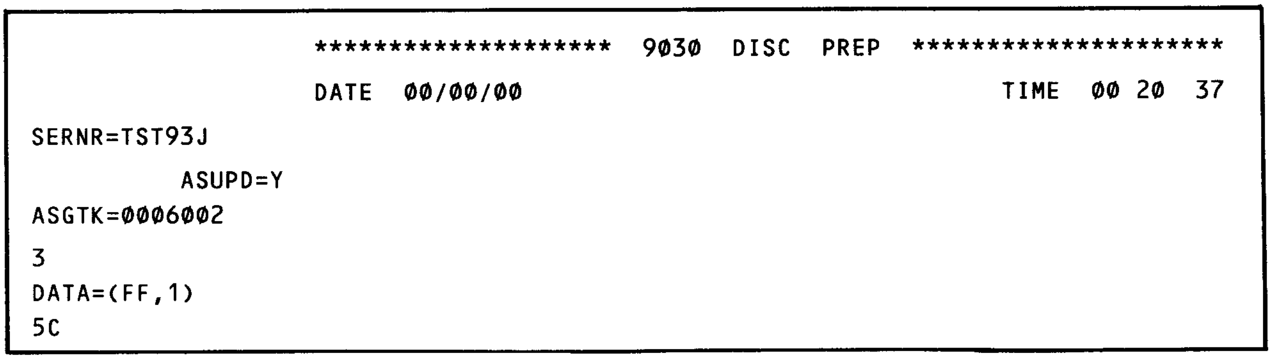
Figure 10-1. AAT Using Update Records

Example 3:

```
1        10     16                                                   72
________________________________________________________________________
// JOB UPDATE
// DVC 20 // LFD PRNTR
// DVC 51 // VOL TST93J // LFD DISKIN
// EXEC DSKPRP
/$
   SERNR=TST93J
     ASUPD=Y
   ASGTK=006002
/*
/$
3
DATA=(FF,1)
5C
/*
/&
// FIN
```

Here, you are again using update records but you omitted the ASGPR keyword. The last byte of sector 3 is being changed to 5C. Since you are using the update function, the record is not printed.

Figure 10-2 shows the output produced by this control stream.



```
                      ********************  9Ø3Ø  DISC  PREP  *********************
                      DATE  ØØ/ØØ/ØØ                                 TIME  ØØ 2Ø  37
SERNR=TST93J
          ASUPD=Y
ASGTK=ØØØ6ØØ2
3
DATA=(FF,1)
5C
```

Figure 10-2. AAT Using Update Records without Printing

11. Tape Prep (TPREP)

11.1. PREPARING YOUR TAPE FOR EXECUTION

Magnetic tapes are shipped blank. You must prepare (prep) these tapes before using them in the SPERRY UNIVAC Operating System/3 (OS/3). The tape prep utility (TPREP) can prep up to 36 tapes in one job step, and will prep them for use with or without block numbers, depending on whether your system is configured to support tape block numbering. If your system is configured to support tape block numbering, you have the option to prep tapes for use without block numbers. You cannot, however, prep a tape for use with block numbers if your system is not configured to do so. You submit all the required information for tape prepping through job control statements. TPREP uses the prep facilities of data management to prep your tapes. The various tape record formats that are generated by the prep facility can be found in the data management user guide, UP-8068 (current version). After your tape has been prepped, it is rewound to load point.

11.2. TAPE PREP CODING INSTRUCTIONS

As mentioned earlier, all the information required for tape prepping is submitted through job control statements. The following job control statements are used for tape 1 prepping:

- // DVC job control statement

  You must specify a logical unit number for the tape being prepped. The range of logical unit numbers is from 90 to 127. An example of the // DVC job control statement used when prepping tapes is shown in the following example:

```
1       10    16
________________
// DVC 9Ø
```

■ // VOL job control statement

You must specify a unique volume serial number for every tape being prepped. The volume serial number can be any alphanumeric character string from one to six characters long, other than the word SCRTCH. Immediately following the last character of your volume serial number, the character string (PREP) must appear. An example of the // VOL job control statement used when prepping tapes is shown in the following example:

```
// VOL DSPØ28(PREP)
```

Notice how the parentheses are coded as part of the parameter.

To prevent your tapes from being prepped for use with block numbers when your system is supporting block numbering, you must include an N parameter in your // VOL statement as follows:

```
1         10    16
__________________________
// VOL    N,DSPØ28(PREP)
```

■ // LBL job control statement

You specify the // LBL job control statement only when you want to assign a file identifier to a tape volume you are prepping. You can assign the same label to multiple volumes to prepare for a multivolume file. If you are familiar with the // LBL job control statement, then you would have recognized that there is a file sequence number parameter used for numbering of files in a multifile tape volume. However, TPREP ignores the file sequence number parameter. An example of the // LBL job control statement is:

```
// LBL MASTERFILE
```

■ // LFD job control statement

You must specify a unique file name for each tape being prepped. The // LFD file name must be in the form TAPExy: under x is any alphanumeric character A through Z or 0 through 9; under y is the character A for ASCII mode or blank for EBCDIC mode. An example of the LFD job control statement used when prepping tapes in both EBCDIC and ASCII modes is:

```
// LFD TAPE1
// LFD TAPE1A
```

- // EXEC job control statement

  You must specify the program TPREP in your // EXEC job control statement to start the prepping of one or more tapes using TPREP. The // EXEC job control statement calls the tape prep utility from $Y$LOD. You code the // EXEC job control statement as:

  ```
  // EXEC TPREP
  ```

Figure 11-1 shows two tape prep examples using TPREP; example a shows the job control statement for prepping of six tapes using two magnetic tape drives, while example b shows the job control statements for prepping six tapes using six different tape drives, with the last three tapes being prepped in ASCII mode. All other tapes are prepped in EBCDIC mode.

```
1        10    16
_____________________________________________
// JOB TAPEPREP
// DVC 9Ø
// VOL TPØØØ1(PREP),TPØØØ2(PREP),TPØØØ3(PREP)
// LFD TAPE1
// DVC 91
// VOL TPØØØ4(PREP),TPØØØ5(PREP),TPØØØ6(PREP)
// LFD TAPE2
// EXEC TPREP
/&
// FIN
```

Example a.

```
// JOB TAPEPREP
// DVC 9Ø // VOL TAPEØ1(PREP) // LFD TAPE1
// DVC 91 // VOL TAPEØ2(PREP) // LFD TAPE2
// DVC 92 // VOL TAPEØ3(PREP) // LFD TAPE3
// DVC 93 // VOL TAPEØ4(PREP) // LFD TAPEAA
// DVC 94 // VOL TAPEØ5(PREP) // LFD TAPEBA
// DVC 95 // VOL TAPEØ6(PREP) // LFD TAPECA
// EXEC TPREP
/&
// FIN
```

Example b.

Figure 11-1. Control Stream Coding Required to Prep Multiple Tape Volumes in a Single Job Step

All messages are displayed on the system console and are written to the communications output printer (COP) if available. Figure 11-2 shows a typical COP listing.

```
14 JC01 JOB TAPEPREP EXECUTING JOB STEP TPREP000 #001

15 TP01 TAPE SP0001 (LFDNAME=TAPE1A) PREPPED IN ASCII , NO BKNO

16 JC02 JOB TAPEPREP TERMINATED NORMALLY
```

Figure 11-2. A Typical COP Listing Showing TPREP Messages

The following message is displayed for each tape successfully prepped:

TP01 TAPE vsn (LFDNAME=filename) PREPPED IN $\left\{\begin{matrix}\text{ASCII}\\ \text{EBCDIC}\end{matrix}\right\}$, $\left\{\begin{matrix}\text{NO}\\ \text{WITH}\end{matrix}\right\}$BKNO

The absence of this message for any of the tapes specified in your job or job step indicates that the tape has not been prepped because of incorrect job control specifications.

If you are creating a file through data management, you may do tape prepping and file creation in one job step. See the data management user guide, UP-8068 (current version).

12. System Utility Copy Routines

12.1. SECTORED DISK COPYING (SU$C16)

You use SU$C16 to make multiple copies with or without verification of any IDA type disk to a similar disk pack and regardless of the disk pack's content (data or libraries). You can execute SU$C16 either in a batch environment or interactively from a terminal. See 12.3 for interactive processing of SU$C16.

12.1.1. SU$C16 Organization

SU$C16 is made up of seven parameters. All these parameters have default values and are associated with the // PARAM statement. However, if you choose all the defaults, you must still have the // PARAM statement present in your control stream.

The format of SU$C16 is:

```
// PARAM [COPY={n}] [,VEFY={NO }] [,PRNT={NO }] [,OVEF={NO }]
               {1}          {YES}          {YES}          {YES}

         [,BGAD=(ccc16   )]  [,EDAD=(ccc16                       )]
         [      {ccch16  }]  [      (ccch16                      )]
         [      {ccchrr16}]  [      (ccchrr16                    )]
         [      (000001  )]  [      {193628 for 8416 and 8418 low}]
                             [      (327628 for 8418 high        )]
         [,UNXF={NO }]       [      (193128 for 8415R            )]
         [      {YES}]       [      (527228 for 8415F            )]
```

You use the COPY parameter to indicate how many copies of your input disk pack are to be made. The value of n may be from 1 to 7. If you omit COPY, one copy is made.

You use the VEFY parameter to verify your new output. If VEFY=YES is specified, each copy is verified against the input. If you omit VEFY, then no verification is performed.

You use the PRNT parameter to print any records found to be in error. If an error is detected, both the input and output records are listed with its corresponding output disk address (ccchrr), in hexadecimal. You would normally choose PRNT=YES if you had chosen VEFY=YES. If you omit PRNT, only fatal errors are written on the system console.

You use the BGAD parameter to indicate the starting address of your copy in hexadecimal. The address you specify may be a cylinder number (ccc_{16}), a cylinder and head number ($ccch_{16}$), or a cylinder, head, and record number ($ccchrr_{16}$). If you omit BGAD, then the beginning address is cylinder 000, head 0, and record 01.

You use the OVEF parameter to indicate that a verify-only operation is to be performed. No copy will be made. If you omit OVEF, one or more copies will be made, with or without specification, depending on other parameter specifications. OVEF may be used in conjunction with BGAD and EDAD.

You use the EDAD parameter to indicate the ending address of your copy in hexadecimal. As explained in the BGAD parameter, the address you can specify can be either ccc_{16}, $ccch_{16}$, or $ccchrr_{16}$. If you omit EDAD, the default values for the hexadecimal ending addresses are as follows:

Disk Subsystem	Cylinder	Head	Record
8416	193	6	28
8418 low	193	6	28
8418 high	327	6	28
8415 removable	193	1	28
8415 fixed	327	2	28

By using both the BGAD and EDAD parameters, you can copy only one record from your input pack to your output pack. It is worth remembering that the input starting and ending addresses are the same as your output addresses, and any information contained in your specified output area is destroyed and the new information is written in that area. For example, if you were copying from cylinders 10–20, any information residing in cylinders 10–20 in your output pack is destroyed and the new information is written in that area.

You use the UNXF parameter to check for the file expiration date of any file on the output disks. If the file date has not expired, a message is displayed that asks you either to continue processing or to skip the file and continue to the next file (providing you are processing more than one file). Using the UNXF parameter eliminates the time and expense of keeping outdated files. It also eliminates the overlaying of files that should not be destroyed.

12.1.2. SU$C16 Interfacing with Job Control

The file name DISCIN must be specified on the // LFD job control statement for your input disk pack. The file name DISCOT must be specified on the // LFD job control statement for your output disk pack. If you are copying onto more than one disk pack, DISCOT01 through DISCOT06 must be specified for each disk pack being copied. If you are running in a minimum system (32K), you must specify X'5800' in the main storage size parameter (MIN) of the // JOB job control statement. Only copy functions are executed in minimum systems; no verification can be performed.

12.1.3. Executing SU$C16

In a multijobbing environment, the use of SU$C16 can cause space allocated to a file to be lost; files may be added to or portions deleted, or a disk pack may be incorrectly copied. These situations can occur when another job is updating the VTOC of a disk pack at the same time SU$C16 is copying that VTOC to another disk pack, or when SU$C16 is copying a file while another job is extending or scratching that file. Therefore, the job executing SU$C16 must be the only job running when the I/O device is either SYSRUN or SYSRES. If the device is not SYSRUN or SYSRES, it must be set to nonsharable by the operator. This ensures that an exact copy will be made by SU$C16.

The following control streams show some typical examples of using SU$C16:

Example 1:

```
1       10    16                                                  72
--------------------------------------------------------------------------
 // JOB STDCOPY
 // DVC 6Ø  // VOL DSPØØ1 // LFD DISCIN
 // DVC 61  // VOL DSPØØ2 // LFD DISCOT
 // EXEC SU$C16
 // PARAM
 /&
 // FIN
```

Notice that, even though you did not specify any parameters, the // PARAM statement still appears in your control stream. This is the basic 8416 disk copy. The printer is not specified; therefore, no printer output is available.

Example 2:

```
1       10    16                                                  72
--------------------------------------------------------------------------
 // JOB VPCOPY
 // DVC 5Ø // VOL DSPØØ1 // LFD DISCIN
 // DVC 51 // VOL DSPØØ2 // LFD DISCOT
 // DVC 2Ø // LFD PRNTR
 // EXEC SU$C16
 // PARAM VEFY=YES,PRNT=YES
 /&
 // FIN
```

Since you are using the print option, the device assignment set for the printer must be specified. The file name PRNTR must be specified on the // LFD job control statement. The verification and printing are specified. If you do not specify a printer, your job is executed but only fatal errors detected are displayed on the system console.

Example 3:

```
1           10      16                                                     72
// JOB COPY7
// DVC 5Ø // VOL DSPØØ1 // LFD DISCIN
// DVC 51 // VOL DSPØØ2 // LFD DISCOT
// DVC 52 // VOL DSPØØ3 // LFD DISCOTØ1
// DVC 53 // VOL DSPØØ4 // LFD DISCOTØ2
// DVC 54 // VOL DSPØØ5 // LFD DISCOTØ3
// DVC 55 // VOL DSPØØ6 // LFD DISCOTØ4
// DVC 56 // VOL DSPØØ7 // LFD DISCOTØ5
// DVC 57 // VOL DSPØØ8 // LFD DISCOTØ6
// DVC 2Ø // LFD PRNTR
// EXEC SU$C16
// PARAM COPY=7,VEFY=YES,PRNT=YES,UNXF=YES
/&
// FIN
```

Here, you are copying your input disk pack to seven output packs with verification, printing, and file expiration date checking. Note the required file names specified on the // LFD job control statements.

Example 4:

```
1           10      16                                                     72
// JOB COPYTRK
// DVC 2Ø // PRNTR
// DVC 6Ø // VOL DSPØØ1 // LFD DISCIN
// DVC 61 // VOL DSPØØ2 // LFD DISCOT
// DVC 62 // VOL DSPØØ3 // LFD DISCOTØ1
// EXEC SU$C16
// PARAM COPY=2,BGAD=ØØ54,EDAD=ØØ54,VEFY=YES
/&
// FIN
```

This example makes two copies of one track and verifies the copies. There is no printer output and verification of the output is written to the console.

Example 5:

```
1           10      16                                                     72
// JOB COPYCYL
// DVC 6Ø // VOL DSPØØ1 // LFD DISCIN
// DVC 61 // VOL DSPØØ2 // LFD DISCOT
// EXEC SU$C16
// PARAM BGAD=Ø7A,EDAD=Ø7A
/&
// FIN
```

This example makes one copy of one cylinder. There is no printer output and no verification.

12.2. NONSECTORED DISK COPYING (SU$CSL)

You use SU$CSL to copy an 8411, 8414, 8424/25, 8430, or 8433 disk pack to a similar device. As was the case with SU$C16, you can copy as little as one track or up to an entire volume regardless of your disk pack's contents. You can execute SU$CSL either in a batch environment or interactively from a terminal. See 12.4 for interactive processing of SU$CSL.

12.2.1. SU$CSL Organization

SU$CSL is made up of seven parameters. All these parameters have default values and are associated with the // PARAM statement as was the case with SU$C16. However, if you choose all the defaults, you must still have the // PARAM statement present in your control stream. SU$CSL checks the file control block to determine the type of disk pack being copied.

The format of SU$CSL is:

```
// PARAM [ COPY={n}] [,VEFY={NO }] [,PRNT={NO } [,OVEF={NO }]
               {1}           {YES}          {YES}          {YES}
         [,BGAD={ccchh16}] [,EDAD=(ccchh16                )]
                {ØØØØØ  }         (ØC7Ø9 for 8411         )
                                  {ØC713 for 8414         }
                                  {18F13 for 8424/25      }
         [,UNXF={NO }]            (19312 for 843Ø         )
                {YES}             (32712 for 8433         )
```

You use the COPY parameter to indicate how many copies of your input disk pack are to be made. The value of n may be from 1 to 7. If you omit COPY, one copy is made.

You use the VEFY parameter to verify your new output. If you are making more than one copy, each copy is verified against the input. If you omit VEFY, no verfication is performed.

You use the PRNT parameter to print any records found to be in error. If an error is detected, both the input and output record is listed with its corresponding output disk address (ccchhrr), in hexadecimal. You would normally choose PRNT=YES if you had chosen VEFY=YES. If you omit PRNT, only fatal errors are written on the system console.

You use the OVEF parameter to indicate that a verify-only operation is to be performed. No copy will be made. If you omit OVEF, one or more copies will be made, with or without verification, depending on other parameter specifications. OVEF may be used in conjunction with BGAD and EDAD.

You use the BGAD parameter to indicate the starting address of the first track to be copied, in hexadecimal. The address you specify must be the cylinder and head number (ccchh). If you omit BGAD, the beginning address is cylinder 000, head 00.

You use the EDAD parameter to indicate the ending address of the last track to be copied in hexadecimal. As explained in the BGAD parameter, the address you specify must be ccchh. If you omit EDAD, the values will be defaulted for each device as shown in the format.

Since the programming logic of SU$CSL is similar to SU$C16, the input addresses are the same as your output addresses and any information contained in your specified output area is destroyed and the new information is written in that area.

You use the UNXF parameter to check for the file expiration date. If the file date has not expired, a message is displayed that asks you either to continue processing or to skip the file and continue to the next file (providing you are processing more than one file). Using the UNXF parameter eliminates the time and expense of keeping outdated files. It also eliminates the overlaying of files that should not be destroyed.

12.2.2. SU$CSL Interfacing with Job Control

The file name DISCIN must be specified on the // LFD job control statement for your input disk pack. The file name DISCOT must be specified on the // LFD job control statement for your output disk pack. If you are copying onto more than one disk pack, then DISCOTO1 through DISCOTO6 must be specified for each additional disk pack being copied.

If you are running in a minimum system (32K) then you must specify X'5C00' in the minimum main storage size parameter (MIN) on the job control statement; however, like SU$C16, only the copy function is permitted under the minimum system using the 8430 or 8433 disk pack. Both copy and verification can be used on 8411 and 8414 disk packs.

12.2.3. Executing SU$CSL

In a multijobbing environment, the use of SU$CSL can cause space allocated to a file to be lost, files may be added to or portions deleted, or a disk pack may be incorrectly copied. These situations can occur when another job is updating the VTOC of a disk pack at the same time SU$CSL is copying that VTOC to another disk pack, or when SU$CSL is copying a file while another job is extending or scratching that file. Therefore, the job executing SU$CSL must be the only job running when the I/O device is either SYSRUN or SYSRES. If the device is not SYSRUN or SYSRES, it must be set to nonsharable by the operator. This ensures that an exact copy will be made by SU$CSL.

The following control streams show some typical examples of using SU$CSL.

Example 1:

```
1        10    16                                                     72
________________________________________________________________________
// JOB COPY8411
// DVC 5Ø // VOL DSPØØ1 // LFD DISCIN
// DVC 51 // VOL DSPØØ2 // LFD DISCOT
// EXEC SU$CSL
// PARAM
/&
// FIN
```

Here is the basic 8411 disk copy. There is no printing or verification. Your starting address is cylinder 000, head 00, while your ending address is cylinder 0C7, head 9.

Example 2:

```
// JOB COPY8411,,,5CØØ
// DVC 5Ø // VOL DSPØØ1 // LFD DISCIN
// DVC 51 // VOL DSPØØ2 // LFD DISCOT
// DVC 2Ø // LFD PRNTR
// EXEC SU$CSL
// PARAM PRNT=YES,VEFY=YES,EDAD=Ø5Ø
/&
// FIN
```

Since you are running in a minimum system (32K), our job control statement has the minimum main storage size specified. You are only copying up to cylinder X'50', as indicated by EDAD=050. You are also using the verification and print options, making sure the information written is the same as that read.

Example 3:

```
1        10    16                                                     72
________________________________________________________________________
// JOB COPY843Ø
// DVC 5Ø // VOL DSPØØ1 // LFD DISCIN
// DVC 51 // VOL DSPØØ2 // LFD DISCOT
// DVC 52 // VOL DSPØØ3 // LFD DISCOTØ1
// DVC 2Ø // LFD PRNTR
// EXEC SU$CSL
// PARAM COPY=2,VEFY=YES,PRNT=YES,BGAD=Ø1Ø,EDAD=1ØØ,UNXF=YES
/&
// FIN
```

Here, you are using all the parameters. You are making two copies by using the 8430 disk packs. Your copy starts at cylinder X'10' and ends at cylinder X'100' with verification and printing of any errors being detected, as well as file expiration date checking.

12.3. EXECUTING SU$C16 IN AN INTERACTIVE ENVIRONMENT

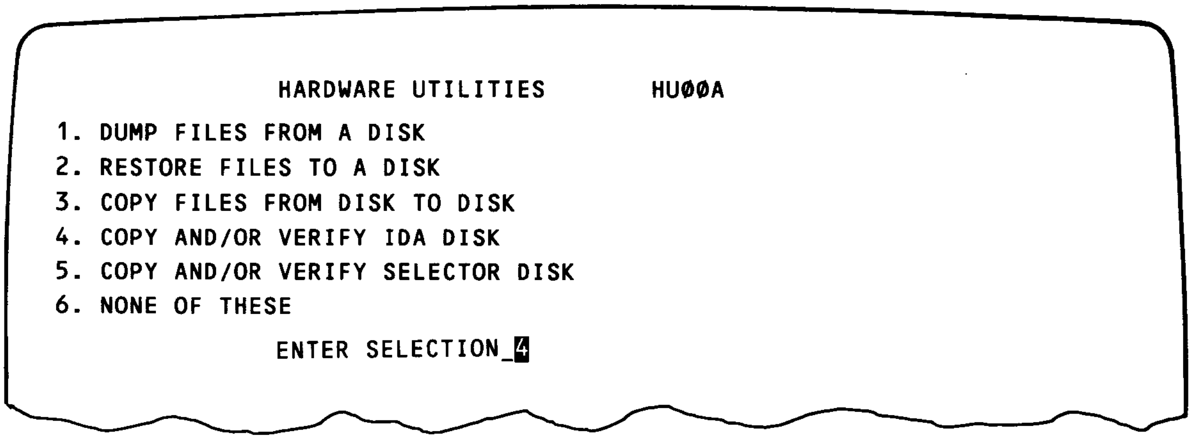
Interactive processing of the SU$C16 routine (and SU$CSL as well) consists of a question and answer session (dialog) using a UNISCOPE 100, UNISCOPE 200, or UTS 400 terminal. The procedures for executing the SU$C16 routine are the same, regardless of which type of terminal you use except for a few adjustments necessitated by differences in the terminal keyboards. With the UTS 400 terminal, for example, use the XMIT key to transmit work screens and function keys 13 and 14 to obtain and exit help screens. For UNISCOPE 100 and 200 terminals, use the TRANSMIT key to transmit work screens. To use function key 13 or 14, you must simulate it in the following manner. Press the MESSAGE WAITING key, then alphabetic F, then the pound sign (#), and then the number 13 or 14.

The following example shows the SU$C16 routine for a UTS 400 terminal. When using a UNISCOPE 100 or 200 terminal, you must adjust the instructions in the manner just discussed.

Help screens are provided for all work screens. To obtain help for any screen, press function key 13. To return to the work screen, press function key 14 or the XMIT key. If multiple help screens are being displayed, press the XMIT key after each help screen. When the last help screen is displayed, press the XMIT key or function key 14; the current work screen is redisplayed.

When you use dialog, your entries are checked and any errors detected are blinked on the screen. For example, invalid entries such as unsupported device types and misspellings of YES or NO are blinked.

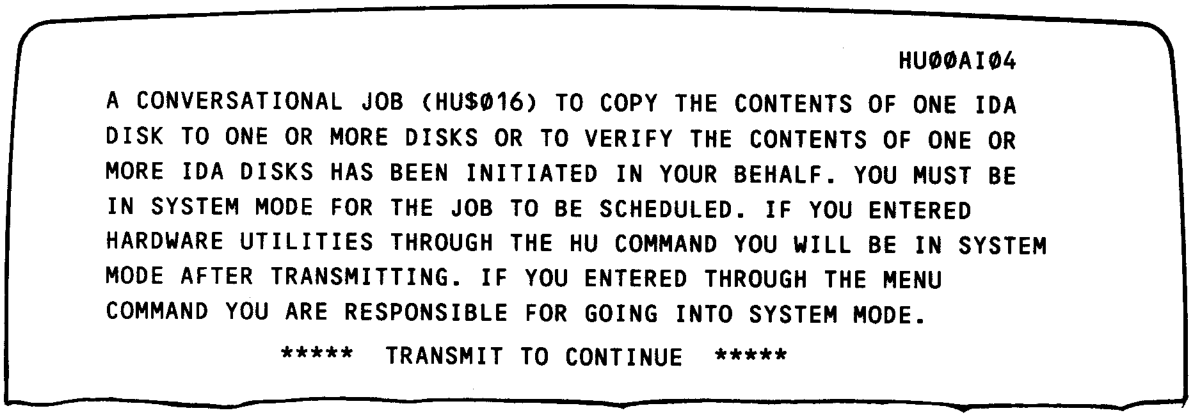
Before executing SU$C16, you must have successfully logged onto the system via the LOGON command. After logging on, key in HU in system mode and press the XMIT key. After pressing the XMIT key, the HARDWARE UTILITIES menu screen appears:



```
                HARDWARE UTILITIES       HUØØA
  1. DUMP FILES FROM A DISK
  2. RESTORE FILES TO A DISK
  3. COPY FILES FROM DISK TO DISK
  4. COPY AND/OR VERIFY IDA DISK
  5. COPY AND/OR VERIFY SELECTOR DISK
  6. NONE OF THESE
                ENTER SELECTION_4
```

Looking at the menu screen, select 4 to start execution of the SU$C16 routine.

