REFERENCE MANUAL

DMF COMMON-RESIDENT LIOCS (LIOCS-C)

SYSTEM TEN COMPUTER BY SINGER

SINGER
BUSINESS MACHINES

REFERENCE MANUAL

DMF COMMON-RESIDENT LIOCS (LIOCS-C)

SYSTEM TEN* COMPUTER BY SINGER

PUBLICATION NO. 40-340
CONTROL NO. B554PA
FEBRUARY 1972

SINGER
BUSINESS MACHINES
2350 WASHINGTON AVE.
SAN LEANDRO, CALIF. 94577

*A trademark of the Singer Company.

PRINTED IN U.S.A.

PREFACE

This manual describes features provided by common-resident LIOCS (LIOCS-C) that augment the capabilities of partition-resident LIOCS described in the DMF Reference Manual. Functions not described in this manual operate in the manner specified in the DMF Reference Manual.

TABLE OF CONTENTS

Section 1
INTRODUCTION

ADDITIONAL CAPABILITIES PROVIDED BY LIOCS-C
DIFFERENCES BETWEEN LIOCS-C AND PARTITION LIOCS
CHOOSING BETWEEN COMMON AND PARTITION LIOCS

ADDITIONAL CAPABILITIES PROVIDED BY LIOCS-C

LIOCS-C differs from its partition-resident counterpart in capability, location, and user interface. The most significant additional capabilities are:

1. Multi-sector records are supported. While the only limit on record length is the amount of core available for the I/O area, any record that is to be accessed by RPG, or is to be sorted, should not exceed 940 data characters (10 sectors).

2. Record contention, i.e., the attempt by two or more partitions to update the same record, can be prevented by utilizing two new operations as follows:

   _GETUP (_GET for update)

   and

   _READU (_READ for update)

3. Disc arm movement caused by switching to another partition during a multi-sector I/O operation, is prevented by "locking" the disc drive to the controlling partition until the operation is complete. This feature can reduce delays in multi-partition systems that are caused by extraneous arm movement.

4. _READ now locates and retrieves the data record in an indexed-linked sequential file, that is, whether or not there is a one-for-one entry in the file index.

5. LIOCS-C contains an overlay routine to fetch or load modules, by file name, that are stored in SYSPOL.

6. A "system lock" is provided to prevent conflict between partitions that are simultaneously allocating free sectors to files in the same pool.

The bulk of LIOCS-C code is located in common and is re-entrant, that is, it allows concurrent use by many partitions. LIOCS-C exits and switches, which are modified during execution, reside in locations 750-999 of each partition.

DIFFERENCES BETWEEN LIOCS-C AND PARTITION LIOCS

The user interface to LIOCS-C differs from partition LIOCS as follows:

1. LIOCS-C is not assembled with user programs. The programmer need only include the common-LIOCS interface macro (CLIOIN) in his program, and all necessary references to the LIOCS-C transfer vector in common will be generated. The transfer vector contains the entry points for all LIOCS-C operations.

2. The file control blocks (FCB's) are 20 characters larger for LIOCS-C. Linked sequential files thus have a 94 character FCB. An FCB macro (FCB) is supplied that will generate FCB's, and it is strongly recommended that this macro be used exclusively to generate FCB's.

3. A new open (OPENC), close (CLOSEC), and conversational loader (C_LOAD) are used with LIOCS-C. These modules, in addition to LIOCSC and P_COMM, must all reside in SYSPOL.

4. The recommended assembler for LIOCS-C is Assembler II; this allows the programmer to use the macros that are supplied for generating FCB's, transfer vector references, and calling sequences (GET, PUT, and so on). These must all be hand-coded each time for use with Assembler I. In addition, future changes to LIOCS-C may require modifications to the FCB's, transfer vectors, and so on. By using Assembler II and the supplied macros, implementing such a change would require only refiling the affected macros and reassembling. When using Assembler I, each program would require program changes that must be provided by hand, thus increasing programmer effort significantly.

5. LIOCS-C must be loaded into common core normally at the start of each day's operation.

CHOOSING BETWEEN COMMON AND PARTITION LIOCS

Deciding whether a System Ten installation should use common or partition LIOCS depends on a number of inter-related factors which follow:

1. Core size and the number of partitions. Partition LIOCS requires approximately 1500 to 4000 positions of core, depending on the operations being used. LIOCS-C currently requires approximately 6000 positions in common core (in addition to the 1000 positions used by the hardware and system constants, tables, and the common mail box). Depending on the operations being used, then, the total amount of core required for partition LIOCS exceeds the core required for LIOCS-C when two to four partitions are simultaneously using partition LIOCS (refer to Figure 1-1).

NUMBER OF PARTITIONS USING LIOCS	TOTAL CORE USED BY PARTITION-RESIDENT LIOCS ROUTINES		
	MINIMUM	AVERAGE	MAXIMUM
1	1,500	2,500	4,000
2	3,000	5,000	8,000
3	4,500	7,500	12,000
4	6,000	10,000	16,000
5	7,500	12,500	20,000

Note: Shaded areas indicate where partition LIOCS uses less total core than LIOCS-C.

Figure 1-1 TOTAL CORE USED BY PARTITION LIOCS

2. Contention versus non-contention use of disc. When applications will be contending for the same pools or files from different partitions, LIOCS-C should be used.

3. LIOCS-C should be used if its capabilities outlined above are needed.

4. If the System Ten installation does not presently indicate the use of LIOCS-C, but it is expected to expand and require LIOCS-C in the future, a conversion effort can be avoided by using LIOCS-C.

5. More advanced future software will require the use of LIOCS-C. If the most capable version of software is wanted (e.g.,RPG), LIOCS-C should be used.

Section 2
LIOCS-C

OPERATION
FILE CONTROL BLOCKS
SUBROUTINES
CONSIDERATIONS

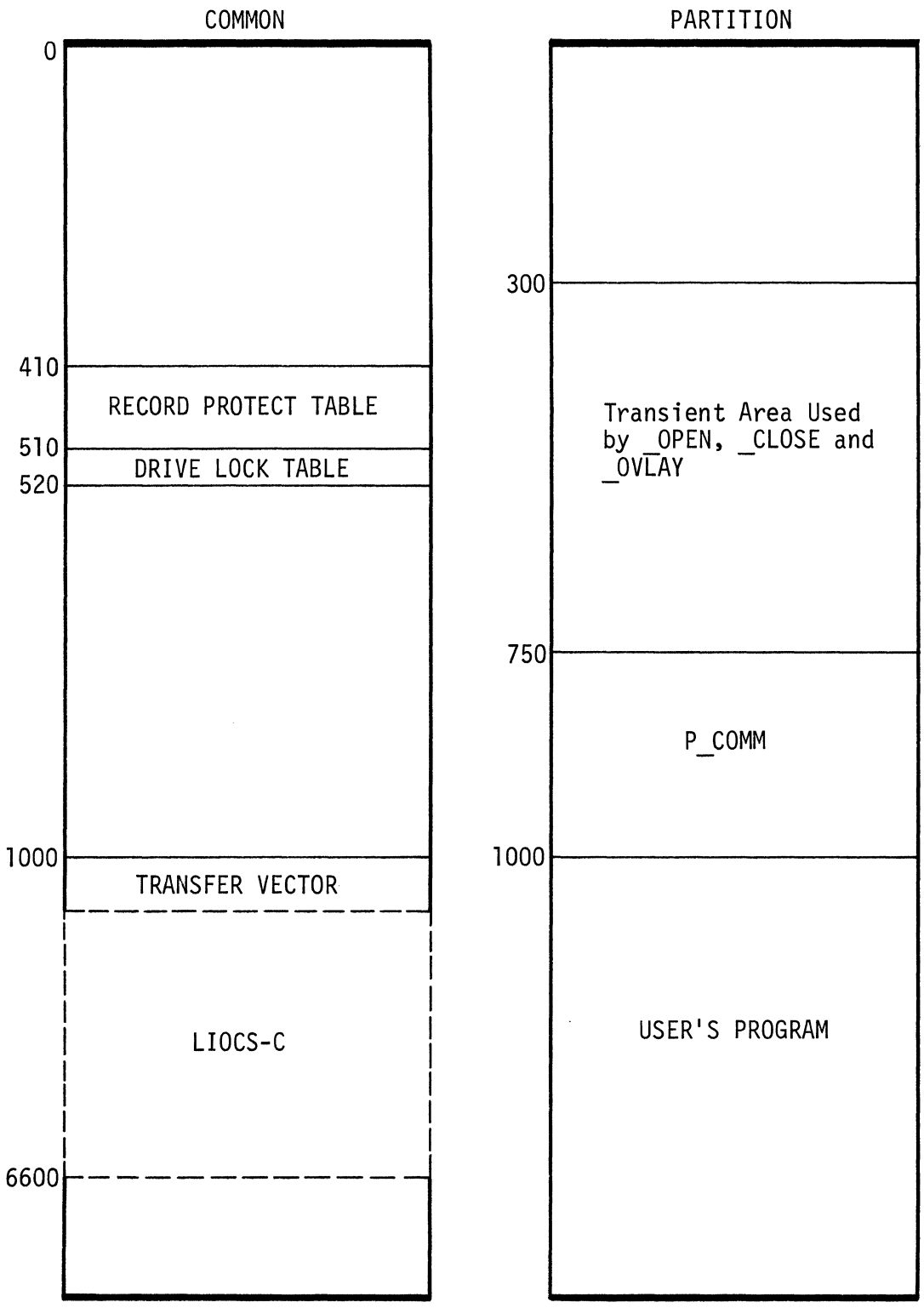

Figure 2-1 LOCATION OF LIOCS-C MODULES

OPERATION

At the beginning of each day, in response to:

```
A)ENTER PROGRAM NAME.
```

the following dialogue takes place with the user on a Model 70 Workstation or Model 80 Video Display:

```
LIOCSC
I)LIOCS-C (xxxxxx: REC-PRO, DR-LOCK OPTIONS) LOADED
```

Note: xxxxxx is date in YYMMDD format

At this point, if the system program UDATE is filed in SYSPOL, it will automatically be loaded to enable the user to set the User Date field in Common:

```
A)SET DATE
```

If no UDATE is found in SYSPOL or after the date has been entered in MM/DD/YY format the following message will be displayed:

```
A)ENTER PROGRAM NAME.
MYPROG              Load User's Program
```

Once LIOCS-C has been loaded, all partitions have access to it. Even if a program crashes or ends it is not necessary to re-load LIOCS-C.

