7. Program Library Details

7.1. GENERAL

The system program library files, which may be 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 the system user to realize the full extent of the capabilities of the librarian, he must be aware of the organization and content of the program libraries in the system. Thus, the organization and content of the system program library are presented in this section of the manual.

The user also may elect to establish a program library of his 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.

7.2. 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 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.

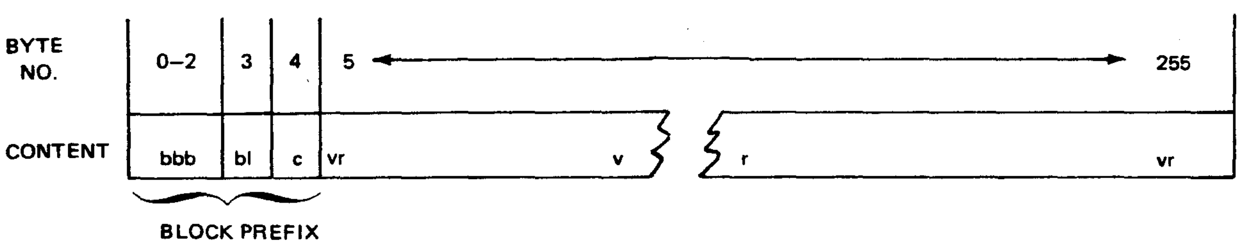
Any programs or data that may be in these files are unrecoverable once their associated jobs have been terminated.

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 version). 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.

7.2.1. Library Blocks

Library blocks are fixed-length, 256-byte blocks (Figure 7—1). 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.



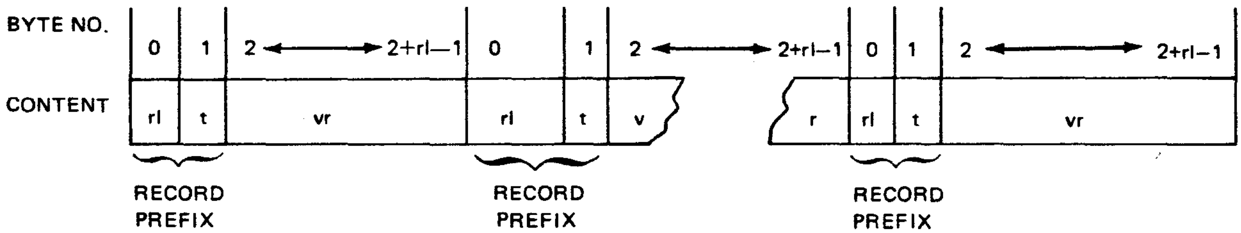
BLOCK FIELD DESCRIPTIONS

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	Check sum (c)	This is a binary value reflecting an exclusive OR of all bytes in the block.
5 — 5+bl-1	Variable records (vr)	Variable-length records comprising the body of data contained in this block

Figure 7—1. Library Block Format

7.2.2. 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 7—2). 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 7—1.)
2 — 2+rl-1	Variable-length record data (vr)	Body of the particular record (up to 249 bytes each)

Figure 7—2. Library Record Format

7.2.3. 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. A list of the record type byte values possible in an OS/3 system library file and their meanings is presented in Table 7—1. Note that the type byte field also exists in disk library directory items, as described in 7.4.

Table 7—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

Table 7—1. Record Type Byte Descriptions (Part 2 of 2)

Type Byte Value (Hexadecimal)	Description
08	Named CSECT records
09	Unnamed CSECT records
0A	Named common records
0B	Unnamed common records
0C	Object code ISD records
12	TEXT/RLD recordsiin 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
A2	Macro/JPROC name header records (in directory only)
A3	Macro/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

7.3. 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) and are described in detail in 7.3.1 through 7.3.4.