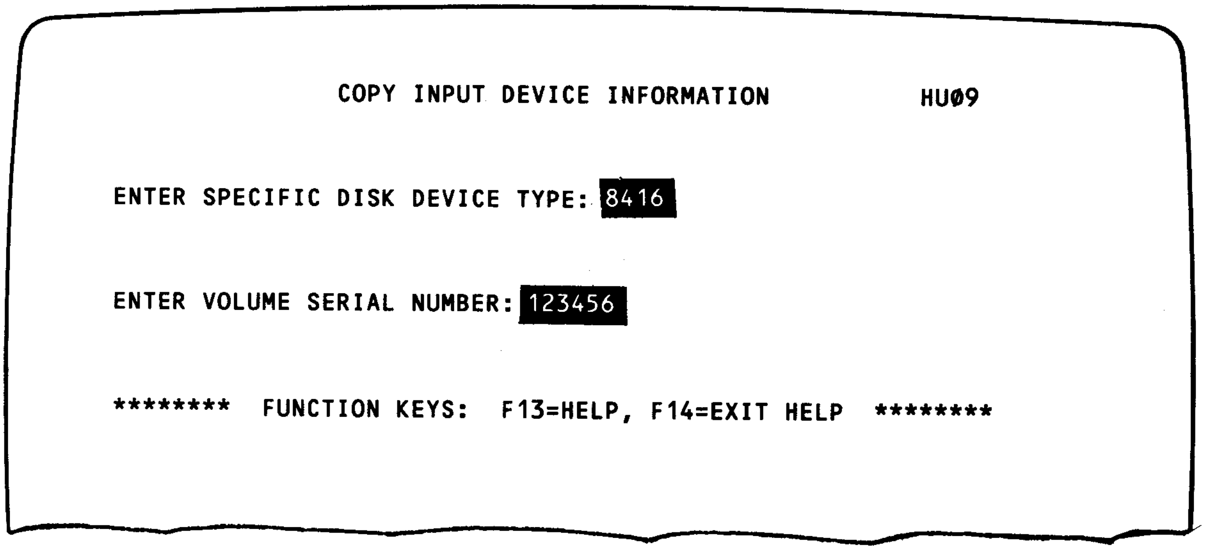
After you press the XMIT key, the following information screen is displayed.



```
                                                        HUØØAIØ4
A CONVERSATIONAL JOB (HU$Ø16) TO COPY THE CONTENTS OF ONE IDA
DISK TO ONE OR MORE DISKS OR TO VERIFY THE CONTENTS OF ONE OR
MORE IDA DISKS HAS BEEN INITIATED IN YOUR BEHALF. YOU MUST BE
IN SYSTEM MODE FOR THE JOB TO BE SCHEDULED. IF YOU ENTERED
HARDWARE UTILITIES THROUGH THE HU COMMAND YOU WILL BE IN SYSTEM
MODE AFTER TRANSMITTING. IF YOU ENTERED THROUGH THE MENU
COMMAND YOU ARE RESPONSIBLE FOR GOING INTO SYSTEM MODE.
          *****  TRANSMIT TO CONTINUE  *****
```

After you press the XMIT key, the following screen appears:

Screen 1 (HU09): Indicating the Specific Disk Device Type and the Volume Serial Number



```
               COPY INPUT DEVICE INFORMATION        HUØ9

ENTER SPECIFIC DISK DEVICE TYPE: 8416

ENTER VOLUME SERIAL NUMBER: 123456

********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

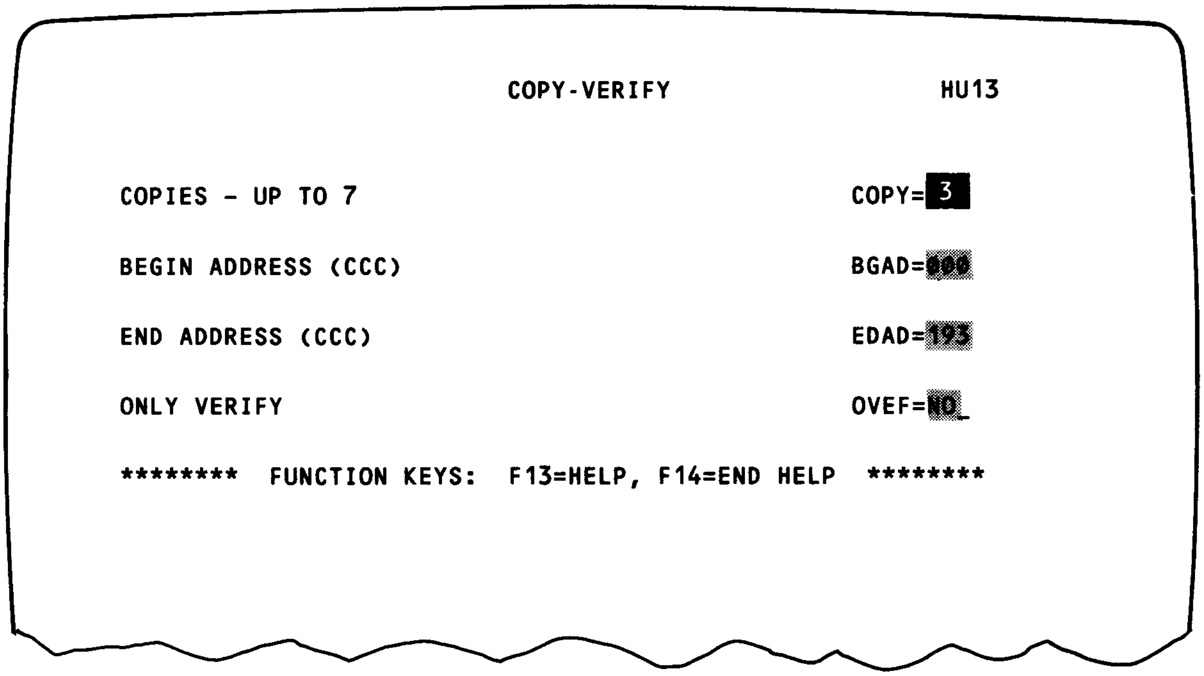
Looking at screen 1, we entered 8416 for the disk device type and 123456 for the volume serial number. The device type entered on this screen must be one of the IDA type disks: 8415, 8416, and 8418.

NOTE:

Regarding the screens, all the default values are shaded while the user entries are shown in reverse lettering (white characters on black background).

Since help is not required, press the XMIT key and the next screen appears:

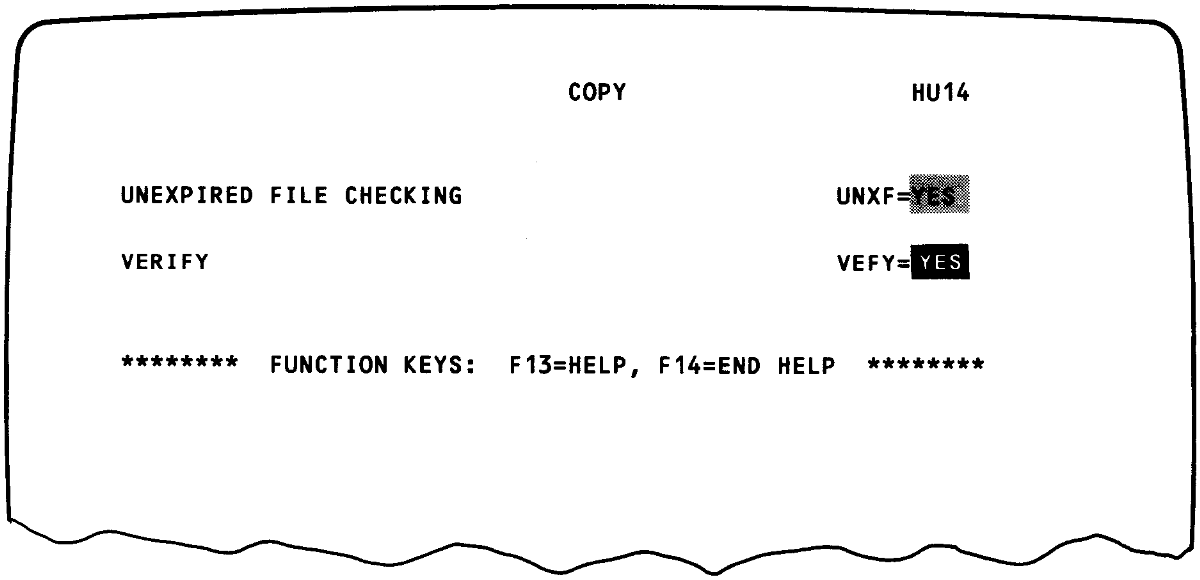
Screen 2 (HU13): Indicating the Number of Copies, Starting and Ending Addresses, and Verification Option



```
                              COPY-VERIFY                   HU13


      COPIES - UP TO 7                                COPY=3

      BEGIN ADDRESS (CCC)                             BGAD=000

      END ADDRESS (CCC)                               EDAD=193

      ONLY VERIFY                                     OVEF=NO_

      ********  FUNCTION KEYS:  F13=HELP, F14=END HELP  ********
```

Looking at screen 2, we entered 3 for the number of copies (overriding the default of 1). We took the default values, which appear on the screen, for the beginning and ending addresses (cylinder 000 for beginning and 193 for ending), as well as the default option NO, indicating that we are doing a copy operation and not a verify-only operation. Since help is not required, press the XMIT key.

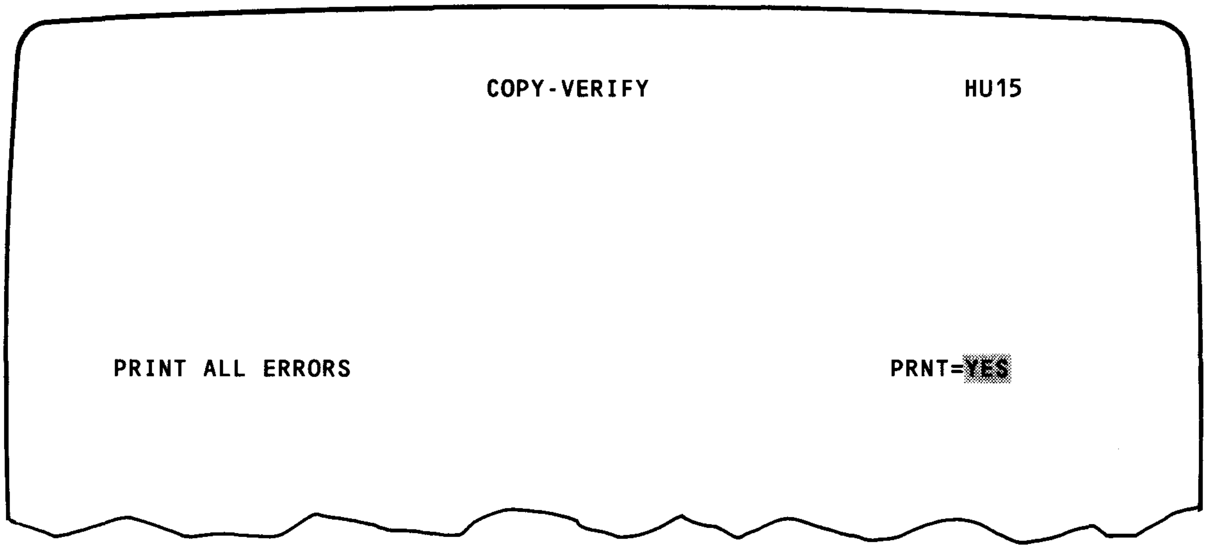
Screen 3 (HU14): Indicating the File Options



```
                                 COPY                      HU14


      UNEXPIRED FILE CHECKING                        UNXF=YES

      VERIFY                                         VEFY=YES


      ********  FUNCTION KEYS:  F13=HELP, F14=END HELP  ********
```

On this screen, we want unexpired file checking and output verification. We entered YES for output verification and allowed the unexpired file checking option (UNXF) to default to YES. Unexpired file checking checks the output volume for file expiration dates. If the date has not expired and your terminal is in system mode, your screen displays a message. This message asks you either to terminate or to skip the file and continue processing. You can skip up to 10 files at a time. Verification ensures that an exact copy of your input is produced. Any errors detected are printed by using the print option (next screen).

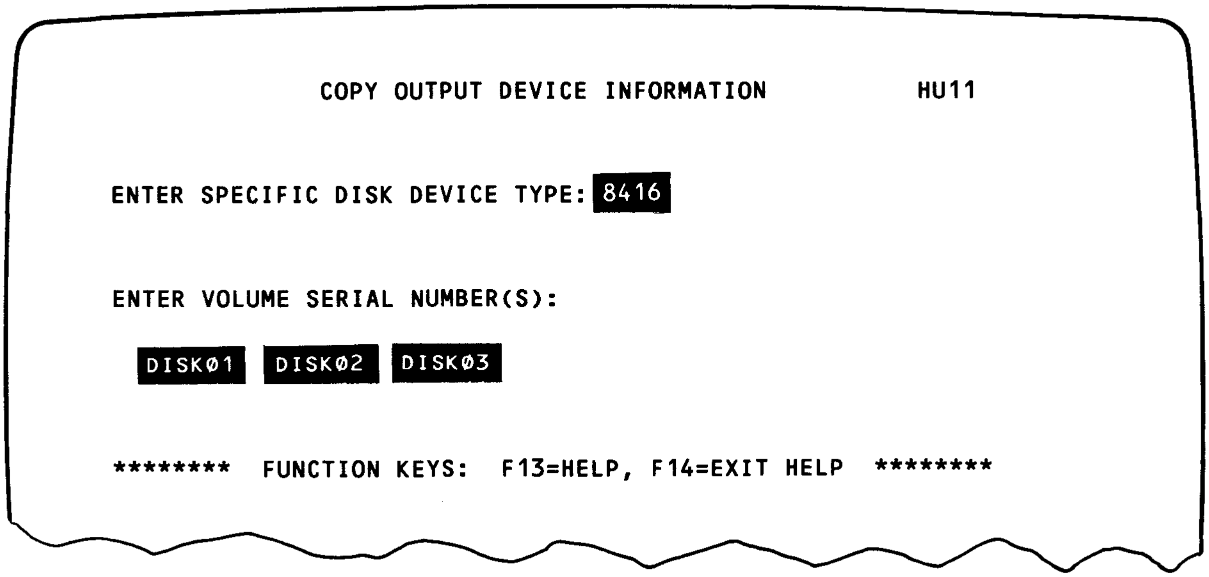
Screen 4 (HU15): Indicating the Print Option



```
                         COPY-VERIFY                   HU15




   PRINT ALL ERRORS                              PRNT=YES
```

Since we specified the verify option on the last screen, we recommend that you take the default YES to print any errors detected during the copy process. Press the XMIT key.

Screen 5 (HU11): Specifying Output Device Information



```
                   COPY OUTPUT DEVICE INFORMATION        HU11

   ENTER SPECIFIC DISK DEVICE TYPE: 8416

   ENTER VOLUME SERIAL NUMBER(S):

    DISKØ1  DISKØ2  DISKØ3

   ********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

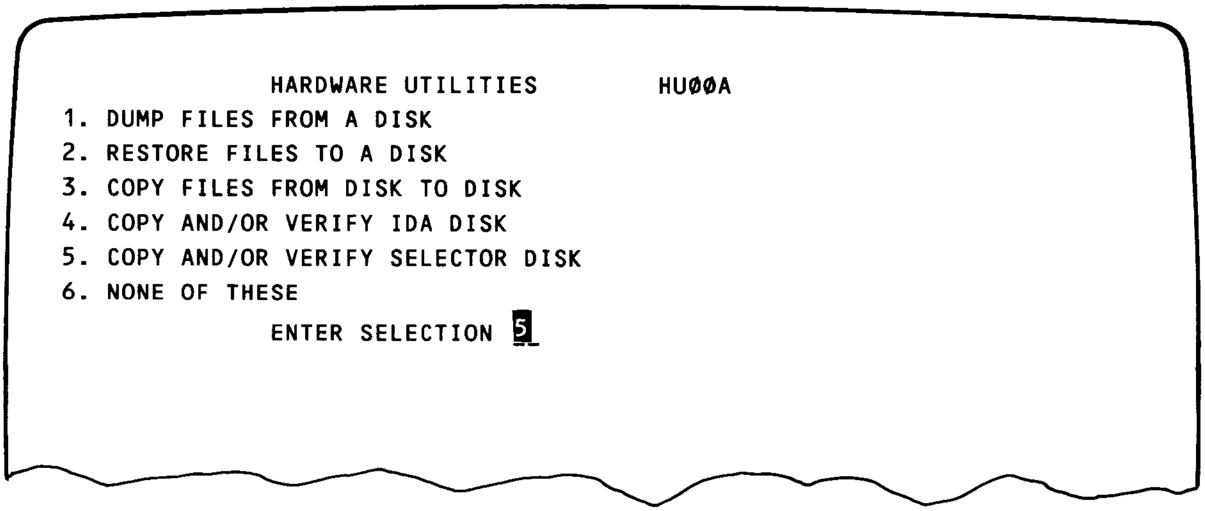
On this screen, we entered 8416 as our output disk device type. Table 12-1 lists the output disk device types that can be used with a given input disk device type. Next, we entered our output volume serial numbers. The number of underscored input fields requesting volume serial numbers that appear on the screen are equal to the number of copies specified on screen 2. All fields that appear must be filled in. Since we specified three copies, fields for three volume serial numbers appear on the screen. We entered volume serial numbers DISK01, DISK02, and DISK03 for the three copies of our output. Press the XMIT key to complete the copy operation. After you press the XMIT key, the terminal becomes free for other uses. When the job is finished, your terminal screen displays an end-of-job message.

12.4. EXECUTING SU$CSL IN AN INTERACTIVE ENVIRONMENT

Interactive processing of the SU$CSL routine consists of a question and answer session (dialog) using either a UNISCOPE 100, UNISCOPE 200, or UTS 400 terminal. The procedures for executing the SU$CSL routine are the same, regardless of which type of terminal you use, except for a few adjustments necessitated by differences in the terminal keyboards. With the UTS 400 terminal, for example, use the XMIT key to transmit work screens and function keys 13 and 14 to obtain and exit help screens. For UNISCOPE 100 and 200 terminals, use the TRANSMIT key to transmit work screens. To use function key 13 or 14, you must simulate it in the following manner. Press the MESSAGE WAITING key, then alphabetic F, then the pound sign (#), and then the number 13 or 14.

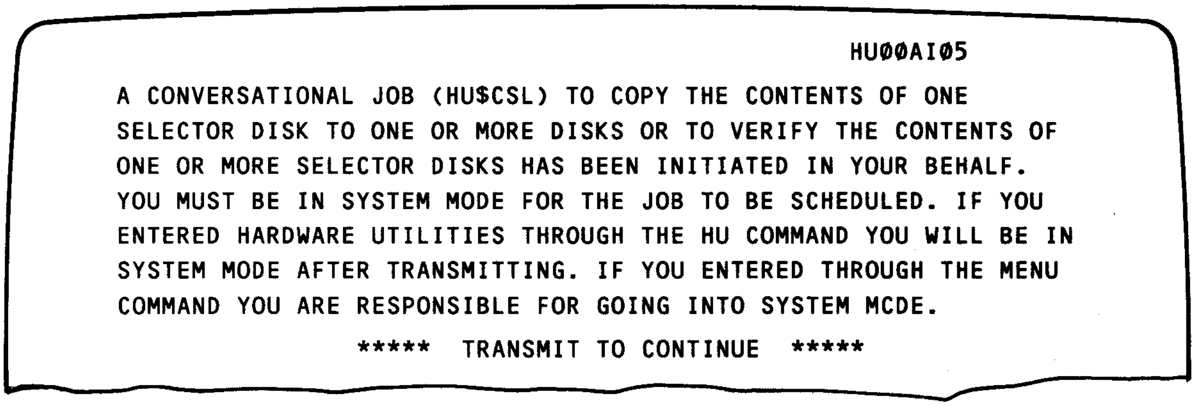
The following example shows the SU$CSL routine for a UTS 400 terminal. When using a UNISCOPE 100 or 200 terminal, you must adjust the instructions in the manner just discussed.

Before executing SU$CSL, you must have successfully logged onto the system via the LOGON command. After logging on, key in HU in system mode and press the XMIT key. After you press the XMIT key, the HARDWARE UTILITIES menu screen appears:



```
              HARDWARE UTILITIES        HUØØA
 1. DUMP FILES FROM A DISK
 2. RESTORE FILES TO A DISK
 3. COPY FILES FROM DISK TO DISK
 4. COPY AND/OR VERIFY IDA DISK
 5. COPY AND/OR VERIFY SELECTOR DISK
 6. NONE OF THESE
              ENTER SELECTION 5
```

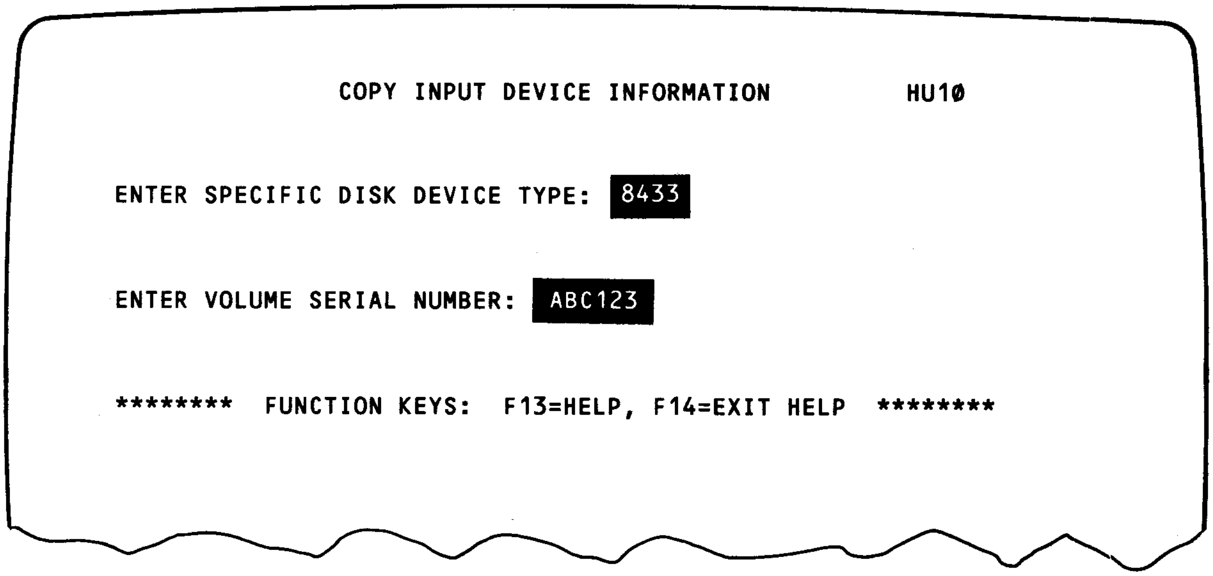
Looking at the menu screen, select 5 to start execution of the SU$CSL routine. After you press the XMIT key, the following informational screen appears:



```
                                                      HUØØAIØ5
     A CONVERSATIONAL JOB (HU$CSL) TO COPY THE CONTENTS OF ONE
     SELECTOR DISK TO ONE OR MORE DISKS OR TO VERIFY THE CONTENTS OF
     ONE OR MORE SELECTOR DISKS HAS BEEN INITIATED IN YOUR BEHALF.
     YOU MUST BE IN SYSTEM MODE FOR THE JOB TO BE SCHEDULED. IF YOU
     ENTERED HARDWARE UTILITIES THROUGH THE HU COMMAND YOU WILL BE IN
     SYSTEM MODE AFTER TRANSMITTING. IF YOU ENTERED THROUGH THE MENU
     COMMAND YOU ARE RESPONSIBLE FOR GOING INTO SYSTEM MCDE.
                    *****  TRANSMIT TO CONTINUE  *****
```

After you press the XMIT key, screen 1 is displayed:

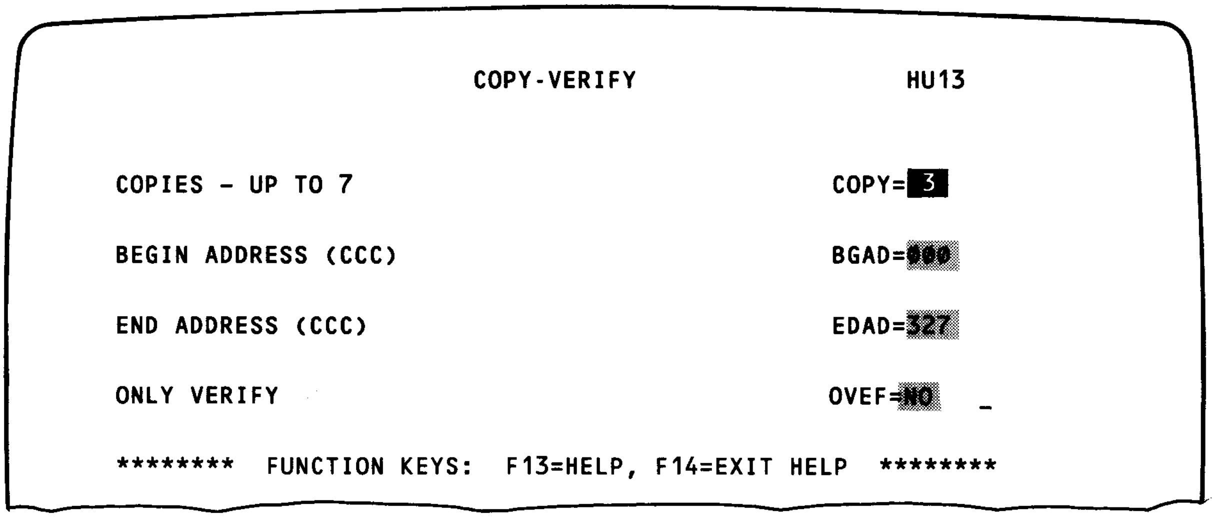
Screen 1 (HU10): Indicating the Specific Disk Device Type and the Volume Serial Number



```
                   COPY INPUT DEVICE INFORMATION        HU1Ø

    ENTER SPECIFIC DISK DEVICE TYPE: 8433

    ENTER VOLUME SERIAL NUMBER: ABC123

    ********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

Looking at screen 1, we entered 8433 for the disk device type and ABC123 for the volume serial number. The device type entered on this screen can be an 8411, 8414, 8424/25, 8430, or 8433 disk pack. Since help is not required, press the XMIT key and the next screen appears:

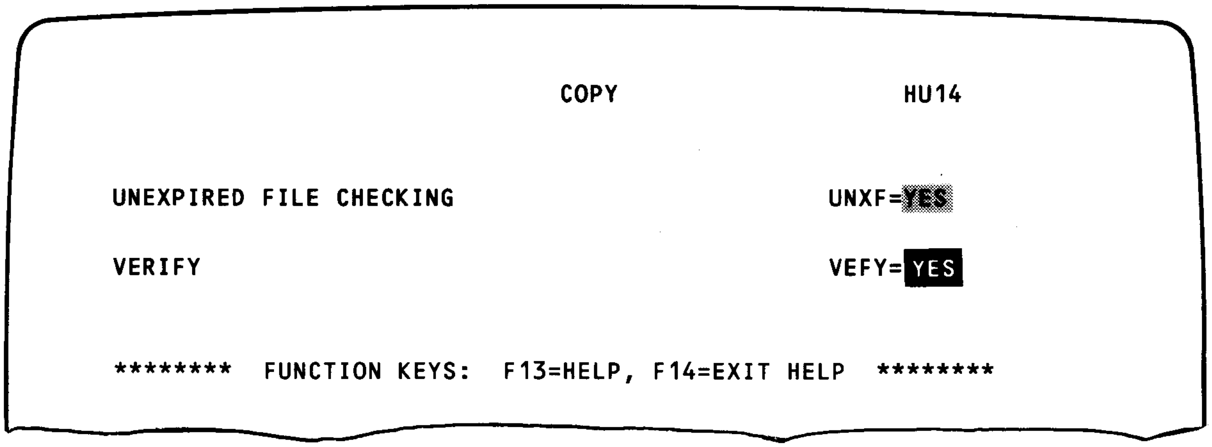
Screen 2 (HU13): Indicating the Number of Copies, Starting and Ending Addresses, and Verification Option



```
                          COPY-VERIFY              HU13


COPIES - UP TO 7                                COPY=3

BEGIN ADDRESS (CCC)                             BGAD=000

END ADDRESS (CCC)                               EDAD=327

ONLY VERIFY                                     OVEF=NO  _

********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

Looking at screen 2, we entered 3 for the number of copies (overriding the default of 1), took the default for the beginning and ending addresses (cylinder 000 for beginning and 327 for ending), and took the default option NO, indicating that we are doing a copy operation and not a verify-only operation. Since help is not required, press the XMIT key.

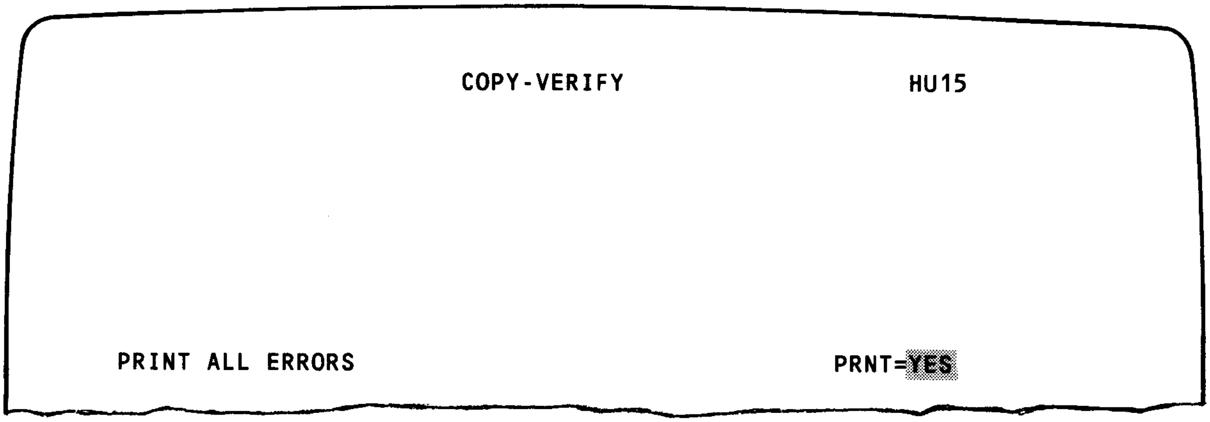
Screen 3 (HU14): Indicating the File Options



```
                             COPY                  HU14


UNEXPIRED FILE CHECKING                         UNXF=YES

VERIFY                                          VEFY=YES


********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

On this screen, we want unexpired file checking and output verification. We entered YES for output verification and allowed the unexpired file checking option (UNXF) to default to YES. Unexpired file checking checks the output volume for file expiration dates. If the date has not expired and you are in system mode, the terminal displays a message. This message asks you either to terminate or to skip the file and continue processing. You can skip up to 10 files at a time. Verification ensures that an exact copy of your input is produced. Any errors detected are printed by using the print option (next screen).

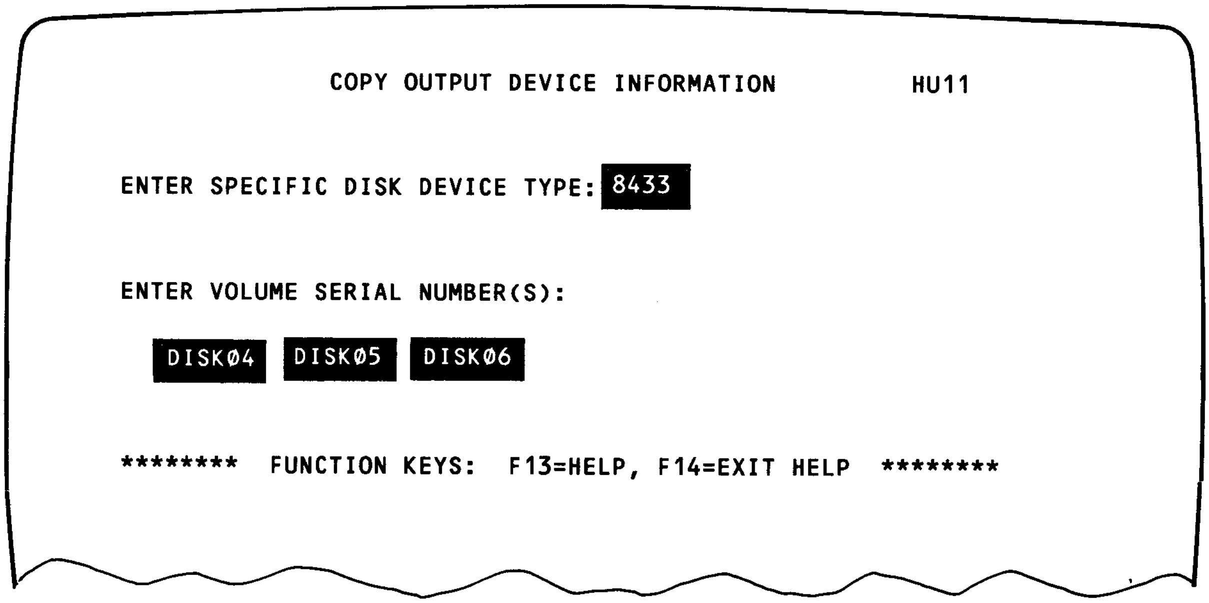
Screen 4 (HU15): Indicating the Print Option



```
                         COPY-VERIFY                 HU15




     PRINT ALL ERRORS                              PRNT=YES
```

Since we specified the verify option on the last screen, we recommend that you take default YES to print any errors detected during the copy process. Press the XMIT key.

Screen 5 (HU11): Specifying Output Device Information



```
                   COPY OUTPUT DEVICE INFORMATION        HU11

     ENTER SPECIFIC DISK DEVICE TYPE: 8433

     ENTER VOLUME SERIAL NUMBER(S):

       DISKØ4  DISKØ5  DISKØ6

     ********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

On this screen, we entered 8433 as our output disk device type. Table 12–1 lists the output disk device types that can be used with a given input disk device type. Next, we entered volume serial numbers DISK04, DISK05, and DISK06 for the three copies of our output.