Hardware requirements are the same as those specified in the DMF Manual, except Common must now be a minimum of 7K to use LIOCS-C and a 10K Common is recommended.

Core memory requirements for LIOCS-C are shown in Figure 2-1. Beginning at location 1000 in common is the LIOCS-C transfer vector, coding the user references to enter LIOCS-C. This coding is referenced via the user's interface generated by macro CLIOIN in the user's program (refer to Figures 4-2 and 4-3). The location of the transfer vector will not change so user programs will not have to be re-assembled should LIOCS-C change.

OPENC and CLOSEC are similar to their counterparts in the existing DMF, with the exception that after execution they return to the user's program via LIOCS-C.

```
          FILE NAME   TYPE    CHANGE    DATE OF     DATE OF        DESCRIPTION
                              LEVEL    EXPIRATION   CREATION

            SAMPLE     0       000      01-04-72    00-00-00     LIOCS-C SAMPLE

RECORD NO.

     1          TITLE  'TYPICAL USES OF LIOCS-C'                                 SAMPLE000100          010521
     2    ************************************************************************ SAMPLE000200        010522
     3    *  MACROS TO BE INCLUDED TO DEFINE SYSTEM AREAS                        * SAMPLE000300        010523
     4    ************************************************************************ SAMPLE000400        010524
     5          -CCDEF       * DEFINE COMMON CORE AREAS                          SAMPLE000500          010525
     6          -LCDEF       * DEFINE PARTITION LOW CORE                         SAMPLE000600          010556
     7          EJECT                                                            SAMPLE000700          010526
     8          CLIOIN       * DEFINE LIOCS-C TRANSFER VECTOR                    SAMPLE000800          010555
     9          EJECT                                                            SAMPLE000900          010527
    10    ************************************************************************ SAMPLE001000        010528
    11    *  BODY OF PROGRAM   - -   LIOCS-C MACRO CALLS                         * SAMPLE001100        010529
    12    ************************************************************************ SAMPLE001200        010561
    13          NORMAL                                                           SAMPLE001300          010562
    14          ORG    1000                                                      SAMPLE001400          010530
    15    START OPEN   EXTFCB,ERROR      *OPEN AN EXTEND FILE                    SAMPLE001500          010531
    16          OPEN   INDFCB,ERROR      *OPEN AN INDEXED FILE                   SAMPLE001600          010532
    17          GETUP  EXTFCB,ERROR      *GET AND LOCK A RECORD                  SAMPLE001700          010533
    18    *<<<<  USER'S PROCESSING TAKES PLACE HERE  >>>>                        SAMPLE001800          010534
    19          UPDATE EXTFCB,ERROR      *UPDATE AND UNLOCK THE RECORD           SAMPLE001900          010535
    20    *<<<<  REMAINDER OF USER'S PROCESSING >>>>                             SAMPLE002000          010563
    21    ***   USER'S EOF ROUTINE.                                              SAMPLE002100          010564
    22    EOF   EQU    *                                                         SAMPLE002200          010565
    23          EOF    INDFCB,*+10                                               SAMPLE002300          010566
    24          CLOSE  INDFCB,ERROR                                              SAMPLE002400          010567
    25          EOF    EXTFCB,*+10                                               SAMPLE002500          010568
    26          CLOSE  EXTFCB,ERROR                                              SAMPLE002600          010557
    27          EJECT                                                            SAMPLE002700          010536
    28    ************************************************************************ SAMPLE002800        010537
    29    *  USER FCB'S GENERATED BY FCB MACRO                                   * SAMPLE002900        010538
    30    ************************************************************************ SAMPLE003000        010558
    31          SPACE  2                                                         SAMPLE003100          010539
    32    EXTFCB FCB   NAME=PNAME.FNAME1,USE=EXTEND,AREA=WKAREA,EXIT=EOF,       XSAMPLE003200          010540
    33                 BLKL=470,TYPE=LS                                          SAMPLE003300          010559
    34          SPACE  2                                                         SAMPLE003400          010541
    35    INDFCB FCB   NAME=PNAME.FNAME2,USE=EXTEND,AREA=WORKA2,EXIT=EOF,       XSAMPLE003500          010542
    36                 BLKL=188,KEYAD=KEY,TYPE=ILS                               SAMPLE003600          010560
    37          EJECT                                                            SAMPLE003700          010543
    38    ************************************************************************ SAMPLE003800        010544
    39    *  PROGRAM CONSTANT AND WORK AREAS                                     * SAMPLE003900        010545
    40    ************************************************************************ SAMPLE004000        010546
    41    ERROR  EQU   *                                                         SAMPLE004100          010547
    42    ***   USER'S ERROR ROUTINE                                             SAMPLE004200          010550
    43    WKAREA DM    470C' '      *WORK AREA FOR EXTEND FILE                   SAMPLE004300          010551
    44    WORKA2 DM    188C' '      *WORK AREA FOR INDEX FILE                    SAMPLE004400          010552
    45    KEY    DM    N4'0'        *INDEX KEY ARGUMENT                          SAMPLE004500          010553
    46    ************************************************************************ SAMPLE004600        010554
    47          END    START                                                     SAMPLE004700          //////

FILE SAMPLE IN POOL SOURCE CONTAINS     47 SECTORS
```

Figure 2-2 SAMPLE PROGRAM UTILIZING LIOCS-C MACROS

```
 SYSTEM TEN ASSEMBLER II      TYPICAL USES OF LIOCS-C                                                       01/04/72    PAGE 0003

SEQ. LOCN  INSTR/DATA OP  A/R   M I  B/S   M I   LINE                                       IMAGE                        C

0008                                              0095           CLIOIN       * DEFINE LIOCS-C TRANSFER VECTOR
0003 0000                0000                     0096G *CLIOI  ORG      *
0004 1000C                                        0097G          COMMON
0005 1000C               1000C                    0098G          ORG      1000
0006 1000C               1000C                    0099G *BASEC  ORG      *                    BASE OF TRANSFER VECTOR
0007 1000C               1020C                    0100G *GET    ORG      *BASEC+20
0008 1020C               1040C                    0101G *PUT    ORG      *BASEC+40
0009 1040C               1060C                    0102G *INSRT  ORG      *BASEC+60
0010 1060C               1080C                    0103G *DLETE  ORG      *BASEC+80
0011 1080C               1100C                    0104G *UPDTE  ORG      *BASEC+100
0012 1100C               1120C                    0105G *BOF    ORG      *BASEC+120
0013 1120C               1140C                    0106G *EOF    ORG      *BASEC+140
0014 1140C               1160C                    0107G *WRTEF  ORG      *BASEC+160
0015 1160C               1180C                    0108G *READ   ORG      *BASEC+180
0016 1180C               1200C                    0109G *WRITE  ORG      *BASEC+200
0017 1200C               1210C                    0110G *OPEN   ORG      *BASEC+210
0018 1210C               1220C                    0111G *CLOSE  ORG      *BASEC+220
0019 1220C               1230C                    0112G *GETUP  ORG      *BASEC+230
0020 1230C               1250C                    0113G *READU  ORG      *BASEC+250
0021 1250C               1270C                    0114G *OVLAY  ORG      *BASEC+270           LOAD OVERLAY FROM SYSPOL
0022 1270C               1300C                    0115G *RETRY  ORG      *BASEC+300           RESUME AFTER DISC FAULT
0023 1300C               1300C                    0116G *OVRET  ORG      *                    OPTIONAL USER 'EXEC' ADDRESS
0024 0000                                         0117G          NORMAL
0025 0000                0990                     0118G          ORG      990
0026 0990                0001        0010         0119G *USERX  DM       C10                 USER EXITS
0027 1000                0000                     0120G          ORG      *CLIOI              RESTORE USER LOCATION COUNTER
```