A. 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

B. PROGRAM SOURCE AND MACRO/JPROC SOURCE MODULE CODE SETS

1 header, type A3 or A4
1 or more source items, type 24 or 25

C. OBJECT CODE SETS

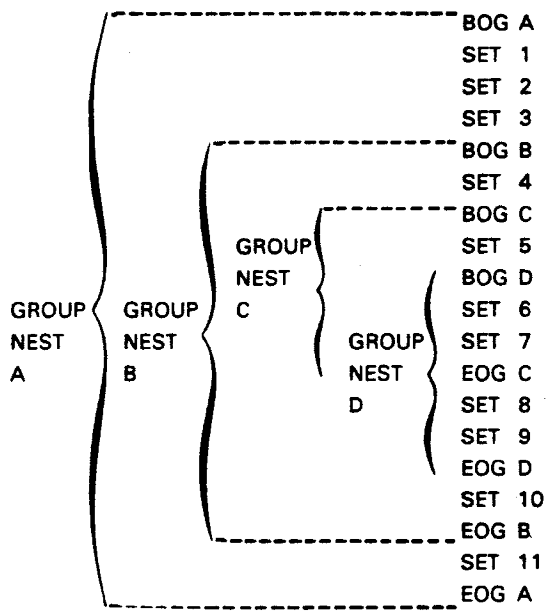
1 header, type 80
1 or more linkage editor control statements, type 40 (optional)
1 or more CSECT, types 08, 09, 0A, 0B
1 or more ESD, types 04, 06, 07 (optional)
1 or more text, type 02
1 transfer, type 03
1 or more linkage editor control statements, type 40 (optional)
1 or more ISD records, type 0C

D. 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

7.3.1. 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 type or may be a mixture of all sets in any order. The grouping is strictly optional and can be performed by the librarian at the user's 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 7—3 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. The library items peculiar to grouped code sets are described in Tables 7—2 through 7—4.



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 7—3. Example of Nested Group Code Sets

Table 7—2. 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 7—3. 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 7—4. 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△△

7.3.2. 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 JPROC's written in job control language. The library items peculiar to source code sets are described in Tables 7—5, 7—6, and 7—7.

Table 7—5. Source Module Code Header Record Format (Part 1 of 2)

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)

Table 7—5. Source Module Code Header Record Format (Part 2 of 2)

Byte Position	Field	Contents
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 7—6. 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 7—7. 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

7.3.3. 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:

Object module header record

Control statement 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.*

All control section records (must precede associated text and entry ESDs)

All ESD records (names must be unique)

* All ISD records

All text/RLD records

Object module transfer record

Control statement records

These records are described in Tables 7—8 through 7—17;

Table 7—8. 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 — Not used 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 byes of pertinent comments as deemed necessary to identify the object module

ISD records are also 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 7—9. 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 7—10.)
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 7—10. 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 7—11. 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 7—12.)
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 7—12. 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 7—13. 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 7—14. 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 7—15.)

Table 7—15. 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 7—4)
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.