The number of underscored input fields requesting volume serial numbers that appear on the screen are equal to the number of copies specified on screen 2. Press the XMIT key to complete the copy operation. After you press the XMIT key, the terminal becomes free for other uses. When the job is finished, your terminal screen displays an end-of-job message.

12.5. STAND-ALONE DISK COPY (SU$IDA and SU$SEL)

Once the stand-alone disk copy routine has been loaded into main storage, it is capable of copying any disk pack to another disk pack of a similar type. As the name implies, this routine does not run under the control of the supervisor nor does it require a SYSRES disk pack to be online. This approach allows the user with only two disk drives to create backup copies of non-SYSRES disk packs. Table 12-1 lists the types of input and output devices that the stand-alone disk copy routine processes.

Table 12-1. Permissible Types of Input and Output Devices

Input	Output	Input	Output
8415R	8415R	8418L	8416
8415F	8415F	8411	8411
8415R	8415F	8414	8414
8415F	8415R ①	8424/25	8424/25
8416	8416	8430	8430
8418L	8418H ③	8433	8433
8418	8418	8430	8433 ②
8418H	8416 ②	8433	8430 ②
8416	8418H ③		
8416	8418L		

NOTES:

① It requires three removable disk packs to store the data of one fixed disk pack.

② Only half of the disk is copied to the new disk. The VTOC shows that 808 cylinders are available.

③ The VTOC for the 8418 disk shows 404 cylinders are available.

The stand-alone disk copy routine executes as a dialog with the operator via console messages. The routine consists of a control phase that determines which of four other phases are to receive control. The control phase first branches to the initialization phase. After the necessary main storage areas have been initialized, the control phase branches to the input processing phase, followed by a branch to the output processing phase. When all processing is completed, the control phase branches to the termination phase.

When control is transferred in either direction, messages are displayed on the console that either are informational or require a reply. If an invalid reply is given, a message is displayed indicating this fact. A message is displayed that also indicates that the correct reply was entered.

12.5.1. Load Procedures

The routine is loaded into main storage by using the absolute load ability of IPL. When the following message appears, the name of the program that you key in is substituted for the underscores.

```
IPL TO LOAD STANDARD SUPERVISOR UNLESS NEW NAME KEYED IN _______,_
```

You should then key in one of the following:

SU$IDA,L – to copy an 8415, 8416, or 8418

SU$SEL,L – to copy an 8411, 8414, 8424/25, 8430, or 8433

Control is then transferred to the initialization phase.

12.5.2. Initialization Phase

This phase updates various tables and areas in main storage. It also determines the type of operation, device type, and device address.

The following message appears on the top line of the console:

```
OS/3 STAND ALONE COPY vv/rr
```

where vv is the version number and rr is the revision number.

Then the following messages are displayed. In responding to any message, only the first character is required; the remaining characters are optional.

```
DC01 TYPE OF OPERATION REQUESTED (COPY,VERIFY)?
- - - - - -
```

Keyin:
    Type in either COPY or VERIFY. The default is COPY.
Response:

```
DC11 YOU HAVE REQUESTED A { COPY   }
                          { VERIFY }

    OPERATION BE PERFORMED
```

```
DC02 DEVICE TYPES SUPPORTED:   8415R,8415F,8416,8418L,8418H
                               8411,8414,8424,8430,8433
```

Which line appears depends on whether SU$IDA (top line) or SU$SEL (bottom line) was specified at load time.

DC03 INPUT DEVICE TYPE?

Keyin:

One of the input device types listed in Table 12-2.

Response:

DC12 INPUT DEVICE TYPE REQUESTED IS (device type)

If keyin is invalid, response is:

DC13 INPUT DEVICE REQUESTED IS (device type)
THIS DEVICE NOT SUPPORTED.

The DC03 message is repeated to allow another device type to be selected.

DC04 INPUT DEVICE ADDRESS?

Keyin:

The 3-digit device address is entered (channel, subchannel, and device number).

Response:

DC14 INPUT DEVICE ADDRESS REQUESTED IS (cnn)

If keyin is invalid, response is:

DC15 INPUT DEVICE ADDRESS REQUESTED IS (cnn)
DEVICE ADDRESS IS INVALID.

The DC04 message is repeated to allow another device address to be entered.

DC05 OUTPUT DEVICE TYPE?

Keyin:

One of the device types listed in Table 12-2.

Response:

DC14 OUTPUT DEVICE TYPE REQUESTED IS (device type)

If keyin is invalid, response is:

DC16 INPUT-OUTPUT DEVICE ADDRESSES ARE EQUAL

This message is output if the address of the output device address is equal to the input device address. The DC03 message is displayed again and the I/O device addresses are resubmitted. Also, a check is made of the correct device address for the 8415R or 8415F disk. If incorrect, the following message is displayed:

DC19 8415 DISC DEVICE ADDRESS INVALID

The I/O devices are then rechecked beginning with the DC03 message.

DC07 OPERATIONAL TYPE: FULL,PARTIAL,COPY,ALLOCATE,RESTORE-FIXED.

Keyin:

FULL

The entire extent of the input device type is used, as listed in Table 12-2. The extent addresses are in the form ccc,hh,rr in decimal.

Table 12-2. Extent Addresses by Input Device Type

Device Type	Beginning Extent	Ending Extent
8415R	000,00,01	403,01,40
8415F	000,00,01	807,02,40
8416	000,00,01	403,06,40
8418L	000,00,01	403,06,40
8418H	000,00,01	807,06,40
8411	000,00,01	202,09,40
8414	000,00,01	202,19,72
8424/25	000,00,01	399,19,72
8430	000,00,01	403,18,96
8433	000,00,01	807,18,96

PARTIAL

This keyin can be used on like disk types only. This operation is invalid for the 8415 disk.

COPY

The allocated space on an 8415 removable disk pack is copied to the unallocated space on an 8415 fixed disk pack.

ALLOCATE

The allocated space on the input device is copied to the output device.

RESTORE-FIXED

A multi-input 8415 removable disk pack can restore an 8415 fixed disk pack.

Response:

DC17 OPERATIONAL TYPE (operational type)

If keyin is invalid, response is:

DC0D OPERATIONAL TYPE (operational type) INVALID

The DC07 message is repeated.

DC08 BEGINNING EXTENT IN DECIMAL: (CCC,HH)?

Keyin:

Cylinder and head address (ccc,hh) of the beginning extent. This message is displayed only when the operational type chosen is PARTIAL.

If keyin is invalid, response is:

DC0A INVALID EXTENT

The extent value does not fall between the permissible beginning and ending extent values. The DC08 message is repeated.

DC0B PARTIAL EXTENT CONTAINS NON NUMERIC

One of the characters in your keyin was nonnumeric. The DC08 message is repeated.

DC09 ENDING EXTENT IN DECIMAL: (CCC,HH)?

Keyin:

Cylinder and head address (ccc,hh) of the ending extent. This message is displayed only when the operational type chosen is PARTIAL.

If keyin is invalid, response is:

DC0A INVALID EXTENT

The extent value does not fall between the permissible beginning and ending extent values. The DC09 message is repeated.

DC0B PARTIAL EXTENT CONTAINS NON NUMERIC

One of the characters in your keyin was nonnumeric. The DC09 message is repeated.

DC20 INPUT BEGIN EXTENT ADDRESS HIGHER THAN END

The DC08 message is repeated.

DC18 INPUT EXTENT

```
BEGINNING EXTENT   =DECIMAL CCC=ccc HH=hh RR=rr
ENDING EXTENT      =DECIMAL CCC=ccc HH=hh RR=rr
```

DC19 OUTPUT EXTENT

```
BEGINNING EXTENT   =DECIMAL CCC=ccc HH=hh RR=rr
ENDING EXTENT      =DECIMAL CCC=ccc HH=hh RR=rr
```

These two messages are displayed to give the user information regarding the extents allocated to the job.

DC25 INITIALIZATION COMPLETE: (GO, REINIT)?

Keyin:

GO

Processing begins.

REINIT

The initialization process begins again with message DC01.

12.5.3. Control Phase

Program control is returned to the control phase after the completion of each of the other phases. If an interrupt has occurred during these phase operations, the following messages appear on the screen.

DC30 REQUEST TYPE OF OPERATION: (DUMP, CANCEL, STATUS, NONE)?

Keyin:

DUMP

After the status information has been displayed, a SYSDUMP is taken.

CANCEL

After the status information has been displayed, the job is terminated.

STATUS

Status information is displayed in the following messages and processing continues.

DC31 INPUT DEVICE ADDRESS: cnn
DC32 OPERATIONAL TYPE:
FULL,PARTIAL,COPY,ALLOCATE,RESTORE-FIXED
DC33 EXTENT FORMAT IN DECIMAL: ccc,hh
DC34 CURRENT EXTENT=ccc,hh ENDING EXTENT=ccc,hh
DC35 OUTPUT DEVICE ADDRESS: cnn

NONE:

This keyin turns the console attention interrupt bit off and returns control to the next instruction in the control routine.

If the keyin is omitted, NONE is assumed.

DC36 MOUNT NEXT SCRATCH ON cnn. REPLY R WHEN READY

Keyin:

R, when the next scratch 8415 removable disk has been mounted.

DC38 RESTORE OPERATION REQUIRES DISC n OF 3 BE MOUNTED. REPLY R WHEN READY

Keyin:

R

DC39 COPY TO FIXED COMPLETED MOUNT SCRATCH ON cnn. REPLY R WHEN READY

Keyin:

R

12.5.4. Input and Output Phases

The input phase reads data one track at a time and stores it in a buffer area. The output phase processes the data and writes it out one track at a time. No messages are displayed during these phases.

12.5.5. Termination Phase

When the job has terminated, the following message is displayed.

$$\text{DC37 STAND ALONE DISC COPY TERMINATED: }\left\{\begin{array}{l}\text{ABNORMALLY}\\ \text{NORMALLY}\end{array}\right\}$$

If you keyed in a dump or cancel operation as a reply to message DC30, an abnormal termination is indicated.

12.5.6. I/O Disk Error Handling Routine

The following messages are displayed, depending on the type of I/O disk error.

DC40 DEVICE cnn IN STOP STATE: (RETRY OR DUMP)?

Keyin:

RETRY
The I/O order is reissued.

DUMP
After the status information has been displayed, a SYSDUMP is taken.

DC41 DEVICE cnn FILE PROTECTED ON WRITE: (RETRY OR DUMP)?

Keyin:

RETRY
The I/O order is reissued.

DUMP
After the status information has been displayed, a SYSDUMP is taken.

DC42 DEVICE cnn I/O ERROR SENSE BYTE=xxxxxxxxxx:
ERROR=error-message- - - - - - - - - - - - - - (RETRY,IGNORE,DUMP)?

Keyin:

RETRY
The I/O order is reissued.

IGNORE
The error is ignored and the I/O order is bypassed.

DUMP
After the status information has been displayed, a SYSDUMP is taken.

The following message is displayed when an error is detected during verification processing.

DC43 ERROR IN VERIFICATION CYLINDER=ccc HEAD=h RECORD=rrr CONTINUE OR DUMP?

Keyin:

CONTINUE

Verification processing continues.

DUMP

After the status information has been displayed, a SYSDUMP is taken.

13. Disk Dump/Restore (DMPRST) Routine

13.1. DMPRST CONCEPT

DMPRST permits you to create backup libraries or data files for possible future use at your installation. You can:

- copy a disk or a portion of a disk to a magnetic tape, called a dump operation;
- copy a magnetic tape to a disk, called a restore operation;
- copy a disk or a portion of a disk to another disk, called a disk copy operation; or
- copy a tape created by DMPRST to another tape, called a tape copy operation.

You must specify similar devices for both the dump and restore operations. However, when you use the copy operation, like devices must be specified except for copying either 8416 or 8418 disk packs. The allowable combinations for copying these packs are: 8416 to 8418 (dual), 8418 (dual) to 8416, 8418 (single) to 8418 (dual), and 8418 (dual) to 8418 (single).

It is important to remember the number of usable tracks per disk pack when downgrading your copy operation. For example, when you copy an 8418 (dual) to an 8416, the 8418 has 808 usable tracks while the 8416 has only 404 usable tracks. Therefore, if you are copying more than 404 tracks, DMPRST will issue a diagnostic message indicating some of your files have been lost.

When you are using a tape as a go-between device, then the disk type that was your input to the tape must also be the same disk type as the output from the tape. For example, if you were dumping an 8411 disk to a UNISERVO VI-C tape, then the UNISERVO VI-C tape must be restored to another 8411 disk. All of the devices you are using must be prepped.

DMPRST does not require a printer in a batch environment. If a fatal error is detected, the error message is displayed on the system console. If you do specify a printer, the // PARAM statements, the FILE statements, and all error messages are listed.

You can perform the dump, restore, and copy operations on either a volume or file basis.

In a multijobbing environment, the use of DMPRST can cause space allocated to a file to be lost; files may be added to or portions deleted, or a disk pack may be incorrectly copied. These situations can occur when another job is updating the VTOC of a disk pack at the same time DMPRST is copying that VTOC to another disk pack, or when DMPRST is copying a file while another job is extending or scratching that file. Therefore, the job executing DMPRST must be the only job running when the I/O device is either SYSRUN or SYSRES. If the device is not SYSRUN or SYSRES, it must be set to nonsharable by using the SET IO operator command. See the current version of the operations handbook for operators, UP-8072 or UP-8511 for details. This will ensure that an exact copy will be made by DMPRST.

NOTE:

DMPRST cannot restore or copy system files (prefixed by Y) on an active SYSRES, i.e., the SYSRES pack you are executing from. To process these files, see the current version of the system installation user guide, UP-8074.

You can execute DMPRST either in a batch environment or interactively from a terminal. Both the interactive and batch methods accomplish the same result; however, there are some differences between what each method supports. Table 13-1 lists these differences. See 13.5 for interactive processing of DMPRST.

Table 13-1. DMPRST Differences between Interactive and Batch Methods

Item	Interactive	Batch
Modes of processing	File	Volume and file
Tape-to-tape copy	Not supported	Supported - volume mode
Printer assignment	Automatically assigned	User responsibility

13.2. EXECUTING DMPRST IN A VOLUME ENVIRONMENT

As stated before, DMPRST enables you to execute four different types of operations: dump, restore, disk copy, and tape copy. // PARAM statements are used to indicate the type of operation to be performed. For convenience, Table 13-2 summarizes the // PARAM statements, along with the appropriate LFD names and the type of operation that will be performed. A complete description of these statements is provided in the paragraphs associated with the appropriate operation.

Table 13-2. Volume Mcde PARAM Statements

Type of Operation	Param Statement	LFD Name
Copy disk to disk	// PARAM IN=DISC // PARAM OUT=DISC	DISCIN DISCOT
Dump disk to tape	// PARAM IN=DISC // PARAM OUT=TAPE	DISCIN TAPEOT
Restore tape to disk	// PARAM IN=TAPE // PARAM OUT=DISC	TAPEIN DISCOT
Copy tape to tape	// PARAM IN=TAPE // PARAM OUT=TAPE	TAPEIN TAPEOT
Suppress file expiration date checking	// PARAM NOEXPCK	

- Dump operation

Here, you are writing from a disk pack to a magnetic tape. Any magnetic tape or disk device may be used. Multiple volume tapes are permitted. Since the dump operation means a disk pack or part of a disk pack is being written to a magnetic tape, the disk pack is your input; therefore, you would use // PARAM IN=DISC. The output is going to a magnetic tape; therefore, you would use // PARAM OUT=TAPE. Notice that we did not indicate what type of disk pack or magnetic tape is being used. DMPRST checks the physical disk or magnetic tape unit for the proper device type. However, DISCIN and TAPEOT must be specified as the file names on the LFD job control statement.

If you are dumping to multivolume tapes, you may specify all of your output tape file names on a single // VOL card. If you do this, only one // DVC card and one // LFD card with the file name TAPEOT specified are needed. Only one tape drive is required, with the operator mounting new tape volumes as needed. However, you may enter a separate device assignment set for each tape volume. If you do this, you must enter a // DVC and // LFD for each tape, and the numbers 01 through 99 must be added to each additional output tape LFD name (TAPEOT01). You must have a separate tape drive for each output tape volume.

If you are dumping the disk to a tape unit connected to a selector channel, but prefer to have the output tape produced in a blocked format that can be read by either multiplexer or selector channel tape units, add the parameter MUX to the // PARAM OUT statement, as shown in example 1. We recommend the MUX parameter be specified for all output tapes because both types of tape units can read the tape and not impact DMPRST processing.

If you wanted only to dump a portion of a disk pack, you would include a // PARAM END=ccc or END=LAST statement in your control stream. The // PARAM END statement is used to terminte the dump operation after a specific cylinder (END=ccc) has been dumped, or after the last cylinder that contains any user data has been dumped (END=LAST). If the // PARAM END statement is omitted, the entire contents of the input disk is dumped to the output tape (including test pattern data written by the disk prep routine), even if only a few cylinders contain user data.

If the // PARAM END=ccc statement is used, ccc must be specified in decimal.

The first record written to your magnetic tape is called the control record. The control record contains the following information in the order shown:

Bytes	Contents
0–1	Control record ID X'COCO'
2–5	System time (packed decimal)
6–9	System date (packed decimal)
10–15	Volume serial number of dumped disk pack
16–17	Unused
18–21	Device type of dumped disk pack
22	Device type of tape
23	Tape channel number
24–25	Unused
26	X'FF'-shows file processing
27–30	Volume table of contents of dumped disk if file processing
31–79	Unused

The following examples show a single and a multiple-volume dump:

Example 1 – Single-Volume Dump:

```
1        10    16
_______________________________________________
// JOB DUMP1
// DVC 2Ø // LFD PRNTR
// DVC 9Ø // VOL SP0366 // LFD TAPEOT
// DVC 5Ø // VOL OS3RES // LFD DISCIN
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=TAPE,MUX
// PARAM END=LAST
/&
// FIN
```

Example 2 – Multiple-Volume Dump:

```
1        10    16
_______________________________________________
// JOB DUMP2
// DVC 2Ø // LFD PRNTR
// DVC 9Ø // VOL SP0366,SP0367 // LFD TAPEOT
// DVC 5Ø // VOL OS3RES // LFD DISCIN
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=TAPE
/&
// FIN
```

Remember, when doing a dump operation, you must specify the // PARAM IN=DISC and // PARAM OUT=TAPE.

- Restore operation

  Here, you are writing from a magnetic tape to a disk pack. Remember, whatever disk type that was used to produce the tape, that same disk type must be used when you are restoring the tape to the disk. Since the restore operation means you are writing from a magnetic tape to a disk pack, the // PARAM IN=TAPE and // PARAM OUT=DISC must be specified. If the input tape was created on a selector channel tape unit and was not specifically produced in the multiplexer blocked format, only a selector channel tape unit can be used to read the input tape. Tapes created in multiplexer blocked format can be read by any tape unit. The file names TAPEIN and DISCOT also must be specified on the // LFD job control statements.

  If restoring from multivolume tapes, you may enter your input tape file names on a single // VOL card. If you do this, only one // LFD card with the name TAPEIN is required. The tape volumes are then mounted on a single drive as they are needed. However, you may enter a device assignment set for each tape appending the numbers 01 through 99 to each additional tape LFD name (TAPEIN01). If you do this, all tape volumes must be mounted on the appropriate tape drive.

The following examples show a single- and a multiple-volume restore operation:

Example 1 – Single-Volume Restore:

```
1         10    16                                                      72
--------------------------------------------------------------------------
// JOB RESTORE1
// DVC 2Ø // LFD PRNTR
// DVC 9Ø // VOL SP0366 // LFD TAPEIN
// DVC 5Ø // VOL DS1111 // LFD DISCOT
// EXEC DMPRST
// PARAM IN=TAPE
// PARAM OUT=DISC
/&
// FIN
```

Example 2 – Multiple-Volume Restore:

```
// JOB RESTORE2
// DVC 2Ø // LFD PRNTR
// DVC 9Ø // VOL SP0366,SP0367 // LFD TAPEIN
// DVC 5Ø // VOL DS1111 // LFD DISCOT
// EXEC DMPRST
// PARAM IN=TAPE
// PARAM OUT=DISC
/&
// FIN
```

- Disk copy operation

Here, you are writing the contents of one disk pack to another disk pack. Both disk packs must be of the same device type except for 8416 and 8418. As stated before, either data or library files are accepted. The // PARAM IN=DISC, and the // PARAM OUT=DISC, along with the file names DISCIN and DISCOT on the LFD job control statements, must be specified. In addition, a // PARAM END=ccc or END=LAST statement can be used to terminate the copy operation after a specific cylinder (END=ccc) has been copied, or after the last cylinder that contains any user data has been copied (END=LAST). If the // PARAM END statement is omitted, the entire contents of the input disk is copied to the output disk (including test pattern data written by the disk prep routine), even if only a few cylinders contain user data.

If the // PARAM END=ccc statement is used, ccc must be specified in decimal.

The following is an example of a typical copy operation:

```
1        10    16
// JOB COPY
// DVC 2Ø // LFD PRNTR
// DVC 5Ø // VOL OS3RES // LFD DISCIN
// DVC 51 // VOL DSP368 // LFD DISCOT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=DISC
// PARAM END=LAST
/&
// FIN
```

- Tape copy operation

Here, you are copying the contents of one tape volume to another. The tape being copied must have been created by a previous DMPRST dump or tape copy operation that was performed in a volume environment. Only tapes created by DMPRST are supported by this function. The tape being copied may have been created on either a multiplexer or selector channel tape drive unit. Likewise, the output tape may be on a multiplexer or selector channel unit. Input tapes created on a selector channel unit, however, cannot be read by a multiplexer channel tape drive unit unless they were created in multiplexer blocked format. Input tapes created on a multiplexer channel tape unit, or in multiplexer format, on a selector channel tape unit can be read by any tape unit. Output tapes produced on a selector channel tape unit can be blocked in multiplexer format by adding the MUX parameter to the // PARAM OUT statement.

DMPRST will not copy a tape produced in the file environment.

The following is an example of a typical job stream used to perform a tape copy operation:

```
1        10    16
// JOB TAPECOPY
// DVC 2Ø // LFD PRNTR
// DVC 9Ø // VOL RELØ3Ø // LFD TAPEIN
// DVC 91 // VOL SP1234 // LFD TAPEOT
// EXEC DMPRST
// PARAM IN=TAPE
// PARAM OUT=TAPE
/&
// FIN
```

13.3. EXECUTING DMPRST IN A FILE ENVIRONMENT

You may execute DMPRST in the file mode to create backups of individual files on your disk. Choose disk or tape as your backup, depending on which DMPRST procedure used. In addition, you can copy, dump, and restore an entire volume of active permanent files. As in volume mode, you can assign a printer to list the // PARAM and FILE statements and any error messages.

For convenience, Table 13-3 summarizes the // PARAM and FILE statements, along with the appropriate LFD names and the type of operation to be performed. A complete description of these statements is provided in the paragraphs associated with the appropriate operation.

Table 13-3. File Mode PARAM and FILE Statements

Type of Operation	Param Statement	LFD Name

Copy disk to disk	// PARAM IN=DISC
	// PARAM OUT=DISC
	// PARAM TYPE=FILE
		DISCIN
		DISCOT

Dump disk to tape	// PARAM IN=DISC
	// PARAM OUT=TAPE
	// PARAM TYPE=FILE
		DISCIN
		TAPEOT

Restore tape to disk	// PARAM IN=TAPE
	// PARAM OUT=DISC
	// PARAM TYPE=FILE
		TAPEIN
		DISCOT

Process only active permanent files	// PARAM TYPE=FILE,ALL	
Avoid waiting for file to be available	// PARAM TYPE=FILE,NOWAIT	
Suppress file expiration date checking	// PARAM NOEXPCK	

13.3.1. Performing a Disk Copy Operation in a File Environment

When performing a disk copy operation in a file environment, you copy the contents of individual files residing on one disk pack to another disk pack. Both disk packs must be of the same device type, except for 8416 and 8418. Either library or data files are accepted.

Input and output in a disk copy operation are always disks. Therefore, specify // PARAM IN=DISC as your input parameter and // PARM OUT=DISC as your output parameter. In addition, you must specify the file names DISCIN on the input // LFD statement and DISCOT on the output // LFD statement.

Since you are copying files, specify // PARAM TYPE=FILE to inform DMPRST that file rather than volume processing is performed. Associated with the // PARAM TYPE=FILE parameter are FILE statements. By including these statements as embedded data in your control stream, you can name which files you want copied by either file name or file prefix.

The FILE statements are free-formatted, meaning that the first character may start anywhere. However, you must place a blank character between the FILE statement and the first operand to act as a delimiter. If you specify a file name containing embedded blanks, enclose it in quotation marks (i.e., FILE "PAY BACKUP").

The following example shows a disk copy operation in the file environment:

```
// JOB DFILCPY
// DVC 2Ø // LFD PRNTR
// DVC 5Ø // VOL DISKØ1 // LFD DISCIN
// DVC 51 // VOL DISKØ2 // LFD DISCOT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=DISC
// PARAM TYPE=FILE
/$
            FILE MONEY
            FILE INTEREST
            FILE BILLS
            FILE.P BANKS
/*
/&
// FIN
```

Files MONEY, INTEREST, and BILLS, along with all files having the prefix of BANKS on disk volume DISK01, are copied to disk volume DISK02. Notice that the FILE statements are included as embedded data in the control stream.

If you want to copy an entire volume of active permanent files, use the FILE,ALL statement. This signifies that all active permanent files on the volume are copied. By active permanent files, we mean all files that have entries in the VTOC and that are not job temporary files (run libraries and scratch files). Any files deleted but not physically removed are not copied. (In volume mode, everything is copied.)

The following example shows a disk copy operation in file mode but copying the entire volume of active permanent files:

```
// JOB COPYALL
// DVC 2Ø // LFD PRNTR
// DVC 5Ø // VOL DISKØ1 // LFD DISCIN
// DVC 51 // VOL DISKØ2 // LFD DISCOT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=DISC
// PARAM TYPE=FILE,ALL
/&
// FIN
```

NOTE:

Job temporary files (run libraries and scratch files) are not processed in this file environment.

Before DMPRST processes any files, it locks them. Therefore, if you copy, dump, or restore a file currently being used in the system, DMPRST waits until the file is available before processing it. You can avoid this delay by specifying the NOWAIT parameter on the FILE statement. DMPRST, upon detecting NOWAIT, skips processing of any of the unavailable files and goes to the next available one. All files skipped are listed on the printer, so you can process them later. The following example shows a disk copy operation in file mode by using the NOWAIT parameter:

```
// JOB COPYALL
// DVC 2Ø // LFD PRNTR
// DVC 5Ø // VOL DISKØ1 // LFD DISCIN
// DVC 51 // VOL DISKØ2 // LFD DISCOT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=DISC
// PARAM TYPE=FILE,NOWAIT
/$
        FILE CREDIT
        FILE ASSETS
        FILE INVENTORY
/*
/&
// FIN
```

13.3.2. Performing a Dump Operation in the File Environment

Here, you can dump one or more files residing on your disk volume to a magnetic tape or dump the entire volume of active permanent files. Since you are writing from disk to tape, the // PARAM IN=DISC and // PARAM OUT=TAPE must be specified. Also, the file names DISCIN and TAPEOT on the // LFD job control statements must be specified.

Up to this point, everything is the same as in the dump operation in the volume environment. Now for the additions: the // PARAM TYPE=FILE must be specified, telling DMPRST that a file is being processed rather than a volume. The // PARAM TYPE=FILE card sets the file operation in motion. Next is your FILE card. You use this card to specify the file name or the file prefix (up to 16 bytes) of the files you want dumped. As stated earlier, the FILE card is free form. When specifying file names, use the word FILE.

Immediately after the word FILE, a blank character must be present; then insert the name of your file to be dumped. If you dump more than one file, a FILE card for each file must be specified. When specifying a file prefix, use the word FILE.P. Again, immediately after the word FILE.P, a blank character must be present; then insert the prefix of the files to be dumped.

When you want to dump the entire volume of active permanent files, use the // PARAM TYPE=FILE,ALL statement. This indicates that all active permanent files (entries in VTOC) are dumped.

Whether you dump individual or active permanent files, the // PARAM END statement cannot be used.

NOTE:

Within a single execution of the restore operation, the files selected for restoration must be requested in the same order in which they appear on the tape. Because of this, we suggest that you dump your files in alphanumeric order. This way, you know the order to put your FILE cards in when restoring.

In example 1, files MYCOBOLMASTER and PROCFILE, as well as all files with the prefix TAXES, are dumped from disk volume DSP001 to magnetic tape SPT001. Notice that, even though you are accessing files on your disk pack, there is no // LBL job control statement present in your job control stream.

Example 1:

```
1        10    16
_____________________________________________
// JOB DUMPFIL
// DVC 2Ø // LFD PRNTR
// DVC 54 // VOL DSPØØ1 // LFD DISCIN
// DVC 9Ø // VOL SPTØØ1 // LFD TAPEOT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=TAPE
// PARAM TYPE=FILE
/$
 FILE MYCOBOLMASTER
 FILE PROCFILE
 FILE.P TAXES
/*
/&
// FIN
```

In example 2, the entire volume of active permanent files is dumped to multivolume tapes by using a single device assignment set.