Figure 2-3 LIOCS-C TRANSFER VECTOR GENERATED BY CLIOIN MACRO

```
 SYSTEM TEN ASSEMBLER II      TYPICAL USES OF LIOCS=C                                  01/04/72     PAGE 0004

SEQ. LOCN  INSTR/DATA OP  A/R  M I  B/S  M I   LINE                    IMAGE                                  C

0010                                           0123  ************************************************************
0011                                           0124  *  BODY OF PROGRAM  - -  LIOCS=C MACRO CALLS                *
0012                                           0125  ************************************************************
0013 0000                                      0126        NORMAL
0014 0000                1000                  0127        ORG    1000
0015                                           0128  START OPEN   EXTFCB,ERROR    *OPEN AN EXTEND FILE
     1000  V0PS15120P 11 0031  6 0 1200C 5 0   0136G START BC     31(6),*OPEN(5)
     1010  P1QV091364 11 1160  0 0 1364  9 0   0138G       BC     EXTFCB(0),ERROR(9)
0016                                           0140        OPEN   INDFCB,ERROR    *OPEN AN INDEXED FILE
     1020  V0PS15120P 11 0031  6 0 1200C 5 0   0148G       BC     31(6),*OPEN(5)
     1030  P1RU491364 11 1254  0 0 1364  9 0   0150G       BC     INDFCB(0),ERROR(9)
0017                                           0152        GETUP  EXTFCB,ERROR    *GET AND LOCK A RECORD
     1040  V0PS15122P 11 0031  6 0 1220C 5 0   0155G       LINK   31,*GETUP
     1050  P1QV091364 11 1160  0 0 1364  9 0   0157G       BC     EXTFCB(0),ERROR(9)
0018                                           0159  *<<<< USER'S PROCESSING TAKES PLACE HERE  >>>>
0019                                           0160        UPDATE EXTFCB,ERROR    *UPDATE AND UNLOCK THE RECORD
     1060  V0PS15108P 11 0031  6 0 1080C 5 0   0171G       BC     31(6),*UPDTE(5)
     1070  P1QV091364 11 1160  0 0 1364  9 0   0173G       BC     EXTFCB(0),ERROR(9)
0020                                           0175  *<<<< REMAINDER OF USER'S PROCESSING >>>>
0021                                           0176  ***   USER'S EOF ROUTINE.
0022 1080                1080                  0177  EOF   EQU    *
0023                                           0178        EOF    INDFCB,*+10
     1080  V0PS15112P 11 0031  6 0 1120C 5 0   0190G       BC     31(6),*EOF(5)
     1090  P1RU491100 11 1254  0 0 1100  9 0   0192G       BC     INDFCB(0),*+10(9)
0024                                           0194        CLOSE  INDFCB,ERROR
     1100  V0PS15121P 11 0031  6 0 1210C 5 0   0202G       BC     31(6),*CLOSE(5)
     1110  P1RU491364 11 1254  0 0 1364  9 0   0204G       BC     INDFCB(0),ERROR(9)
0025                                           0206        EOF    EXTFCB,*+10
     1120  V0PS15112P 11 0031  6 0 1120C 5 0   0218G       BC     31(6),*EOF(5)
     1130  P1QV091140 11 1160  0 0 1140  9 0   0220G       BC     EXTFCB(0),*+10(9)
0026                                           0222        CLOSE  EXTFCB,ERROR
     1140  V0PS15121P 11 0031  6 0 1210C 5 0   0230G       BC     31(6),*CLOSE(5)
     1150  P1QV091364 11 1160  0 0 1364  9 0   0232G       BC     EXTFCB(0),ERROR(9)
```

Figure 2-4 SAMPLE LIOCS-C MACRO CALLS

```
SYSTEM TEN ASSEMBLER II      TYPICAL USES OF LIOCS-C                                        01/04/72    PAGE 0005

SEQ. LOCN  INSTR/DATA OP  A/R   M I  B/S   M I   LINE                                  IMAGE                        C

0028                                             0235  *******************************************************************
0029                                             0236  *  USER FCB'S GENERATED BY FCB MACRO                                *
0030                                             0237  *******************************************************************

0032                                             0239  EXTFCB FCB    NAME=PNAME,FNAME1,USE=EXTEND,AREA=WKAREA,EXIT=EOF,      X
0033                                                                 BLKL=470,TYPE=LS
0004                                             0240G **LIOCS FCB (711220)
      1160               1160                    0263G EXTFCB EQU    *(94)
      1160  0            0001      0001          0267G        DM     C'0'                        FILE TYPE : LS
      1161  PNAME        0001      0006          0269G        DM     C6'PNAME'                 POOL NAME
      1167  FNAME1       0001      0006          0271G        DM     C6'FNAME1'                  FILE NAME
      1173  1364         0001      0004          0273G        DM     A'WKAREA'                      WORK AREA ADDRESS
      1177  0470         0001      0004          0275G        DM     A'470'                       BLOCK LENGTH
      1181  1080         0001      0004          0277G        DM     A'EOF'                    USER EXIT ADDRESS
      1185               0001      0006          0279G        DM     C6'       '                 PASSWORD
      1191  010          0001      0003          0281G        DM     C'010'                      ACTION FLAGS : EXTEND
      1194  $$$$         0001      0004          0283G        DM     C'$$$$'                        SECONDARY ALLOCATION
      1198  $$$$         0001      0004          0285G        DM     C'$$$$'                        PRIMARY ALLOCATION
      1202               0001      0052          0289G        DM     C052' '                     (RESERVED)

0035                                             0292  INDFCB FCB    NAME=PNAME,FNAME2,USE=EXTEND,AREA=WORKA2,EXIT=EOF,      X
0036                                                                 BLKL=188,KEYAD=KEY,TYPE=ILS
0004                                             0293G **LIOCS FCB (711220)
      1254               1254                    0318G INDFCB EQU    *(110)
      1254  I            0001      0001          0322G        DM     C'I'                        FILE TYPE : ILS
      1255  PNAME        0001      0006          0324G        DM     C6'PNAME'                 POOL NAME
      1261  FNAME2       0001      0006          0326G        DM     C6'FNAME2'                  FILE NAME
      1267  1834         0001      0004          0328G        DM     A'WORKA2'                      WORK AREA ADDRESS
      1271  0188         0001      0004          0330G        DM     A'188'                       BLOCK LENGTH
      1275  1080         0001      0004          0332G        DM     A'EOF'                    USER EXIT ADDRESS
      1279               0001      0006          0334G        DM     C6'       '                 PASSWORD
      1285  010          0001      0003          0336G        DM     C'010'                      ACTION FLAGS : EXTEND
      1288  $$$$         0001      0004          0338G        DM     C'$$$$'                        SECONDARY ALLOCATION
      1292  $$$$         0001      0004          0340G        DM     C'$$$$'                        PRIMARY ALLOCATION
      1296  2022         0001      0004          0343G        DM     A'KEY'                    KEY ADDRESS
      1300               0001      0064          0346G        DM     C064' '                     (RESERVED)
```

Figure 2-5 SAMPLE FCB MACRO GENERATION

LIOCS-C

UFCB 1	UFCB 2	UFCB 3	UFCB 4	UFCB 5	UFCB 6
0 0	1 6	7 12	13 16	17 20	21 24
FCB TYPE	POOL NAME	FILE NAME	WORK AREA ADDRESS	LOGICAL RECORD SIZE	USER EOF ROUTINE ADDRESS

UFCB 7	UFCB 8	UFCB 9	UFCB 10	UFCB 11	UFCB 12
25 30	31 33	34 37	38 41	42 45	46 n*
FILE ACCESS PASSWORD	ACTION FLAGS	SECONDARY ALLOCATION SPECS	PRIMARY ALLOCATION SPECS	KEY ARGUMENT ADDRESS	USED BY OPEN

* 93 FOR LINKED SEQUENTIAL FILES, 109 FOR INDEXED LINKED SEQUENTIAL FILES.

Figure 2-6 USER FILE CONTROL BLOCK(UFCB)

FILE CONTROL BLOCKS

User FCB:

The user file control block is as described in the DMF Manual, with the following modifications (refer to Figure 2-6).

1. The FCB must not be in common core. (Same as before)

2. Work Area Address - Field 4, Positions 13 to 16. User's Work Area must be in partition.

3. Logical Record Size - Field 5, Positions 17 to 20. The maximum record size is 9,999 (but full record must fit in partition).

4. User EOF Routine Address - Field 6, Positions 21 to 24. Must not be in common core.

5. Reserved for Post Open FCB - Field 12, Positions 46 to n. If the file is not indexed, this field must be 52 characters (including field 11). If the file is indexed, this field must be 64 characters in length to allow for expansion by the _OPEN operation.

Post Open FCB:

This is as described in the DMF Manual, with the following modification (refer to Figure 2-7).

1. Save Start of Record (_FCBSF) contains the start address of a multi-sector record.

2. Record Protect Switch (_FCBGU) set ON when a file is using record protect.

3. Open flag (_FCBOP) is set to '!' after the file is opened.

_FCBTY	_FCBPL	_FCBFL	_FCBWA*	_FCBLR*	_FCBEF*
0 0	1 6	7 12	13 16	17 20	21 24
FCB TYPE	POOL LABEL ADDRESS	FILE LABEL ADDRESS	WORK AREA ADDRESS	LOGICAL RECORD SIZE	USER EOF ROUTINE ADDRESS

_FCBST	_FCBCT	_FCBCN	_FCBSC*	_FCBFS	_FCBCS
25 25	26 29	30 33	34 37	38 43	44 49
USER STATUS CODE	CONTENTION CONTROL	CONTROL FLAGS	SECONDARY SECTOR ALLOCATION	PRIOR SECTOR ADDRESS	CURRENT SECTOR ADDRESS

_FCBNS	_FCBAL	_FCBEL	_FCBEX	_FCBSF	_FCBGU
50 55	56 61	62 67	68 73	74 91	92 92
NEXT SECTOR ADDRESS	SECTORS ALLOCATED	EOF MARK DISC ADDRESS	POINTER TO EXTENSION PORTION	PRIOR SECTOR, CURRENT SECTOR, NEXT SECTOR OF START OF MULTI-SECTOR RECORD	RECORD PROTECT SWITCH

_FCBØP	_FCBAR*	_FCBDP	_FCBKL	_FCBDR
93 93	94 97	98 101	102 103	104 109
OPEN FLAG	(ACTUAL) KEY ARGUMENT ADDRESS	KEY DISPLACEMENT	KEY LENGTH	POINTER TO INDEX ROOT

*FIELDS THAT CAN BE MODIFIED BY USER AFTER FILE HAS BEEN OPENED.

