Licensed Material — Property of IBM
LY24-5192-1
File No. S370-30

Program Product

VSE/VSAM
VSAM Logic, Volume 2:
Record Management

Program Number 5746-AM2

Component 5745-SC-VSM

Release 2

Second Edition (December 1979)

This edition, LY24-5192-1, applies to Release 2 of Virtual Storage Extended/Virtual Storage Access Method (VSE/VSAM), Program Product 5746-AM2, and to subsequent releases and modifications until otherwise indicated in new editions or Technical Newsletters. Changes are periodically made to the information contained herein; before using this publication in connection with the operation of IBM systems, consult the *IBM System/370 and 4300 Processors Bibliography,* GC20-0001, for the editions that are applicable and current.

Summary of Amendments

For a list of changes, see page iii.

Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Publications are not stocked at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Programming Publications, Department G60, P.O. Box 6, Endicott, New York, U.S.A. 13760. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Summary of Amendments for VSE/VSAM VSAM Logic, Volume 2

Summary of Amendments for LY24-5192-1 Release 2

This major revision contains information about the following items for this release:

- CA split integrity: If a control area split in a KSDS is interrupted by a system failure, the file can have duplicate records that were copied into it. The control area split integrity enhancement detects that the duplicate data exists and erases the original versions of the copied records.
- Share option 4 improvements: The number of I/O's required in keyed processing of a KSDS is reduced by locking a sequence set record rather than a data control area. Also, there is a time limit that prevents sequential processing from locking out direct requests for an excessively long time.
- Space Management for SAM Feature: This manual documents hooks in Record Management for the Space Management for SAM Feature. For information about the internal logic of that feature, refer to *VSE/VSAM Space Management for SAM Feature Logic,* LY24-5204.

Two new control blocks have been added to facilitate processing of share option 4 files:

- The File Sharing Work Area (SHRW) is used as a work area in processing of SHAREOPTIONS(4) files.
- The Hold Block (THB) replaces the Track Hold Block (THB) control block and contains information necessary to lock a control area of a SHAREOPTIONS(4) file during updates and inserts.

Some of the existing control blocks have had fields changed, deleted, and added to support the new items for VSE/VSAM Release 2.

Two new modules (IKQLNA and IKQIOD) and four new macros (IKQSHRW, LOCK, MODDTL, and UNLOCK) are now used by Record Management for processing of files.

Various editorial changes are also included to improve the usefulness of this manual.

Preface

This logic manual is one of three volumes providing detailed information about VSE/VSAM. The three volumes are:

VSE/VSAM VSAM Logic, Volume 1: Catalog Management, Open/Close, DADSM, IIP, Control Block Manipulation, LY24-5191

VSE/VSAM VSAM Logic, Volume 2: Record Management, LY24-5192

VSE/VSAM Access Method Services Logic, LY24-5195

This volume contains all VSAM record management, I/O management, buffer management and EOV logic documentation.

This manual is mainly intended for persons involved in program maintenance and for system programmers who are altering the program design. Logic information is not necessary for the operation of the programs described.

This manual and the code it supports should be viewed as a maintenance set. This means that the module prologues and comments contain certain types of information and that this manual contains other kinds of information. Thus, the listings provide the description of the internal logic of modules, and the manual uses Method of Operations diagrams to show what the functions of VSAM are and how the modules work together to carry out those functions. The term *data set* is used in this manual instead of *file* to conform to the program listings.

Effective use of this publication requires an understanding of system operation, PL/S language, assembler language, and its associated macros.

Organization of This Publication

This publication is organized in the following manner:

- *Section 1. Introduction* describes the major components of VSAM.
- *Section 2. Method of Operation* contains HIPO diagrams describing record management, buffer management, I/O management, and EOV.
- *Section 3. Program Organization* describes the information contained in VSAM program listings and the relationship of the program structures to the issued macro.
- *Section 4. Directory* contains lists of phases, components, modules, routines, catalog external entry points, and data areas.
- *Section 5. Data Areas* describes control blocks used by VSAM record management, I/O management, and buffer management.
- *Section 6. Diagnostic Aids* contains diagnostic aids, such as error codes.
- *Glossary* defines terms relevant to VSAM.
- *Index* is a subject index to the publication.

Required Publications

The following publications should be read and understood before using this publication:

VSE/VSAM General Information, GC24-5143, explains basic VSAM concepts and facilities and how to use them.

Using VSE/VSAM Commands and Macros, SC24-5144, tells how to code VSAM macros in application programs and describes VSAM data management. Access Method Services commands and their use are also described.

Related Publications

Other publications that may be of interest in conjunction with this manual are:

VSE/VSAM Programmer's Reference, SC24-5145, describes installation and operating procedures, sysgen information, storage estimates, debugging techniques, performance tips, and recovery procedures.

VSE/VSAM VSAM Logic, Volume 1, LY24-5191, describes the logic of VSAM catalog management, open/close, DADSM, ISAM Interface Program, and control block manipulation.

VSE/VSAM Access Method Services Logic, LY24-5195 documents the logic of Access Method Services.

VSE/VSAM Space Management for SAM Feature Logic, LY24-5204, describes the interfaces between Record Management and that feature.

VSE/VSAM Documentation Subset, SC24-5191, contains a subset of the information contained in Using VSE/VSAM Commands and Macros.

VSE/VSAM Messages and Codes, SC24-5146, includes all messages and codes originated by VSAM and Access Method Services.

Using This Publication

This publication is designed to be used with the VSAM program listings. The diagrams in *Method of Operation* describe the major functions performed by VSAM; these diagrams are intended to be your key to a module name (and routine name, as appropriate) in the listing. See the *Method of Operation* chapter for a description of how to read these diagrams. For information on what is available in the program listings, see the chapter *Program Organization.*

The module directory in the *Directory* chapter lists the modules by symbolic name (all of which start with IKQ, IIP, IGG0, $S, or $$B) and contains page references to the appropriate method of operation diagram or program structure that applies to each module. If you wish to see how modules are grouped according to component, see the component directory. The routine directory, where relevant, further shows how the modules are subdivided into routines.

The *Index* to this volume contains a list of all VSAM modules and indicates whether each is documented in Volume 1, Volume 2, or both.

Contents

Illustrations

Figures

Method of Operation Diagrams

Acronyms and Abbreviations

AC	allocation chain
ACB	access method control block
ACC	AMS Catalog communication area
ACE	argument control entry
ADDR	address
ADR	addressed accessing
AIX	alternate index
AMBL	access method block list
AMCBS	access method control block structure
AMDSB	access method data statistics block
AMDTF	access method define the file (ISAM only)
ANCHT	anchor table
ARDB	address range definition block
AU	allocation unit
BCB	buffer control block
BCR	base cluster record
BHD	buffer header
BKPHD	block pool header
BKWD (also BWD)	backward
BLK	block
BSPH	buffer subpool header
BUFF	buffer
BUFH	buffer header
CA	control area
CAT	catalog
CAXWA	catalog auxiliary work area
CB	control block
CCA	catalog communications area
CCB	command control block
CCR	catalog control record
CI or CNV	control interval
CIDF	control interval definition field
CINV	control interval
CIWA (also CIW)	control interval work area
CKD	count-key-data
CM	catalog management
CMS	catalog management services
CNV or CI	control interval
COMREG	communications region
CP	channel program
CPA	channel program area
CPAH	channel program area header
CPL	catalog parameter list
CRA	catalog recovery area
CTGFL	catalog field parameter list
CTGFV	catalog field vector table
CTGPL	catalog parameter list
CVH	common VTOC handler
DADSM	direct-access device space management
DDname	data definition name
DIR	direct processing
DLBL	DASD label
DS	data set
DSA	dynamic storage area
DSCB	data set control block (in VSE, VTOC label)
DSN (also DSNAME)	data set name
DSORG	data set organization
DTF	define the file
ECB	event control block
EDB	extent definition block
EOD	end of data
EOF	end of file
EOV	end of volume
ESDS	entry-sequenced data set
EXCP	execute channel program
EXLST	exit list
FBA	fixed block architecture
FCDB	field control and data block
FKS	full key search
Fn	format n
FPL (also FL)	field parameter list
FS	free space
FVT	field vector table
FWD	forward
FXL	fix list
GEN	generic key search
GO	group occurrence
GOP	group occurrence pointer
ID	identifier
IIP	ISAM interface program
I/O	input/output
ISAM	indexed sequential access method
JIB	job information block
KEQ	search on key equal
KEY	keyed accessing
KGE	search on key greater or equal
KRDR	key range determination routine
KSDS	key-sequenced data set
KWTC	keyword type code
LOC	locate
LPMB	logical-to-physical mapping block
LRD	last record
LUB	logical unit block
MVE	move
n	number
NSP	note string position
NUB	no user buffer
NUP	no update
OAL	open ACB list
O/C/EOV	open/close/end of volume
OPNWA	open work area
PIB	program information block
PL/S	programming language/system
PLH	placeholder
PSW	program status word
PT or PTR	pointer
PUB	physical unit block
RAB	record area block
RBA	relative byte address
RDF	record definition field
REP	replication
Rn	register n
RPHD	resource pool header
RPL	request parameter list
RRDS	relative-record data set
RSCB	resource sharing control block

SCIB	search compressed index block
SEOF	software end of file
SEQ	sequential
SHRW	file sharing work area
SKP	skip sequential
SS	sequence set
SVC	supervisor call
THB	the hold block
TIC	transfer in channel
UBF	user buffer
UCAT	user catalog
UPD	update mode (or data modify)
USB	upgrade set block
USVR	user security verification routine
VOLID	volume identification
VRPPL	BLDVRP parameter list
VSAM	virtual storage access method
VSRT	VSAM shared resource table
VTOC	volume table of contents
WA	work area

Section 1. Introduction

Virtual Storage Access Method (VSAM) is an access method that operates with direct-access storage to provide fast storage and retrieval of data.

VSAM is divided into modules, which are logically grouped into components.

- Record management, which reads and writes records in response to user-issued VSAM and ISAM macro instructions. This component also reads and writes records for the catalog management component.
- End of Volume, which mounts volumes and allocates space. End of Volume modifies the existing control blocks to reflect the newly mounted volumes and newly allocated space.
- Service aids, which enable program maintenance and Field Engineering personnel to obtain dumps, maintain VTOC labels, and load phases.

The following components are documented in *VSE/VSAM VSAM Logic, Volume 1.*

- Control block manipulation, which allows the user program to create, modify, display, and test the contents of some VSAM control blocks (the ACB, EXLST, and RPL, which are described under *Data Areas* in this publication), and to build or delete a VSAM resource pool.
- Open, which connects a user's program to a VSAM data set and builds the control blocks required to permit the user to read from and write to the data set.
- ISAM interface, which allows the user program to issue ISAM macro instructions to process records in a VSAM data set.
- Catalog management, which writes and updates catalog records. Catalog management processes the catalog to obtain information for Open, Close, end-of-volume, and Access Method Services.
- DADSM, which allows the system to maintain VTOC labels for data spaces. In VSAM, DADSM is used by the catalog to create and delete data spaces, both unique and nonunique.
- Close, which disconnects a user's program from a data set and releases the data set's control blocks built by Open. Close also updates statistics in the VSAM catalog.