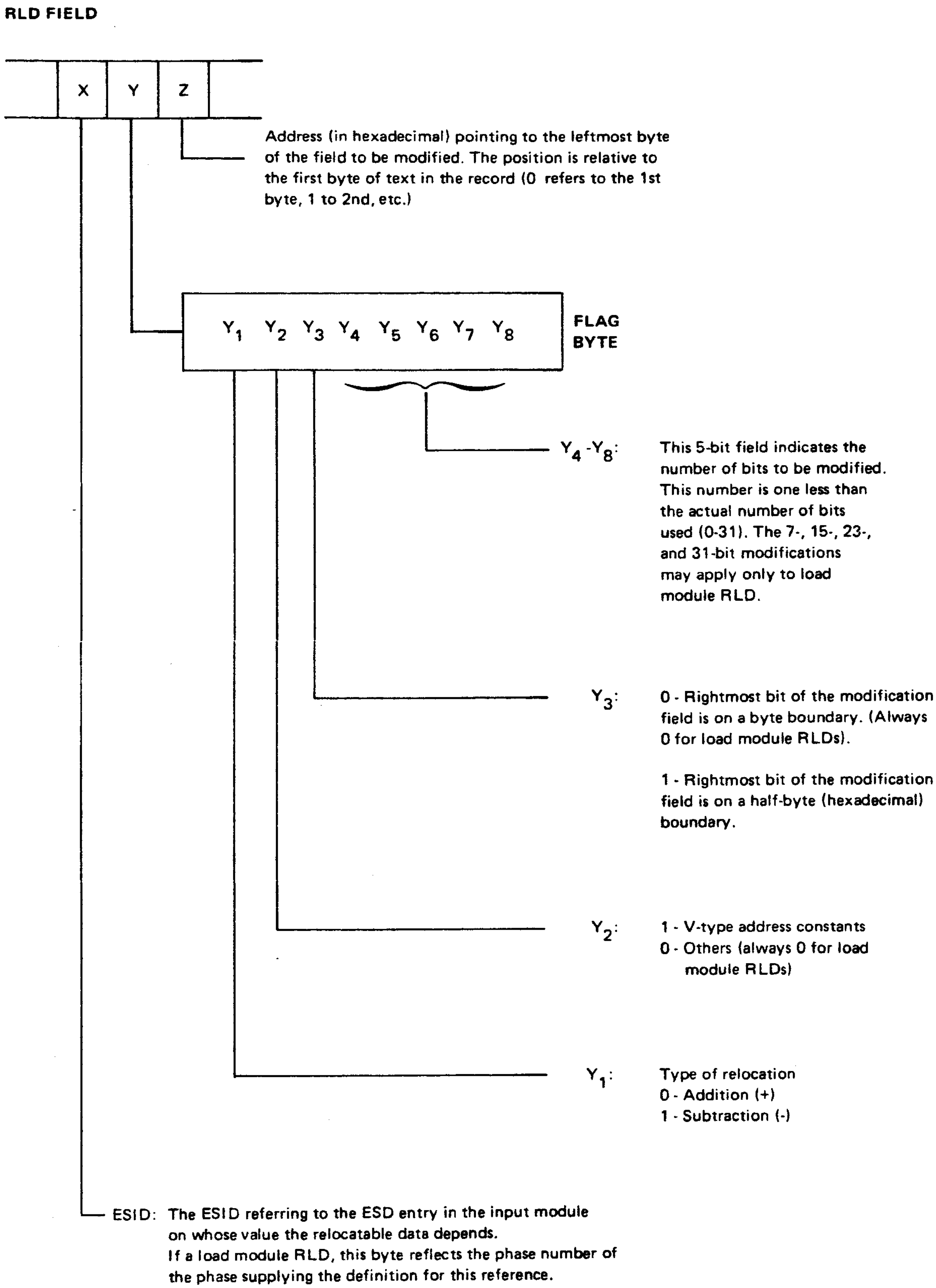


Figure 7–4. Relocation Mask Field

Table 7—16. 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 (1 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 7—17. 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.

7.3.4. 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. The load code set records are described in Table 7—18 through 7—22.

Table 7—18. 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	<u>Byte 3</u> 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 Bits 3–7 — Not used <u>Byte 4</u> Bit 0 — Set to indicate that module has been patched Bits 1–7 — Not used
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)

Table 7—18. Load Code Phase Definition Record Format (Part 2 of 2)

Byte Position	Field	Contents
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 7—19. 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 7—20. Load Code ISD Record Format (Part 1 of 2)

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

Table 7—20. Load Code ISD Record Format (Part 2 of 2)

Byte Position	Field	Contents
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 7—21. 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 7—15)

Table 7—22. 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	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

7.3.5. 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 one and two are similar to the standard load module data in that they are structured as index and data partitions. However, the data in partition three is not structured and is made up of contiguous text data, free of any control information. In other words, partition three is made up of the actual block module text records. The data in partition two describes the boundaries of each phase in partition three. The block module text data (partition three) is in sequential load order and is binary zero-filled when appropriate.

The order of all modules within the block load code set is shown in Tables 7—23 through 7—28.