Example 2:

```
// JOB DUMPALL
// DVC 2Ø // LFD PRNTR
// DVC 5Ø // VOL MYPACK // LFD DISCIN
// DVC 9Ø // VOL BACKØ1,BACKØ2
// LFD TAPEOT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=TAPE
// PARAM TYPE=FILE,ALL
/&
// FIN
```

13.3.3. Performing a Restore Operation in the File Environment

When you dump files from a disk, the backup created is not an exact copy of the original. Therefore, you cannot use the backup on your system in the event your original data is lost. Before actually using any backup data on your system, you must restore it to the original disk or a disk of the same type as the original.

In a restore operation, the output is always a disk. Therefore, specify // PARAM OUT=DISC as your output parameter, and DISCOT as the file name on your disk // LFD statement.

Since you are restoring files, include the // PARAM TYPE=FILE parameter in your job stream. This parameter informs DMPRST that file rather than volume processing is performed. If you want to restore an entire volume of active permanent files, use // PARAM TYPE=FILE,ALL.

Associated with the // PARAM TYPE=FILE parameter are FILE statements. Like the FILE statements used in file dump operations, they name which files you want processed. In a restore operation, however, they also allow you to rename files and specify the type of processing you want performed.

The FILE statements are free-formatted, meaning that the first character may start anywhere. However, you must place a blank between the FILE statement and the first operand, and a comma between the operands to act as a delimiter. If you specify a file name containing embedded blanks, enclose it in quotation marks.

The formats of the FILE statement used in a restore operation are:

Format 1:

```
FILE old-name ⎡,⎧ABS⎫⎤ [,new-name]
              ⎢ ⎪REL⎪⎥
              ⎢ ⎨LOG⎬⎥
              ⎣ ⎩PRE⎭⎦
```

The old-name parameter specifies the name of the file you are restoring to disk.

Format 2:

```
FILE.P prefix-name ⎡,⎧ABS⎫⎤
                   ⎢ ⎪REL⎪⎥
                   ⎢ ⎨LOG⎬⎥
                   ⎣ ⎩PRE⎭⎦
```

The prefix-name parameter specifies the prefix name of all the files to be restored. Regardless of the format, when restoring more than one file, you must restore the files in the order they were dumped. Also, when you restore files by prefix, the files to be restored must be in the order on the tape. Once it has begun, processing of files by prefix is terminated when a file is found without that prefix name. For example, suppose you have a tape containing files in the following order: three files with the prefix BANK, one file with the prefix AMT, and two more files with the prefix BANK. If you use the prefix name BANK to restore these files, only the first three files will be restored.

13.3.3.1. Using the Allocation Parameter to Control Restore Processing

The second parameter in the FILE statement is the allocation parameter. This optional parameter enables you to control file processing. The allocation parameters are:

ABS

Locates a file on the same absolute extents. If a file cannot be allocated to the same extents, an error message is issued and DMPRST processes the next FILE statement.

REL

Relocates a file. If relocation to the same size file is not possible, a file large enough to hold the original file is allocated. If this is unsuccessful, an error message is issued and DMPRST processes the next FILE statement.

LOG

Relocates a file and deletes all unassigned space for that file.

PRE

Relocates a file in a space that was preallocated.

You can only specify one allocation parameter in each FILE statement. Enter it after the old-name parameter, with a comma separating the two.

When you want to perform standard restore processing, omit the allocation parameter. However, if you omit the allocation parameter and specify the new-name parameter, substitute a comma for the missing allocation parameter.

During standard restore processing, DMPRST scratches the file from the output disk and makes up to three attempts to reallocate it on the disk.

When reallocating the file, DMPRST first attempts to allocate the same absolute extents occupied by the original file. If this is unsuccessful, the file relocation facility attempts to allocate the same size file at a different location. If this second try is also unsuccessful, the file relocation facility determines how much of the original space is actually assigned. It then tries to allocate a file large enough to hold the assigned space from the original file. If all three attempts fail, an error message is issued to the system console.

13.3.3.2. Using the File Prefix Parameter

The file prefix parameter gives you the capability to restore all files having a certain prefix in the file name. For example, suppose you want to restore all files having the file name prefix of OLD. In addition, the restored files are to be located on the same absolute extents. The following example shows this restore operation by using the prefix-name and ABS parameters:

```
// JOB UPDATE
// DVC 2Ø // PRNTR
// DVC 9Ø // VOL SAVEØ1,SAVEØ2,SAVEØ3
// LFD TAPEIN
// DVC 5Ø // VOL NEVOL1,NEVOL2,NEVOL3
// LFD DISCOT
// EXEC DMPRST
// PARAM IN=TAPE
// PARAM OUT=DISC
// PARAM TYPE=FILE
/$
        FILE.P OLD,ABS
/*
/&
// FIN
```

13.3.3.3. Using the New-name Parameter to Rename Files

The third parameter in the FILE statement is the new-name parameter. You use it to change the name of a file being restored to disk.

For example, suppose you want to restore the disk file MYFILE to a file named PAYROLL on disk. In order to copy MYFILE to PAYROLL, you specify PAYROLL as the new-name parameter in the FILE statement with the old-name MYFILE.

In this case, DMPRST first scratches any file having the name PAYROLL from the output disk. Next, DMPRST allocates a file large enough for MYFILE. The old file, MYFILE, bearing the new name, PAYROLL, is then written to the allocated space on the output disk.

The following example shows a restore operation using the new-name parameter:

```
// JOB RENAME
// DVC 2Ø // LFD PRNTR
// DVC 9Ø // VOL SAVEØ1,SAVEØ2,SAVEØ3
// LFD SEQDIN
// DVC 5Ø // VOL BAKVOL // LFD DISCOT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=DISC
// PARAM TYPE=FILE
/$
          FILE OLDNAME,REL,NEWNAME
          FILE OLDACCOUNTS,,NEWACCOUNTS
/*
/&
// FIN
```

The files are restored to disk with the new file name specified by the new-name parameter. For the first file, OLDNAME, DMPRST scratches any file having the name NEWNAME from the output disk. DMPRST then allocates a file large enough for the file OLDNAME. The old file OLDNAME bearing the new file name NEWNAME is then written to allocated space on the output disk. The same procedure is followed for the second FILE statement. Here, however, a comma is substituted for the missing allocation parameter. As in all restore operations, the files are restored in the same order they were dumped.

13.3.3.4. Restoring from Tape in the File Environment

If you are using magnetic tape as input, specify // PARAM IN=TAPE as your input parameter, and TAPEIN as the file name on your tape // LFD statement. When using more than one tape as your input, you can assign them in your control stream in one of two ways.

One way uses a single device assignment set (DVC - LFD sequence), having a file name of TAPEIN. Included in this set are // VOL statements listing the volume serial numbers of the restore operation's input tapes.

Another way is to supply a separate device assignment set for each tape. Here, a separate device is assigned for each tape volume. Again, the file name on your // LFD statement is TAPEIN. However, the numbers 01 through 99 are attached to each tape file name, following the first (i.e., FILE // LFD TAPEIN01).

NOTE:

If you perform a tape dump operation using separate device assignment sets and then restore by using a single device assignment set, you will receive an error message from data management. The method of assigning tapes used in the dump operation should also be used in the related restore operation.

Since you are executing DMPRST in the file environment, include the // PARAM TYPE=FILE parameter in your job stream. Also, supply the appropriate FILE statements as embedded data. Remember, files must be restored in the same order they were dumped. If you want to restore the entire volume of active files, use // PARAM TYPE=FILE,ALL.

The following examples show restore operations using magnetic tape.

Example 1: Multiple-file restore from single-volume tape file

```
// JOB TFILRST1
// DVC 2Ø1// LFD PRNTR
// DVC 9Ø // VOL PAYØØ2 // LFD TAPEIN
// DVC 5Ø // VOL DISKØ1 // LFD DISCOT
// EXEC DMPRST
// PARAM IN=TAPE
// PARAM OUT=DISC
// PARAM TYPE=FILE
/$
         FILE CREDT,ABS
         FILE ASSETS,REL
/*
/&
// FIN
```

The first file, CREDIT, is restored to the same absolute extents that it originally occupied. This function is controlled by the allocation parameter, ABS. The second file, ASSETS, is restored to a different location on disk by the REL allocation parameter.

Example 2: Restoring only active files from a single-volume tape file

```
// JOB TFILRST1
// DVC 2Ø // LFD PRNTR
// DVC 9Ø // VOL PAYØØ2 // LFD TAPEIN
// DVC 5Ø // VOL DISKØ1 // LFD DISCOT
// EXEC DMPRST
// PARAM IN=TAPE
// PARAM OUT=DISC
// PARAM TYPE=FILE,ALL
/*
/&
// FIN
```

Here, the entire volume of active files (entries in VTOC) are RESTORED.

13.3.4. Copying Files in a Single Disk Environment

You can make copies of a file on the same disk using the new-name parameter of the FILE statement. Here, the copy you create is written on the same disk as the original file. The new file, however, is assigned a new file name.

Since the same disk is used as both the input and output volume, the same device must be assigned having the // LFD names of DISCIN and DISCOT. Also, specify // PARAM IN=DISC and // PARAM OUT=DISC.

The new-name parameter of the file statement must be specified when you're copying files on the same disk, since two files with the same name can't exist on the same disk volume. With the new-name parameter, you can copy the file and give it a new name.

For example, suppose you are copying the MYFILE file on your disk. In order to copy MYFILE, specify the name MYFILE02 as the new-name parameter on the FILE statement.

In this case, DMPRST first scratches any file having the name MYFILE02 from the disk. Next, DMPRST allocates a file large enough for MYFILE. The old file, MYFILE, is then copied to the allocated space and given the new name, MYFILE02. Now, two identical files having different file names exist on the disk.

The following example shows a single volume file copy operation:

```
// JOB FILCOPY
// DVC 2Ø // LFD PRNTR
// DVC 5Ø // VOL DISK01 // LFD DISCIN
// DVC 5Ø // VOL DISK01 // LFD DISCOT
// EXEC DMPRST
// PARAM IN=DISC
// PARAM OUT=DISC
// PARAM TYPE=FILE
/$
            FILE MYFILE,,MYFILEØ2
/*
/&
// FIN
```

In this example, LFD names DISCIN and DISCOT are still required, even though you are using the same volume.

13.4. CHECKING FOR FILE EXPIRATION DATE

The DMPRST routine automatically checks for file expiration dates on your output volume, thus saving you the time and expense of keeping outdated files. DMPRST checks the file expiration date by comparing that date with the system date. If the date has not expired, a message is displayed that asks you to process the file (ignoring the date) or terminate the job. The expiration date function is used in both volume and file modes when copying or restoring files.

To suppress the file expiration date checking feature, you must include the // PARAM NOEXPCK statement.

The following example shows a typical control stream using the expiration date checking function in the file mode:

```
// JOB DSKRST
// DVC 2Ø // LFD PRNTR
// DVC 9Ø // VOL PAYØØ3 // LFD TAPEIN
// DVC 5Ø // VOL DISKØ1 // LFD DISCOT
// EXEC DMPRST
// PARAM IN=TAPE
// PARAM OUT=DISC
// PARAM TYPE=FILE
/$
      FILE TAXES
      FILE CREDITS
      FILE.P MAST
/*
/&
// FIN
```

In this example, the file expiration date function is done automatically since we omitted the // PARAM NOEXPCK statement. In this tape-to-disk restore operation, we specified the FILE PREFIX statement, which will restore all files having the prefix MAST.

13.5. EXECUTING DMPRST IN AN INTERACTIVE ENVIRONMENT

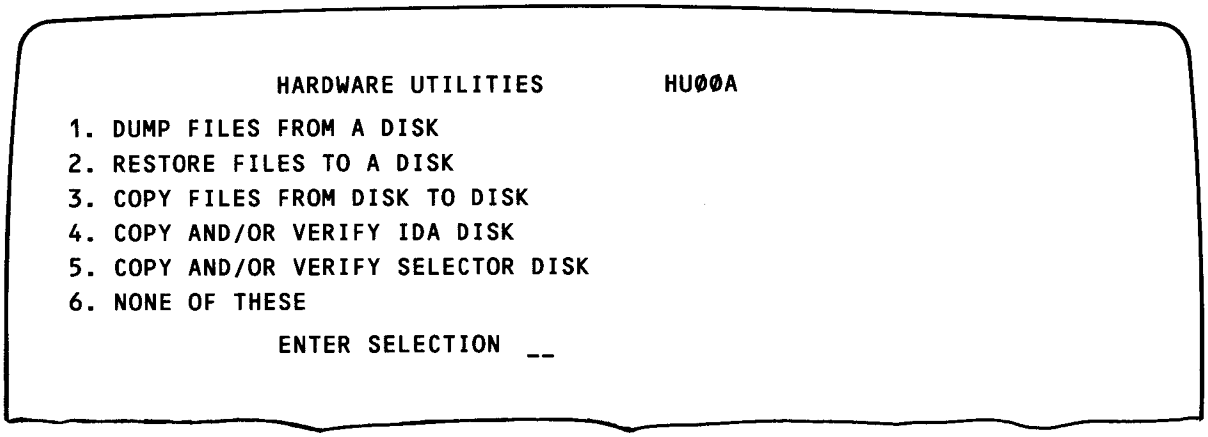
Interactive processing of the DMPRST routine consists of a question and answer session (dialog) using a UNISCOPE 100, UNISCOPE 200, or UTS 400 terminal. The procedures for executing the DMPRST routine are the same, regardless of which type of terminal you use, except for a few adjustments necessitated by differences in the terminal keyboards. With the UTS 400 terminal, for example, use the XMIT key to transmit work screens and function keys 13 and 14 to obtain and exit help screens. For UNISCOPE 100 and 200 terminals, use the TRANSMIT key to transmit work screens. To use function key 13 or 14, you must simulate it in the following manner. Press the MESSAGE WAITING key, then alphabetic F, then the pound sign (#), and then the number 13 or 14.

The answers you give during the dialog tell the DMPRST routine what function to do, assign the necessary input and output devices, and execute DMPRST in the file mode. The DMPRST routine supplies default values whenever possible. When you use the dialog, your entries are checked and any errors detected are blinked on the screen with explanatory messages. For example, invalid entries such as unsupported device types and misspellings of YES or NO are blinked.

The following examples show the DMPRST routine for a UTS 400 terminal. When using a UNISCOPE 100 or 200 terminal, you must adjust the instructions in the manner just discussed.

Help screens are provided for all work screens. To obtain help for any screen, press function key 13. To return to the work screen, press function key 14 or the XMIT key. If multiple help screens are being displayed, press the XMIT key after each help screen. When the last help screen is displayed, press the XMIT key or function key 14; the current work screen is redisplayed.

To conduct an interactive dialog with the DMPRST routine, you must first log onto the system by entering the LOGON command. After you have successfully logged on, key in HU in system mode and then press the XMIT key. After you press the XMIT key, the following menu screen appears (Figure 13-1):



```
                HARDWARE UTILITIES          HUØØA
   1. DUMP FILES FROM A DISK
   2. RESTORE FILES TO A DISK
   3. COPY FILES FROM DISK TO DISK
   4. COPY AND/OR VERIFY IDA DISK
   5. COPY AND/OR VERIFY SELECTOR DISK
   6. NONE OF THESE
                ENTER SELECTION __
```

Figure 13-1. HARDWARE UTILITIES Menu Screen

Depending on the number you select, an appropriate set of screens is displayed. You simply enter the requested information on the screen and DMPRST does the rest.

NOTE:

After you transmit the menu screen, an informational message is displayed on your terminal screen, indicating that an interactive job has been initiated. Press the XMIT key to continue.

The next three subsections (13.5.1 - 13.5.3) show typical examples of the DMPRST functions: copy, dump, and restore.

NOTE:

Regarding the screens, all the default values are shaded while the user entries are shown in reverse type (white characters on a black background).

13.5.1. Performing a Disk Copy Operation

During a disk copy operation, you are copying the contents of a disk to another disk of the same type. You perform a disk copy by selecting 3 on the menu screen. Up to three screens will be displayed. They are:

- Screen 1 (HU12)

  Asks you to specify the input and output device information.

- Screen 2 (HU16)

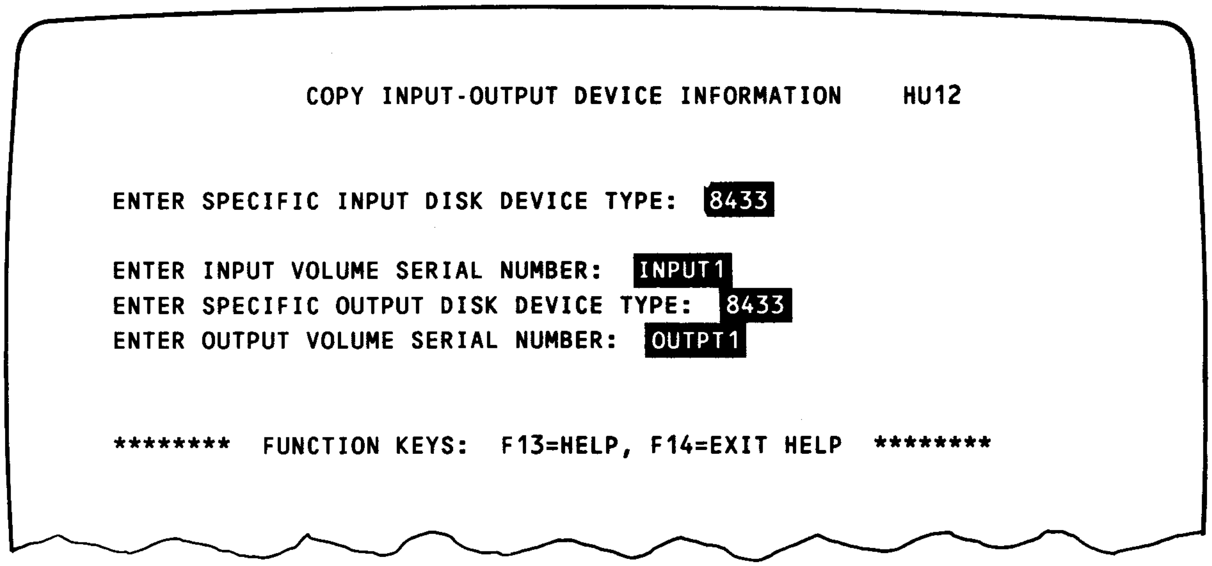
  Asks you to specify the file options that are in effect.

- Screen 3 (HU17)

  Appears only if an individual file is being copied and it asks you for the file names, relocation option, and rename specifications.

Now, let's see a typical disk copy operation.

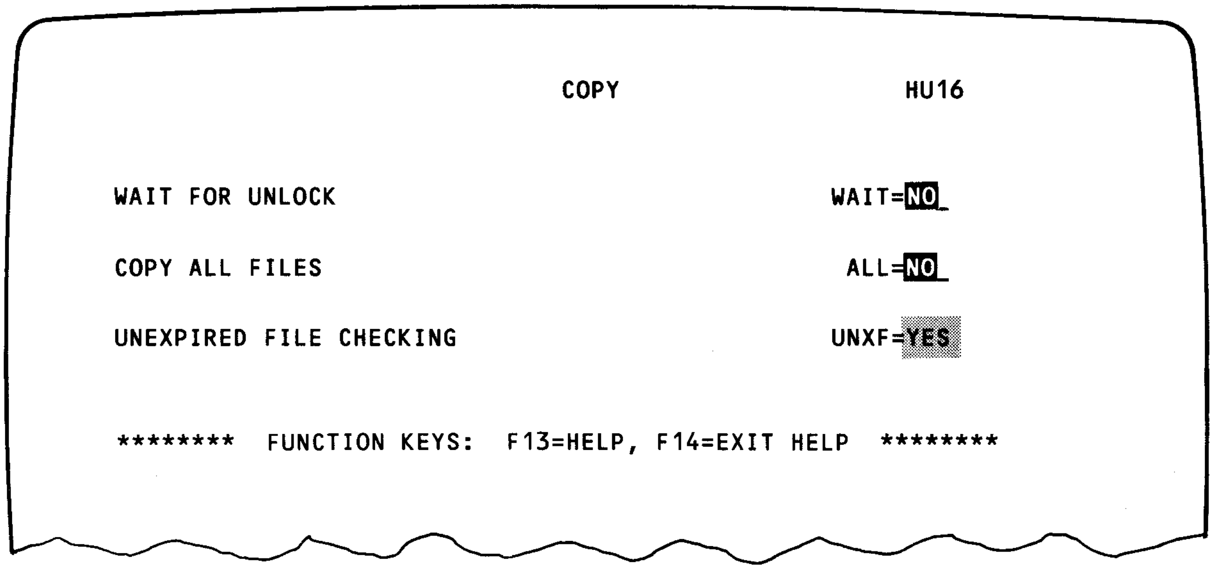
Screen 1 (HU12): Defining Our Input and Output Devices



```
                COPY INPUT-OUTPUT DEVICE INFORMATION    HU12

      ENTER SPECIFIC INPUT DISK DEVICE TYPE: 8433

      ENTER INPUT VOLUME SERIAL NUMBER: INPUT1
      ENTER SPECIFIC OUTPUT DISK DEVICE TYPE: 8433
      ENTER OUTPUT VOLUME SERIAL NUMBER: OUTPT1


      ********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

Looking at screen 1, we see that both our input and output device type is 8433, our input volume serial number is INPUT1, and our output volume serial number is OUTPT1. Since we don't need any help, press the XMIT key.

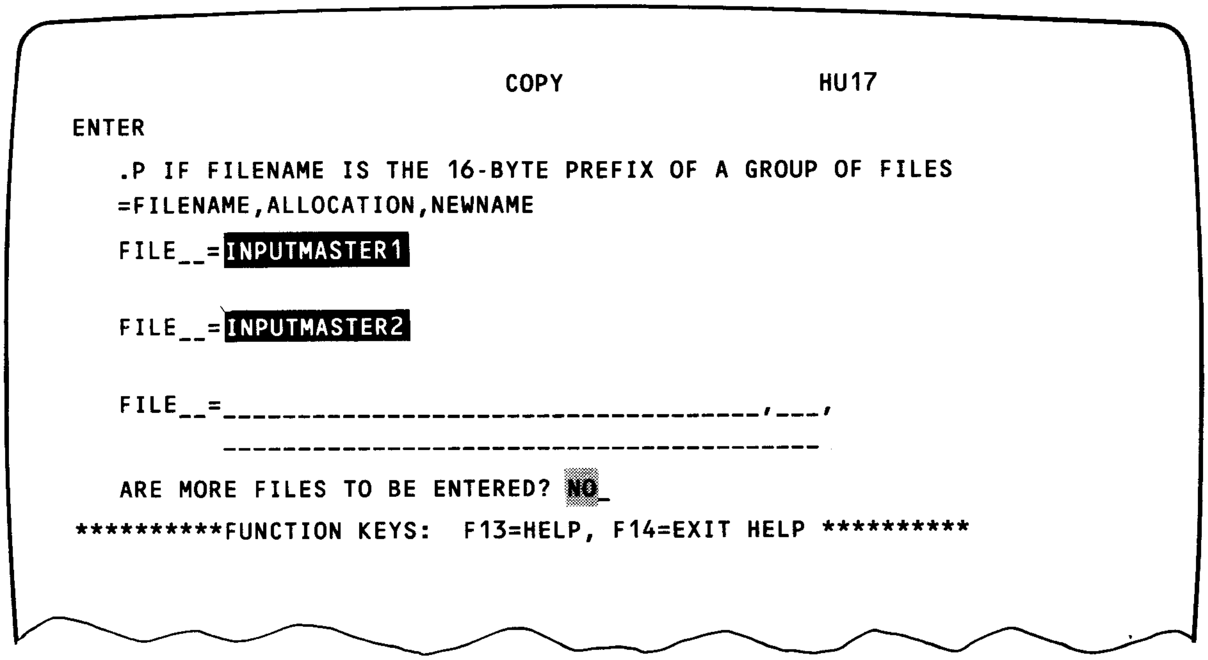
Screen 2 (HU16): Indicating File Options



```
                              COPY                 HU16


    WAIT FOR UNLOCK                                WAIT=NO_

    COPY ALL FILES                                  ALL=NO_

    UNEXPIRED FILE CHECKING                        UNXF=YES


    ********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

On screen 2, we entered NO, overriding the default of YES for the WAIT FOR UNLOCK option (indicating whether the file is unavailable to be locked), skip the file, and continue to the next one. Any file skipped has its file name listed on the printer. In a subsequent run, we would copy all the unavailable files (all file names printed). Since we do not want to copy all the files, we overwrite the default (YES) with NO. This provides for file selection. Finally, because we want UNEXPIRED FILE CHECKING, we simply press the XMIT key (assume the default YES). This option prevents copying over on the output volume any files whose dates have not expired. (The expiration date was generated at file creation time by using the // LBL job control statement.)

Screen 3 (HU17): Specifying File Names



```
                            COPY               HU17
  ENTER
     .P IF FILENAME IS THE 16-BYTE PREFIX OF A GROUP OF FILES
     =FILENAME,ALLOCATION,NEWNAME
     FILE__=INPUTMASTER1

     FILE__=INPUTMASTER2

     FILE__=_________________________________________,___,
            _________________________________________
     ARE MORE FILES TO BE ENTERED? NO_
  **********FUNCTION KEYS:  F13=HELP, F14=EXIT HELP **********
```

Here, we specified our file names as INPUTMASTER1 and INPUTMASTER2. Any other files residing on our volume are not copied. You also have the option of specifying a file prefix rather than the entire file name. To indicate a prefix, enter .P immediately before the equal sign (=); enter the prefix name (up to 16 characters) immediately after the equal sign. Since we don't need to copy our file to a specific area on our output volume, we ignore the allocation and rename options. Since we are copying only two files, we take the default (NO) to the query ARE MORE FILES TO BE ENTERED? and press the XMIT key.

NOTES:

1. *If you answered YES to COPY ALL FILES on screen 2 (HU16), then screen 3 (HU17) is not displayed and the copy operation starts immediately after you press the XMIT key.*

2. *For a brief description of the file allocation parameters, see Table 13-4. For a complete description, see 13.3.3.1.*

Table 13-4. File Allocation Parameters

Parameter	Function
ABS	Locates a file on the same absolute extents. If a file cannot be allocated to the same extents, an error message is displayed, the file is bypassed, and processing continues.
REL	Requests that the file be relocated to a file the same size as the original
LOG	Relocates a file and deletes all unassigned space
PRE	Relocates a file in a space that was preallocated
Omit parameters	Standard restore processing: DMPRST scratches the file from the output disk and makes up to three attempts to reallocate it on the disk.

13.5.2. Performing a Dump Operation

As we mentioned earlier, when you perform a dump operation you also must do a restore operation. The dump operation copies the contents of your disk volume to a magnetic tape, while the restore operation copies the information back to your original disk volume from tape. We'll discuss the dump opertion here. See 13.5.3 for the restore operation.

You perform a dump operation by selecting 1 on the menu screen. Up to four screens are displayed. They are:

- Screen 1 (HU18)

  Asks you for the input device information.

- Screen 2 (HU20)

  Asks you to specify your volume serial numbers.

- Screen 3 (HU22)

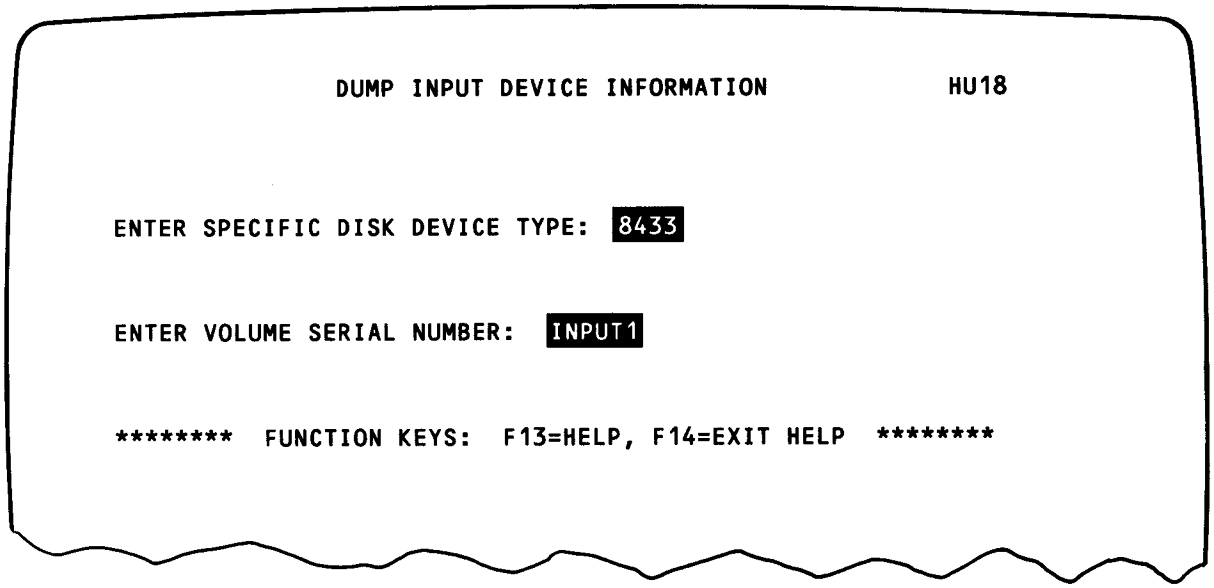
  Asks you to specify the file options.

- Screen 4 (HU23)

  Appears only if individual files are being dumped rather than dumping all the files on the volume; and it asks you for the file names.

Now, let's do a dump operation.