For a list of the modules in these components, see the *Directory* in this publication.

Section 2. Method of Operation

Reading Method of Operation Diagrams

Method of operation diagrams depict the internal functions of a programming system, in this case, an access method. The internal functions are categorized by the macro instructions issued by the user, such as the GENCB, MODCB, OPEN, GET, PUT, CLOSE and ENDREQ macro instructions.

Diagram AB shows the basic organization of the method of operation diagrams according to the macro instructions mentioned above. References lead from the high-level charts showing subfunctions required to carry a request to its completion.

Note the relationship of function (exemplified by the macro instructions) to component. Starting with an OPEN issued by the user, a logical progression is made from Open modules to supporting Catalog modules. When a record management macro instruction such as PUT is issued by the user, not only the Record Management modules are involved (which include modules that perform buffer and I/O management and end-of-volume processing) but the Catalog modules which, in turn, call upon the DADSM modules for space management.

The diagrams contain three blocks of information: input, processing, and output. The left-hand side of the diagram shows the data that serves as input to the processing steps on the center of the diagram, and the right-hand side shows the data that is output from the processing steps. Input is anything significant that program processing steps refer to or get. Processing is the steps that support the function or subfunction represented by the diagram. Output is any significant change effected by a processing step, for example, register contents, or control blocks created or modified. The processing steps are numbered and the numbers correspond to notes, if any, on the pages following the appropriate diagram(s). If notes are given, they include references to modules, routines, and/or labels shown on the extreme right-hand side of the diagram. These references are your link to the program listings. Figure 2.1 shows the symbols used in these diagrams and describes their meaning.

As an example of how to interpret a typical method of operation diagram, see page 1 of Diagram CG, which graphically depicts the control interval space reclamation function. The left-hand side of the diagram shows the significant input required by the processing steps shown in the diagram. The data-set information in the AMDSB is input to steps 3, 4, and 6 in the processing portion of the diagram. The processing portion of the diagram shows the processing steps required to fulfill the function described by the diagram. Note that the function described by one diagram may be performed by one or more VSAM modules; that is, the diagrams not only show program flow, but show the subfunctions that are required to carry out the function and that are subsequently shown in separate diagrams.

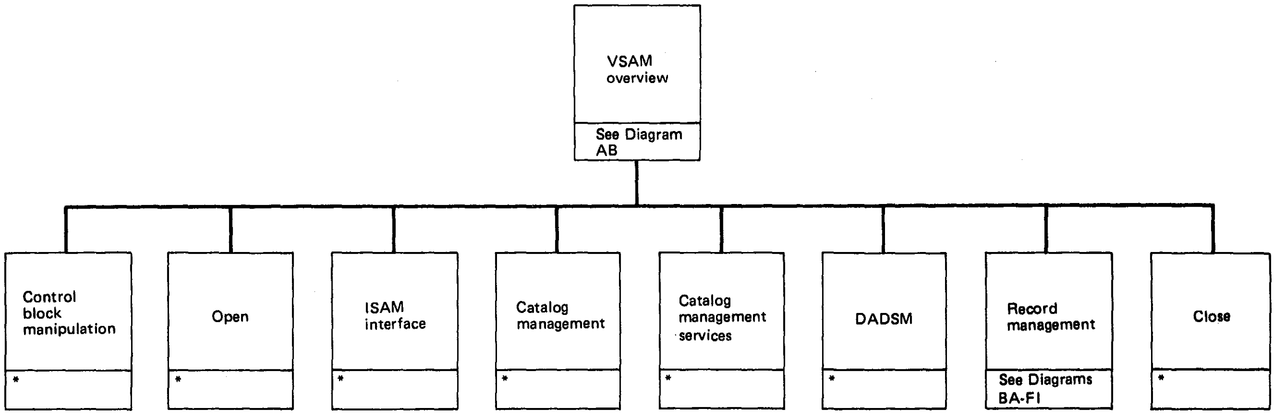
Note that some diagrams have more than one entry point.

The notes provide details about the processing shown in the diagram. For example, note 13 tells how index entries are found (in step 13). The diagrams are numbered in a sequence that follows the pattern *ccn,* where the first character, in general, represents a part of VSAM such as buffer management, the second character represents a category within buffer management, and the number represents the first, second, third, etc., page of that particular diagram. Thus, DG1 would be the first page (1) of the share option 4 hold function for buffer management (D). See the list of diagrams for details.

⇨	DATA MOVE -- shows input and output of data to and from the processing steps
➡	FLOW OF ACTION -- shows the flow of action through the processing steps
▪▪▪▪➡	ERROR FLOW -- shows the flow of action in the case of an error or for unusual conditions. This symbol is normally used only in the steps where an error causes a change of flow. Further error processing uses the normal symbols for flow of action.
⟶	POINTERS -- these are the interconnections between the control blocks and data areas
- - - - →	REFERENCE INDICATOR -- means *refer to this item*
Ⓐ	I/O INDICATOR -- used where arrows had to be interrupted, or where one input or output is used for several steps. Circles within one diagram with identical letters are regarded as being logically connected.
①	ONPAGE CONNECTOR -- used together with the flow of action symbols to indicate a branch, within the same diagram, to the step number within the circle, regardless of whether the diagram is on more than one page.
⬠	OFFPAGE CONNECTOR -- used for backward reference between diagrams.
↑	ADDRESS -- indicates, within control blocks, that this field contains the address of the specified field.
PLH PLHWAREA WKABB WKACC WKAHH	In the input and output colums of some of the HIPO diagrams, control block sub-fields that are contained within a larger field are indicated by dashed lines. In this example, sub-fields WKABB, WKACC, and WKAHH are all contained within the field PLHWAREA. (In this particular case, these sub-fields are mapped by a different mapping macro, which accounts for their labels beginning with "WKA.")

Figure 2.1 **Symbols used on method of operation diagrams**

Diagram AA1. Method of Operation Contents



*See VSE/VSAM VSAM Logic, Volume 1

Diagram AB1. VSAM Overview

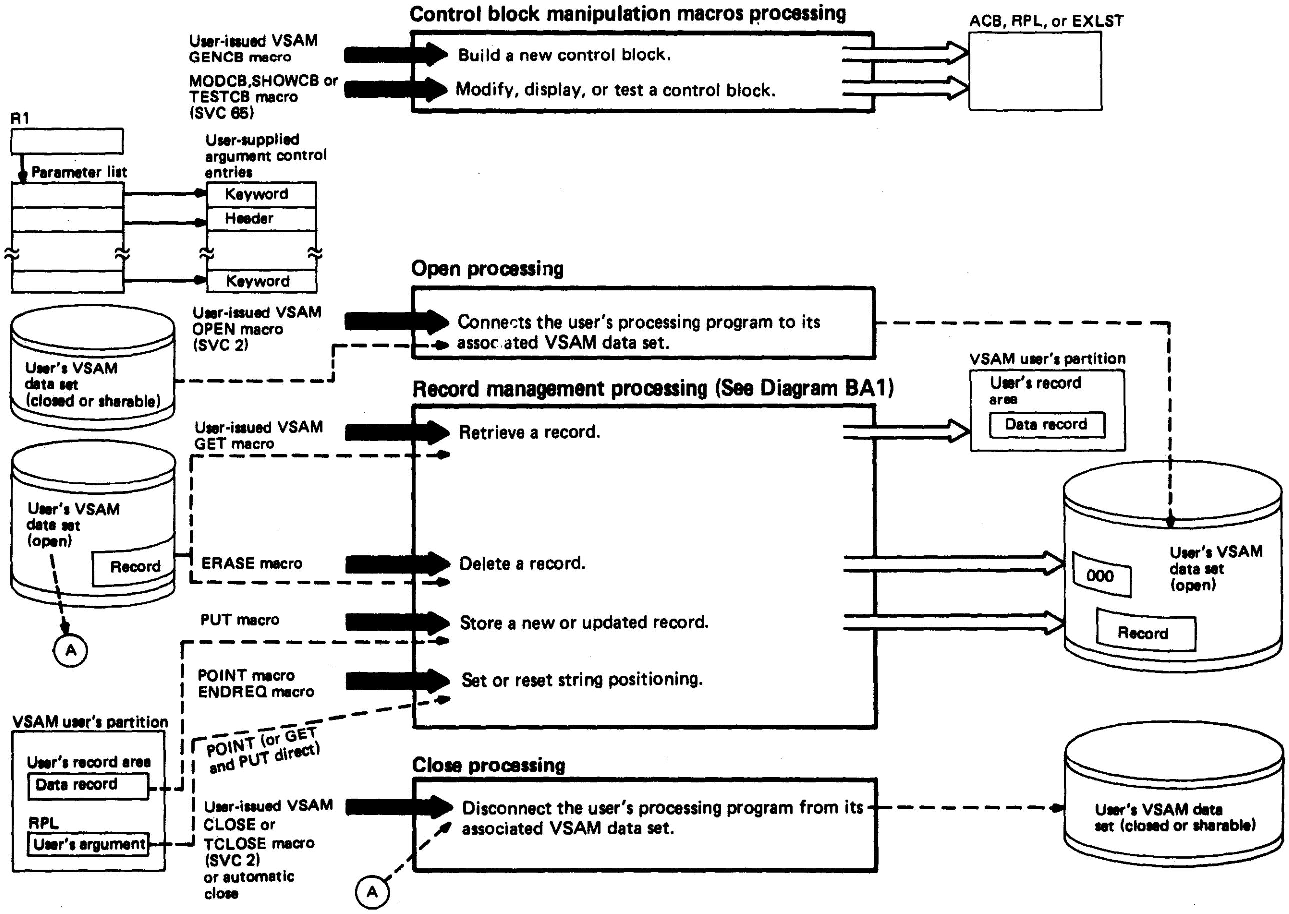

Control block manipulation macros processing
ACB, RPL, or EXLST
User-issued VSAM GENCB macro
Build a new control block.
MODCB,SHOWCB or TESTCB macro (SVC 65)
Modify, display, or test a control block.
R1
Parameter list
User-supplied argument control entries
Keyword
Header
Keyword
Open processing
User-issued VSAM OPEN macro (SVC 2)
Connects the user's processing program to its associated VSAM data set.
User's VSAM data set (closed or sharable)
VSAM user's partition
User's record area
Data record
Record management processing (See Diagram BA1)
User-issued VSAM GET macro
Retrieve a record.
User's VSAM data set (open)
Record
ERASE macro
Delete a record.
User's VSAM data set (open)
000
PUT macro
Store a new or updated record.
Record
A
POINT macro
ENDREQ macro
Set or reset string positioning.
VSAM user's partition
User's record area
Data record
RPL
User's argument
POINT (or GET and PUT direct)
Close processing
User-issued VSAM CLOSE or TCLOSE macro (SVC 2) or automatic close
Disconnect the user's processing program from its associated VSAM data set.
A
User's VSAM data set (closed or sharable)