Figure 2-7 POST OPEN FILE CONTROL BLOCK (POFCB)

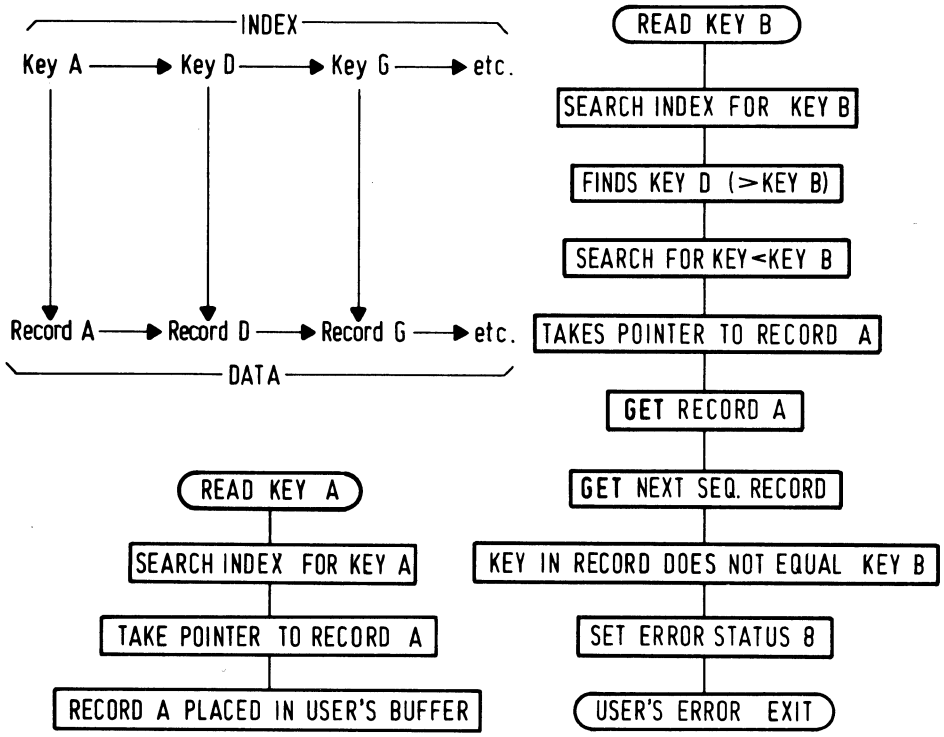

Figure 2-8 DATA RETRIEVAL FROM AN INDEXED FILE

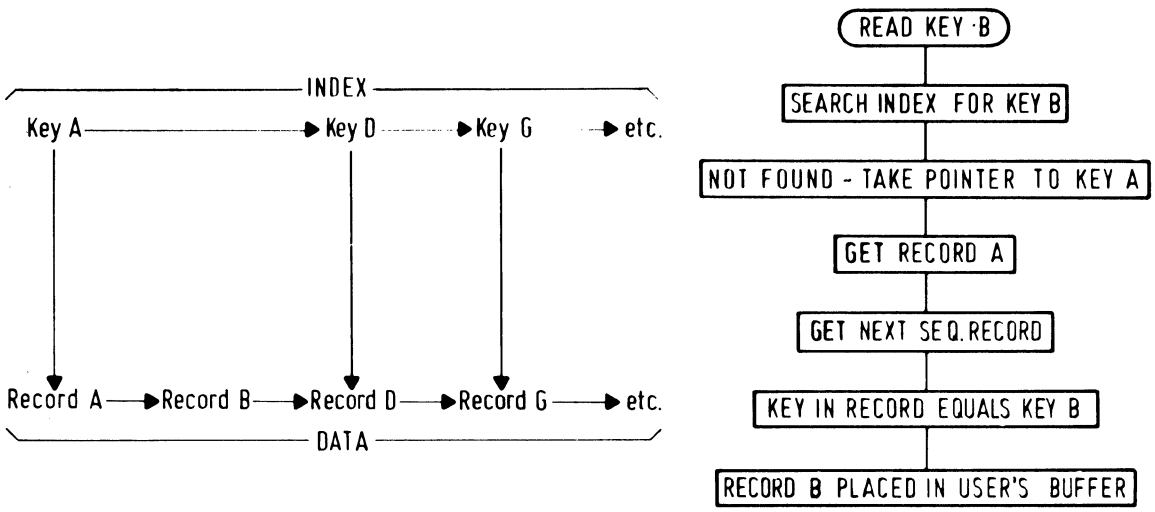

Figure 2-9 DATA RETRIEVAL AFTER RECORD INSERTION IN AN INDEXED FILE

SUBROUTINES

All subroutines not listed below operate as described in the DMF Reference Manual provided "multi-sector record" is substituted for "sector" in all the discussions.

Open A File

Main Subroutine Name: *_OPEN*

Function:

The OPEN subroutine calls the system program OPENC, which converts the UFCB into the POFCB.

Description of Operation:

_OPEN functions in the same way as the DMF OPEN subroutine, though it uses a transient named OPENC which returns to the user's program via LIOCS-C.

Close A File

Main Subroutine Name: *_CLOSE*

Function:

To finalize the state of the files created by the user's program.

Description of Operation:

_CLOSE functions in the same way as the DMF CLOSE subroutine, though it uses a transient named CLOSEC which causes returns to the user's program via LIOCS-C.

Read A Record

Main Subroutine Name: _READ

Function:

To access an indexed linked sequential data file in a random manner and to read a desired record, or its logical successor, into the user's buffer.

Description of Operation:

_READ enables the location and retrieval of data in a file, whether or not there is a one-for-one entry in the file index. If the required data is not found, then the record having the next higher key to that requested is placed in the user's buffer.

Figure 2-8 illustrates an indexed file and two attempts to read from that file, one of which is successful while the other is not, since key/record B does not exist. After the resulting error exit the user's FCB points to record D. The user can now employ _INSRT to link record B between records A and D. Figure 2-9 shows the result of this insert, and the way in which the inserted record is retrieved.

A further example of the use of _READ is given in Figure 2-10. Here, the index contains one entry for every four keys, and the consequent search and retrieval sequence is detailed in the flow diagram.

For the data retrieval logic to operate successfully, the data file must be in ascending key sequence. But, if the user has more than one index per file it is possible that the data file is out of key sequence for one of the file indexes. If a data file is not in key sequence, therefore, there must be a one-for-one index, and in this case, the function _INSRT cannot be used.

Conditions:

1. A key of high value (e.g. underscores) should be placed at the end of a file to prevent any attempted _INSRT off the end of the file.

2. Since the search logic reads each record into the user's buffer, any record to be inserted must only be placed in the buffer immediately prior to the _INSRT taking effect.

3. If indexed access is used on an extend file, the function EOF must be used before _CLOSE.

4. When setting up the index for a file having multi-sector records, the NENTRIES parameter must contain a multiple of the number of sectors per record.

5. If a record is not found in an indexed file and its key is not lower than the lowest key in the index, error status 8 is set in _FCBST and the User's Error Exit is taken.

6. If a record is not found in an indexed file and its key is lower than the lowest key in the index, error status 9 is set in _FCBST and the User's Error Exit is taken.