Table 7—23. Partition One—Directory Entry

Byte Position	Field
0—7	Symbolic name
8	Type flag (80_{16})
9—11	Block relative pointer
12	Record relative pointer

Table 7—24. Partition Two — Block Load Module Header Record (Part 1 of 2)

Byte Position	Field	Contents
0	Length prefix	75 (binary format)
1	Type prefix	80_{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 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.

Table 7—24. Partition Two — Block Load Module Header Record (Part 2 of 2)

Byte Position	Field	Contents
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 recorded
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 three)
72—74	Block number	Pointer to first text or transfer block of this phase in partition two
75	Displacement	Pointer to first text or transfer record of this phase in partition two
76	Checksum	XOR of first byte of each text block of partition three

Table 7—25. Partition Two — 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	5 byte RLD masks (see Table 7—26)

Table 7—26. 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 7—27. Partition Two — Block Load Modules 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 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 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 7—15)

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 7—28. Partition Two — 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 3 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

7.4. 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 not a unique corresponding record in the prime data partition. This item points to the module header record.)

7.4.1. 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 7—5 illustrates the disk library file structure and the format of each directory record.

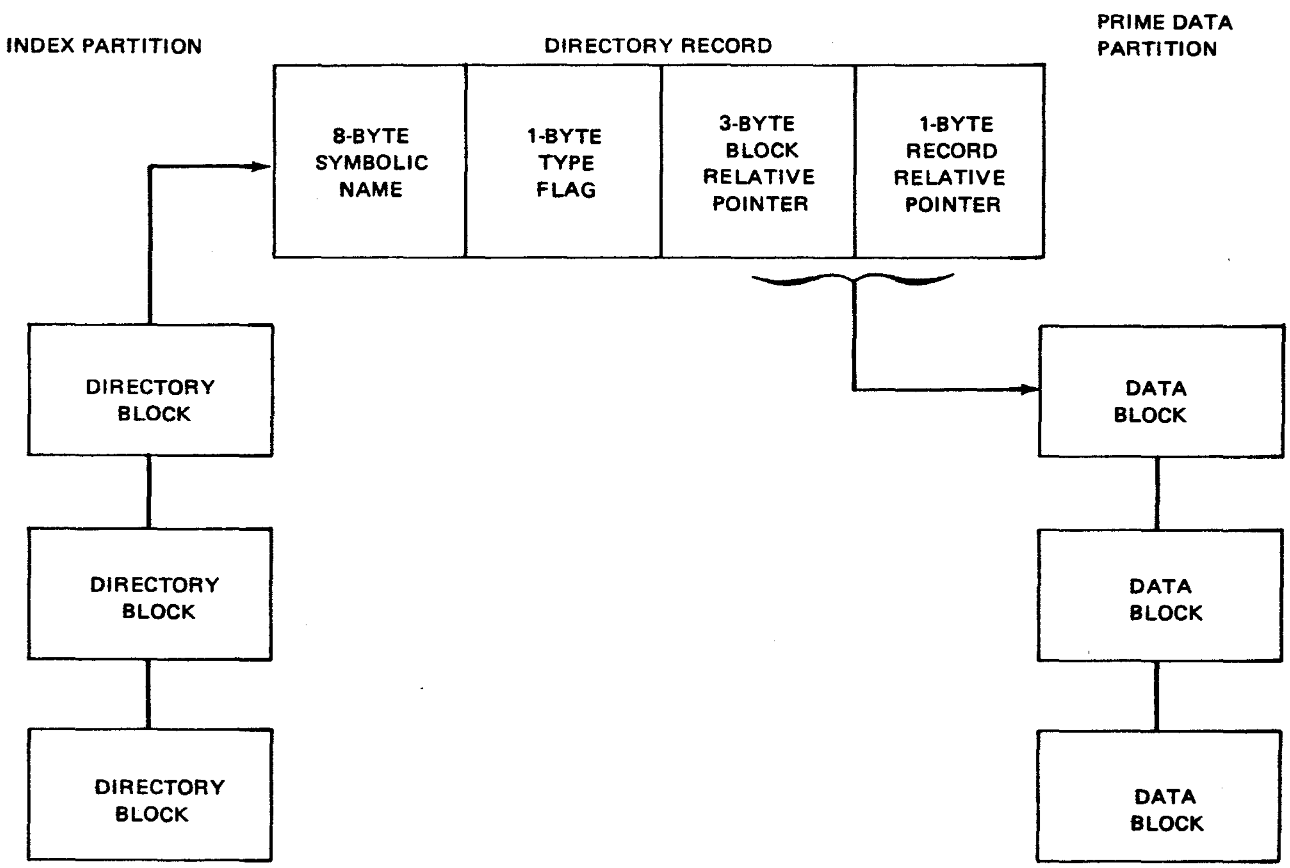


Figure 7—5. Disk Library File Structure

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. The type flags possible in an index item are listed in Table 7—29.

Table 7—29. 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.

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.

7.5. CARD LIBRARIES

The librarian can punch libraries into cards and, in turn, access card files are 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.

7.6. 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: BLK, DEL, PAC, REC, REN, and REPRO.

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 using either the prep option of the VOL job control statement or through the tape prep routine (TPREP, Section 9).