Diagram BA1. Record Management Contents

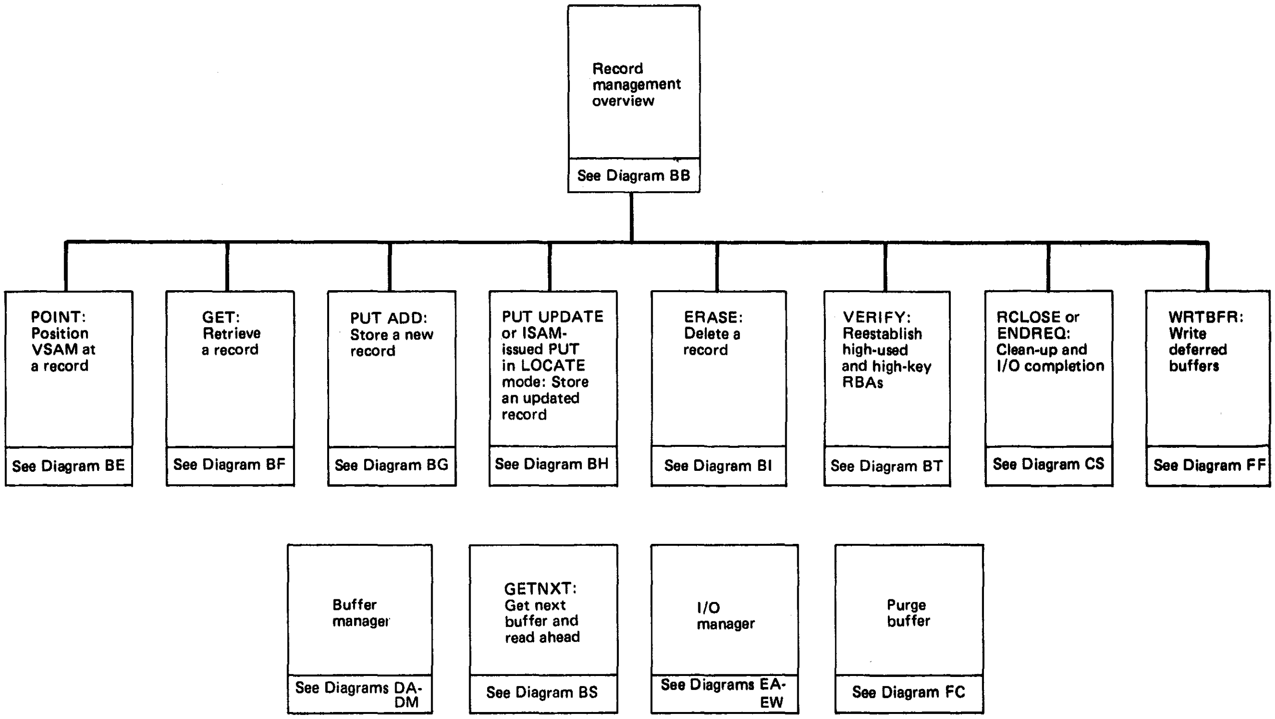
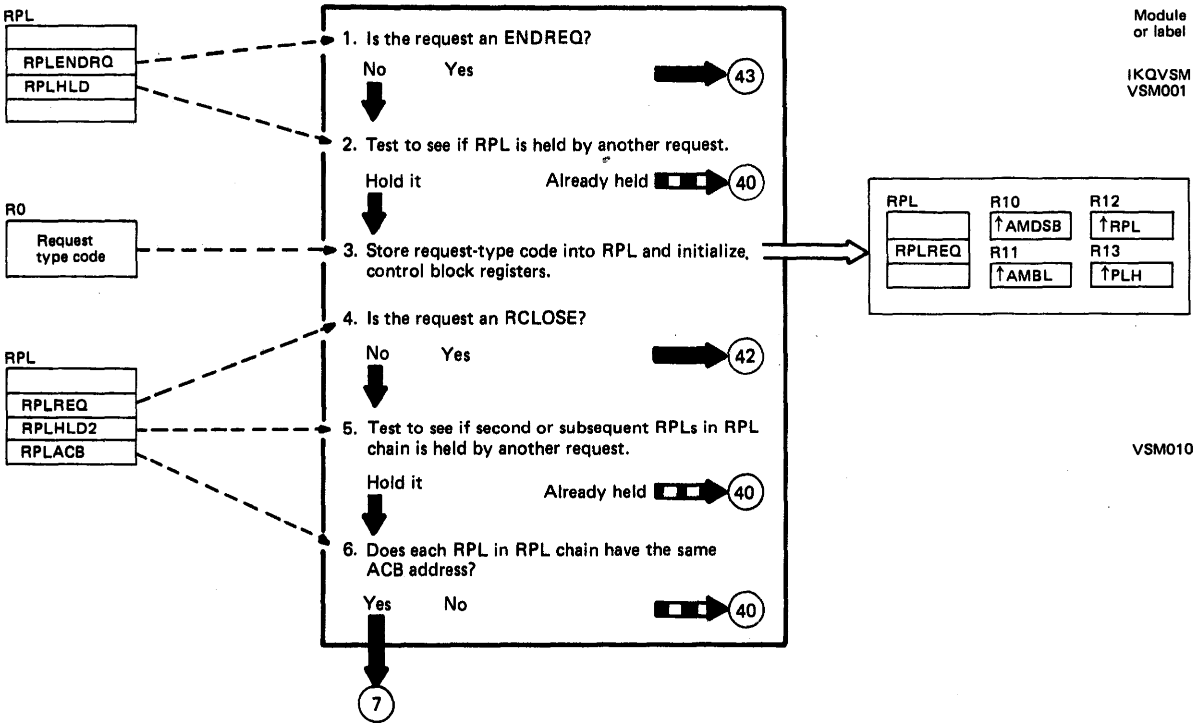

Record management overview
See Diagram BB
POINT: Position VSAM at a record
See Diagram BE
GET: Retrieve a record
See Diagram BF
PUT ADD: Store a new record
See Diagram BG
PUT UPDATE or ISAM-issued PUT in LOCATE mode: Store an updated record
See Diagram BH
ERASE: Delete a record
See Diagram BI
VERIFY: Reestablish high-used and high-key RBAs
See Diagram BT
RCLOSE or ENDREQ: Clean-up and I/O completion
See Diagram CS
WRTBFR: Write deferred buffers
See Diagram FF
Buffer manager
See Diagrams DA-DM
GETNXT: Get next buffer and read ahead
See Diagram BS
I/O manager
See Diagrams EA-EW
Purge buffer
See Diagram FC

Diagram BB1. Record Management Overview



1. Is the request an ENDREQ?

2. Test to see if RPL is held by another request.

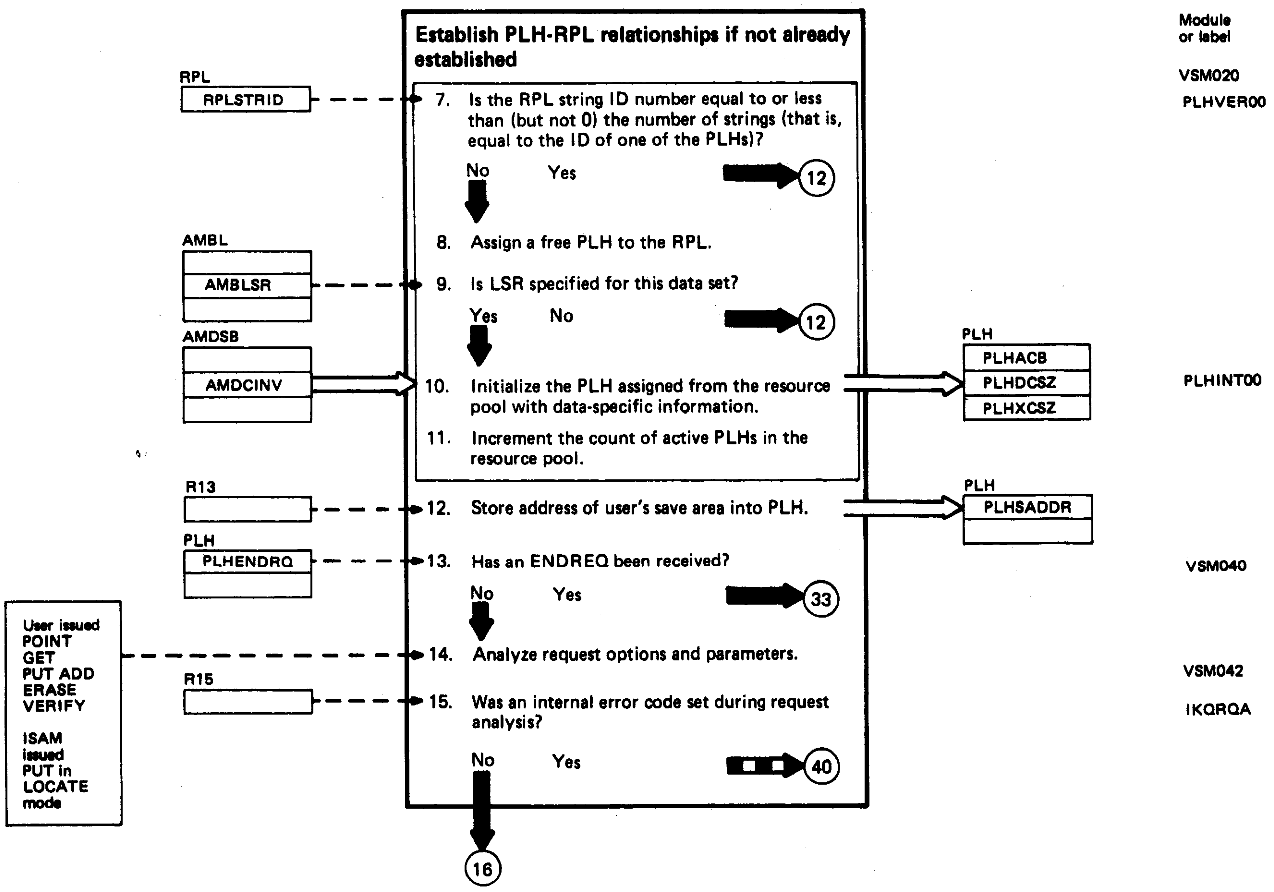
3. Store request-type code into RPL and initialize control block registers.