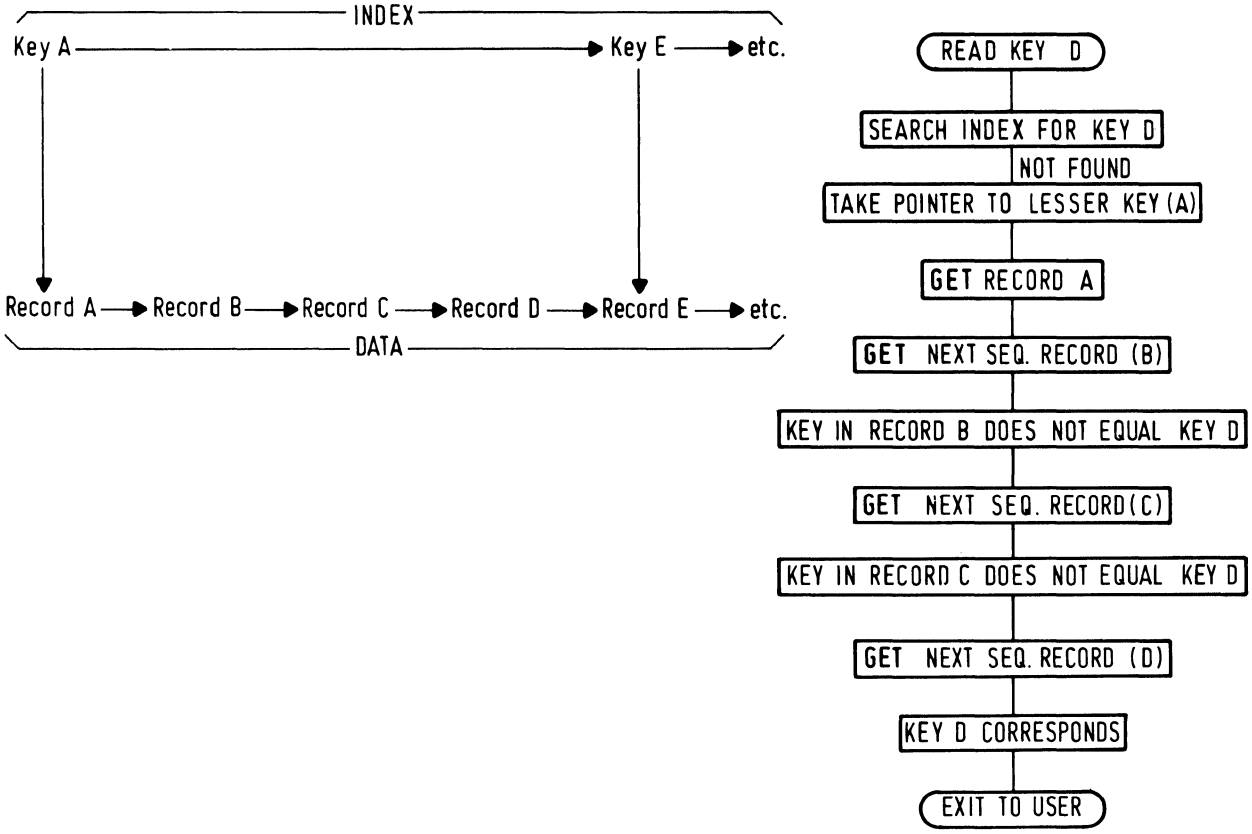

Figure 2-10 DATA RETRIEVAL FROM A MULTI-SECTOR RECORD INDEXED FILE

Read And Lock A Record

Main Subroutine Name: _READU

Function:

To access an indexed linked sequential data file in a random manner, lock the desired record and read it into the user's buffer.

Description of Operation:

_READU operates in a manner similar to _READ. _READU also inspects the Record Protect Table (see Record Protection), locks each record in turn, inspects it and unlocks it if it is not the required record. If the required record is not encountered in the file, its logical successor is locked and read into the user's buffer. _READU honors record locks in its search for a record. If a record it is attempting to inspect is locked, it will switch out until that record is unlocked.

A record locked by _READU will be unlocked by any subsequent LIOCS-C operation from the same partition on the same FCB. Only one record per FCB may be locked at a time.

Write A Record

Main Subroutine Name: _WRITE

Function:

To write the user's work area into an indexed linked sequential file, provided its key exists in the relevant index file.

Description of Operation:

Upon issuing a _WRITE, the index is searched for the desired key. If the key is found in the index, the user's work area is written into the sectors pointed to by that index entry. No check of the subject data record is made to ascertain it does indeed contain the desired key. If inserts have been made into the file, erroneous results can be obtained using _WRITE.

If the desired key is not found in the index, the user's error exit will be taken with the status code set to 8 or 9. FCB pointers are set so a subsequent _GET will obtain the record pointed to by the index entry with the next lower key. It is then the user's responsibility to issue _GET's until the desired record is found and then issue an _UPDTE or _INSRT. _WRITE does not provide any record protect facility.

Conditions:

1. _WRITE can only be used if no inserts have been made in the file since the last MAINT/INDEX run.

2. The only efficient use of _WRITE is with an index built with a density of one index entry per data record.

3. If both conditions above are not met, _READ/_READU followed by _UPDTE or _INSRT should be used.

Get and Lock A Record

Main Subroutine Name: _GETUP

Function:

To lock the next logically sequential record and to read its contents into the user's buffer.

Description of Operation:

The operation of _GETUP is similar to that of the main subroutine _GET in partition LIOCS, with the exception that _GETUP locks any record it retrieves. If the record it is attempting to read has been previously locked by _GETUP or _READU, the partition will switch out until the record is unlocked. A record locked by _GETUP will be unlocked by any subsequent LIOCS-C operation from the same partition on the same FCB. Only one record per FCB may be locked at a time.

Update A Record

Main Subroutine Name: _UPDTE

Function:

To write the contents of the user's buffer into the current record.

Description of Operation:

_UPDTE behaves in a similar manner to the existing DMF main subroutine, with the exception that if it refers to a locked record after a _GETUP or _READU operation, it will unlock that record after it has been completely updated.

Insert A Record

Main Subroutine Name: _INSRT

Function:

To write from the user's buffer into a record, which is then linked between the prior and current records.

Description of Operation:

_INSRT operates in the manner described in the DMF Manual, with the addition that all partitions have the possibility to insert into the same file.

Multi-partition Insert into the Same File:

Primary and secondary allocation must be specified in the FCB by the user; a primary allocation of zero, for an extend file, is acceptable. There is no conflict during _OPEN, _CLOSE and allocation, since the pool directory is automatically locked for all allocating of sectors to private free sector lists. When an _INSRT is made, the following sequence takes place. The last sector of the previous record is reread, and the link is compared with the sector address of the first sector in the current record: non-equality indicates that another partition has simultaneously made an _INSRT at this point. Consequently, the FCB is set up so issuing a _GET will retrieve the record just inserted by the other partition, and the user's error exit is taken with error status V. Figure 2-11 illustrates an example of two partitions (0 and 1) attempting simultaneous inserts between records 1 and 2. Partition 0 succeeded in inserting record 1A, while partition 1 took an error exit and its insert failed. A _GET was then taken by partition 1, and record 1A was obtained; and since partition 1 did not want an insert between records 1 and 1A, another _GET was taken and record 2 was obtained. _INSRT B then inserts record 1B in the desired location between records 1A and 2.

Note that partition 1 obtained record 2 twice, owing to the alteration of the sequence of the records. This factor must be taken into account by the user in the preparation of his program. Note also that after an error exit with status V, the only valid operation is _GET, after which all I/O operations are valid. However, _GET will overwrite the data in the user's buffer; the user must be able either to re-create his data in the buffer or to _GET into another buffer.

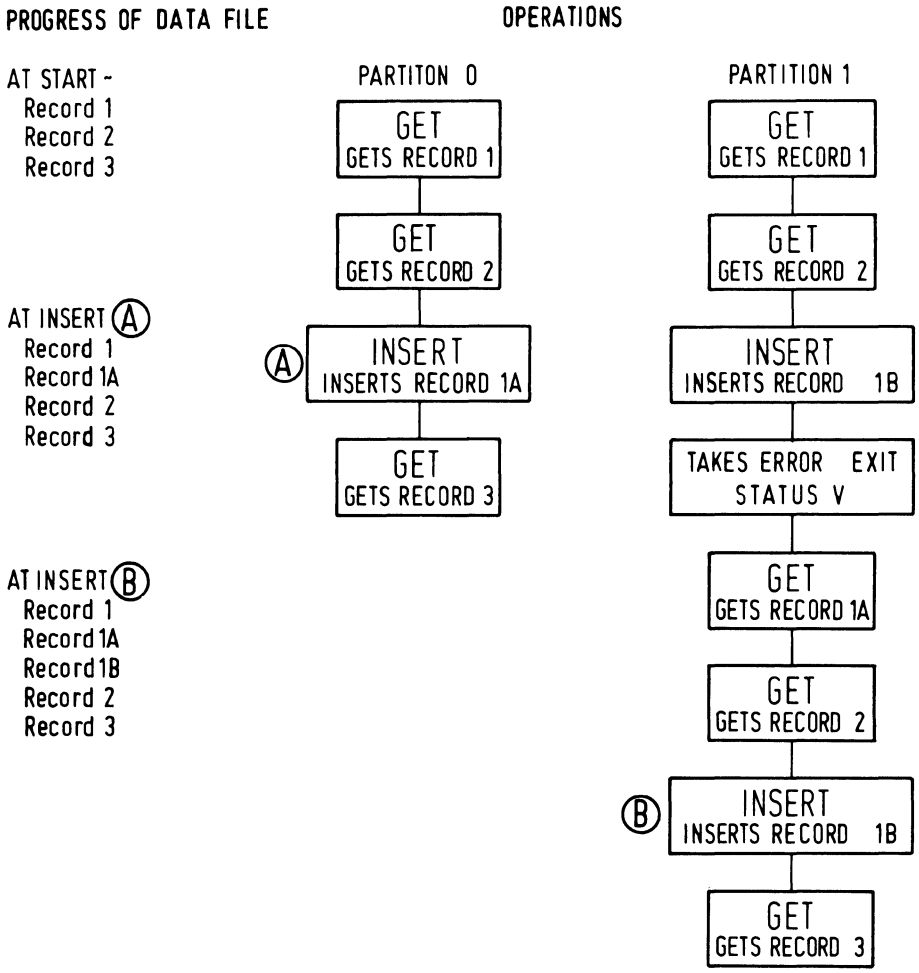

Figure 2-11 MULTI-PARTITION INSERT

Find End of File

Main Subroutine Name: _EOF

Function:

To position the file at the record that contains either a temporary or an absolute end-of-file mark.

Description of Operation:

_EOF operates in the same way as its DMF counterpart, with the addition that it will set up the user's FCB so that for subsequent operations:

_GET will take the user's EOF exit.