7.7. DISKETTE LIBRARIES

The librarian can either be input from a diskette, or punch to a diskette. Diskette library processing is the same as card library processing (7.5.).

```
                             515 *  PRINT AN ERROR MESSAGE AND TERMINATE READING FOR THIS LIBRARY.
                             516 *
2440            004A0        517 GETSPACE L     9,NEXTSPAC                START OF AVAILABLE SPACE
1000            00000        518 INCRSPAC LA    15,0(9,1)                 END OF DESIRED ENTRY
24A4            004A4        519          C     15,ATABEND                IS THERE ROOM?
2376            00376        520          BH    OVERFLOW                  NO
2440            004A0        521          ST    15,NEXTSPAC               UPDATE NEXT SPACE POINTER
                             522          BR    14                        EXIT
2549 278F 00549 0078F        523 OVERFLOW MVC   LINE(39),=CL39'**** MEMORY OVERFLOW - INPUT TERMINATED'
229             0029C        524          BAL   14,PRINT                  PRINT ERROR MESSAGE
2230            00230        525          B     PRNTXREF                  START XREF IMMEDIATELY
                             526 *
                             527 * LIBRARY FILE GET ROUTINE
                             528 *  THE CURRENT RECORD, IF A SOURCE RECORD, WILL BE PROPERLY EXPANDED
                             529 *  INTO "CARD" - OTHERWISE IT WILL BE LOADED INTO "CARD" AS IS.
                             530 *  IN EITHER CASE, THE RECORD TYPE CODE WILL BE PLACED INTO "RECTYPE".
                             531 *  IF THE RECORD IS AN EOF SENTINEL, THIS ROUTINE WILL BRANCH
                             532 *  DIRECTLY TO "PRNTXREF".
                             533 *
24AC            004AC        534 GETS     ST    14,SAVE14                 SAVE RETURN ADDRESS
2498            00498        535          L     15,BPSOURCE               CURRENT POSITION IN BLOCK
2758            00758        536          C     15,=A(IOLIBIN+5)          START OF A NEW BLOCK?
23B4            003B4        537          BH    GETSREC                   NO
                             538          WAITF LIBIN                     WAIT FOR COMPLETION OF READ
                             544          SR    1,1
289F            0089F        545          IC    1,IOLIBIN+3               BLOCK LENGTH
28A1            008A1        546          LA    1,IOLIBIN+5(1)
24A8            004A8        547          ST    1,BLOCKEND                LOGICAL END OF BLOCK
2498            00498        548          L     15,BPSOURCE               RESTORE REGISTER 15
26ED F001 006ED 00001        549 GETSREC  MVC   RECTYPE,1(15)             RECORD TYPE CODE
24F8      004F8              550          MVI   CARD,C' '                 CLEAR RECORD IMAGE AREA
24F9 24F8 004F9 004F8        551          MVC   CARD+1(79),CARD
                             552          SR    1,1
F000            00000        553          IC    1,0(,15)                  RECORD LENGTH
F002            00002        554          LA    15,2(,15)                 SKIP OVER RECORD PREFIX
                             555          LTR   1,1                       IS RECORD LENGTH ZERO?
2426            00426        556          BZ    GETSNEXT                  YES - THERE'S NOTHING TO MOVE
24F8            004F8        557          LA    14,CARD                   R14 SCANS CARD IMAGE
26ED      006ED              558          CLI   RECTYPE,X'25'             COMPRESSED SOURCE RECORD?