4. Is the request an RCLOSE?

5. Test to see if second or subsequent RPLs in RPL chain is held by another request.

6. Does each RPL in RPL chain have the same ACB address?

Diagram BB2. Record Management Overview



Establish PLH-RPL relationships if not already established

7. Is the RPL string ID number equal to or less than (but not 0) the number of strings (that is, equal to the ID of one of the PLHs)?
   No Yes → 12

8. Assign a free PLH to the RPL.

9. Is LSR specified for this data set?
   Yes No → 12

10. Initialize the PLH assigned from the resource pool with data-specific information.

11. Increment the count of active PLHs in the resource pool.

12. Store address of user's save area into PLH.

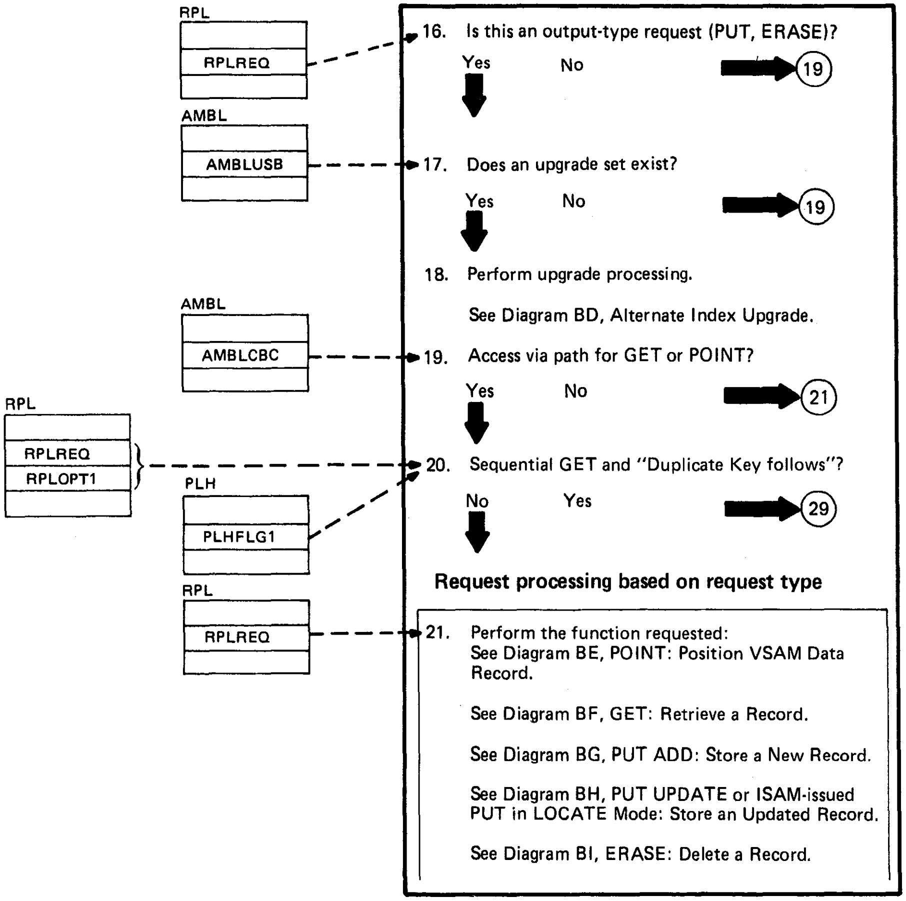
13. Has an ENDREQ been received?
    No Yes → 33

14. Analyze request options and parameters.

15. Was an internal error code set during request analysis?
    No Yes → 40

16

Diagram BB3. Record Management Overview



16. Is this an output-type request (PUT, ERASE)?

17. Does an upgrade set exist?

18. Perform upgrade processing.

See Diagram BD, Alternate Index Upgrade.

19. Access via path for GET or POINT?

20. Sequential GET and "Duplicate Key follows"?

Request processing based on request type

21. Perform the function requested:
See Diagram BE, POINT: Position VSAM Data Record.

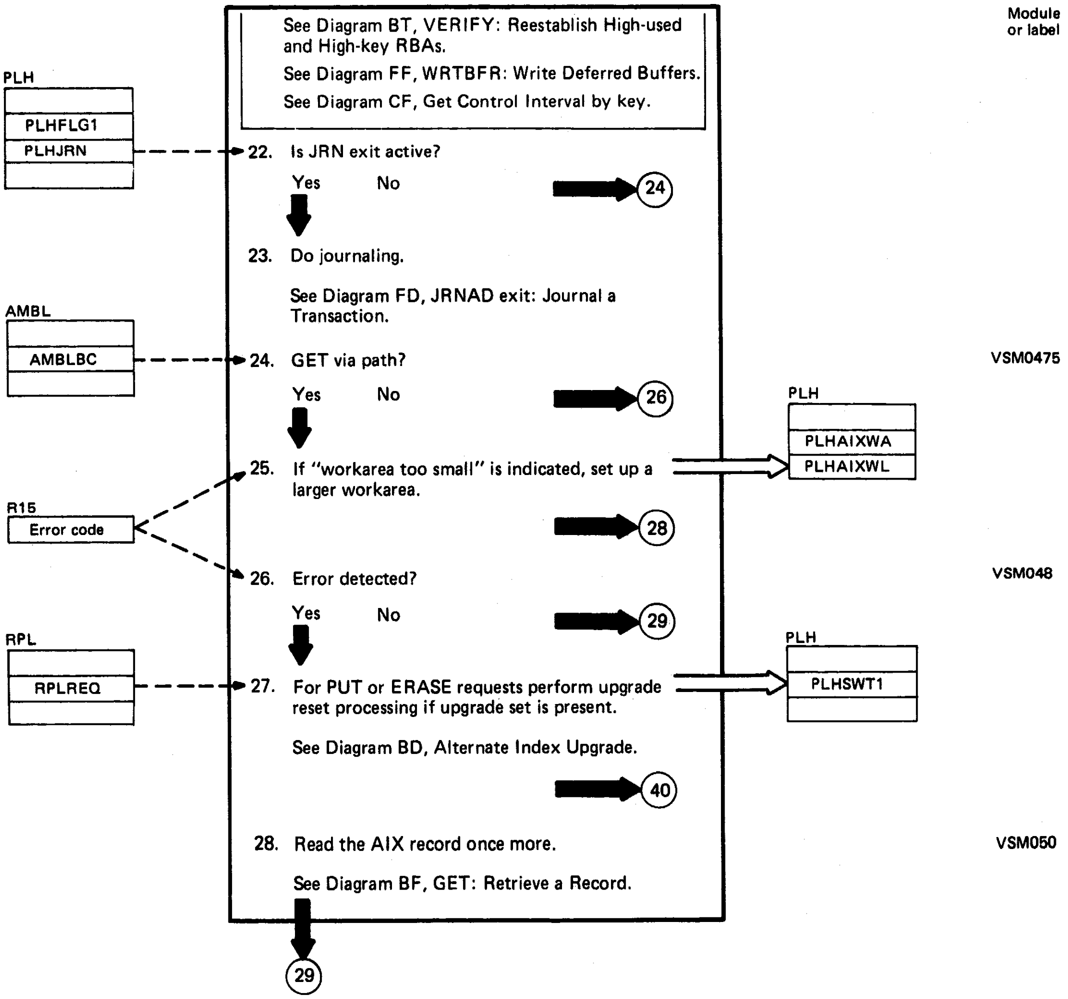
See Diagram BF, GET: Retrieve a Record.

See Diagram BG, PUT ADD: Store a New Record.

See Diagram BH, PUT UPDATE or ISAM-issued PUT in LOCATE Mode: Store an Updated Record.

See Diagram BI, ERASE: Delete a Record.

Diagram BB4. Record Management Overview



See Diagram BT, VERIFY: Reestablish High-used and High-key RBAs.

See Diagram FF, WRTBFR: Write Deferred Buffers.

See Diagram CF, Get Control Interval by key.

22. Is JRN exit active?

23. Do journaling.

See Diagram FD, JRNAD exit: Journal a Transaction.

24. GET via path?

25. If "workarea too small" is indicated, set up a larger workarea.

26. Error detected?

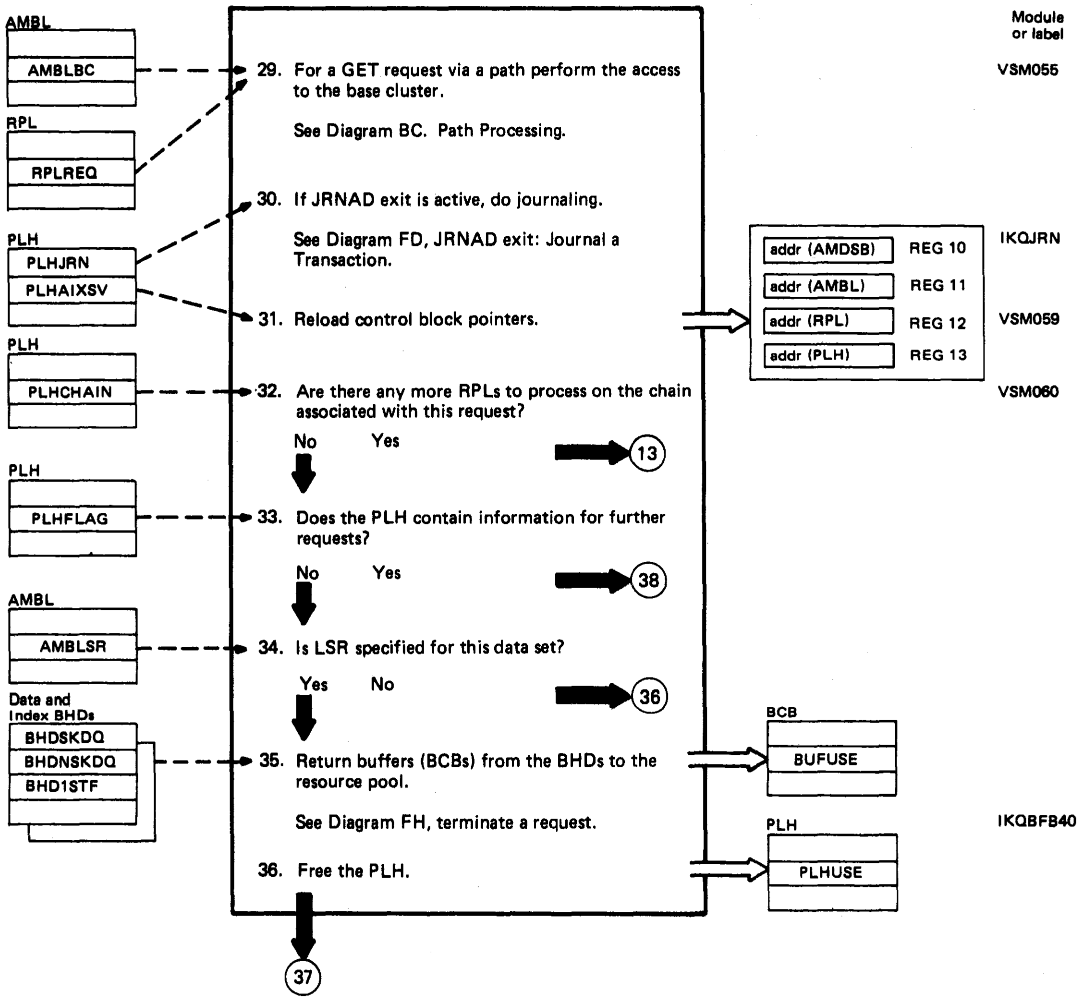
27. For PUT or ERASE requests perform upgrade reset processing if upgrade set is present.