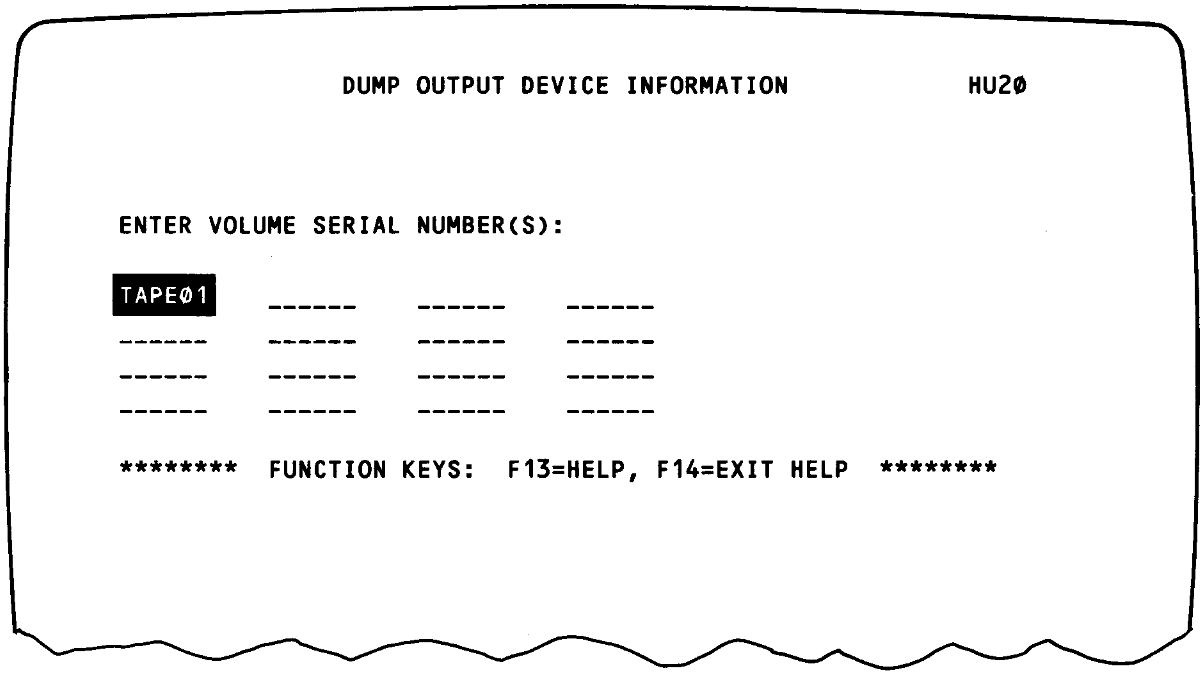
Screen 1 (HU18): Defining Our Input Device



```
                  DUMP INPUT DEVICE INFORMATION            HU18


   ENTER SPECIFIC DISK DEVICE TYPE: 8433


   ENTER VOLUME SERIAL NUMBER: INPUT1


   ********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

On screen 1, we indicated that our disk device type is an 8433 and its volume serial number is INPUT1. No help is required, so press the XMIT key.

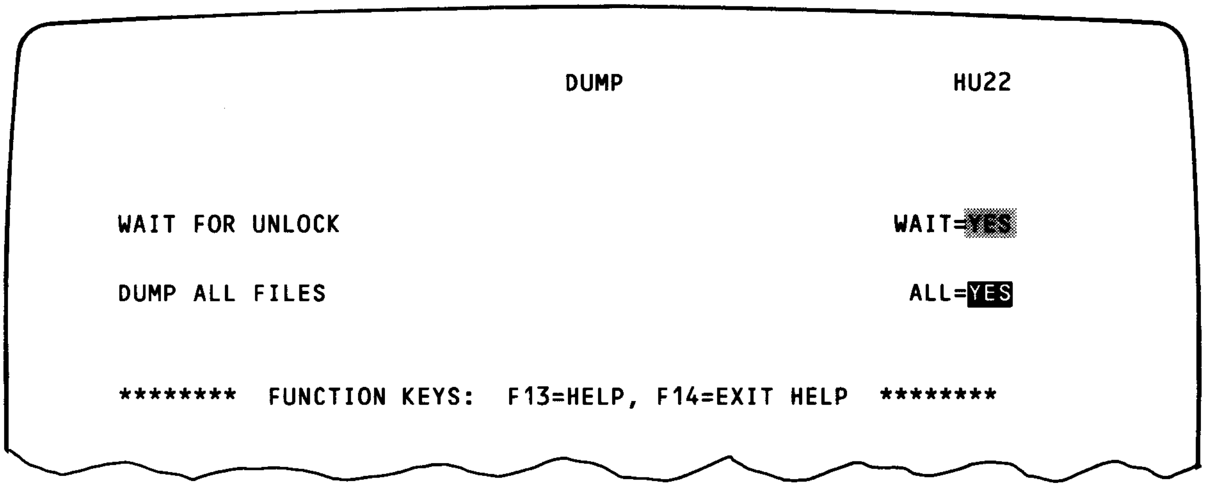
Screen 2 (HU20): Specifying Our Output Volumes



```
                 DUMP OUTPUT DEVICE INFORMATION          HU2Ø


   ENTER VOLUME SERIAL NUMBER(S):

   TAPEØ1   ------   ------   ------
   ------   ------   ------   ------
   ------   ------   ------   ------
   ------   ------   ------   ------

   ********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

On screen 2, we entered volume serial number TAPE01 for out output. When you specify volume serial numbers, the first entry contains your first volume serial number, the second your second volume serial number, and so on. Up to 16 volume serial numbers can be specified. Since no help is required, press the XMIT key.

Screen 3 (HU22): Indicating File Options



```
                              DUMP                      HU22


   WAIT FOR UNLOCK                                  WAIT=YES

   DUMP ALL FILES                                    ALL=YES

   ********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

On this screen, we took the default value YES for WAIT FOR UNLOCK (waiting for our file to be available) and entered YES for DUMP ALL FILES. Press the XMIT key.

NOTES:

1. *If certain files (rather than all the files) on the volume are to be dumped, then screen 4 (HU23) is displayed, asking you for the file names.*

2. *When you dump all files, only active files are dumped. (This is the same as dumping the entire volume.) Active files are defined as those having entries in the VTOC and are not job temporary files (run libraries and scratch files).*

13.5.3. Performing a Restore Operation

After you have successfully completed a dump operation, you must use the restore operation to copy the files back to disk. You execute the restore operation by selecting 2 on the menu screen. Up to four screens are displayed.

These screens are:

- Screen 1 (HU03)

  Asks you for your input volume serial numbers.

- Screen 2 (HU05)

  Asks you for the output device type, which must be the same device type as the original device that was dumped.

- Screen 3 (HU07)

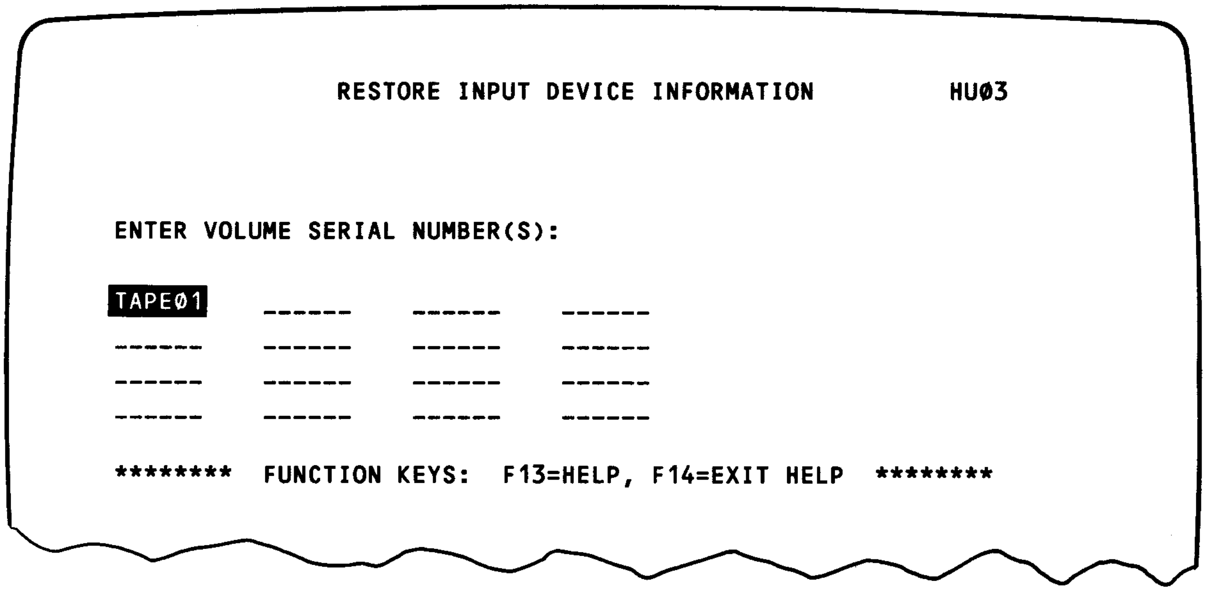
  Asks you to specify the file options.

- Screen 4 (HU08)

  Appears only if you are restoring individual files rather than all files on the volume; and it asks you to specify the file names.

Now let's do a typical restore operation. Remember, in our dump operation we used magnetic tape as our medium. This same tape is now used as input for our restore operation.

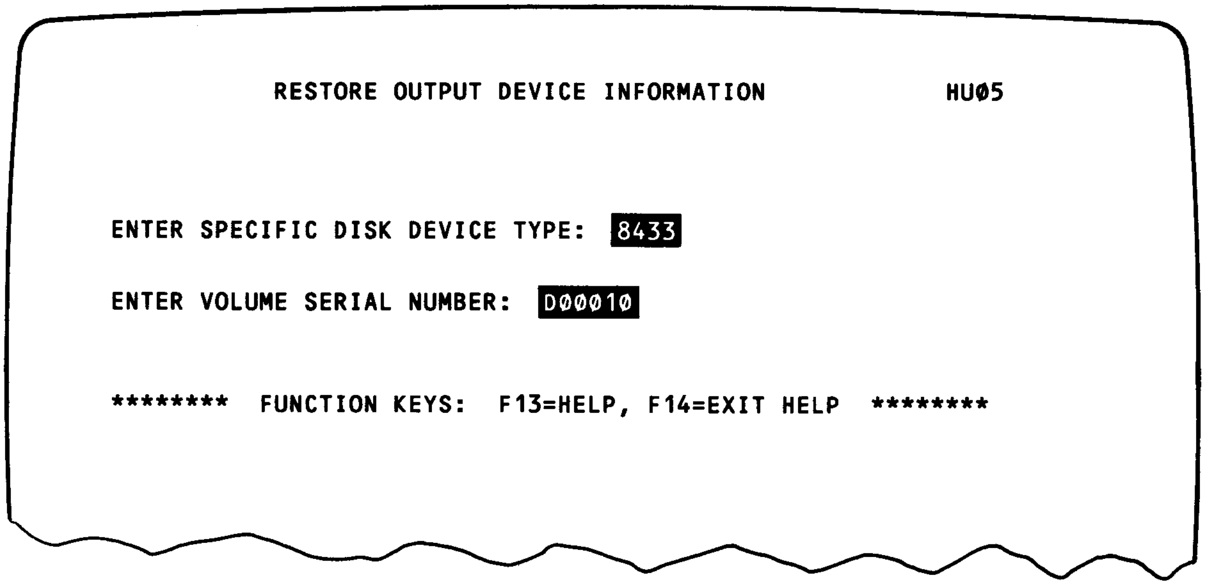
Screen 1 (HU03): Specifying Our Input Volume Serial Number



```
                  RESTORE INPUT DEVICE INFORMATION          HUØ3


    ENTER VOLUME SERIAL NUMBER(S):

    TAPEØ1   ------   ------   ------
    ------   ------   ------   ------
    ------   ------   ------   ------
    ------   ------   ------   ------

    ********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

On this screen, we entered TAPE01, which identifies the volume we are using as input for our restore operation. When you specify more than one volume serial number, the first entry contains your first volume serial number, the second your second volume serial number, and so on. Up to 16 volume serial numbers can be specified. Since no help is needed, press the XMIT key.

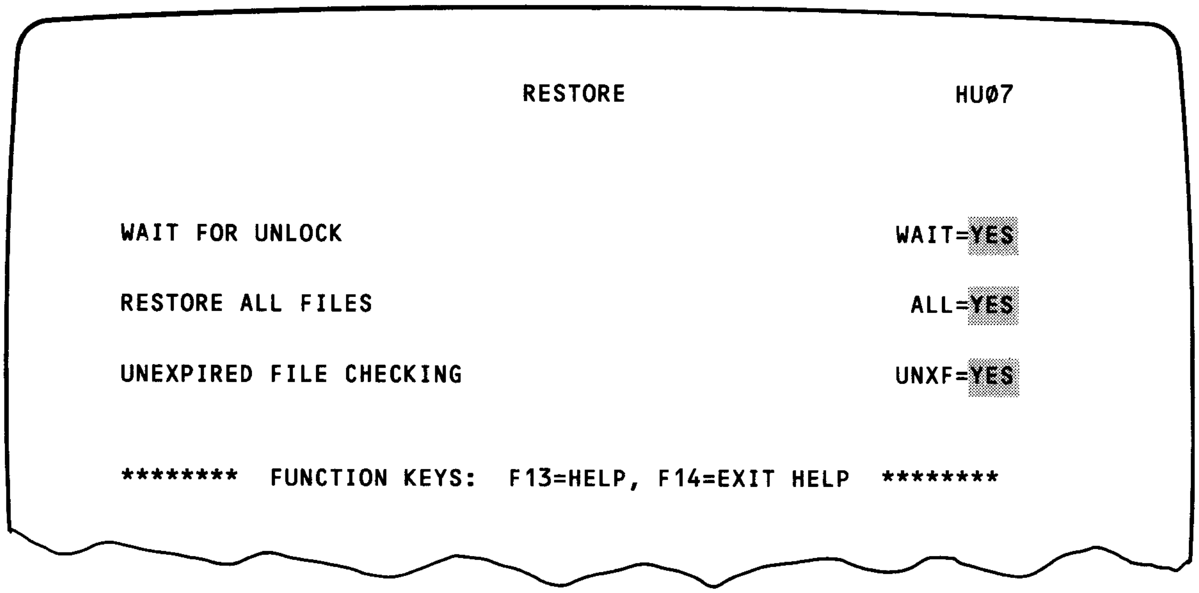
Screen 2 (HU05): Specifying Our Output Device Type



```
               RESTORE OUTPUT DEVICE INFORMATION            HUØ5


    ENTER SPECIFIC DISK DEVICE TYPE: 8433

    ENTER VOLUME SERIAL NUMBER: DØØØ1Ø


    ********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

We indicated our device type is 8433 and the volume serial number is D00010. Press the XMI~~T key~~.

Screen 3 (HU07): Indicating the File Options



```
                              RESTORE                  HUØ7


    WAIT FOR UNLOCK                                WAIT=YES

    RESTORE ALL FILES                               ALL=YES

    UNEXPIRED FILE CHECKING                        UNXF=YES

    ********  FUNCTION KEYS:  F13=HELP, F14=EXIT HELP  ********
```

On this screen, we are taking the default values, specifically, YES for all file options. Our result is:

- DMPRST will not start processing until the file becomes available;
- all files are restored; and
- the file expiration date is checked on the output volume. If the date has not expired, then don't process the file.

Press the XMIT key.

Since we are restoring all the files on the volume, no other screens will be displayed. The restore operation is executed and, upon completion, a message indicating that the job is terminated is displayed at our terminal.

14. The Diskette Utility (CREATE)

14.1. FUNCTIONS

The diskette utility provides a way for you to create and maintain files on diskettes by using the system console as the input device. You can, for example, place job control streams, normally kept on punched cards, onto diskettes for easier access and maintainability. The record images that you place in the created diskette files can be altered and sorted; added to or deleted from; or completely overwritten by using the diskette utility.

To use the diskette utility, you first run a canned job control stream named WRT, and then communicate with the diskette utility via the system console. If the RUN command used to call WRT indicates that the diskette file being addressed is to be sorted, WRT will cause the execution of three programs: CREATE, CPYFLE, and SORT. CREATE is the utility that allows you to build and maintain diskette files, CPYFLE transfers an image of the created diskette file into a SYSRES work file for sorting, and SORT performs the actual sorting operation. If sorting is not called for, only CREATE is executed.

There must be four cylinders of space available on your SYSRES volume to support the file maintenance functions of the utility. These cylinders provide space for file sorting and copying and are completely scratched after the utility has completed processing. If this space is not available, the job WRT will not be run.

14.2. USING THE DISKETTE UTILITY

The first step in using the diskette utility is to create, or access, the required file space on the diskette device. You do so by running the canned control stream, named WRT, that identifies the file and also specifies the processing options you want performed on that file. The format of the RUN command used to call the canned job control stream is:

```
RV WRT,[,D={n  }],V=vsn[,F={filename}][,B={n }][,O={Y}][,S={Y}][,E={Y}]
           {130}            {IMAGE   }      {52}     {N}     {N}     {N}
      [,R={Y}][,A={Y}]
          {N}      {N}
```

You use the D=, V=, and F= keywords to locate an already existing file or to allocate a new file. The D= keyword identifies the diskette device required. The default is 130. The V= keyword specifies the volume serial number of the diskette to be used. You must make a 1- to 6-character entry for this keyword parameter. The F= keyword specifies the file name of your file. If you do not enter a file name, it is defaulted to IMAGE.

Your file automatically has 52 sectors assigned to it. If your file requires more or fewer sectors, you can specify the required amount with the B= keyword. The maximum number of sectors you can specify is limited only by the total number of sectors available on the diskette.

The remaining keyword parameters are all used to specify the function you wish to perform on the file. You must specify the appropriate keyword parameter with the Y (yes) option selected to have the function performed. The default is N (no) in every case. The functions of the keyword parameters are:

- O Open an existing file.
- S Scratch all or part of a file.
- E Extend (add to) an existing file.
- R Recreate (overwrite) an existing file.
- A Sort the file in numeric order.

Suppose, for example, you want to create a file on a diskette but do not have any record information to go into the file and want to reserve space on a diskette other programmers are also using. Assume the volume serial number of the diskette is TSDSDT; you wish to reserve 78 sectors; and the name you want to assign to your file is JOB1. The RUN command you enter is:

```
RV WRT,,V=TSDSKT,F=JOB1,B=78
```

The job control stream effectively run is:

```
// JOB WRT*
// DVC 13Ø*
// VOL TSDSKT
// EXT SQ,C,Ø,BLK,(128,78)
// LBL JOB1
// LFD DIMAGE*
// EXEC CREATE*
```

*Supplied by the canned job control stream

After you enter the RUN command, several lines of text relative to your job are displayed on the system console. The first two lines (prefixed with JC) give your diskette device assignment and volume serial number, and inform you that the system is executing the CREATE program. The third line states the CREATE program writes JCL streams to diskette files. The fourth line is an informational line stating that, if you enter an M in column 54, the format for columns 54 through 80 will be displayed and you will be able to insert record information into those columns. The next line asks whether the special character space value should be set. You respond to this by either entering any nonblank character or making no entry and transmitting a blank line. If you make an entry, the utility will suppress all leading spaces on any record containing that character. For example, if you enter a pound sign (#), any record containing a pound sign will have any leading spaces suppressed, even if you enter the record with them.

You can now begin to enter your record data into the file. The data is entered in response to a series of queries displayed on the system console. Each query consists of a 2-digit hexadecimal message number; followed by a question mark; a C; four Ns for the record number; and the column format for that line. You respond with the message number and one of the following commands, depending upon which function you wish performed:

- A To put a record into a file
- C To identify a line you wish to change
- R To change a line identified
- I To have a previously issued C command ignored
- L To list the contents of a file
- F To close a particular file
- E To terminate the utility
- D To display the available commands and their functions

Getting back to our example, since you do not intend to put any data into the file, terminate the utility by entering an E command.

Suppose you are now ready to begin inserting records into the file. To open the file and to have processing begin at the first record, enter the following run command:

```
RV WRT,,V=TSDSKT,F=JOB1,O=Y
```

You can now start adding your records. The record images you put into the diskette files are handled as if they are sequential cards. Therefore, you must number your records (lines). The numbering enables you to more easily locate and alter records and insert new records into a file. The numbering of the records is also essential for sorting. The diskette utility uses the OS/3 independent sort/merge routine to sort a diskette file after the CREATE program is terminated.

To add a record to a file, you enter the requesting message number, an A, then the record number, and finally the record information itself. Do this until all the information is in the file.

If you make an error in one of the lines, you enter a C followed by the line number of the line in error. The system responds by displaying the present contents of the line. To change the line, enter an R followed by the correct line information. If after viewing the displayed line you do not wish to change the line, enter an I, and the request for a change is ignored. After the information is in the file, enter an E to terminate the utility.

Suppose you wish to add new information to the file, sort the new information, and merge it with the original file. To do so, enter this command:

```
RV WRT,,V=TSDSKT,F=JOB1,O=Y,E=Y,A=Y
```

The job control stream effectively produced is the same as the previous example except the // EXT statement will not be produced and the // LFD card will have the EXTEND parameter specified.

If you wish to recreate a file, that is you want to overwrite all the record information in the file, you can do so by entering the following command:

```
RV WRT,,V=TSDKST,F=JOB1,O=Y,R=Y
```

Enter the data just as you would for a newly created file. The file pointers are reset when the file is opened.

The following command can be used to completely scratch a file from the diskette:

```
RV WRT,,V=TSDSKT,F=JOB1,S=Y
```

The effectively produced job control stream is the same as for the first example except the // EXT statement is not produced and a // SCR DIMAGE is included.

To obtain a listing of the contents of a file, enter the L command. The file is displayed five lines at a time; after each five lines, you are given the option to display the next five lines or terminate the listing.

To close a file without terminating the diskette utility, enter the F command. The system responds by asking whether processing is to continue with the same file. If you respond with a yes, processing continues. If, however, you respond with a no, the diskette utility terminates.

As an added convenience, you can have the available diskette utility commands displayed on the system console by entering the D command.

14.3. DISKETTE INDEX SCAN

The diskette index scan allows access to the diskette index track from sectors C7 through C26 and provides the ability to create, alter, or re-create the VOL1/HDR1 index data set labels. The index scan is executed via a RUN command entered at the system console. The format for the RUN command is:

```
RV ISCAN,⎡,D=⎧n  ⎫⎤[,V=vsn]
         ⎣   ⎩130⎭⎦
```

You use the D keyword to identify the desired diskette device; the default is 130. The V keyword is used to specify the volume serial number of the diskette. If the parameter is not specified, the volume serial number will default to the volume serial number of the diskette mounted on the device specified in the D keyword.

After entering the RUN command, you are asked to set the special character space value. You respond by entering any nonblank character or transmitting a blank line. This functions as explained for a normal diskette utility operation. The following message then appears:

```
?SS=Ø7-26
```

Your response is to enter the sector number of the data set label to be manipulated. If you wish to terminate the index scan, enter END in response to this query. After the desired sector number is entered, the following template appears on the screen:

```
1       10        20        30        40        50
12345678901234567890123456789012345678901234567890
LBL. DATA----SET---ID.BKLH. BOE.. EOE.. B P MS.CDA
 E X I S T I N G  D A T A  I N  R E C O R D

        60        70        80

123456789012345678901234567890
TE.RLH.NXRSP    EDATE.V.EOD..
 E X I S T I N G  D A T A  I N  R E C O R D
```

See Table 14-1 for an explanation of the template fields. The template display is followed by:

```
?C=CHANGE,I=IGNORE
```

If, after examining the exisiting data in the record, you determine that no change is required, enter I. You will then be asked to enter another sector number. If C is entered, the template is displayed again with the data lines blank. You then enter the data you wish to appear in that index label. You must rewrite the entire record if you enter the C. Figure 14-3 is an example of a typical ISCAN operation.

Table 14-1. Disk Index Scan Record Template Fields

Field	Explanation
LBL.	Label ID
DATA	Data set name
BKLH.	Block/record length
BOE..	Address of first sector of data set
EOE..	Address of last sector of data set
B	Bypass data set
P	Write protect
M	Multivolume indicator
S	Volume sequence number
CDATE	Creation date
EDATE.	Expiration data
V.	Verify mark
EOD..	Address of next unused sector

14.4. DISKETTE UTILITY PROGRAMMING EXAMPLES

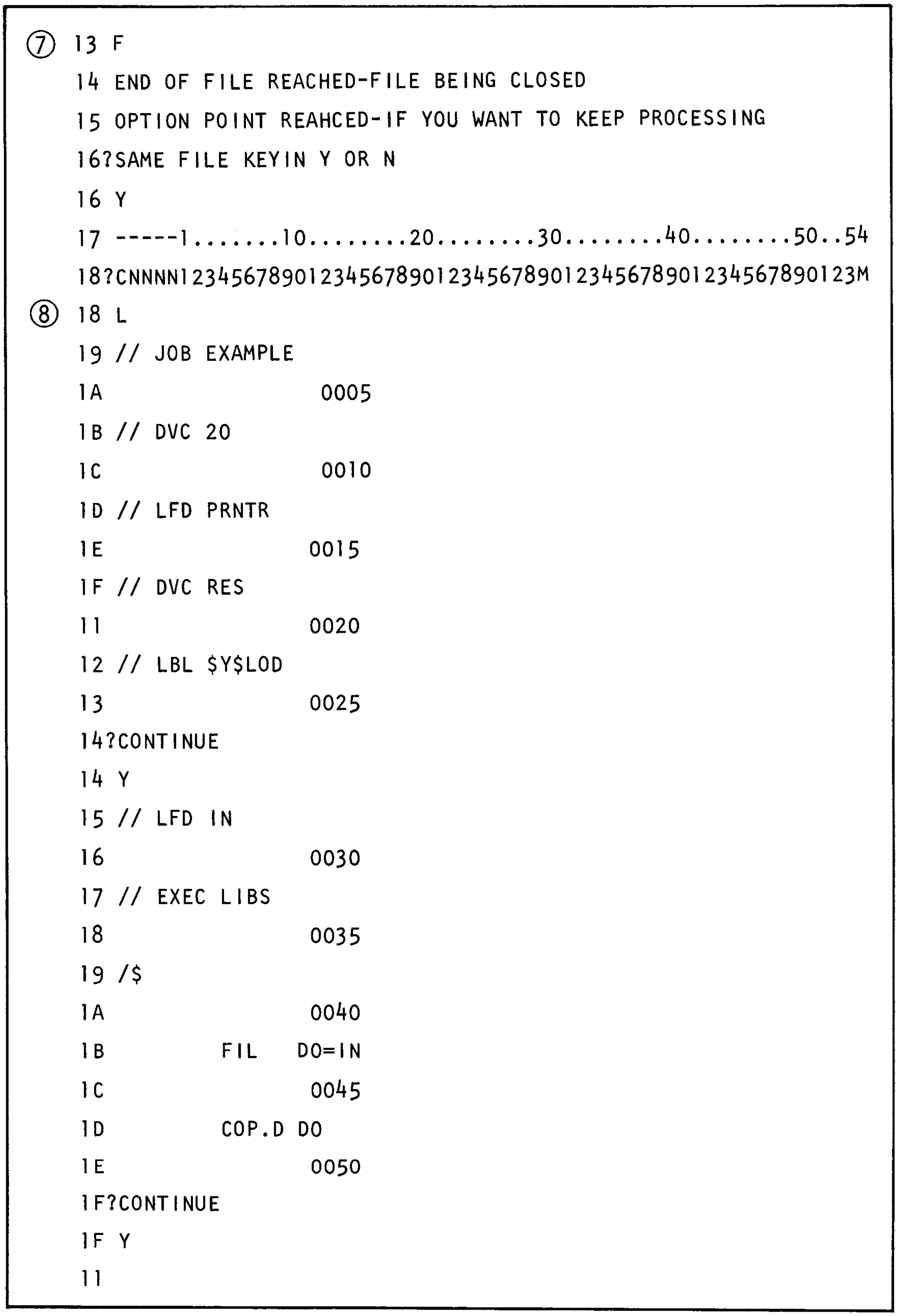
The following figures illustrate typical ways of using the diskette file creation utility. Figure 14-1 illustrates the commands and messages used to create and insert records into a file. This example also shows how to correct an error in the file. Figure 14-2 is an example of how new records can be inserted into a file, then sorted numerically.