_UPDTE will update the last record in the file.

_INSRT will insert ahead of the last record in the file.

_DLETE will delete the last record in the file.

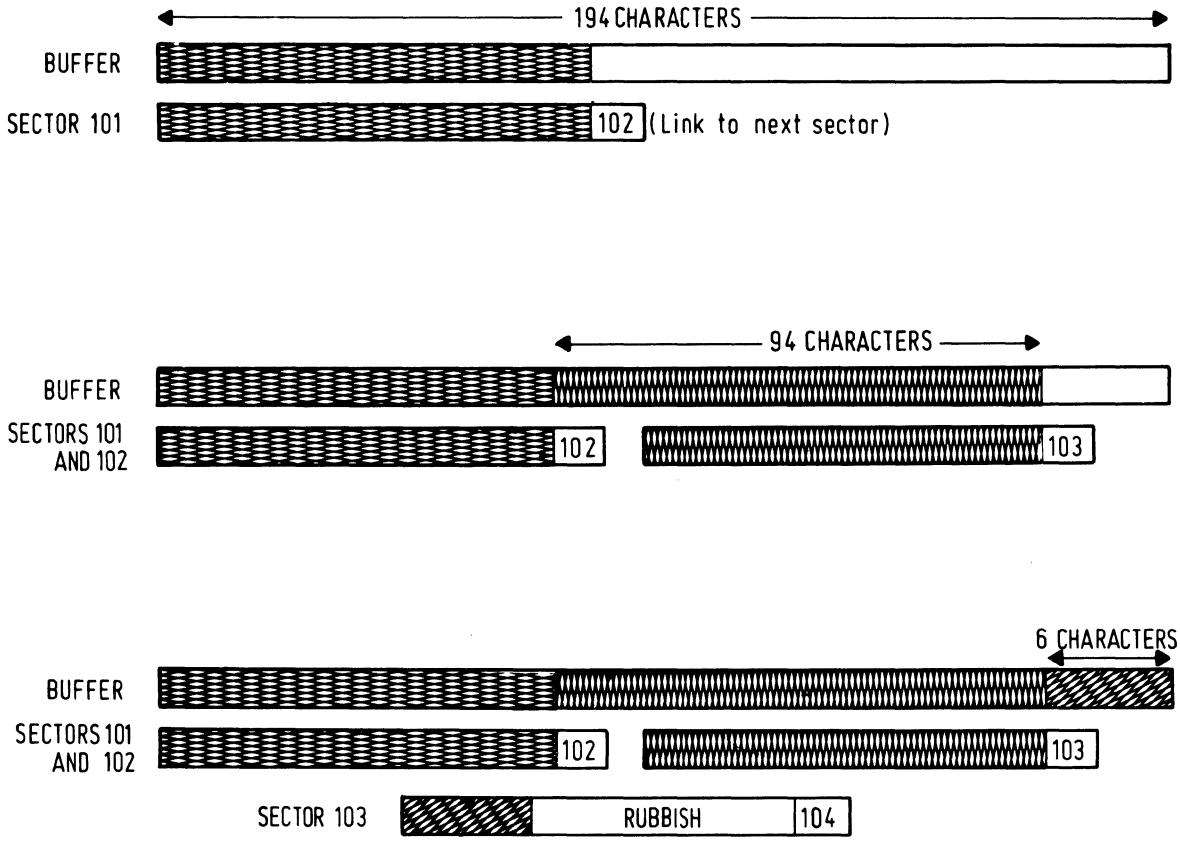

Figure 2-12. PUTTING A MULTI-SECTOR RECORD TO DISC

CONSIDERATIONS

Multi-sector Records:

The maximum permissible record size is limited only by the amount of partition core the user has available. If the user specifies a block size of, say, 194 characters and then initiates a _PUT, the following sequence occurs (refer to Figure 2-12).

1. LIOCS-C writes the first 94 characters directly from the user's buffer into the first sector.

2. This is followed by the next 94 characters being written into the next sector.

3. Finally, the remaining six characters are written into the next sector.

No exit is made to the user until the contents of the entire buffer is written. In response to the next _PUT the first part of the record will be written in sector 104, the remaining space in sector 103 being left unused.

_GET, _READ, _INSRT, _DLETE and _UPDTE operate in a similar manner to _PUT, though _GET and _READ do not transfer any of the 'rubbish' from the last sector into the user's buffer.

Record Access Time Considerations

LIOCS-C has the ability to read all sectors of a multi-sector record "on the fly" (i.e. on the same revolution of the disc) provided the sectors reside in consecutive disc addresses and the record is an integral multiple of 94 characters in length. All sectors of a record are read directly into the user's buffer unless the record does not end on a sector boundary. Then the last sector must first be read into the system buffer (200-299 in partition) and a portion of it moved to the user's buffer. This eliminates overlaying the area immediately after the user's buffer when reading this last "short" sector. This added processing makes it impossible to read the first sector of the next record on the same revolution. Since LIOCS-C automatically scans through the file for the proper key on a _READ, this is an important consideration, especially when a file has a density of many records per index entry.

It is recommended that record length be specified as a multiple of 94 characters. It costs no disc space; each record must begin on a sector boundary anyway. The potential time savings are worth the cost of a few core positions to pad out the user's buffers to integral multiples of 94 characters.

Record Protection:

LIOCS-C provides a facility to prevent the simultaneous update of the same record by more than one partition. This is afforded by _GETUP (_GET for update), _READU (_READ for update), and a ten entry Record Protect Table in Common. When a partition issues a _GETUP or _READU, the partition number and the address of the first sector of the record obtained are placed in the Record Protect Table. The record is unlocked when the locking partition performs any subsequent LIOCS-C operation on the file in question (normally an _UPDTE). While a record is locked, any reference to it by _GETUP or _READU from another partition will cause that partition to switch out until the record is unlocked.

Locked records are only protected from access by _GETUP or _READU; all other LIOCS-C operations by-pass the record protect logic. For instance, a record locked by a _READU from partition 1 may be accessed simultaneously by a _READ from partition 3. LIOCS-C does not restrict partition 3 from now issuing an _UPDTE on the record it has just obtained via a _READ, which would probably overlay any updating done on the record by partition 1. This feature allows read-only accessing of files being updated by _READU and _GETUP. If two or more partitions have the possiblity of updating the same record simultaneously, it is the programmer's responsibility to utilize _GETUP and _READU so LIOCS-C can coordinate the use of records by all partitions.

There is no limit to the number of FCB's that may have records locked, but the Record Protect Table can only accommodate ten entries at a time. Should it become filled, any partition attempting to lock a record will switch out until an entry in the table is freed. Filling the table is an unlikely event. Still, good system design dictates that any partition lock a record for only as long as absolutely necessary to perform an update. This is primarily a consideration to increase system response when many partitions are referencing the same records but it will also minimize competition for table entries.

If a partition should go to a load condition, the address of any records it has locked will remain in the table. Whenever the conversational loader (C_LOAD) is used to load a program, it purges all entries in the Record Protect Table for all partitions in a load and for the partition in which it is running.

Drive Lock:

A drive lock routine is included to prevent any partition from accessing a drive already accessed by another partition, and to maintain the lock on the drive until the controlling partition has completed I/O operation, i.e. until every sector in a multi-sector record has been read or written. This facility reduces I/O times by minimizing head movement. The drive lock routine references a ten-character field; known as the Drive Lock Table, which is situated in locations 510C to 519C.

The left-most position (510C) refers to drive 0, the next to drive 1, and so on, the right-most position (519C) relating to drive 9. The state of each position indicates whether or not the relevant drive is locked; if the position contains a blank, the drive is free; if it contains the partition number the drive is locked. (In Figure 2-13, drives 0, 2 and 3 are locked by partitions 0, 4 and 2 respectively. A single-character partition number is used: 10, 11, 12 etc. are represented by P, Q, R etc., respectively.)

Any attempt by a program to use a locked drive will cause control to be switched to the next partition. However, the partition is not informed that this event has occurred, and it attempts to access the locked drive each time it has control.

If the controlling partition assumes a load condition before the I/O operation is completed, it is possible that the drive will remain locked. The conversational loader clears from the table all locks if the associated partition is in a load condition.

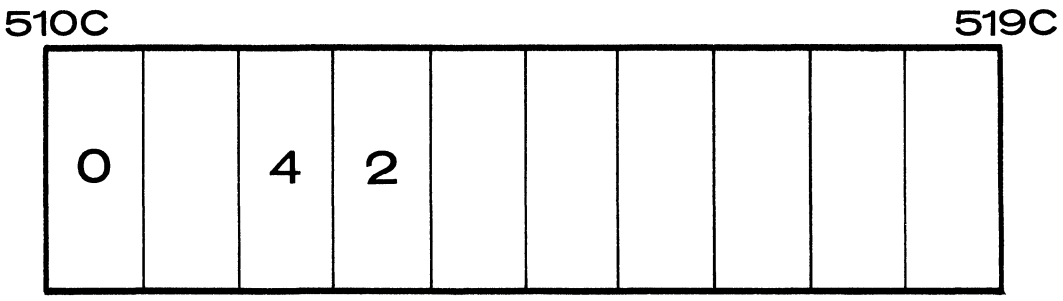

Figure 2-13 DRIVE LOCK TABLE

Error Exit:

During an error exit, location 990 in partition contains the instruction

```
BC ERROR(5),NORMAL(5)
```

so that the instruction

```
MN _USERX+6(4),...
```

can be used to obtain the address of the calling sequence.