See Diagram BD, Alternate Index Upgrade.

28. Read the AIX record once more.

See Diagram BF, GET: Retrieve a Record.

Diagram BB5. Record Management Overview



29. For a GET request via a path perform the access to the base cluster.

See Diagram BC. Path Processing.

30. If JRNAD exit is active, do journaling.

See Diagram FD, JRNAD exit: Journal a Transaction.

31. Reload control block pointers.

32. Are there any more RPLs to process on the chain associated with this request?

No Yes 13

33. Does the PLH contain information for further requests?

No Yes 38

34. Is LSR specified for this data set?

Yes No 36

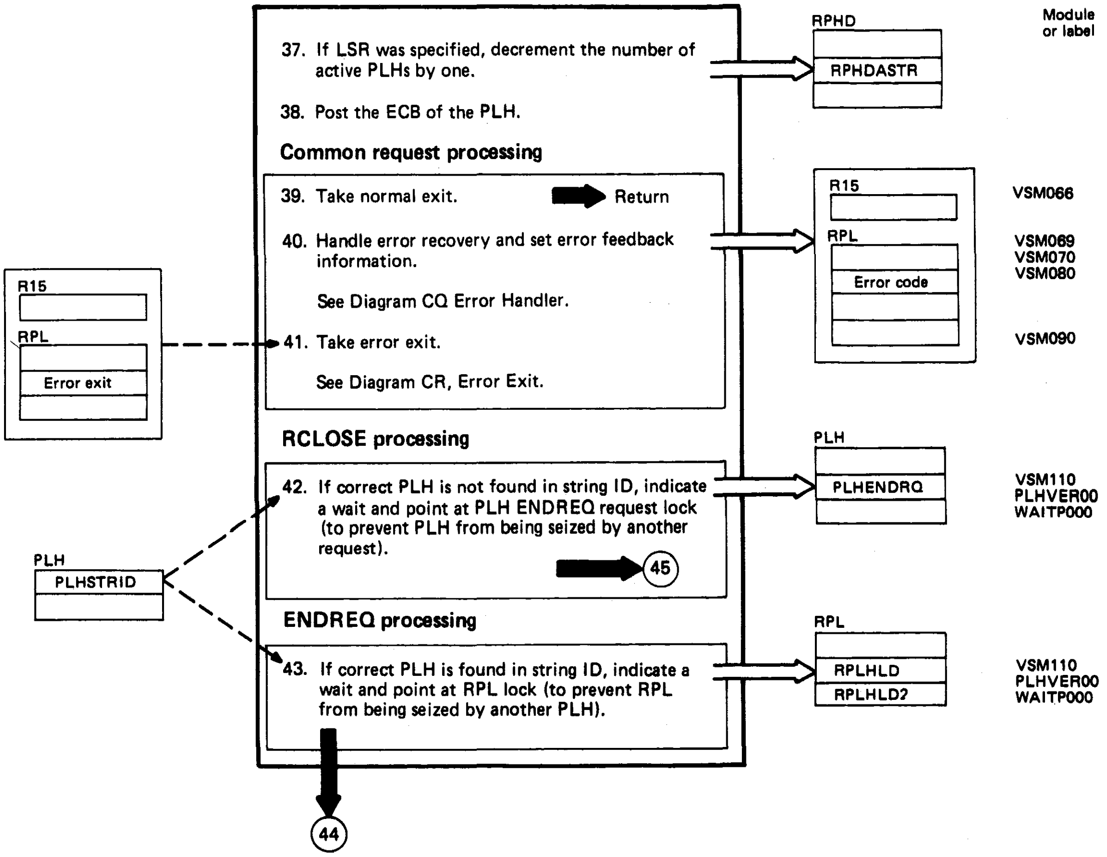
35. Return buffers (BCBs) from the BHDs to the resource pool.

See Diagram FH, terminate a request.

36. Free the PLH.

37

Diagram BB6. Record Management Overview



37. If LSR was specified, decrement the number of active PLHs by one.

38. Post the ECB of the PLH.

Common request processing

39. Take normal exit. ➡ Return

40. Handle error recovery and set error feedback information.

See Diagram CQ Error Handler.

41. Take error exit.

See Diagram CR, Error Exit.

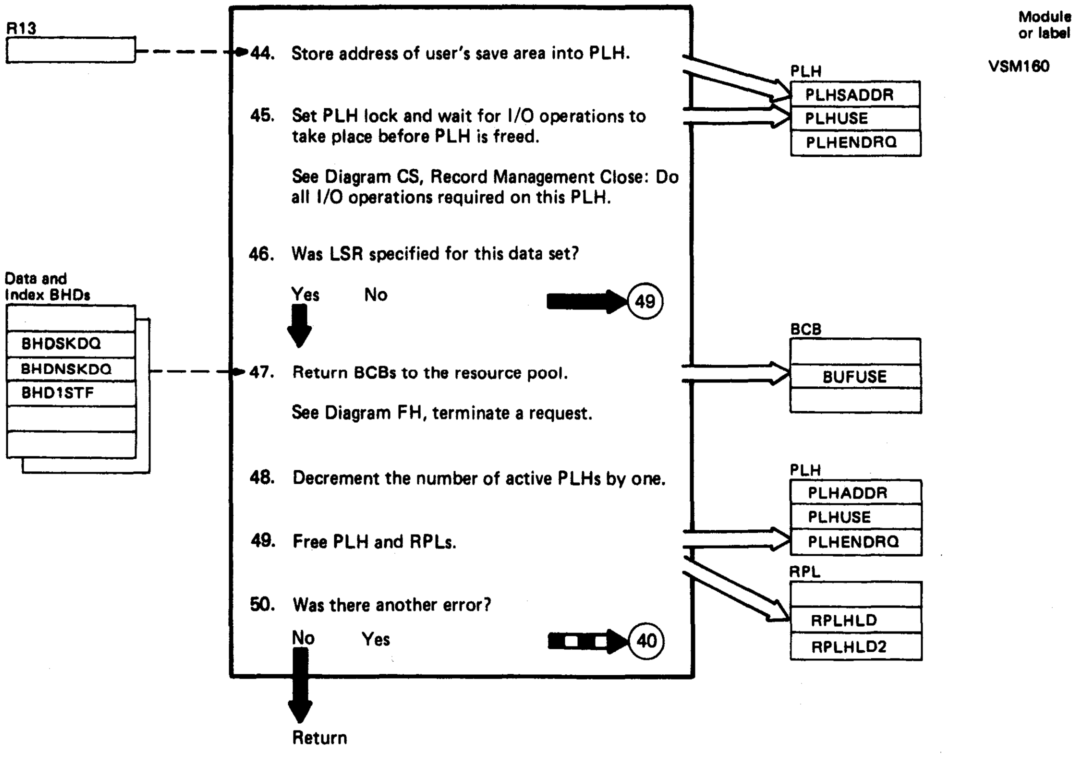
RCLOSE processing

42. If correct PLH is not found in string ID, indicate a wait and point at PLH ENDREQ request lock (to prevent PLH from being seized by another request). ➡ 45

ENDREQ processing

43. If correct PLH is found in string ID, indicate a wait and point at RPL lock (to prevent RPL from being seized by another PLH).

Diagram BB7. Record Management Overview



44. Store address of user's save area into PLH.

45. Set PLH lock and wait for I/O operations to take place before PLH is freed.

See Diagram CS, Record Management Close: Do all I/O operations required on this PLH.

46. Was LSR specified for this data set?

Yes No 49

47. Return BCBs to the resource pool.

See Diagram FH, terminate a request.

48. Decrement the number of active PLHs by one.

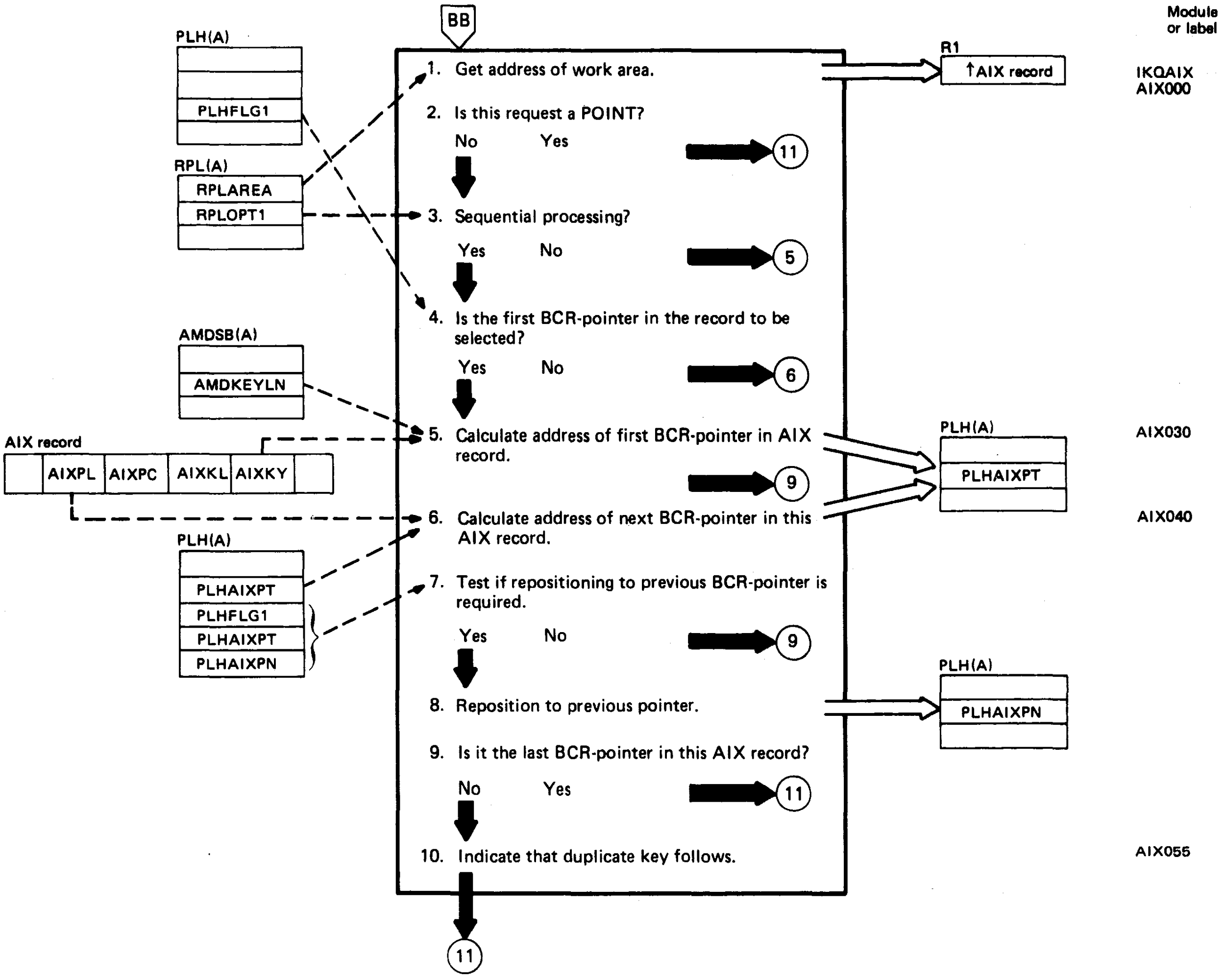
49. Free PLH and RPLs.