```
① RV WRT,,D=130,V=VANBRO,F=TEST1,B=78
  11 JC06  USING   DEV=010 VSN=VANBRO
  12  JC01 JOB WRT     EXECUTING JOB STEP CREATE00 #001 12:29:07
  13  THIS PROGRAM WRITES JCL IMAGES TO DISKETTE FILE
  14  IF COL. M IS SPECIFIED YOU WILL GET COLS 54 THRU 80
  15?SET SPECIAL CHARACTER SPACE VALUE
② 15 #
③ 16 -----1.......10........20........30........40........50..54
④ 17?CNNNN1234567890123456789012345678901234567890123456789012
⑤ 17 A0005// JOB EXAMPLE
  18 -----1.......10........20........30........40........50..54
  19?CNNNN1234567890123456789012345678901234567890123456789012
  19 A0010// DVC 20
  1A -----1.......10........20........30........40........50..54
  1B?CNNNN1234567890123456789012345678901234567890123456789012
  1B A0015// LFD PRNTR
  1C -----1.......10........20........30........40........50..54
```

Figure 14-1. File Creation and Record Correction (Part 1 of 5)

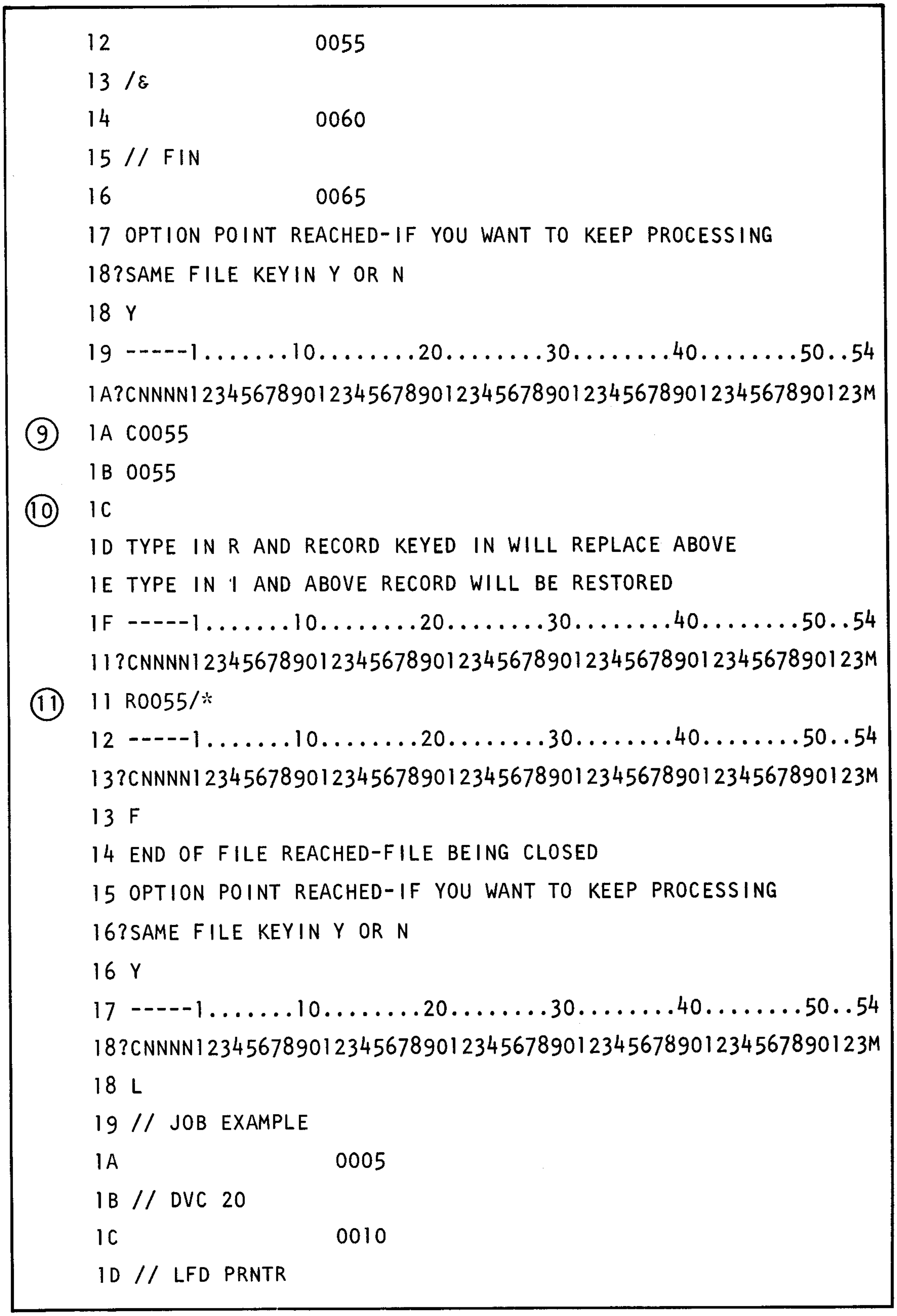
```
  1D?CNNNN12345678901234567890123456789012345678901234567890123M
  1D A0020// DVC RES
  1E -----1.......10........20........30........40........50..54
  1F?CNNNN12345678901234567890123456789012345678901234567890123M
  1F A0025// LBL $Y$LOD
  11 -----1.......10........20........30........40........50..54
  12?CNNNN12345678901234567890123456789012345678901234567890123M
  12 A0030// LFD IN
  13 -----1.......10........20........30........40........50..54
  14?CNNNN12345678901234567890123456789012345678901234567890123M
  14 A0035// EXEC LIBS
  15 -----1.......10........20........30........40........50..54
  16?CNNNN12345678901234567890123456789012345678901234567890123M

  16 A0040/$
  17 -----1.......10........20........30........40........50..54
  18?CNNNN12345678901234567890123456789012345678901234567890123M
  18 A0045         FIL   DO=IN
  19 -----1.......10........20........30........40........50..54
  1A?CNNNN12345678901234567890123456789012345678901234567890123M
  1A A0050         COP.D DO
  1B -----1.......10........20........30........40........50..54
  1C?CNNNN12345678901234567890123456789012345678901234567890123M
⑥ 1C A0055
  1D -----1.......10........20........30........40........50..54
  1E?CNNNN12345678901234567890123456789012345678901234567890123M
  1E A0060/&
  1F -----1.......10........20........30........40........50..54
  11?CNNNN12345678901234567890123456789012345678901234567890123M
  11 A0065// FIN
  12 -----1.......10........20........30........40........50..54
  13?CNNNN12345678901234567890123456789012345678901234567890123M
```

Figure 14-1. File Creation and Record Correction (Part 2 of 5)



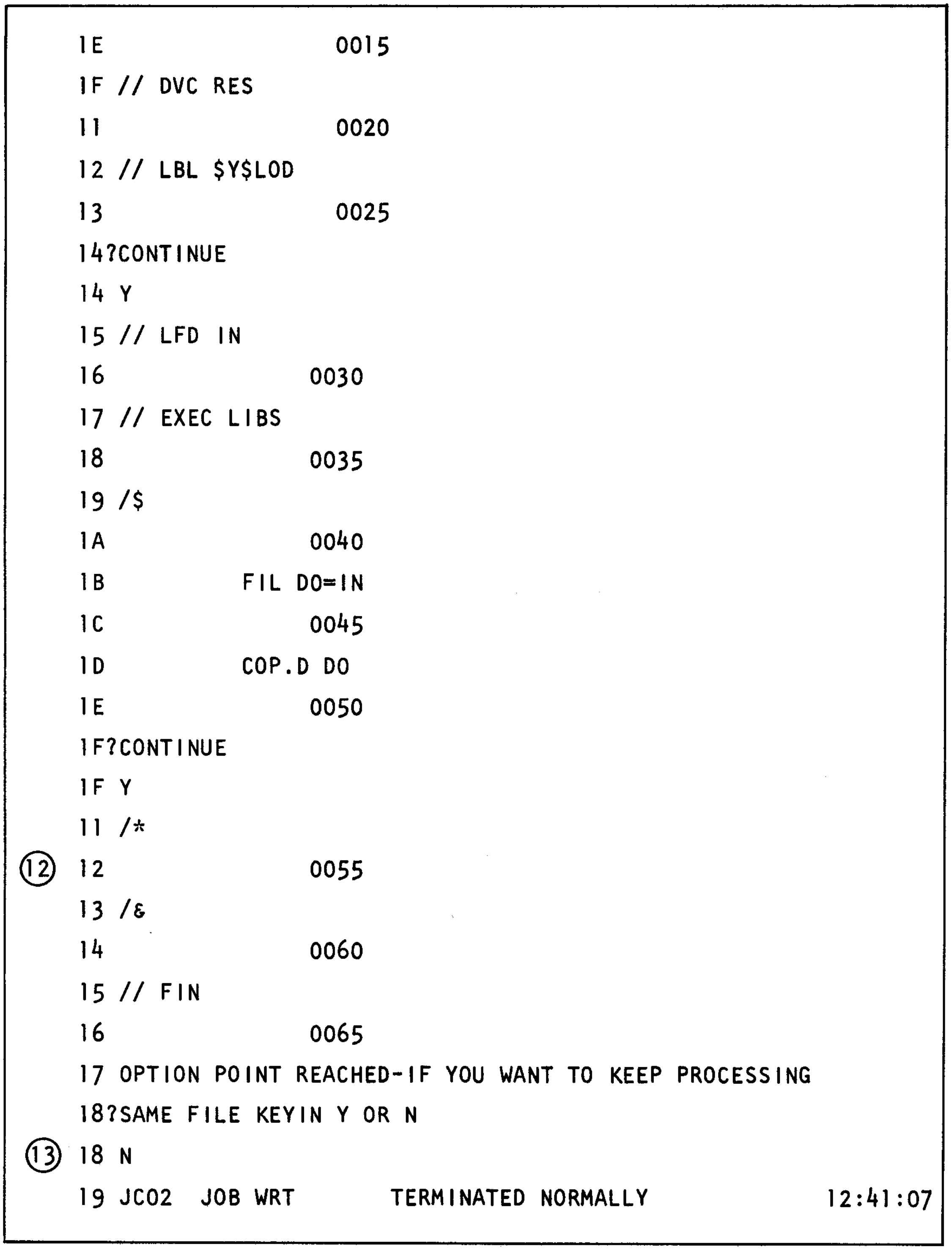
```
⑦ 13 F
  14 END OF FILE REACHED-FILE BEING CLOSED
  15 OPTION POINT REAHCED-IF YOU WANT TO KEEP PROCESSING
  16?SAME FILE KEYIN Y OR N
  16 Y
  17 -----1.......10........20........30........40........50..54
  18?CNNNN12345678901234567890123456789012345678901234567890123M
⑧ 18 L
  19 // JOB EXAMPLE
  1A                  0005
  1B // DVC 20
  1C                  0010
  1D // LFD PRNTR
  1E                 0015
  1F // DVC RES
  11                 0020
  12 // LBL $Y$LOD
  13                 0025
  14?CONTINUE
  14 Y
  15 // LFD IN
  16                 0030
  17 // EXEC LIBS
  18                 0035
  19 /$
  1A                 0040
  1B          FIL   DO=IN
  1C                 0045
  1D          COP.D DO
  1E                 0050
  1F?CONTINUE
  1F Y
  11
```

Figure 14-1. File Creation and Record Correction (Part 3 of 5)



```
    12                  0055
    13 /&
    14                  0060
    15 // FIN
    16                  0065
    17 OPTION POINT REACHED-IF YOU WANT TO KEEP PROCESSING
    18?SAME FILE KEYIN Y OR N
    18 Y
    19 -----1.......10........20........30........40........50..54
    1A?CNNNN1234567890123456789012345678901234567890123456789012 3M
⑨   1A C0055
    1B 0055
⑩   1C
    1D TYPE IN R AND RECORD KEYED IN WILL REPLACE ABOVE
    1E TYPE IN I AND ABOVE RECORD WILL BE RESTORED
    1F -----1.......10........20........30........40........50..54
    11?CNNNN12345678901234567890123456789012345678901234567890123M
⑪   11 R0055/*
    12 -----1.......10........20........30........40........50..54
    13?CNNNN12345678901234567890123456789012345678901234567890123M
    13 F
    14 END OF FILE REACHED-FILE BEING CLOSED
    15 OPTION POINT REACHED-IF YOU WANT TO KEEP PROCESSING
    16?SAME FILE KEYIN Y OR N
    16 Y
    17 -----1.......10........20........30........40........50..54
    18?CNNNN12345678901234567890123456789012345678901234567890123M
    18 L
    19 // JOB EXAMPLE
    1A                  0005
    1B // DVC 20
    1C                  0010
    1D // LFD PRNTR
```

Figure 14-1. File Creation and Record Correction (Part 4 of 5)



```
   1E                  0015
   1F // DVC RES
   11                  0020
   12 // LBL $Y$LOD
   13                  0025
   14?CONTINUE
   14 Y
   15 // LFD IN
   16                0030
   17 // EXEC LIBS
   18                0035
   19 /$
   1A                0040
   1B          FIL DO=IN
   1C                0045
   1D          COP.D DO
   1E                0050
   1F?CONTINUE
   1F Y
   11 /*
⑫ 12                0055
   13 /&
   14                0060
   15 // FIN
   16                0065
   17 OPTION POINT REACHED-IF YOU WANT TO KEEP PROCESSING
   18?SAME FILE KEYIN Y OR N
⑬ 18 N
   19 JC02  JOB WRT       TERMINATED NORMALLY          12:41:07
```

Figure 14-1. File Creation and Record Correction (Part 5 of 5)

Explanation for Figure 14–1:

① Initiates the diskette utility requesting DVC 130, VSN VANBRO, assigning a file name of TEST1, and requesting 78 sectors for the file

② Specifies special character to suppress leading spaces

③ System responds by displaying record column format.

④ Second line of system reply: Question mark requests a reply; C represents a command identifier, NNNN indicates record number field; 123...123 indicates record data columns; M indicates continuation column.

⑤ First record is being added to the file. Record number is 0005 and increment by 5 for ease of insertion of any new records.

⑥ Blank record accidentally written

⑦ End-of-file (F) command entered

⑧ List (L) command entered. System responds with display of file contents.

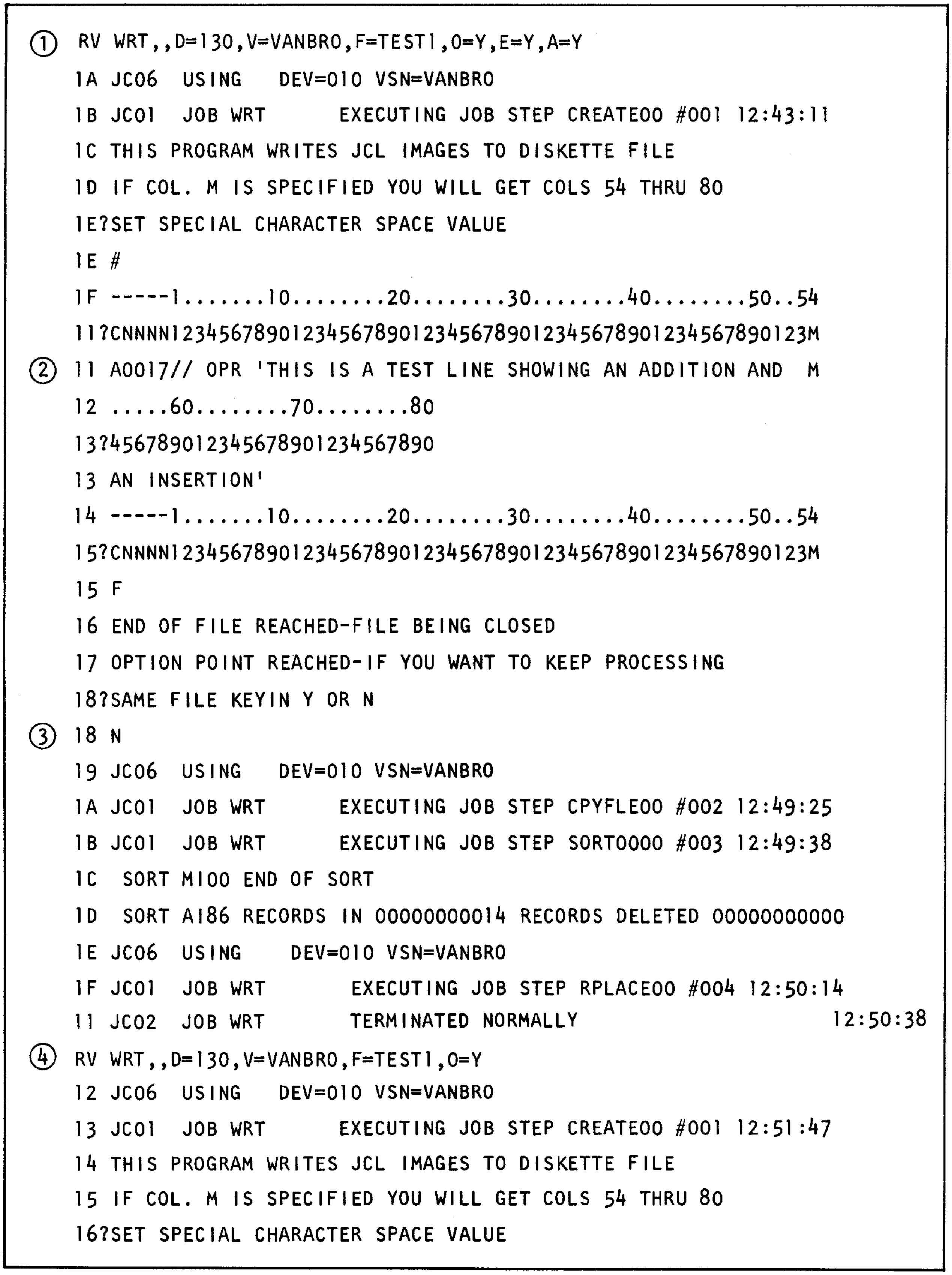
⑨ Change (C) command entered with line number of record to be changed

⑩ Lines 1B and 1C display the present contents of record 0055, which is empty.

⑪ Replace (R) command entered with line number and new record data

⑫ Display shows new record data properly inserted.

⑬ Response to terminate diskette utility



```
① RV WRT,,D=130,V=VANBRO,F=TEST1,O=Y,E=Y,A=Y
  1A JC06  USING   DEV=010 VSN=VANBRO
  1B JC01  JOB WRT      EXECUTING JOB STEP CREATE00 #001 12:43:11
  1C THIS PROGRAM WRITES JCL IMAGES TO DISKETTE FILE
  1D IF COL. M IS SPECIFIED YOU WILL GET COLS 54 THRU 80
  1E?SET SPECIAL CHARACTER SPACE VALUE
  1E #
  1F -----1.......10........20........30........40........50..54
  11?CNNNN1234567890123456789012345678901234567890123M
② 11 A0017// OPR 'THIS IS A TEST LINE SHOWING AN ADDITION AND  M
  12 .....60........70........80
  13?4567890123456789012345678901234567890
  13 AN INSERTION'
  14 -----1.......10........20........30........40........50..54
  15?CNNNN1234567890123456789012345678901234567890123M
  15 F
  16 END OF FILE REACHED-FILE BEING CLOSED
  17 OPTION POINT REACHED-IF YOU WANT TO KEEP PROCESSING
  18?SAME FILE KEYIN Y OR N
③ 18 N
  19 JC06  USING   DEV=010 VSN=VANBRO
  1A JC01  JOB WRT      EXECUTING JOB STEP CPYFLE00 #002 12:49:25
  1B JC01  JOB WRT      EXECUTING JOB STEP SORT0000 #003 12:49:38
  1C  SORT M100 END OF SORT
  1D  SORT A186 RECORDS IN 00000000014 RECORDS DELETED 00000000000
  1E JC06  USING    DEV=010 VSN=VANBRO
  1F JC01  JOB WRT       EXECUTING JOB STEP RPLACE00 #004 12:50:14
  11 JC02  JOB WRT       TERMINATED NORMALLY                   12:50:38
④ RV WRT,,D=130,V=VANBRO,F=TEST1,O=Y
  12 JC06  USING   DEV=010 VSN=VANBRO
  13 JC01  JOB WRT      EXECUTING JOB STEP CREATE00 #001 12:51:47
  14 THIS PROGRAM WRITES JCL IMAGES TO DISKETTE FILE
  15 IF COL. M IS SPECIFIED YOU WILL GET COLS 54 THRU 80
  16?SET SPECIAL CHARACTER SPACE VALUE
```

Figure 14-2. Sample Program to Add to and Sort a File (Part 1 of 3)

```
  16 #
  17 -----1.......10........20........30........40........50..54
  18?CNNNN12345678901234567890123456789012345678901234567890123M
⑤ 18 L
  19 // JOB EXAMPLE
  1A                 0005
  1B // DVC 20
  1C                 0010
  1D // LFD PRNTR
  1E                 0015
  1F // OPR 'THIS IS A TEST LINE SHOWING AN ADDITION AND A INSER
⑥ 11 TION'           0017
  12 // DVC RES
  13                 0020
  14?CONTINUE
  14 Y
  15 // LBL $Y$LOD
  16                 0025
  17 // LFD IN
  18                 0030
  19 // EXEC LIBS
  1A                 0035
  1B /$
  1C                         0040
  1D                FIL      DO=IN
  1E                         0045
  1F?CONTINUE
  1F Y
  11                COP.D DO
  12                         0050
  13 /*
  14                         0055
  15 /&
  16                         0060
```

Figure 14-2. Sample Program to Add to and Sort a File (Part 2 of 3)

```
    17 // FIN
    18                          0065
    19 OPTION POINT REACHED-IF YOU WANT TO KEEP PROCESSING
    1A?SAME FILE KEYIN Y OR N
    1A Y
    1B -----1.......10........20........30........40........50..54
    1C?CNNNN12345678901234567890123456789012345678901234567890123M
⑦   1C D
    1D L = LIST FILE
    1E C = CHANGE LINE SEQ NO REQ
    1F A = ADD TO FILE
    11 R = REPLACE LINE
    12 D = DISPLAY OPTIONS
    13 E = END FUNCTIONS
    14 I = IGNORE CHANGE
    15 F = FINISHED WITH ADD
    16 -----1.......10........20........30........40........50..54
    17?CNNNN12345678901234567890123456789012345678901234567890123M
⑧   17 E
    18 JC02  JOB WRT       TERMINATED NORMALLY             12:54:15
```

Figure 14-2. Sample Program to Add to and Sort a File (Part 3 of 3)

Explanation for Figure 14-2:

① Initiates the diskette utility, opens the specified file, indicates your intention to add to the file, and requests a numeric sort

② This is the new record to be added. The selected line number will place the line in its proper sequence when the file is sorted. The M in column 54 indicates the line is continued.

③ Informs the utility that no more changes are being made to the file

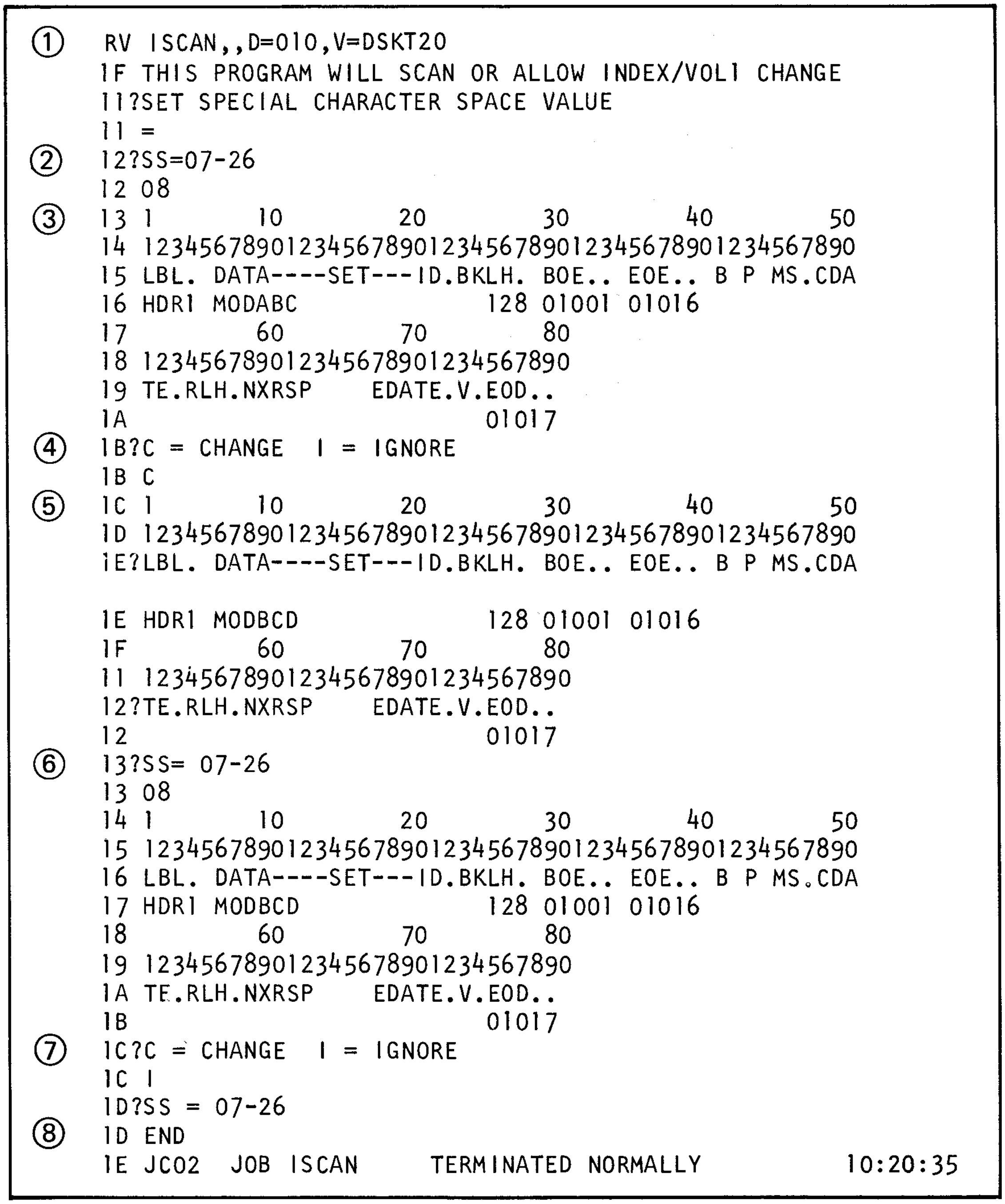
④ Initiates the diskette utility and opens the specified file

⑤ The list (L) command causes a display of the file.

⑥ Display shows the added line in proper sequence in the file.

⑦ The display (D) command causes the utility to display the available commands.

⑧ Terminates the diskette utility



```
①  RV ISCAN,,D=010,V=DSKT20
    1F THIS PROGRAM WILL SCAN OR ALLOW INDEX/VOL1 CHANGE
    11?SET SPECIAL CHARACTER SPACE VALUE
    11 =
②  12?SS=07-26
    12 08
③  13 1        10        20        30        40        50
    14 12345678901234567890123456789012345678901234567890
    15 LBL. DATA----SET---ID.BKLH. BOE.. EOE.. B P MS.CDA
    16 HDR1 MODABC              128 01001 01016
    17          60        70        80
    18 123456789012345678901234567890
    19 TE.RLH.NXRSP    EDATE.V.EOD..
    1A                          01017
④  1B?C = CHANGE  I = IGNORE
    1B C
⑤  1C 1        10        20        30        40        50
    1D 12345678901234567890123456789012345678901234567890
    1E?LBL. DATA----SET---ID.BKLH. BOE.. EOE.. B P MS.CDA

    1E HDR1 MODBCD              128 01001 01016
    1F          60        70        80
    11 123456789012345678901234567890
    12?TE.RLH.NXRSP    EDATE.V.EOD..
    12                          01017
⑥  13?SS= 07-26
    13 08
    14 1        10        20        30        40        50
    15 12345678901234567890123456789012345678901234567890
    16 LBL. DATA----SET---ID.BKLH. BOE.. EOE.. B P MS.CDA
    17 HDR1 MODBCD              128 01001 01016
    18          60        70        80
    19 123456789012345678901234567890
    1A TE.RLH.NXRSP    EDATE.V.EOD..
    1B                          01017
⑦  1C?C = CHANGE  I = IGNORE
    1C I
    1D?SS = 07-26
⑧  1D END
    1E JC02  JOB ISCAN    TERMINATED NORMALLY          10:20:35
```

Figure 14–3. Example of Diskette Index Scan

Explanation for Figure 14-3:

① RUN command to begin index scan

② Request for sector number. Your response, 08, entered on following line

③ System responds by displaying the template fields and the existing information in sector 08.

④ System query for change or ignore. Your response for change (C) follows.

⑤ System displays template, leaving data lines blank. You respond by typing in new information.

⑥ Query for new sector number. You enter 08 and system responds by displaying sector.

⑦ Query for change or ignore. You enter I for ignore. System responds by querying for new sector.

⑧ You enter END and index scan terminates.

15. System Utility Symbiont

The system utility symbiont (SL$$SU) is a multipurpose utility that allows you to perform various functions using cards, tapes, disks, or diskettes. Table 15-1 breaks down the different functions to the media associated with them.

Table 15-1. SL$$SU Functions (Part 1 of 2)

Function Code	Function Performed
	Card Functions
CC	Reproducing cards punched in Hollerith code
CC96	Reproducing 96-column cards
CCB	Reproducing cards punched in binary and Hollerith code
CCS	Reproducing and resequencing source programs
CS96	Reproducing and resequencing source programs contained on 96-column cards
CT	Writing card to tape in unblocked format
CT96	Writing 96-column cards to tape in unblocked format
CTR	Writing card to tape in blocked format
CP	Listing cards
CP96	Listing 96-column cards in character format
CH	Listing cards containing compressed mode
CH96	Listing 96-column cards in vertical hexadecimal format
JCP	Punching cards from the system console

Table 15-1. SL$$SU Functions (Part 2 of 2)

Function Code	Function Performed
Tape Functions	
TT	Copying a tape to another tape
TH	Printing a tape in character and hexadecimal format
THR	Printing a tape in character, hexadecimal, deblocked format
TP	Printing a tape containing only standard characters
TPR	Printing a tape in character and deblocked format
TRS	Locating a specific record on tape
TRL	Changing existing records on tape
TC	Punching cards from tape
INT	Prepping a tape
FSF	Forward space to a specific file
BSF	Backward space to a specific file
FSR	Forward space to a specific record
BSR	Backward space to a specific record
WTM	Writing tape marks
REW	Rewinding a tape
RUN	Rewinding a tape with interlock
ERG	Erasing a portion of a tape
Disk Functions	
DD	Printing a disk in unblocked format
DDR	Printing a disk in reblocked format
VTP	Printing the volume table of contents of a disk
SVT	Printing a short format VTOC file
AVX	Displaying available disk extents on console screen
DID	Changing the volume serial number
Diskette Functions	
DD	Printing a diskette in unblocked format
VTP	Printing the data set labels of a diskette
DID	Changing the volume serial number

Two system utility symbionts are available under OS/3: SU and TU. The two symbionts are interchangeable; both respond to the same function codes, and both are requested and controlled from the system console. However, we recommend the TU symbiont be used for tape operations, since doing so will increase the buffer size for all selector channel tapes from the 8189 bytes used for SU to 32,767 bytes.

For detailed information on the system utility symbiont, refer to the appropriate operations handbook for operators for your system.

16. List Software Maintenance Corrections (SMCLIST)

16.1. SMCLIST FUNCTION

You can use the SMCLIST canned job control stream to print a listing of the software maintenance corrections contained in the SMCLOG file. This listing may be printed in either a full or condensed format. Also, by specifying certain parameters, you can produce listings sorted by SMC number, component number, program-product-type number, date, and time.

16.2. EXECUTING SMCLIST

The format of the SMCLIST canned job control stream is:

```
RV SMCLIST,[,FMT={F}][,SEQ1={COMP   }][,SEQ2={COMP   }]
                 {C}        {DATE   }        {DATE   }
                            {TIME   }        {TIME   }
                            {PP-TYPE}        {PP-TYPE}
                            {SMC#   }        {SMC#   }
```

where:

```
FMT={F}
    {C}
```

Specifies the format of the listing being produced.