NORMAL is the normal return address.

N.B. A move numeric instruction must be used.

Physical I/O Errors:

If a physical I/O error (status code 1) occurs within a file, it is not advisable to continue using that file after an error exit; the file should be closed, and both the file and pool free sector list should be checked for correct link addresses.

Drive Not Ready:

If a drive is not ready, the message

```
A)READY DEVICE Dn.
```

will be displayed on the CONO device. If the user responds via the CONI device by setting a FLAG condition (pressing any control key on the workstation or video display), the user's error exit is taken with status code 1 set. Any other response via the CONI device will cause the I/O operation to be retried.

If either the CONO or CONI device is unassigned (assigned as NODEV), the user's error exit will be taken with a status code X set. The only legitimate option the user may then exercise in his error routine is to branch to _RETRY within LIOCS-C with a particular condition code set. At _RETRY is a branch instruction which:

1. For a condition code of 2, will retry the I/O operation.
2. For a condition code other than 2, will take the user's error exit with condition code 1 set.

This allows I/O error retries from "blind" partitions under program control.

Error Status Code:

Table 4-7 of the DMF Manual is replaced by Table A-1.

Sector Allocation and Contention Problems:

As described in the DMF Manual, with two additions; multi-partition inserts (which have already been dealt with under the heading _INSRT) and multi-partition deletes.

Multi-partition Delete in the Same File:

Multi-partition deletes in the same file are not permitted. If one partition references a file in any way, and a second partition deletes a record that the first partition is about to access, the results are unpredictable. If it is required to delete a record in these circumstances, the relevant record must be obtained using Record Protect, and be _UPDTE'd with characters that will indicate to all programs referencing that file, that the record has been deleted. The record may be deleted at some later time by a user-written program operating in a non-contention environment.

Multi-partition Extend in the Same File:

LIOCS-C does not provide any additional capabilities for physically extending (i.e. _PUT beyond EOF marker) an extend type file than Partition LIOCS provided. A physical extension is still linked onto the file at Close. If two physical extensions are made to the same file simultaneously, when the second one is closed it will overlay the link to the extension of the first. If physical extension is to be used, one FCB must be passed back and forth through Common to each contending partition as was necessary in Partition LIOCS. Common flags must be used to make sure only one partition uses the common FCB at a time.

Logical extension from many partitions may be accomplished under LIOCS-C contention control by using _INSRT in front of a trailer record.

Overlay Routine:

LIOCS-C contains an overlay routine to load modules stored in SYSPOL. The user places the name of the program to be loaded in partition locations 25-30 (_LNAM) and branches to _OVLAY in the LIOCS-C transfer vector.

If the user wishes to execute the overlay after loading (begin executing at the address defined in the EXEC card), the calling sequence

```
    B       _OVLAY
```

is used.

If the user wishes to load the overlay and return to the calling program, the calling sequence

```
LINK    _LXR3,_OVLAY
```

is used and the overlay module must be terminated with an EXEC card image specifying the address 1300C generated by

```
EXEC    _OVRET
```

in the overlay module's source deck (_OVRET is defined by including the macro CLIOIN in the source deck). An overlay loaded in this manner must not modify index register 3, otherwise the user's return address will be lost.

In neither case is it necessary for the user to load the Locator or set its search arguments in low core. All overlays loaded via _OVLAY must be previously filed in SYSPOL. If an overlay is not found, the message

```
L)progname NOT FOUND.
```

is displayed on CONO, if it is defined. Whether or not CONO is defined, the partition will then go to a load with the Program Check Area (_LCKR;40P to 44P) containing the address of the above message.

The overlay routine does not require P_COMM or the Locator to be loaded by the user; the Locator is loaded automatically by LIOCS-C and P_COMM is not used.

Appendix A
LIOCS-C STATUS CODES

Table A-1 STATUS CODE SETTINGS (0-9)

STATUS CODE	LIOCS MAIN SUBROUTINE	CONDITION INDICATED
0	(NONE)	(NOT USED)
1	ALL	An irrecoverable read parity or a flag error was encountered while attempting to perform linked sequential disc I/O or A write flag error has been encountered while attempting to perform linked sequential disc I/O.
2	(NONE)	(NO LONGER USED)
3	_PUT _INSRT	The pool free sector list is exhausted.
4	_READ	The user's key argument was found in the index file, but is missing from the indexed data file. The desired data record has either been deleted, improperly updated, or a record with a higher key inserted in front of it. The record in the user's work area contains the first record encountered with a key greater than the one desired. An _INSRT could now be performed if the sequence of the data file can still be assumed.
5	_PUT, _UPDTE, or _WRTEF	Invalid operation was attempted on a read-only file.
6	_PUT	An attempt was made to write beyond the absolute end-of-file mark on a fixed-allocation file.
	_INSRT, _DLETE	Invalid operations on fixed-allocation or read-only file.
7	(NONE)	(No longer used -- if a read error occurs during _READ or _WRITE, status code 1 is set.)
8		Key argument not found in the index file, but it is higher than the lowest key in the index file.
	_READ	No record containing the specified key has been found in the data file. The record in the user's work area and referenced by the FCB pointers is the first one encountered in the data file with a key greater than the one specified.
	_WRITE	FCB pointers set so _GET will retrieve record with next lower key contained in the index. User's work area is undisturbed.
9		Key argument not found in the index file. It is lower than the lowest key in the index file.
	_READ	No record containing the specified key has been found in the data file. The record in the user's work area and referenced by the FCB pointers is the first one encountered in the data file with a key greater than the one specified.
	_WRITE	FCB pointers set so _GET will retrieve record with lowest key in the index. User's work area is undisturbed.

NOTE: THE CONTENTS OF THE STATUS CODE FIELD OF THE POFCB ARE NOT INDICATIVE OF ANY CONDITION UNLESS THE PROGRAM BEING EXECUTED BRANCHES TO AN ERROR ROUTINE SPECIFIED IN A LIOCS MAIN SUBROUTINE CALLING SEQUENCE.

NOTE: ERROR CONDITIONS THAT APPLY TO _READ ALSO APPLY TO _READU; THOSE THAT APPLY TO _GET ALSO APPLY TO _GETUP.

Table A-1 STATUS CODE SETTINGS (A-G)

STATUS CODE	LIOCS MAIN SUBROUTINE	CONDITION INDICATED
A	(NONE)	(NOT USED)
B	_READ,_WRITE	_READ or _WRITE attempted to a non-indexed file.
		_READ or _WRITE attempted on an output or work file.
	_GET,_BOF _EOF,_INSRT, _DLETE,_READ, _WRITE	One of these operations was attempted in the extension portion of an extend file; they are permitted only in the original portion of the file.
		Or, one of these operations was attempted in a work or output file before a temporary end-of-file mark was written or after the temporary end-of-file mark was overwritten by a _PUT.
	_UPDTE,_DLETE _WRTEF	Current sector address is null. One of these operations was attempted immediately after a _DLETE,_BOF,_WRTEF, or _OPEN.
	_WRTEF	The operation was attempted on an extend file. It is prohibited.
		Or an attempt was made to shorten an output file which already had an existing temporary end-of-file mark.
	Any LIOCS-C Operation	An operation other than _OPEN attempted on an unopened file.
C	_OPEN	A disc error has occurred in processing a directory entry or allocating pool free sectors.
		CONO unit display will be:
		S)OPEN pppppp.ffffff:DISC I/O ERROR.
D	_OPEN	The system was unable to locate a pool specified (in UFCB Field 2).
		CONO unit display will be:
		S)OPEN pppppp.ffffff:POOL NOT FOUND.
E	_OPEN	The system was unable to locate the file specified (in UFCB Field 3) in the pool specified (in UFCB Field 2).
		CONO unit display will be:
		S)OPEN pppppp.ffffff:FILE NOT FOUND.
F	_OPEN	The user's file type does not agree with the file type of the requested file.
		CONO unit display will be:
		D)OPEN pppppp.ffffff:INVALID USER FCB TYPE.
G	_OPEN	The action flag field of the UFCB does not contain one of the five acceptable values (000,100,010,001 or W00).
		CONO unit display will be:
		S)OPEN pppppp.ffffff:INVALID ACTION FLAGS.

NOTE: THE CONTENTS OF THE STATUS CODE FIELD OF THE POFCB ARE NOT INDICATIVE OF ANY CONDITION UNLESS THE PROGRAM BEING EXECUTED BRANCHES TO AN ERROR ROUTINE SPECIFIED IN A LIOCS MAIN SUBROUTINE CALLING SEQUENCE.

NOTE: ERROR CONDITIONS THAT APPLY TO _READ ALSO APPLY TO READU; THOSE THAT APPLY TO _GET ALSO APPLY TO _GETUP.