50. Was there another error?

No Yes 40

Return

Diagram BC1. Path Processing


BB
Module or label
PLH(A)
PLHFLG1
RPL(A)
RPLAREA
RPLOPT1
AMDSB(A)
AMDKEYLN
AIX record
AIXPL
AIXPC
AIXKL
AIXKY
PLH(A)
PLHAIXPT
PLHFLG1
PLHAIXPT
PLHAIXPN
1. Get address of work area.
2. Is this request a POINT?
No
Yes
11
3. Sequential processing?
Yes
No
5
4. Is the first BCR-pointer in the record to be selected?
Yes
No
6
5. Calculate address of first BCR-pointer in AIX record.
9
6. Calculate address of next BCR-pointer in this AIX record.
7. Test if repositioning to previous BCR-pointer is required.
Yes
No
9
8. Reposition to previous pointer.
9. Is it the last BCR-pointer in this AIX record?
No
Yes
11
10. Indicate that duplicate key follows.
11
R1
↑AIX record
PLH(A)
PLHAIXPT
PLH(A)
PLHAIXPN
IKQAIX
AIX000
AIX030
AIX040
AIX055

Notes for Diagram BC1

General note for sequential processing

During sequential processing, the base records are returned in the order of the BCR pointers in the AIX record, regardless of the "direction" of sequential processing (forward or backward).

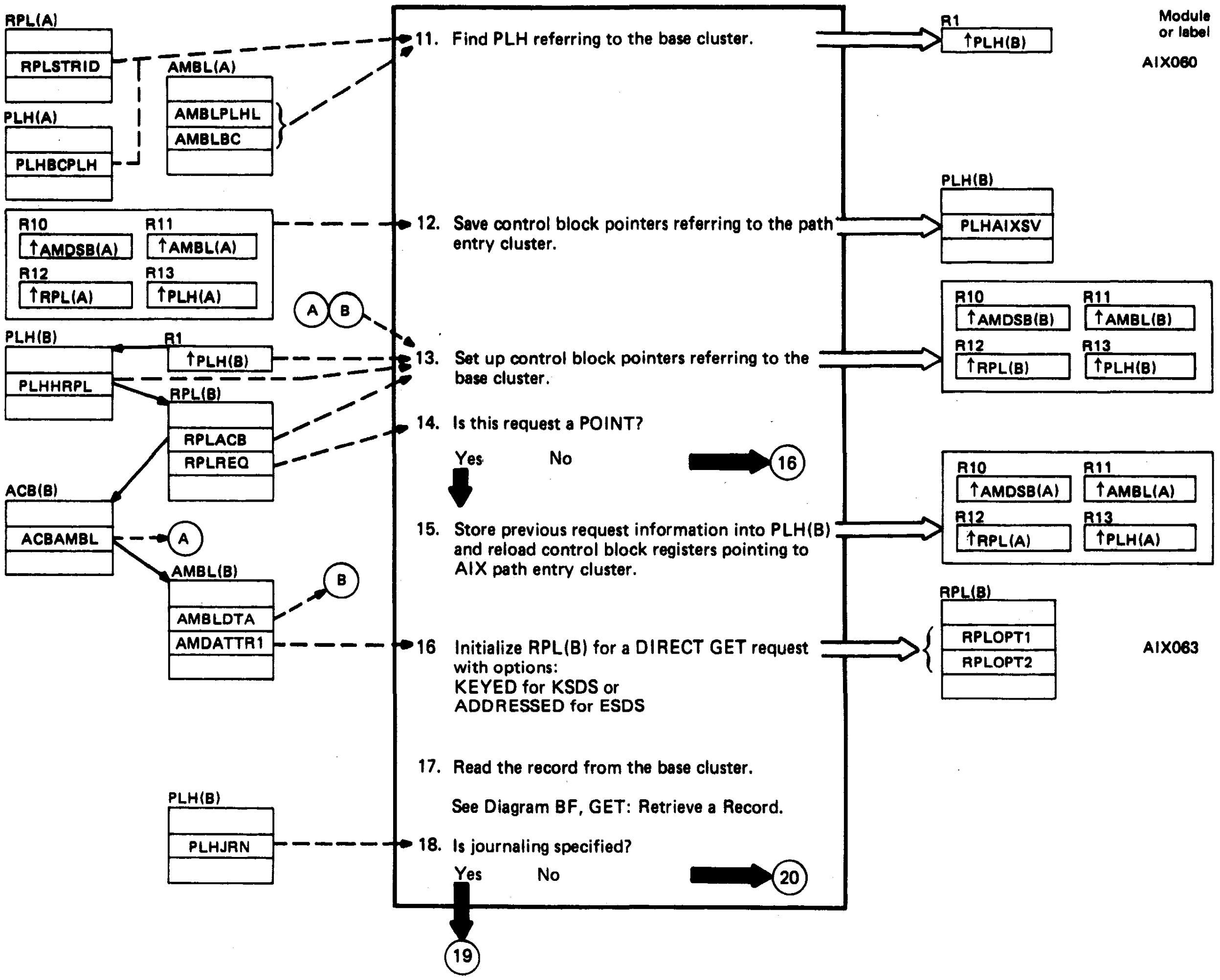
Control block notation

RPL(A), PLH(A), etc.: control blocks referring to the path entry (AIX).

RPL(B), PLH(B), etc.: control blocks referring to the base cluster.

3. If sequential processing is not specified, only the first base record associated with a given AIX key is retrieved. The indicator "duplicate key follows" is set if applicable.
7. Repositioning to the previous BCR pointer is required only if the previous sequential request ended with a "no record found" condition in the base cluster. To prevent the user from simply skipping this BCR pointer (which represents a lack of compatibility between AIX and base cluster), the PLH, which had been incremented in the previous request to point to the next BCR pointer, is returned to the faulty BCR pointer. A series of sequential GETs will thus return a series of "no record found" conditions, all for the same base record.

Diagram BC2. Path Processing



11. Find PLH referring to the base cluster.

12. Save control block pointers referring to the path entry cluster.

13. Set up control block pointers referring to the base cluster.

14. Is this request a POINT?

15. Store previous request information into PLH(B) and reload control block registers pointing to AIX path entry cluster.

16 Initialize RPL(B) for a DIRECT GET request with options:
KEYED for KSDS or
ADDRESSED for ESDS

17. Read the record from the base cluster.

See Diagram BF, GET: Retrieve a Record.

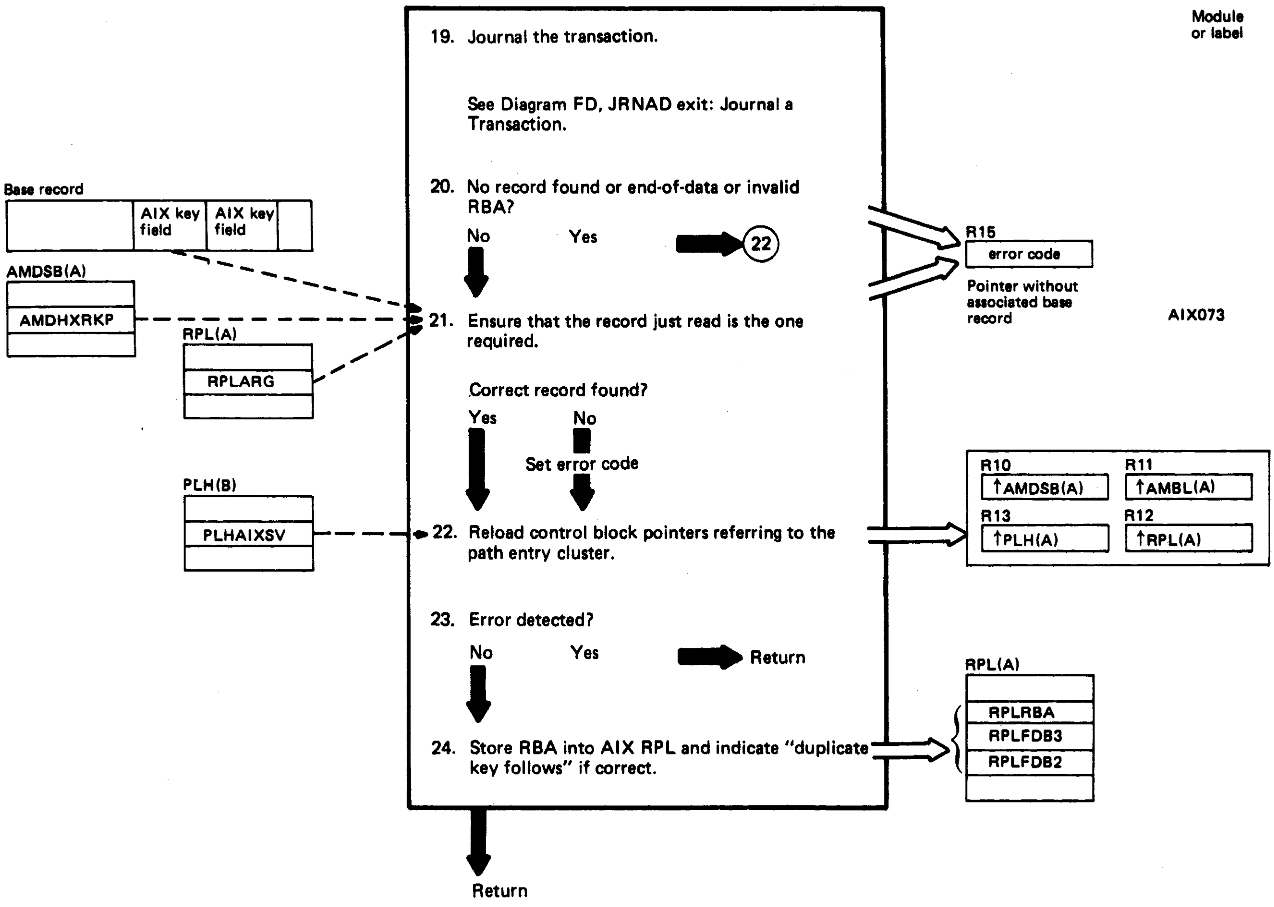
18. Is journaling specified?

Notes for Diagram BC2

11. A direct pointer (PLHBCPLH) is used for local shared resources. Otherwise the base cluster PLH is indexed by the AMBL (base cluster).