23FC            003FC        559          BE    GETSCOMP                  YES - USE SPECIAL ROUTINE
```



```
CT CODE   ADDR1 ADDR2   LINE    SOURCE STATEMENT                                          OS/3 ASM 77/12/1
                             560          BCTR  1,0                       DECREMENT FOR EXECUTE
23F6            003F6        561          EX    1,GETSMOVE                LOAD RECORD IMAGE AS IS
F001            00001        562          LA    15,1(1,15)                INCREMENT BLOCK POINTER
26ED      006ED              563          CLI   RECTYPE,X'A1'             END OF FILE SENTINEL?
2426            00426        564          BNE   GETSNEXT                  NO
2230            00230        565          B     PRNTXREF                  START PRINTING CROSS-REFERENCE
E000 F000 00000 00000        566 GETSMOVE MVC   0(0,14),0(15)             EXECUTED TO MOVE PART OF RECORD
                             567 * EXPANSION ROUTINE FOR COMPRESSED SOURCE RECORDS
                             568 GETSCOMP LR    0,15                      CURRENT POSITION IN BLOCK
                             569          AR    0,1                       ADD RECORD LENGTH
2488            00488        570          ST    0,WORK                    THE RECORD ENDS HERE
F001            00001        571 GETSCMPL IC    1,1(,15)                  NUMBER OF BLANKS
                             572          AR    14,1                      INCREMENT CARD IMAGE POINTER
F000            00000        573          IC    1,0(,15)                  LENGTH OF FOLLOWING TEXT
F002            00002        574          LA    15,2(,15)                 SKIP OVER CONTROL FIELD
23F6            003F6        575          EX    1,GETSMOVE                MOVE FOLLOWING TEXT
E001            00001        576          LA    14,1(1,14)                INCREMENT CARD IMAGE POINTER
F001            00001        577          LA    15,1(1,15)                INCREMENT BLOCK POINTER
2488            00488        578          C     15,WORK                   END OF COMPRESSED RECORD?
2404            00404        579          BL    GETSCMPL                  NO - CHECK FOR MORE
2498            00498        580 GETSNEXT ST    15,BPSOURCE               UPDATE BLOCK POINTER
24A8            004A8        581          C     15,BLOCKEND               END OF BLOCK?
244A            0044A        582          BL    GETSX                     NO - EXIT
                             583          GET   LIBIN,LIBIN2              START READING THE NEXT BLOCK
2498 2758 00498 00758        590          MVC   BPSOURCE,=A(IOLIBIN+5)    RESET BLOCK POINTER
24AC            004AC        591 GETSX    L     14,SAVE14                 RESTORE RETURN ADDRESS
                             592          BR    14                        EXIT
                             593 *
                             594 * I/O ERROR ROUTINES
                             595 *
2549 2785 00549 00785        596 ERLIBIN  MVC   LINE(23),=CL23'**** LIBIN I/O ERROR AT'
2488            00488        597          ST    14,WORK
2561 2488 00561 00488        598          UNPK  LINE+24(7),WORK(5)
2561 25ED 00561 005ED        599          TR    LINE+24(6),HXTR-X'F0'
2567 27CC 00567 007CC        600          MVC   LINE+30(17),=CL17' - STATUS BYTES ='
2579 2822 00579 00822        601          UNPK  LINE+48(5),LIBIN+50(3)
2579 25ED 00579 005ED        602          TR    LINE+48(4),HXTR-X'F0'
```