Table A-1 STATUS CODE SETTINGS (H-N)

STATUS CODE	LIOCS MAIN SUBROUTINE	CONDITION INDICATED
H	_OPEN	The user has attempted to open a null file (one containing no data records) as a read-only, fixed-allocation or extend file; these three categories must contain data records at _OPEN time. CONO unit display will be: S)OPEN pppppp.ffffff:INVALID ACTION FOR NULL FILE.
I	_OPEN	An attempt has been made to open a file containing data records as a work or output file; these two types of file must be null at open time. CONO unit display will be: S)OPEN pppppp.ffffff:INVALID ACTION FOR NON-NULL FILE.
J	_OPEN	An attempt has been made to open an indexed file as an output or work file. CONO unit display will be: S)OPEN pppppp.ffffff:INVALID ACTION FOR INDEX FILE.
K	_OPEN	UFCB Field 4 (Work Area Address) does not contain a valid partition address. User's Work Area must be in partition. CONO unit display will be: S)OPEN pppppp.ffffff:INVALID WORK AREA ADDRESS.
L	_OPEN	UFCB Field 5 (Logical Record Size) does not contain a numeric value. (Note that a 0 size record is accepted but will default to the record size specified in the file label Field 9.) CONO unit display will be: S)OPEN pppppp.ffffff:INVALID RECORD SIZE.
M	_OPEN	UFCB Field 6 (User EOF Routine Address) does not contain a valid System Ten partition address. User error routines must not be in common. CONO unit display will be: S)OPEN pppppp.ffffff:INVALID USER EOF ADDRESS.
N	_OPEN	UFCB Field 11 (Key Argument Address) does not contain a valid partition or common address. In the UFCB source coding, this field should contain either the label or the address where the key argument will be stored during program execution. CONO unit display will be: S)OPEN pppppp.ffffff:INVALID KEY ARG ADDRESS.

NOTE: THE CONTENTS OF THE STATUS CODE FIELD OF THE POFCB ARE NOT INDICATIVE OF ANY CONDITION UNLESS THE PROGRAM BEING EXECUTED BRANCHES TO AN ERROR ROUTINE SPECIFIED IN A LIOCS MAIN SUBROUTINE CALLING SEQUENCE.

NOTE: ERROR CONDITIONS THAT APPLY TO _READ ALSO APPLY TO _READU; THOSE THAT APPLY TO _GET ALSO APPLY TO _GETUP.

Table A-1 STATUS CODE SETTINGS (O-Z)

STATUS CODE	LIOCS MAIN SUBROUTINE	CONDITION INDICATED
O	(NONE)	(NOT USED)
P	(NONE)	(NOT USED)
Q	_OPEN	UFCB Fields 9 and 10 (Secondary and Primary Allocation fields) do not contain $$$$, ////, or numeric values; or the fields are not in agreement. If contention is specified in either field, it must be specified in both. (//// = non-contention, numeric value = special allocation requested, and $$$$ = default allocation requested). CONO unit display will be: S)OPEN pppppp.ffffff:INVALID SECTOR ALLOCATION.
R	(NONE)	(NOT USED)
S	_OPEN	A primary allocation was requested but no pool free sectors were available. CONO unit display will be: S)OPEN pppppp.ffffff:NO POOL FREE SECTORS AVAILABLE
T	(NONE)	(NOT USED)
U	(NONE)	(NOT USED)
V	_INSRT	Between the time this partition obtained the pointers to the prior and current sectors and initiated an _INSRT, another partition has made an _INSRT into the file at the same place. This partition did not insert the record or alter the file. It's FCB has been set up so the next _GET will obtain the record just inserted by the other partition. User Response: issue _GET's until a key greater than the one to be inserted is encountered, then re-issue the _INSRT. Remember to _GET into a different buffer or the record to be inserted will be overlayed.
W	Any LIOCS-C Operation	A previous _READ or _WRITE encountered an error with a status code of X and the user program failed to respond by branching to _RETRY The partition is missing either a CONO or CONI device or both making communication with the operator impossible. User Response: Close all other files, abort the run, and check the integrity of the file in question.
X	_READ,_WRITE	An attempt was made to access a disc drive that is not ready. The partition is missing either the CONO or CONI device or both, making displaying the normal error message and responding to it impossible. ONLY USER PROGRAM RESPONSE: Branch to _RETRY in LIOCS-C with condition code set. CC2=retry I/O operation; any other condition code = the user's error exit will be taken with status code 1 set.
Y	_CLOSE	A disc error was encountered during execution of the system program CLOSEC. The file has not been properly closed. CONO unit display will be: S)I/O ERROR (nnnnnn):CLOSE NOT COMPLETED. (nnnnnn) specifies the physical disc address of the sector on which the error was encountered.
Z	_CLOSE	A disc I/O error was encountered during execution of the system program CLOSEC. The file may be closed but unused sectors were not returned to the pool free sector list. CONO unit display will be: S)I/O ERROR (nnnnnn):SECTORS LOST DURING CLOSE. (nnnnnn) specifies the physical disc address of the sector on which the error was encountered.

NOTE: THE CONTENTS OF THE STATUS CODE FIELD OF THE POFCB ARE NOT INDICATIVE OF ANY CONDITION UNLESS THE PROGRAM BEING EXECUTED BRANCHES TO AN ERROR ROUTINE SPECIFIED IN A LIOCS MAIN SUBROUTINE CALLING SEQUENCE.

NOTE: ERROR CONDITIONS THAT APPLY TO _READ ALSO APPLY TO _READU; THOSE THAT APPLY TO _GET ALSO APPLY TO _GETUP.

Appendix B
FCB MACRO PARAMETERS

Table B-1 FCB MACRO PARAMETERS

FCB MACRO FORMAT: Label⌀FCB⌀NAME=p-name.f-name,USE=action, AREA=wkarea,
EXIT=eofaddr,BLKL=blklen,PASSW='password',
KEYAD=keyaddr,ALLOC=alspec,TYPE=fcbtype

EXAMPLE: FCB NAME=PNAME.FNAME1,USE=EXTEND,AREA=WKAREA,EXIT=EOF,BLKL=470,ALLOC=(0,10),
TYPE=LS

PARAMETER	EXPLANATION	DEFAULT
Label	Label for beginning of FCB --Optional	Optional
FCB	Macro name	Required
NAME=p-name.f-name	Names of pool and file to be accessed	Required -- no default pool.
USE=action	Sets Action Flags. action specified — file type INPUT — read-only file ⌀UTPUT — output file EXTEND — extend file FIXED — fixed-allocation file WORK — work file	Defaults to INPUT if not specified
AREA=wkarea	wkarea is the address (an expression) of the User's Work Area	Required
EXIT=eofaddr	eofaddr is the address (an expression) of the User's End-of-File Routine	Required
BLKL=blklen	Logical Block Length (an expression)	Defaults to zero if not specified (Logical Block Length to be picked up from file label)
PASSW='password'	File Access Password; password is 1-6 characters enclosed in apostrophes	Defaults to blanks if not specified
KEYAD=keyaddr	keyaddr is the address of the Key Argument Field (an expression).	Required for Indexed Linked Sequential Files (TYPE=ILS)
ALLOC=alspec	alspec is the Allocation Specification alspec: — meaning: YES — System default values (field set to '$$$$') N⌀ — No allocation-- Input or FIXED file (field set to '////') (primary, secondary) Explicit specification of primary and secondary allocation; both are expressions	Defaults to YES if not specified.
TYPE=fcbtype	FCB type specification: LS-- for Linked Sequential FCB (will generate a 94 character FCB) ILS-- for Indexed Linked Sequential FCB (will generate a 110 character FCB)	Required

Drive Lock

A LIOCS-C facility that minimizes disc seek time by locking a disc drive to a partition until it has completely read or written a record. The ten-entry Drive Lock Table begins in location 510 in common.

Record Lock

LIOCS-C functions _GETUP and _READU lock a record so it is protected from being accessed by _GETUP or _READU from another partition as long as it is locked. Any subsequent LIOCS-C operation from the same partition on the same FCB will automatically unlock the record. The ten-entry Record Protect Table begins in location 410 in common.

System Lock

The system lock makes sure only one partition at a time modifies the Pool Directory.

Transfer Vector

A transfer vector is a set of branch instructions in a fixed location which are used as entry points to routines whose locations may vary.

DMF COMMON—RESIDENT LIOCS (LIOCS-C)
REFERENCE MANUAL
Publication No. 40-340

We produce manuals for you, and we want you to find them useful and informative. That's our job.

So we're asking you to help us furnish you with the best possible publications. Please take a few minutes to answer the following questions. Add any comments you wish. If you desire a reply to any question, be sure to include your name and address.

Thank you.

DETACH HERE

- Does this manual meet your needs? Yes ☐ No ☐
  If not, what additional information would be of help to you?

- Can you find what you're looking for quickly and easily? Yes ☐ No ☐
  How can the organization be improved?

- Is the material easy to read and to understand? Yes ☐ No ☐
  Are there enough illustrations to support the text? Yes ☐ No ☐
  Comments

- Did you find any errors or ambiguities in the manual? Yes ☐ No ☐
  If yes, please cite page, line, and/or figure number with your comments.

- Other comments.

- What is your relationship to the product described?
  - ☐ Operator.
  - ☐ Programmer.
  - ☐ Other (please specify)

SINGER
BUSINESS MACHINES

STAPLE

STAPLE

FOLD BACK

FIRST CLASS
PERMIT No. 320
San Leandro, Calif.

BUSINESS REPLY MAIL
No postage stamp necessary if mailed in the United States

POSTAGE WILL BE PAID BY

FRIDEN DIVISION
THE SINGER COMPANY
2350 Washington Ave.
San Leandro, California 94577

Attn: Customer Technical Publications,
Department 753

FOLD BACK

SINGER
BUSINESS MACHINES
PUBLICATION NO. 40-340
CONTROL NO. B554PA