14. A POINT request is completed when the PLH is positioned to the correct BCR pointer. A GET request continues and actually retrieves the base cluster record indicated by the current BCR pointer.

Diagram BC3. Path Processing



19. Journal the transaction.

See Diagram FD, JRNAD exit: Journal a Transaction.

20. No record found or end-of-data or invalid RBA?

No Yes → 22

21. Ensure that the record just read is the one required.

Correct record found?

Yes No

Set error code

22. Reload control block pointers referring to the path entry cluster.

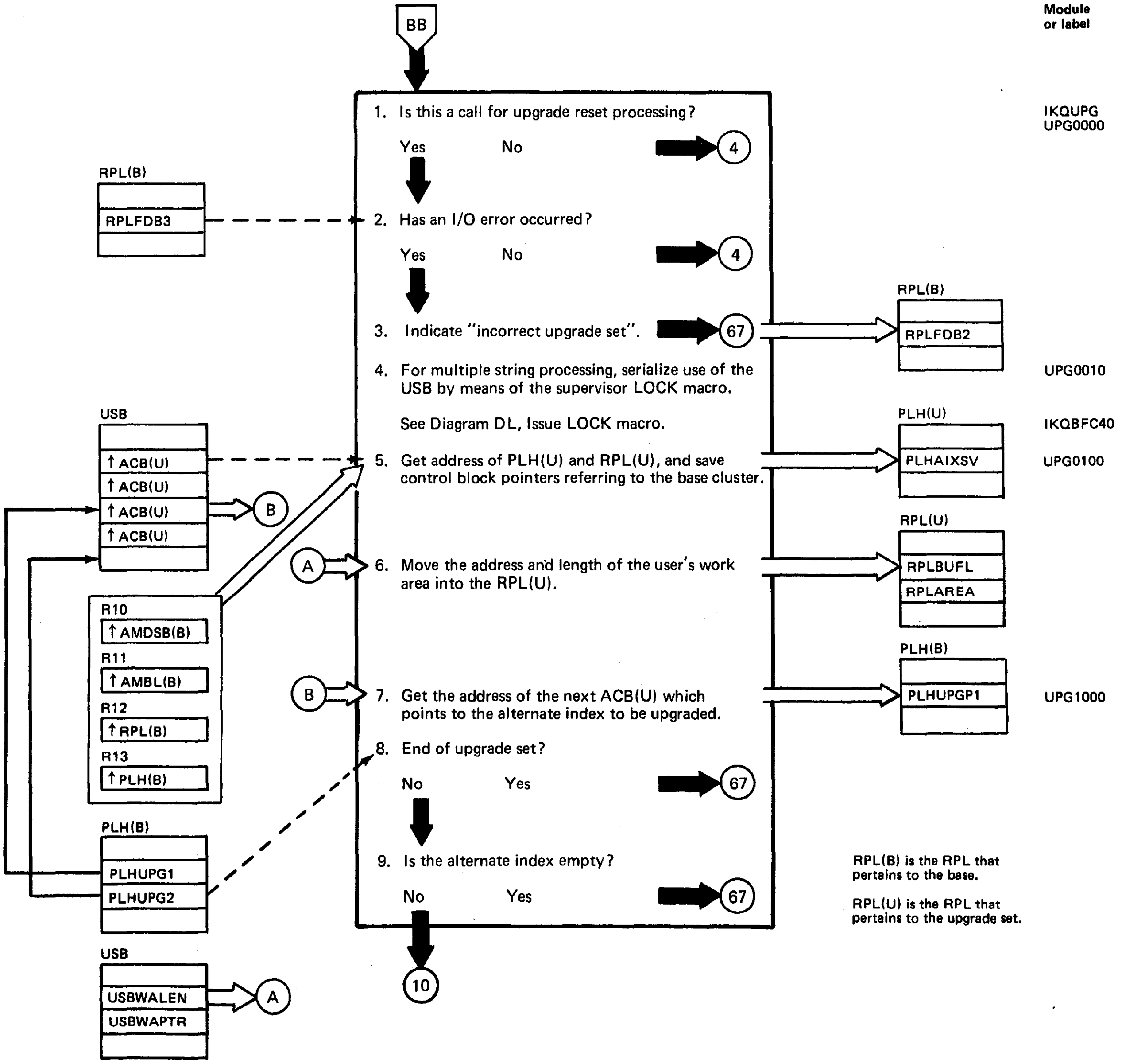
23. Error detected?

No Yes → Return

24. Store RBA into AIX RPL and indicate "duplicate key follows" if correct.

Return

Diagram BD1. Alternate Index Upgrade



1. Is this a call for upgrade reset processing?
   Yes No
2. Has an I/O error occurred?
   Yes No
3. Indicate "incorrect upgrade set".
4. For multiple string processing, serialize use of the USB by means of the supervisor LOCK macro.

   See Diagram DL, Issue LOCK macro.
5. Get address of PLH(U) and RPL(U), and save control block pointers referring to the base cluster.
6. Move the address and length of the user's work area into the RPL(U).
7. Get the address of the next ACB(U) which points to the alternate index to be upgraded.
8. End of upgrade set?
   No Yes
9. Is the alternate index empty?
   No Yes

RPL(B) is the RPL that pertains to the base.

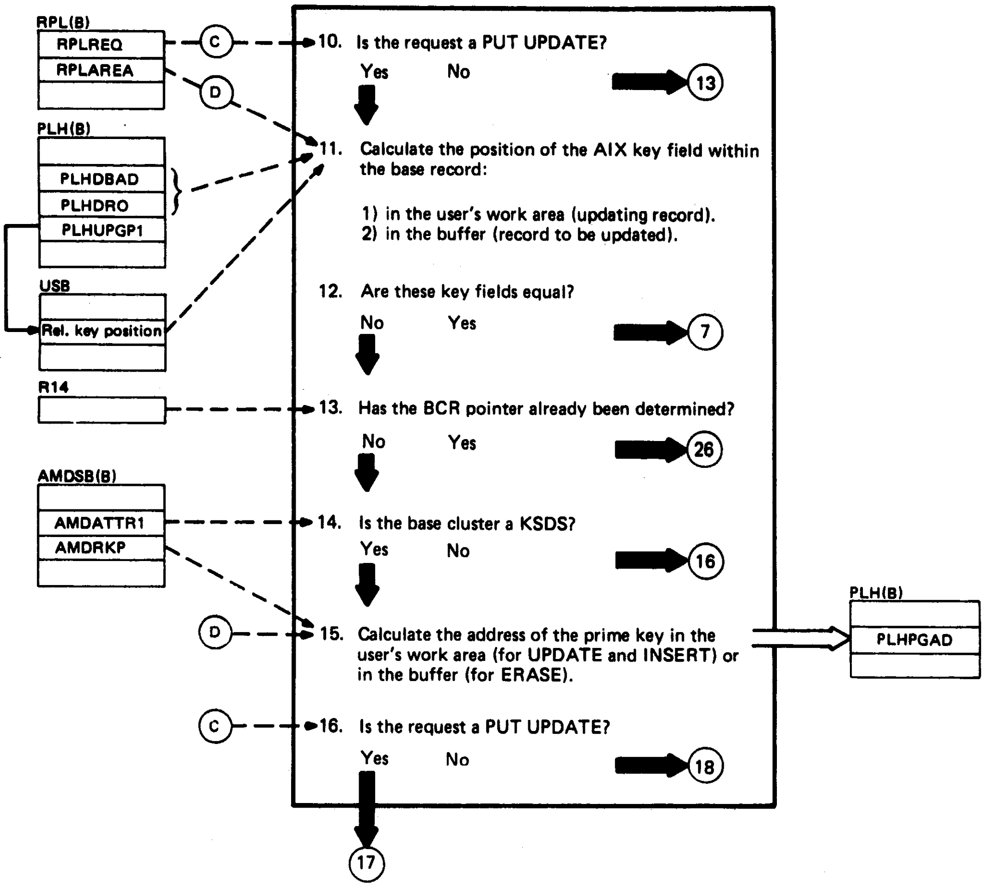
RPL(U) is the RPL that pertains to the upgrade set.

Notes for Diagram BD1

4. The LOCK macro (rather than a test-and-set lock) serializes use of the USB so that attempts to do multiple concurrent upgrades under the same VSE task can be detected and a return code issued. Because the LOCK macro is used, all multiple string upgrades over the same base cluster throughout the system are serialized.

   The LOCK name for USB serialization is the standard LOCK name for the base cluster (see note for step 1 of Diagram DG), except that the last two bytes are X'0004'.

Diagram BD2. Alternate Index Upgrade



Module or label

10. Is the request a PUT UPDATE?

11. Calculate the position of the AIX key field within the base record: UPG1020

    1) in the user's work area (updating record).
    2) in the buffer (record to be updated).

12. Are these key fields equal?

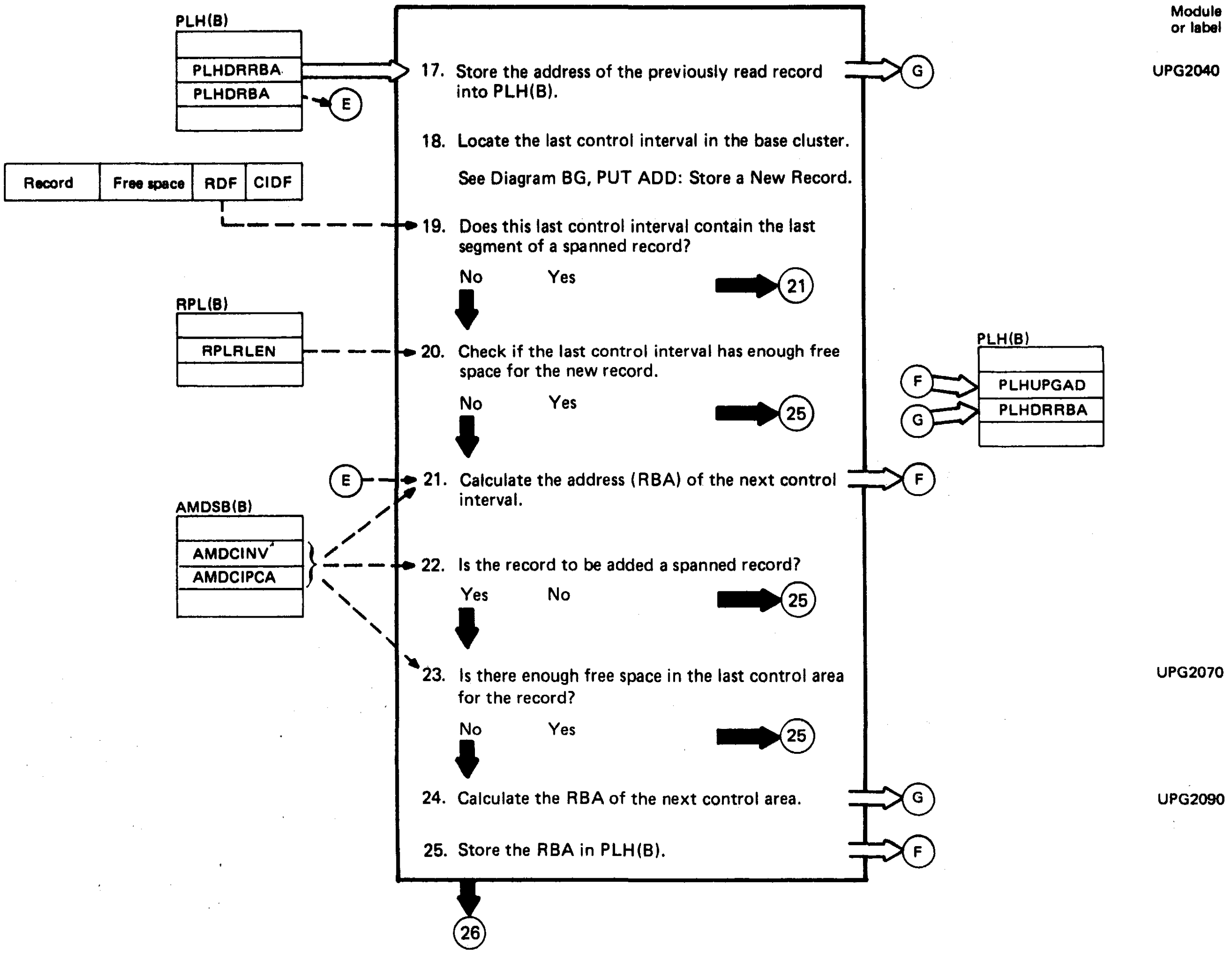
13. Has the BCR pointer already been determined?