```
FMT=F
```

Specifies that a full listing is printed.

```
FMT=C
```

Specifies that a condensed listing is printed.

A full listing is a listing sorted primarily by component number and then by SMC number. It gives more information about each SMC than a condensed listing, such as the regenerations an SMC requires or the method used to install it.

Since you don't always need as much information as the full listing shows, we also provide a condensed listing. A condensed listing contains only SMC numbers in ascending order and an indication of whether any SMCs were backed out, replaced, or not installed because of an error during installation. See Figures 16–1 and 16–2 for an example of each listing.

The default for the FMT parameter is C for condensed. Unless you specify FMT=F on your SMCLIST run command, you will always receive a condensed listing of the SMCs in the SMCLOG file.

```
SEQ1=⎧COMP   ⎫
     ⎪DATE   ⎪
     ⎨TIME   ⎬
     ⎪PP-TYPE⎪
     ⎩SMC#   ⎭
```

Specifies the primary sorting key to be used.

SEQ1=COMP

Specifies component number.

SEQ1=DATE

Specifies the date the SMCs were applied.

SEQ1=TIME

Specifies the time the SMCs were applied.

SEQ1=PP-TYPE

Specifies program-product-type number.

SEQ1=SMC#

Specifies SMC number.

If you omit this keyword, the full listing is sorted primarily by component number.

```
SEQ2=⎧COMP   ⎫
     ⎪DATE   ⎪
     ⎨TIME   ⎬
     ⎪PP-TYPE⎪
     ⎩SMC#   ⎭
```

Specifies the secondary sorting key to be used.

SEQ2=COMP

Specifies component number.

SEQ2=DATE

Specifies the date the SMCs were applied.

SEQ2=TIME

Specifies the time the SMCs were applied.

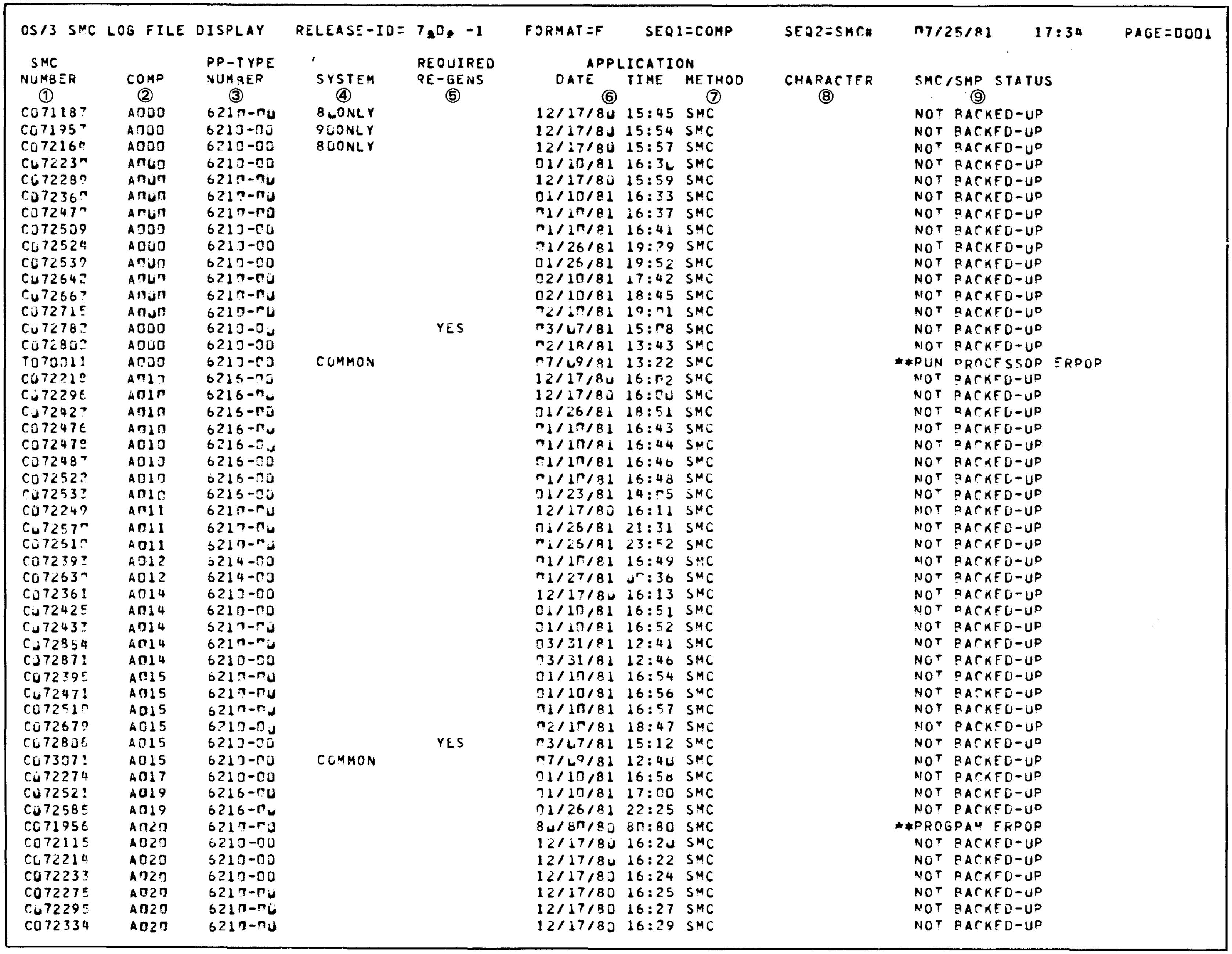
SEQ2=PP-TYPE
Specifies program-product-type number. ←

SEQ2=SMC#
Specifies SMC number. ←

If you omit the SEQ2 keyword, the secondary sorting key is the SMC number.

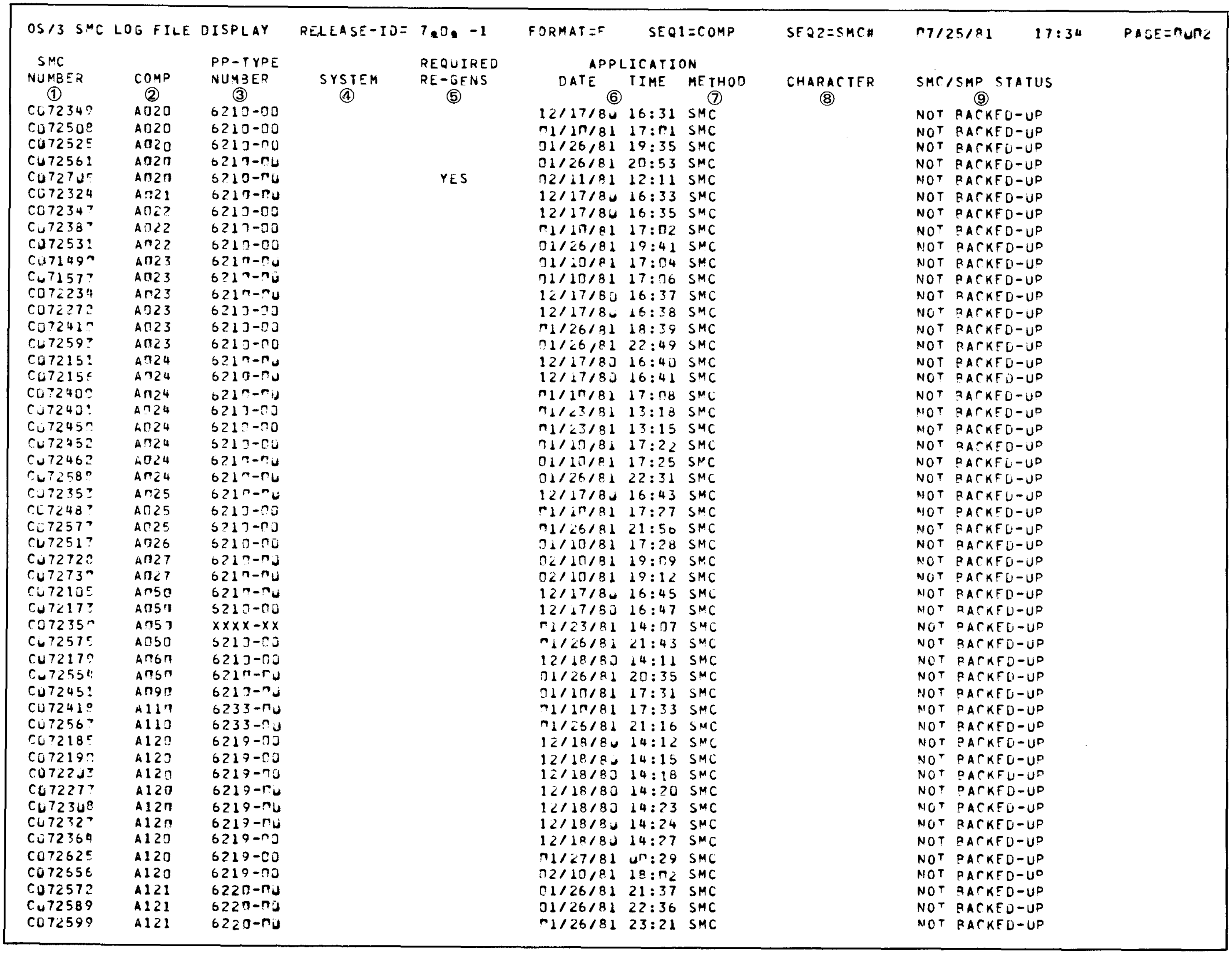
NOTE:

The SEQ1 specification may not be the same as the SEQ2 specification.



```
OS/3 SMC LOG FILE DISPLAY   RELEASE-ID= 7.0. -1   FORMAT=F   SEQ1=COMP   SEQ2=SMC#   07/25/81   17:34   PAGE=0001

 SMC                 PP-TYPE              REQUIRED          APPLICATION
NUMBER     COMP      NUMBER      SYSTEM   RE-GENS        DATE   TIME  METHOD  CHARACTER        SMC/SMP STATUS
  ①          ②          ③          ④         ⑤             ⑥          ⑦         ⑧                 ⑨
C071187    A000      6210-00     80ONLY                12/17/80 15:45 SMC                      NOT BACKED-UP
C071957    A000      6210-00     90ONLY                12/17/80 15:54 SMC                      NOT BACKED-UP
C072168    A000      6210-00     80ONLY                12/17/80 15:57 SMC                      NOT BACKED-UP
C072230    A000      6210-00                           01/10/81 16:30 SMC                      NOT BACKED-UP
C072289    A000      6210-00                           12/17/80 15:59 SMC                      NOT BACKED-UP
C072360    A000      6210-00                           01/10/81 16:33 SMC                      NOT BACKED-UP
C072470    A000      6210-00                           01/10/81 16:37 SMC                      NOT BACKED-UP
C072509    A000      6210-00                           01/10/81 16:41 SMC                      NOT BACKED-UP
C072524    A000      6210-00                           01/26/81 19:29 SMC                      NOT BACKED-UP
C072539    A000      6210-00                           01/26/81 19:52 SMC                      NOT BACKED-UP
C072642    A000      6210-00                           02/10/81 17:42 SMC                      NOT BACKED-UP
C072667    A000      6210-00                           02/10/81 18:45 SMC                      NOT BACKED-UP
C072715    A000      6210-00                           02/10/81 19:01 SMC                      NOT BACKED-UP
C072782    A000      6210-00              YES          03/07/81 15:08 SMC                      NOT BACKED-UP
C072802    A000      6210-00                           02/18/81 13:43 SMC                      NOT BACKED-UP
T070011    A000      6210-00     COMMON                07/09/81 13:22 SMC                    **RUN PROCESSOR ERROR
C072219    A010      6215-00                           12/17/80 16:02 SMC                      NOT BACKED-UP
C072296    A010      6216-00                           12/17/80 16:00 SMC                      NOT BACKED-UP
C072427    A010      6215-00                           01/26/81 18:51 SMC                      NOT BACKED-UP
C072476    A010      6216-00                           01/10/81 16:43 SMC                      NOT BACKED-UP
C072478    A010      6216-00                           01/10/81 16:44 SMC                      NOT BACKED-UP
C072487    A010      6216-00                           01/10/81 16:46 SMC                      NOT BACKED-UP
C072522    A010      6216-00                           01/10/81 16:48 SMC                      NOT BACKED-UP
C072533    A010      6215-00                           01/23/81 14:05 SMC                      NOT BACKED-UP
C072249    A011      6210-00                           12/17/80 16:11 SMC                      NOT BACKED-UP
C072570    A011      6210-00                           01/26/81 21:31 SMC                      NOT BACKED-UP
C072510    A011      6210-00                           01/26/81 23:52 SMC                      NOT BACKED-UP
C072393    A012      6214-00                           01/10/81 16:49 SMC                      NOT BACKED-UP
C072630    A012      6214-00                           01/27/81 00:36 SMC                      NOT BACKED-UP
C072361    A014      6210-00                           12/17/80 16:13 SMC                      NOT BACKED-UP
C072425    A014      6210-00                           01/10/81 16:51 SMC                      NOT BACKED-UP
C072433    A014      6210-00                           01/10/81 16:52 SMC                      NOT BACKED-UP
C072854    A014      6210-00                           03/31/81 12:41 SMC                      NOT BACKED-UP
C072871    A014      6210-00                           03/31/81 12:46 SMC                      NOT BACKED-UP
C072395    A015      6210-00                           01/10/81 16:54 SMC                      NOT BACKED-UP
C072471    A015      6210-00                           01/10/81 16:56 SMC                      NOT BACKED-UP
C072510    A015      6210-00                           01/10/81 16:57 SMC                      NOT BACKED-UP
C072679    A015      6210-00                           02/10/81 18:47 SMC                      NOT BACKED-UP
C072806    A015      6210-00              YES          03/07/81 15:12 SMC                      NOT BACKED-UP
C073071    A015      6210-00     COMMON                07/09/81 12:40 SMC                      NOT BACKED-UP
C072274    A017      6210-00                           01/10/81 16:58 SMC                      NOT BACKED-UP
C072521    A019      6216-00                           01/10/81 17:00 SMC                      NOT BACKED-UP
C072585    A019      6216-00                           01/26/81 22:25 SMC                      NOT BACKED-UP
C071956    A020      6210-00                           80/80/80 80:80 SMC                    **PROGRAM ERROR
C072115    A020      6210-00                           12/17/80 16:20 SMC                      NOT BACKED-UP
C072218    A020      6210-00                           12/17/80 16:22 SMC                      NOT BACKED-UP
C072233    A020      6210-00                           12/17/80 16:24 SMC                      NOT BACKED-UP
C072275    A020      6210-00                           12/17/80 16:25 SMC                      NOT BACKED-UP
C072295    A020      6210-00                           12/17/80 16:27 SMC                      NOT BACKED-UP
C072334    A020      6210-00                           12/17/80 16:29 SMC                      NOT BACKED-UP
```

Figure 16-1. Example of Full SMC Listing (Part 1 of 3)

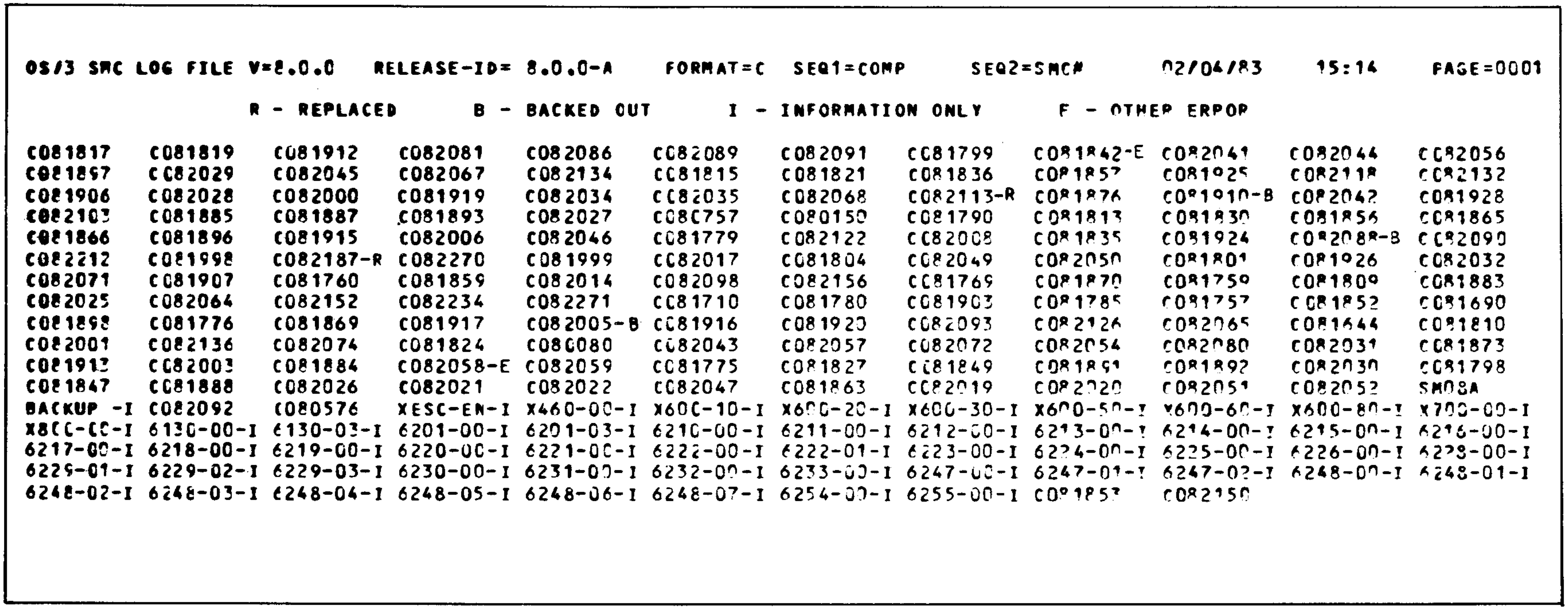


```
OS/3 SMC LOG FILE DISPLAY   RELEASE-ID= 7.0. -1    FORMAT=F     SEQ1=COMP      SEQ2=SMC#      07/25/81     17:34      PAGE=0002

 SMC                 PP-TYPE              REQUIRED          APPLICATION
NUMBER     COMP      NUMBER      SYSTEM   RE-GENS     DATE     TIME  METHOD     CHARACTER      SMC/SMP STATUS
  ①         ②          ③           ④        ⑤            ⑥            ⑦            ⑧              ⑨
C072349    A020      6210-00                          12/17/80 16:31 SMC                       NOT BACKED-UP
C072508    A020      6210-00                          01/10/81 17:01 SMC                       NOT BACKED-UP
C072525    A020      6210-00                          01/26/81 19:35 SMC                       NOT BACKED-UP
C072561    A020      6210-00                          01/26/81 20:53 SMC                       NOT BACKED-UP
C072705    A020      6210-00              YES         02/11/81 12:11 SMC                       NOT BACKED-UP
C072324    A021      6210-00                          12/17/80 16:33 SMC                       NOT BACKED-UP
C072347    A022      6210-00                          12/17/80 16:35 SMC                       NOT BACKED-UP
C072387    A022      6210-00                          01/10/81 17:02 SMC                       NOT BACKED-UP
C072531    A022      6210-00                          01/26/81 19:41 SMC                       NOT BACKED-UP
C071490    A023      6210-00                          01/10/81 17:04 SMC                       NOT BACKED-UP
C071577    A023      6210-00                          01/10/81 17:06 SMC                       NOT BACKED-UP
C072234    A023      6210-00                          12/17/80 16:37 SMC                       NOT BACKED-UP
C072272    A023      6210-00                          12/17/80 16:38 SMC                       NOT BACKED-UP
C072410    A023      6210-00                          01/26/81 18:39 SMC                       NOT BACKED-UP
C072593    A023      6210-00                          01/26/81 22:49 SMC                       NOT BACKED-UP
C072151    A024      6210-00                          12/17/80 16:40 SMC                       NOT BACKED-UP
C072156    A024      6210-00                          12/17/80 16:41 SMC                       NOT BACKED-UP
C072400    A024      6210-00                          01/10/81 17:08 SMC                       NOT BACKED-UP
C072401    A024      6210-00                          01/23/81 13:18 SMC                       NOT BACKED-UP
C072450    A024      6210-00                          01/23/81 13:15 SMC                       NOT BACKED-UP
C072452    A024      6210-00                          01/10/81 17:22 SMC                       NOT BACKED-UP
C072462    A024      6210-00                          01/10/81 17:25 SMC                       NOT BACKED-UP
C072588    A024      6210-00                          01/26/81 22:31 SMC                       NOT BACKED-UP
C072353    A025      6210-00                          12/17/80 16:43 SMC                       NOT BACKED-UP
C072487    A025      6210-00                          01/10/81 17:27 SMC                       NOT BACKED-UP
C072577    A025      6210-00                          01/26/81 21:56 SMC                       NOT BACKED-UP
C072517    A026      6210-00                          01/10/81 17:28 SMC                       NOT BACKED-UP
C072720    A027      6210-00                          02/10/81 19:09 SMC                       NOT BACKED-UP
C072730    A027      6210-00                          02/10/81 19:12 SMC                       NOT BACKED-UP
C072105    A050      6210-00                          12/17/80 16:45 SMC                       NOT BACKED-UP
C072173    A050      6210-00                          12/17/80 16:47 SMC                       NOT BACKED-UP
C072350    A050      XXXX-XX                          01/23/81 14:07 SMC                       NOT BACKED-UP
C072575    A050      6210-00                          01/26/81 21:43 SMC                       NOT BACKED-UP
C072179    A060      6210-00                          12/18/80 14:11 SMC                       NOT BACKED-UP
C072554    A060      6210-00                          01/26/81 20:35 SMC                       NOT BACKED-UP
C072451    A090      6210-00                          01/10/81 17:31 SMC                       NOT BACKED-UP
C072418    A110      6233-00                          01/10/81 17:33 SMC                       NOT BACKED-UP
C072567    A110      6233-00                          01/26/81 21:16 SMC                       NOT BACKED-UP
C072185    A120      6219-00                          12/18/80 14:12 SMC                       NOT BACKED-UP
C072190    A120      6219-00                          12/18/80 14:15 SMC                       NOT BACKED-UP
C072203    A120      6219-00                          12/18/80 14:18 SMC                       NOT BACKED-UP
C072277    A120      6219-00                          12/18/80 14:20 SMC                       NOT BACKED-UP
C072308    A120      6219-00                          12/18/80 14:23 SMC                       NOT BACKED-UP
C072327    A120      6219-00                          12/18/80 14:24 SMC                       NOT BACKED-UP
C072369    A120      6219-00                          12/18/80 14:27 SMC                       NOT BACKED-UP
C072625    A120      6219-00                          01/27/81 00:29 SMC                       NOT BACKED-UP
C072656    A120      6219-00                          02/10/81 18:02 SMC                       NOT BACKED-UP
C072572    A121      6220-00                          01/26/81 21:37 SMC                       NOT BACKED-UP
C072589    A121      6220-00                          01/26/81 22:36 SMC                       NOT BACKED-UP
C072599    A121      6220-00                          01/26/81 23:21 SMC                       NOT BACKED-UP
```

Figure 16-1. Example of Full SMC Listing (Part 2 of 3)

Field	Description
① SMC NUMBER	Software maintenance correction number assigned
② COMP	Software component number (internal use only)
③ PP-TYPE NUMBER	Program product type number; e.g., control system (6210)
④ SYSTEM	No entry indicates SMC is common to both System 80 and Series 90. 80 only indicates SMC pertains to System 80 only. 90 only indicates SMC pertains to Series 90 only.
⑤ REQUIRED RE-GENS	YES indicates some SYSGEN was required. No entry indicates SYSGEN was not required.
⑥ APPLICATION DATE - TIME	Date and time SMC or SMP was applied.
⑦ METHOD	SMC applied via the librarian SMP applied via dump/restore
⑧ CHARACTER	No entry indicates SMC/SMP was required. OPTIONAL indicates SMC/SMP was not required.
⑨ SMC/SMP STATUS	Not BACKED UP indicates SMC/SMP cannot be deleted but was successfully applied. NOT APPLIED,DEPENDENCY indicates other SMCs not contained on the volume were needed; SMC/SMP was not applied. **SMC LOOPED indicates SMC was not applied because specified time limit was exceeded for applying SMC. **SMC EXECUTION ERROR indicates SMC was not applied due to abnormal termination. **RUN PROCESSOR ERROR indicates SMC, not being scheduled, was not applied.

Figure 16-1. Example of Full SMC Listing (Part 3 of 3)



```
OS/3 SMC LOG FILE V=8.0.0   RELEASE-ID= 8.0.0-A    FORMAT=C  SEQ1=COMP     SEQ2=SMC#      02/04/83     15:14     PAGE=0001

                    R - REPLACED       B - BACKED OUT       I - INFORMATION ONLY       F - OTHER ERROR

C081817     C081819     C081912     C082081     C082086     C082089     C082091     C081799     C081842-E   C082041     C082044     C082056
C081897     C082029     C082045     C082067     C082134     C081815     C081821     C081836     C081857     C081925     C082118     C082132
C081906     C082028     C082000     C081919     C082034     C082035     C082068     C082113-R   C081876     C081910-B   C082042     C081928
C082103     C081885     C081887     C081893     C082027     C080757     C080150     C081790     C081813     C081830     C081856     C081865
C081866     C081896     C081915     C082006     C082046     C081779     C082122     C082008     C081835     C081924     C082088-B   C082090
C082212     C081998     C082187-R   C082270     C081999     C082017     C081804     C082049     C082050     C081801     C081926     C082032
C082071     C081907     C081760     C081859     C082014     C082098     C082156     C081769     C081870     C081759     C081809     C081883
C082025     C082064     C082152     C082234     C082271     C081710     C081780     C081903     C081785     C081757     C081852     C081690
C081898     C081776     C081869     C081917     C082005-B   C081916     C081920     C082093     C082126     C082065     C081644     C081810
C082001     C082136     C082074     C081824     C080080     C082043     C082057     C082072     C082054     C082080     C082031     C081873
C081913     C082003     C081884     C082058-E   C082059     C081775     C081827     C081849     C081891     C081892     C082030     C081798
C081847     C081888     C082026     C082021     C082022     C082047     C081863     C082019     C082020     C082051     C082052     SM08A
BACKUP -I   C082092     C080576     XESC-EN-I   X460-00-I   X600-10-I   X600-20-I   X600-30-I   X600-50-I   X600-60-I   X600-80-I   X700-00-I
X800-00-I   6130-00-I   6130-03-I   6201-00-I   6201-03-I   6210-00-I   6211-00-I   6212-00-I   6213-00-I   6214-00-I   6215-00-I   6216-00-I
6217-00-I   6218-00-I   6219-00-I   6220-00-I   6221-00-I   6222-00-I   6222-01-I   6223-00-I   6224-00-I   6225-00-I   6226-00-I   6228-00-I
6229-01-I   6229-02-I   6229-03-I   6230-00-I   6231-00-I   6232-00-I   6233-00-I   6247-00-I   6247-01-I   6247-02-I   6248-00-I   6248-01-I
6248-02-I   6248-03-I   6248-04-I   6248-05-I   6248-06-I   6248-07-I   6254-00-I   6255-00-I   C081853     C082150
```

Figure 16-2. Sample of Condensed SMC Listing

This condensed listing shows all of the SMCs contained in the SMCLOG file. It is sorted by SMC number and shows all of the SMCs in ascending order. Some of the SMCs show codes after them, indicating that they were:

- replaced by Sperry Univac because they produced adverse effects on OS/3. Any SMCs of this kind show a code of –R after them. For example: `C082187-R`;
- backed out by you under the direction of a Sperry Univac representative because they produced adverse effects on your particular system. Any SMCs of this kind show a code of –B after them. For example: `C082005-B`; and
- not applied to your system because of an error during installation. These SMCs are followed by a code of –E. For example: `C082058-E`.

NOTE:

An additional code of –I may also appear on your condensed listing. This code applies to non-SMC records and indicates that Sperry Univac recorded information about that record in the SMCLOG file to be used by the SMC job stream. You can ignore any records containing this code.

PART 5. APPENDIXES

Appendix A. Canned Job Control Streams

A.1. GENERAL PURPOSE OF THE CANNED JOB CONTROL STREAMS

The following canned job control streams provide you with a more convenient method of performing some system utilities without the need of punching the parameters and job control statements normally required to run them. The utilities reside in the system load library file ($Y$LOD), and their corresponding job control streams reside in the system job control stream library file ($Y$JCS). The utilities are initiated from the system console by keying in their associated job control stream name.

Table A-1 shows the job names associated with the utilities, the functions performed, and the manuals where they can be found.

Table A-1. Canned Job Control Streams (Part 1 of 2)

Job Name	Function	Described in Document (current version)
COPYREL	Copies SYSRES files to a new SYSRES volume	System installation UG/PR UP-8074; SSP UG UP-8062
COPY$10	Creates a backup copy of an 8410 disk	Emulation/conversion (360/20) UG/PR UP-8064
COPY$11	Creates a backup copy of an IBM 2311 disk onto a disk supported by Series 90	Emulation/conversion (360/20) UG/PR UP-8064
DCOP	Copies SYSRES from one disk to another	System installation UG/PR UP-8074
DUMPLOG	Dumps job or console log records to disk	Spooling and job accounting concepts and facilities UP-8869
DUMPLOGT	Dumps job or console log records to tape	Spooling and job accounting concepts and facilities UP-8869
DUMP20	Dumps the image of an IBM 360/20 disk pack	Emulation/conversion (360/20) UG/PR UP-8064
ECDC	Feeds in cards with names for an emulation carriage tape loops display	Emulation/conversion (9200/ 9300) and (360/20) UG/PR UP-8063 and UP-8064

Table A-1. Canned Job Control Streams (Part 2 of 2)

Job Name	Function	Described in Document (current version)
ECDK	Keys in names for an emulation carriage tape loops display program	Emulation/conversion (9200/ 9300) and (360/20) UG/PR UP-8063 and UP-8064
IMPLDSKT	Creates an IMPL diskette	System installation UG/PR UP-8074
JBLOG	Produces a job accounting report with SYSLOG residing on disk	Spooling and job accounting concepts and facilities UP-8869
JBLOGT	Produces a job accounting report with SYSLOG residing on tape	Spooling and job accounting concepts and facilities UP-8869
LNKUPL	Executes UTS 400 upline linker routine	UTS 400-OS/3 interface UG/PR UP-8611
PIMAGE	Creates a copy of an 8410 disk image onto a disk supported by Series 90	Emulation/conversion (9200/ 9300) UG/PR UP-8063
PNCH9300	Punches card deck to use 9300 system as a peripheral device	Emulation/conversion (9200/ 9300) UG/PR UP-8063
PRNT9300	Prints source module needed to use 9300 system as a peripheral device	Emulation/conversion (9200/ 9300) UG/PR UP-8063
SCLIST	Lists the shared code ($Y$SCLOD) modules	System installation UG/PR UP-8074
SYSDUMP	Prints a complete system dump from SYSRES or another system disk	Dump analysis UG/PR UP-8837
SYSDUMPO	Prints a complete system dump or a portion of a system dump from SYSRES or another system disk	Dump analysis UP/PR UP-8837
UPLCNV	Executes UTS 400 upline conversion routine	UTS 400-OS/3 interface UG/PR UP-8611
UPLDELT	Deletes the specified UTS 400 upline dump file	UTS 400-OS/3 interface UG/PR UP-8611
UPLDMPN	Prints the specified UTS 400 upline dump file	UTS 400-OS/3 interface UG/PR UP-8611
UPLDUMP	Prints and deletes the specified UTS 400 upline dump file	UTS 400-OS/3 interface UG/PR UP-8611
VTOC20	Prints a volume table of contents listing of an IBM 360/20 disk pack image	Emulation/conversion (360/20) UG/PR UP-8064

A.2. COPYING RELEASE OR SYSRES LIBRARIES (COPYREL)

For convenience, we are showing the format of the COPYREL canned job control stream. Not only is it useful during your system installation, it can be used any time you need to copy all the RELEASE or SYSRES volume libraries to a second disk pack of any type.

The format of COPYREL is:

```
RV△COPYREL,[,V=vsn][,T=disk-type][,S=first-file]
    [,E=last-file]
```

For a complete description of COPYREL, see the current version of the system installation user guide, UP-8074.

Appendix B. Code Set Components

Code set components are defined as those records that, when combined in a particular sequence, make up a program source module, a macro/jproc source module, an object module, a load module, or a grouped code set module. The elements, or records, comprising these code sets are listed, as follows, by module type (in hexadecimal).

1. GROUPED CODE SETS

   1 beginning of group demarcator, type A0
   Separate or mixed sets of source, macro/jproc, object, or load modules
   1 end of group demarcator, type A8
   1 EOF code sentinel, type A1

2. PROGRAM SOURCE AND MACRO/JPROC SOURCE MODULE CODE SETS

   1 header, type A3 or A4
   1 or more source items, type 24 or 25

3. OBJECT CODE SETS

   1 header, type 80
   1 or more linkage editor control statements, type 40 (optional)
   1 or more CSECT, types 08, 09, 0A, 0B
   1 or more ESD, types 04, 06, 07 (optional)
   1 or more text, type 02
   1 transfer, type 03
   1 or more linkage editor control statements, type 40 (optional)
   1 or more ISD records, type 0C

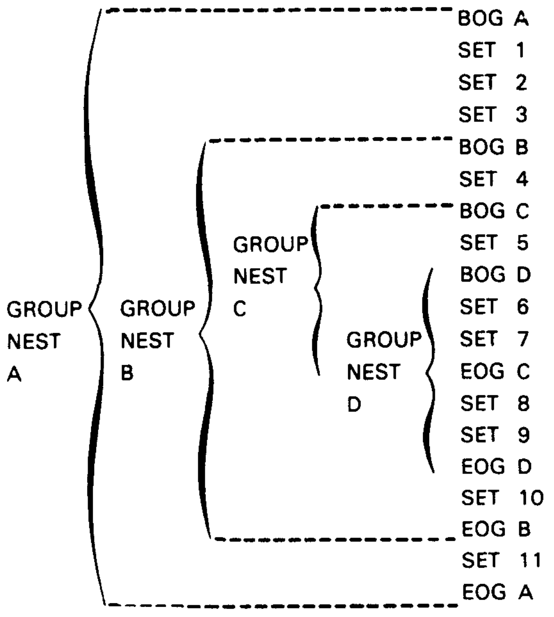
4. LOAD CODE SETS

   1 header, type 90 or B0 (root phase definition)
   1 or more SENTRY, type C4 (optional)
   1 or more sets of resource and SEXTERN records, type C8 and C6 (optional)

   1 or more text, type 12 or 32
   1 transfer, type 13
   1 or more sets phase definition (type 90 or B0), text (type 12 or 32), and transfer (type 13) records, depending on the number of phases in the load module (optional)
   1 or more ISD records, type 1C

B.1. GROUPED CODE SETS

Library files may contain group demarcators that divide different sets of elements into specific groups. Groups may be composed of any one code set or may be a mixture of all sets in any order. The grouping is strictly optional and can be performed by the librarian at your option. The librarian can manipulate code within libraries on a group basis and these files may then, in turn, be accessed by processing routines at a group level. Groups may overlap other groups and may be nested to any level. (Figure B-1 illustrates the nesting of groups.) Beginning- and end-of-group (BOG and EOG) records (type A0 and A8, respectively) demarcate and name the grouped code sets. Tables B-1 through B-3 describe the library items peculiar to grouped code sets.



NOTE:

All sets are contained within Group Nest A. Some sets are subnested and overlapped as follows:

A. Sets 6, 7, 8, and 9 are contained within Group Nest D, which is contained within Group Nest B, which is contained within Group Nest A. Group Nest C and Group Nest D overlap within Group Nest B.

B. Sets 5, 6, and 7 are contained within Group Nest C, which is contained within Group Nest B, which is contained within Group Nest A.

C. Sets 4 through 10 are contained within Group Nest B, which is contained within Group Nest A.

D. Sets 1, 2, 3, and 11 are contained only within Group Nest A.

Figure B-1. Example of Nested Group Code Sets

Table B—1. Beginning-of-Group (BOG) Header Record Format

Byte Position	Field	Contents
0	Length prefix	38 (binary format)
1	Type prefix	$A0_{16}$
2—9	Group name	Symbolic name of the logical group of code sets contained within this group and terminated by this record (left-justified and space-filled)
10—39	Comments	Up to 30 bytes of pertinent comments (as deemed necessary to identify the group)

Table B—2. End-of-Group (EOG) Trailer Record Format

Byte Position	Field	Contents
0	Length prefix	8 (binary format)
1	Type prefix	$A8_{16}$
2—9	Group name	Symbolic name of the logical group of code sets contained within this group and terminated by this record (left-justified and space-filled)

Table B—3. End-of-File (EOF) Sentinel Record Format

Byte Position	Field	Contents
0	Length prefix	20 (binary format)
1	Type prefix	$A1_{16}$
2—13	Unused	00_{16}
14—21	Name	ENDLIB△△

B.2. SOURCE MODULE CODE SETS

Source module code sets within library files may be composed of any type of source module statements from BAL macro definitions or own-code specifications up through specific language processor parameters and JPROCs written in job control language. Tables B-4, B-5, and B-6 describe the library items peculiar to source code sets.

Table B-4. Source Module Code Header Record Format

Byte Position	Field	Contents
0	Length prefix	56 (binary format)
1	Type prefix	$A3_{16}$ or $A4_{16}$
2	Unused	00_{16}
3, 4	Flags	00_{16}, or 80_{16} if module has been corrected
5—13	Unused	00_{16}
14—21	Module name	Symbolic name of the source code set originated by this record (left-justified and space-filled)
22—24	Date	In the form as it appears in the preamble
25—26	Time	In the form: hour-minute (packed decimal less zone field)
27	Unused	00_{16}
28—57	Comments	Up to 30 bytes of pertinent comments as deemed necessary to identify the source module.

Table B-5. Source Module Code Statement Record Format

Byte Position	Field	Contents
0	Length prefix	Variable; 2+ length
1	Type prefix	24_{16}
2—81	Source record	Source statement

Table B-6. Compressed Source Module Code Statement Record Format

Byte Position	Field	Contents
0	Length prefix	Variable; 2+ compressed source length
1	Type prefix	25_{16}
2—81	Source record	Compressed source statement

B.3. OBJECT CODE SETS

Object code within library files is composed mostly of text and relocation data generated as output of the various language processors. This code exists in a format acceptable to the linkage editor and contains additional record types used by the linkage editor for load module generation. Object module records are variable in length and are packed as densely as possible within a given library block. The desired order of appearance of all records within an object code set is:

1. Object module header record
2. Control statement records*
3. All control section records (must precede associated text and entry ESDs)
4. All ESD records (names must be unique)
5. All ISD records**
6. All text/RLD records
7. Object module transfer record
8. Control statement records

Tables B-7 through B-16 describe these records.

* *Control statement records are generated by certain language processors and may be used to designate control information necessary to a subsequent linkage editor run.*

** *ISD records also are generated by certain language processors and are used by JOBDUMP to produce a formatted dump if an abnormal termination occurs in your load module.*

Table B-7. Object Code Header Record Format

Byte Position	Field	Contents
0	Length prefix	55 (binary format)
1	Type prefix	80_{16}
2	ESID	00_{16}
3	Unused	
4	Flag	Bit 0 Set to indicate that the module has been patched Bits 1–6 Reserved Bit 7 Set to indicate that the object module is reentrant
5—8	Address	Assembled or compiled origin of the object module
9—12	Module length	Total number of bytes required for the object module
13—20	Module name	Symbolic name of the object module originated by this record (left-justified and space-filled)
21—23	Date	In the form as it appears in the preamble
24, 25	Time	Hour-minute (packed decimal less zone field)
26	Unused	00_{16}
27—56	Comments	Up to 30 bytes of pertinent comments as deemed necessary to identify the object module

Table B-8. Object Code Control Section Record Format

Byte Position	Field	Contents
0	Length prefix	19 (binary format)
1	Type prefix	08_{16}, 09_{16}, $0A_{16}$, or $0B_{16}$ (See Table B-9.)
2	ESID	External symbol identification assigned to this control or common section
3, 4	Flag bytes	8000_{16} indicates a deferred length specified in the transfer record of this object module; ignore bytes 9—12
5—8	Section address	Compiled address of the start of this control or common section
9—12	Section size	Total length in bytes of this control or common section
13—20	Section name	Symbolic name of the control or common section (left-justified and space-filled)

Table B-9. Possible Control Section Record Types

Type of Control Section	Record Type	Record Length	Field Contents				
			2	3,4	5–8	9–12	13–20
Named control section	08	19	ESID	0000_{16} or 8000_{16}	Address	Length	Control section name
Unnamed control section	09						Blanks (40_{16})
Named common section	0A						Common section name
Unnamed common section	0B						Blanks (40_{16})

Table B-10. Object Code ESD Record Format

Byte Position	Field	Contents
0	Length prefix	15 (binary format)
1	Type prefix	04_{16}, 06_{16}, or 07_{16} (See Table B-11.)
2	ESID	External symbol identification assigned to this ESD reference
3, 4	Unused	00_{16}
5—8	Relative address	Processor-generated address or value assigned to this ESD reference
9—16	ESD name	Symbolic name of the ESD reference

Table B-11. Possible ESD Record Types

ESD Type	Record Type	Record Length	Field Contents			
			2	3,4	5–8	9–16
ENTRY	04	15	ESID	0000_{16}	Assembled address	Symbol
EXTRN	06					
V-CON	07					

Table B-12. Object Code ISD Record Format

Byte Position	Field	Contents
0	Length prefix	Variable
1	Type prefix	$0c_{16}$
2	ESID	External symbol identification of CSECT assigned to the ISD
3	Flag	Bits 0—1 unused Bit 2 set to indicate Type 3 ISD Bit 3 set to indicate Type 4 ISD (comment) Bits 4—7 unused
4	Flag	Unused
5—8	Compile origin	Processor-generated address assigned to this ISD
9—246	Attributes	Symbolic name and attributes of the ISD item

Table B-13. Object Code Text/RLD Record Format

Byte Position	Field	Contents
0	Length prefix	Variable: 7 + text length + RLD length (binary format)
1	Type prefix	02_{16}
2	ESID	External symbol identification with which the text data in this record is associated.
3	Text length	Number of bytes less one byte of text data in this record
4	RLD length	Number of bytes of relocation data in this record (a multiple of three bytes)
5	Flag	01_{16} if patched text item
6—8	Relative address	Processor-assigned relative address of first byte of text data in this record
9—9+ text length	Text data	Instructions and/or data generated by a processor and relative to the ESID specified
9+ text length + RLD length backward thru 9 + text length	RLD data	Three-byte relocation masks used to modify the various fields of preceding text data in this record (See Table B-14.)

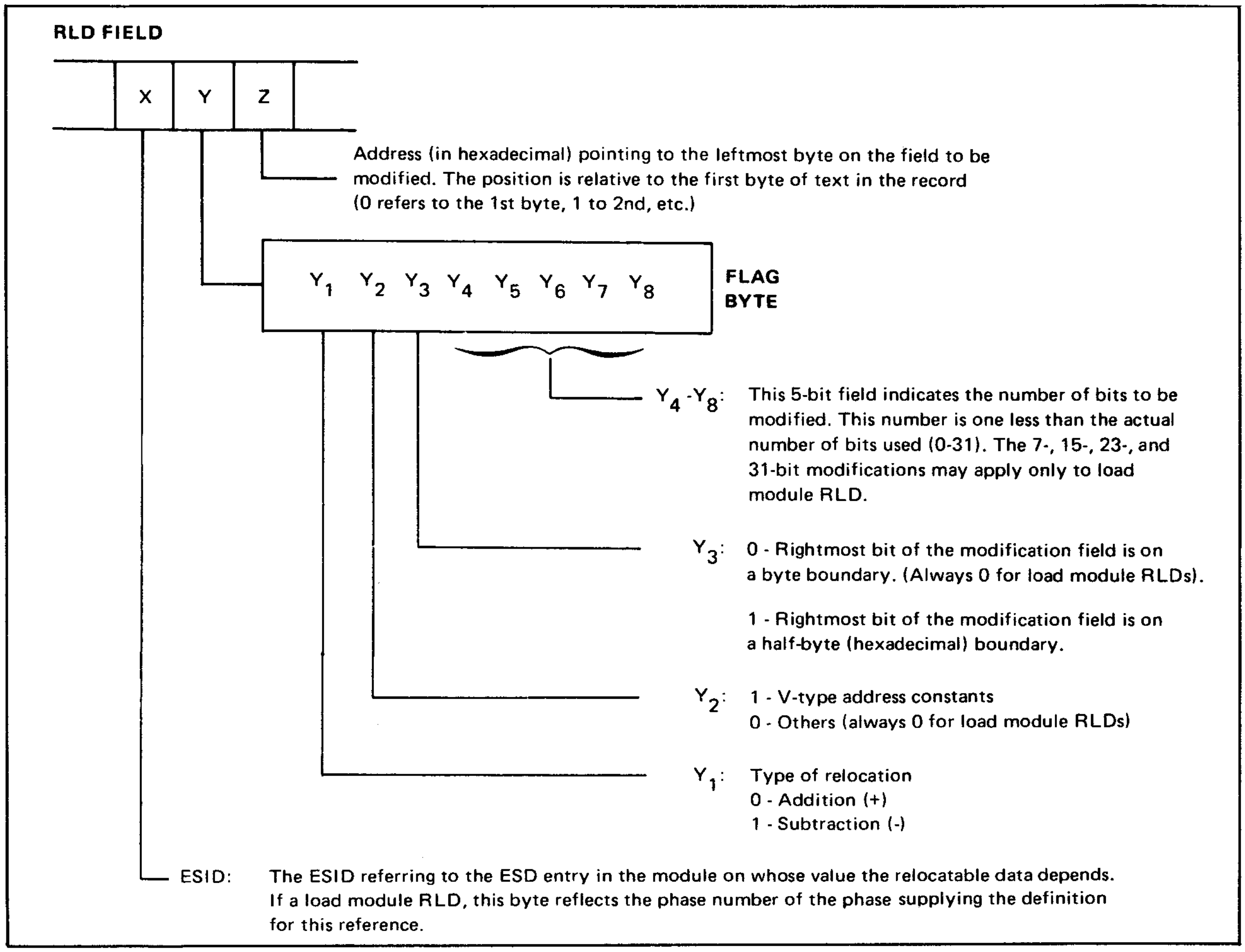
Table B-14. Relocation Mask Formats

Byte Position	Field	Contents
0	ESID	External symbol identification of the external reference whose subsequent value will be used to modify the addressed field
1	Flag	Designator byte reflecting type, size, and position of the modification field (Figure B–2)
2	Address	Relative record pointer indicating the most significant (leftmost) byte of text data at which the modification is to begin (first text byte, 0; 2nd byte, 1, etc.)

NOTES:

1. Each RLD data field in a given text record is composed of three bytes of relocation information designating the field size, field position, and associated external index relevant to the modification of the addressed data bytes in this text record. The field may be positively or negatively relocated at link-edit time and can be modified by one or more relocation masks. The text and its associated relocation masks always must appear within the same logical record.

2. Load module relocation masks are identical, except that the ESID field represents the phase number assigned to the definition referenced by the address constant in the linked load module.



Address (in hexadecimal) pointing to the leftmost byte on the field to be modified. The position is relative to the first byte of text in the record (0 refers to the 1st byte, 1 to 2nd, etc.)

Y_4-Y_8: This 5-bit field indicates the number of bits to be modified. This number is one less than the actual number of bits used (0-31). The 7-, 15-, 23-, and 31-bit modifications may apply only to load module RLD.

Y_3: 0 - Rightmost bit of the modification field is on a byte boundary. (Always 0 for load module RLDs).

1 - Rightmost bit of the modification field is on a half-byte (hexadecimal) boundary.

Y_2: 1 - V-type address constants
0 - Others (always 0 for load module RLDs)

Y_1: Type of relocation
0 - Addition (+)
1 - Subtraction (-)

ESID: The ESID referring to the ESD entry in the module on whose value the relocatable data depends. If a load module RLD, this byte reflects the phase number of the phase supplying the definition for this reference.

Figure B-2. Relocation Mask Field

Table B-15. Object Code Transfer Record Format

Byte Position	Field	Contents
0	Length prefix	11 + RLD (binary format)
1	Type prefix	03_{16}
2	ESID	External symbol identification assigned to the transfer reference
3	Text length	3 (binary format)
4	RLD length	Number of bytes of relocation data in this record (a multiple of 3 bytes)
5	Flag	80_{16} if deferred length is present in bytes 6—8
		40_{16} if the transfer record does not terminate the object module (one or more control statements follow)
6—8	Deferred length	One CSECT or common section (named, unnamed, or blank) may have its respective record flagged to indicate that the object module transfer record specifies the actual length
9—12	Transfer address	Processor-generated object module transfer address
13—13 + RLD length	RLD data	Relocation data used to modify the transfer address

Table B-16. Object Code Control Statement Record Format

Byte Position	Field	Contents
0	Length prefix	80 (binary format)
1	Type prefix	40_{16}
2—81	Control statement	Source control statement

NOTE:

Any control statements appearing in an object module must directly follow a header record or directly follow a transfer record. The latter case is indicated by the appropriate setting of the flag byte in the transfer record.

B.4. LOAD CODE SETS

Load modules are produced by the linkage editor and are loaded in the system at program execution time by the system load facility. Load programs may be composed of more than one phase or program segment. The initial phase is called the root phase. The composition of each phase of a load program is:

- a phase definition record;
- one or more SENTRY records (optional);
- one or more resource records (optional);
- one or more SEXTRN records (optional);
- one or more ISD records (optional);
- one or more text/RLD records; and
- a transfer record.

All load programs (segmented or not) contain root phases. If the automatic overlay mechanism is used, standard text records reflecting that facility are generated into the root phase. (Automatically included modules also become resident in the root phase.) Each phase segment contains its own transfer record signaling termination of the phase and a possible start of execution address. Tables B-17 through B-21 describe the load code set records.

Table B-17. Load Code Phase Definition Record Format (Part 1 of 2)

Byte Position	Field	Contents
0	Length prefix	67 (binary format)
1	Type prefix	90_{16}
2	Phase number	Linkage editor assigned phase number of this phase
3, 4	Flag	Byte 3 Bit 0 Set in root phase header to indicate clear module partition as defined in bytes 27–30 Bit 1 Set to indicate that the load module calls reentrant code Bit 2 Set to identify the load module as reentrant Bit 3 Set to identify the load module as base 0 shared code Bit 4 Set to identify the load module as key 0 shared code Bits 5–7 Reserved Byte 4 Bit 0 Set to indicate that module has been patched Bit 1 Reserved Bits 2–7 Reserved

Table B-17. Load Code Phase Definition Record Format (Part 2 of 2)

Byte Position	Field	Contents
5–8	Phase load address	Linkage editor assigned relative origin of this phase
9–12	Phase length	Total number of bytes required for this phase segment; value represents the highest zero relative address assigned to this phase
13–20	Phase name	Symbolic name assigned to this loadable phase segment
21–23	Date	Month-day-year (packed decimal less zone field)
24, 25	Time	Hour-minute (packed decimal less zone field)
26	SENTRY count	Number of SENTRY records contained in the load module
27–30	Module length	Total number of bytes required for loading the module; value represents the highest zero relative address assigned to the load module
31–38	Alias phase name	Symbolic name assigned to this loadable phase segment by the linkage editor OVERLAY or REGION control statement that created the phase
39–68	Comments	Up to 30 bytes of pertinent comments as deemed necessary to identify the load module segment

Table B-18. Load Module Shared Code Record Formats

Byte Position	Field	Contents		
		Resource Records	SEXTRN Records	SENTRY Records
0	Length prefix	15 (binary format)	15 (binary format)	15 (binary format)
1	Type prefix	$C8_{16}$	$C6_{16}$	$C4_{16}$
2	Number	Resource number	SINDEX number	SENTRY number
3, 4	Unused			
5–8	Length	Resource size	Byte 5 has resource number Bytes 6–8 unused	Link address
9–16	Name	Resource name left-justified, and zero-filled	SEXTRN name left-justified and blank-filled	SENTRY name left-justified and blank-filled

Table B-19. Load Code ISD Record Format

Byte Position	Field	Contents
0	Length prefix	Variable
1	Type prefix	1c
2	Phase number	Linkage editor assigned phase number of this phase
3	Flag	Bit 0 set to indicate Type 1 ISD (CSECT) Bit 1 set to indicate Type 2 ISD (comment) Bit 2 set to indicate Type 3 ISD Bit 3 set to indicate Type 4 ISD (comment) Bits 4—7 unused
4	Flag	Unused
5—8	Link origin	Linkage editor assigned relative origin for this ISD record
9—12	Compile origin	Language processor generated address to the ISD record
13—16	Size	Size of this ISD record
17—250	Attributes	Symbolic name and attributes of this ISD record

Table B-20. Load Code Text/RLD Record Format

Byte Position	Field	Comments
0	Length prefix	Variable: 7 + text length + RLD length (binary format)
1	Type prefix	12_{16}
2	Phase number	Linkage editor assigned phase number of text data in this record
3	Text length	Number of bytes less 1 of text data in this record
4	RLD length	Number of bytes of relocation data in this record (a multiple of 3 bytes)
5	Flag	01_{16} if a patched text item
6—8	Load address	Linkage editor assigned phase segment load address assigned to the first byte of text data in this record
9—9+ text length	Text data	Instructions or data to be loaded relative to the load address
9 + text length + RLD length backward thru 9 + text length	RLD data	Three-byte relocation masks used to modify text in the record (Table B-14)

Table B-21. Load Code Transfer Record Format

Byte Position	Field	Comments
0	Length prefix	11 + RLD data length (binary format)
1	Type prefix	13_{16}
2	Phase number	Linkage editor assigned phase number of this phase
3	Text length	3 (binary format)
4	RLD length	Number of bytes of relocation data in this record (a multiple of 3 bytes)
5—8	Reserved	00_{16}
9—12	Transfer address	Linkage editor assigned phase segment transfer address
13—13 + RLD length	RLD data	Relocation data used to modify the transfer address

B.5. BLOCK LOAD CODE SETS

Unlike the standard load module, which has data in two partitions, the block load module has data in three partitions. The data in partitions 1 and 2 are similar to the standard load module data in that they are structured as index and data partitions. However, the data in partition 3 is not structured and is made up of contiguous text data, free of any control information. In other words, partition 3 is made up of the actual block module text records. The data in partition 2 describes the boundaries of each phase in partition 3. The block module text data (partition 3) is in sequential load order and is binary zero-filled when appropriate.

Tables B-22 through B-27 show the order of all modules within the block load code set.

Table B-22. Partition 1-Directory Entry

Byte Position	Field
0—7	Symbolic name
8	Type flag ($B0_{16}$)
9—11	Block relative pointer
12	Record relative pointer

Table B-23. Partition 2 - Block Load Module Header Record (Part 1 of 2)

Byte Position	Field	Contents
0	Length prefix	75 (binary format)
1	Type prefix	BO$_{16}$
2	Phase number	Linkage editor assigned phase number of this phase
3	Flag	80_{16} indicates clear module partition as defined in bytes 27–30. 40_{16} indicates that this module calls shared code. 20_{16} indicates that this is a base 0 shared load module. 10_{16} indicates that this is a key 0 shared load module.
4	Flag	80_{16} indicates this module has been patched.
5–8	Phase load address	Linkage editor assigned relative origin of this phase
9–12	Phase length	Total number of bytes required for this phase segment; value represents the highest relative zero address assigned to this phase.
13–20	Phase name	Symbolic name assigned to this loadable phase segment
21–23	Date	In the form as it appears in the preamble
24, 25	Time	Hour-minute (packed decimal less zone field)
26	SENTRYs	Number of SENTRYs reported
27–30	Module length	Total number of bytes required for loading the module; value represents the highest relative zero address assigned to the load module
31–38	Alias phase name	Symbolic name assigned to this loadable phase segment by the linkage editor OVERLAY or REGION control statement that created the phase
39–68	Comments	Up to 30 bytes of pertinent comments as deemed necessary to identify the load module segment
69–71	Block number	Pointer to text block (beginning of this phase in partition 3)

Table B-23. Partition 2 - Block Load Module Header Record (Part 2 of 2)

Byte Position	Field	Contents
72—74	Block number	Pointer to first text or transfer block of this phase in partition 2
75	Displacement	Pointer to first text or transfer record of this phase in partition 2
76	Checksum	XOR of first byte of each text block of partition 3

Table B-24. Partition 2 - Block Load Module RLD Record

Byte Position	Field	Contents
0	Length prefix	1 + no. of RLD times 5 (binary format)
1	Type prefix	32_{16}
2	Length of RLDs	Number of RLD masks times 5
3 (3 + n x 5—1)	RLD masks	Five-byte RLD masks (Table B—25)

Table B-25. RLD Mask

Byte Position	Contents
0	Phase number (in load module RLD mask)
1	Bits (in load module RLD masks)
2—4	Load module relative address

Table B-26. Partition 2 – Block Load Module Nonphase Text/RLD Record

Byte Position	Field	Comments
0	Length prefix	Variable: 7 + text length + RLD length (binary format)
1	Type prefix	12_{16}
2	Phase number	Linkage editor assigned phase number of text data in this record
3	Text length	Number of bytes less 1 of text data in this record
4	RLD length	Number of bytes of relocation data in thi srecrod (a multiple of three bytes)
5	Flag	01_{16} if a patched text item
6—8	Load address	Linkage editor assigned phase segment load address assigned to the first byte of text data in this record
9—9 + text length	Text data	Instructions and/or data to be loaded relative to the load address
9 + text length + RLD length backward thru 9 + text length	RLD data	Three-byte relocation masks used to modify text in the record (Table B-14)

NOTE:

Nonphase text records are present in block load modules when text/RLD items are detected that are not part of a given phase. Such text/RLD items outside the phase being loaded are to be loaded at the same time.

Table B—27. Partition 2 — Block Load Module Transfer Record

Byte Position	Field	Comments
0	Length prefix	11 + RLD data length (binary format)
1	Type prefix	13_{16}
2	Phase number	Linkage editor assigned phase number of this phase
3	Text length	3 (binary format)
4	RLD length	Number of bytes of relocation data in this record (a multiple of three bytes)
5—8	Unused	00_{16}
9—12	Transfer address	Linkage editor assigned phase segment transfer address
13—13 + RLD length	RLD data	Relocation data used to modify the transfer address

Index

SPERRY UNIVAC

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

____

(Document Title)

____________________ ____________________ ____________________

(Document No.) *(Revision No.)* *(Update No.)*

Comments:

From:

____

(Name of User)

____

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

Cut along line.

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

FOLD

CUT

SPERRY UNIVAC

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

____
(Document Title)

____________________ ____________________ ____________________
(Document No.) *(Revision No.)* *(Update No.)*

Comments:

From:

____
(Name of User)

____
(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

Cut along line.

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

FOLD

CUT

SPERRY UNIVAC

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

(Document Title)

(Document No.) *(Revision No.)* *(Update No.)*

Comments:

From:

(Name of User)

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

Cut along line.

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

FOLD

CUT