14. Is the base cluster a KSDS?

15. Calculate the address of the prime key in the user's work area (for UPDATE and INSERT) or in the buffer (for ERASE).

16. Is the request a PUT UPDATE? UPG2030

Diagram BD3. Alternate Index Upgrade



17. Store the address of the previously read record into PLH(B).

18. Locate the last control interval in the base cluster.

See Diagram BG, PUT ADD: Store a New Record.

19. Does this last control interval contain the last segment of a spanned record?

20. Check if the last control interval has enough free space for the new record.

21. Calculate the address (RBA) of the next control interval.

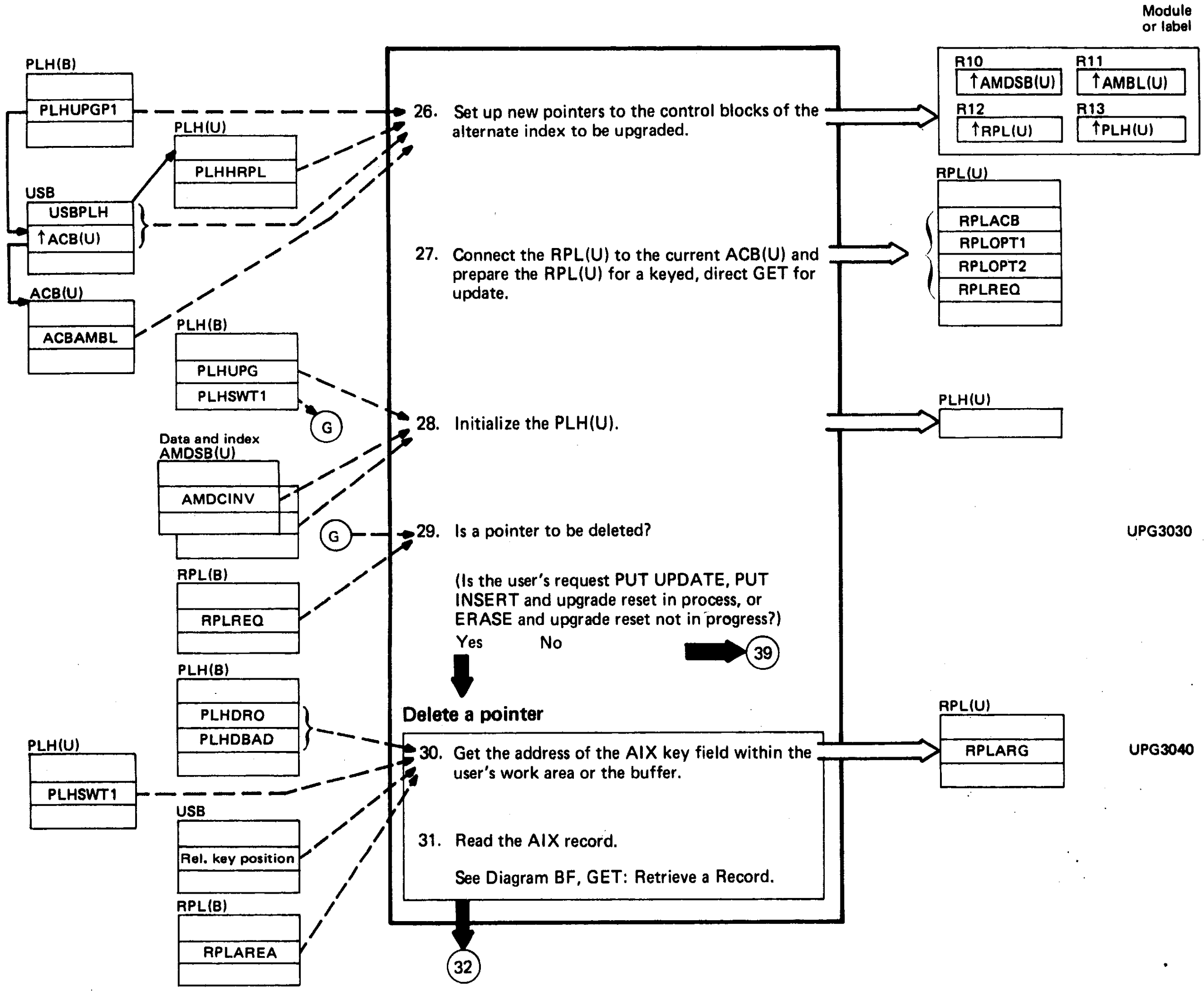
22. Is the record to be added a spanned record?

23. Is there enough free space in the last control area for the record?

24. Calculate the RBA of the next control area.

25. Store the RBA in PLH(B).

Diagram BD4. Alternate Index Upgrade



26. Set up new pointers to the control blocks of the alternate index to be upgraded.

27. Connect the RPL(U) to the current ACB(U) and prepare the RPL(U) for a keyed, direct GET for update.

28. Initialize the PLH(U).

29. Is a pointer to be deleted?

(Is the user's request PUT UPDATE, PUT INSERT and upgrade reset in process, or ERASE and upgrade reset not in progress?)

Yes No

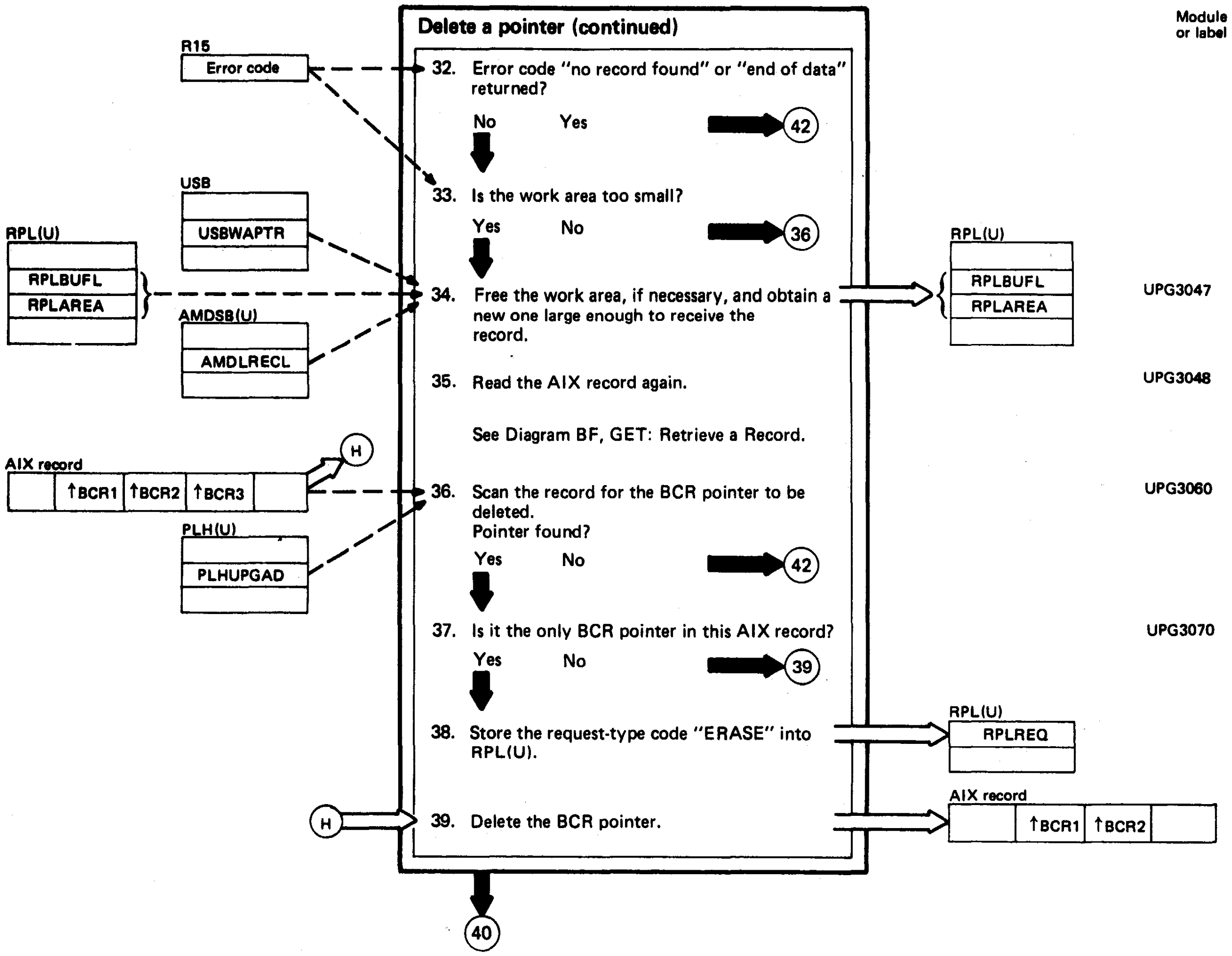
Delete a pointer

30. Get the address of the AIX key field within the user's work area or the buffer.

31. Read the AIX record.

See Diagram BF, GET: Retrieve a Record.

Diagram BD5. Alternate Index Upgrade



Delete a pointer (continued)

Module or label

32. Error code "no record found" or "end of data" returned?
    No Yes → 42

33. Is the work area too small?
    Yes No → 36

34. Free the work area, if necessary, and obtain a new one large enough to receive the record. — UPG3047

35. Read the AIX record again. — UPG3048

    See Diagram BF, GET: Retrieve a Record.

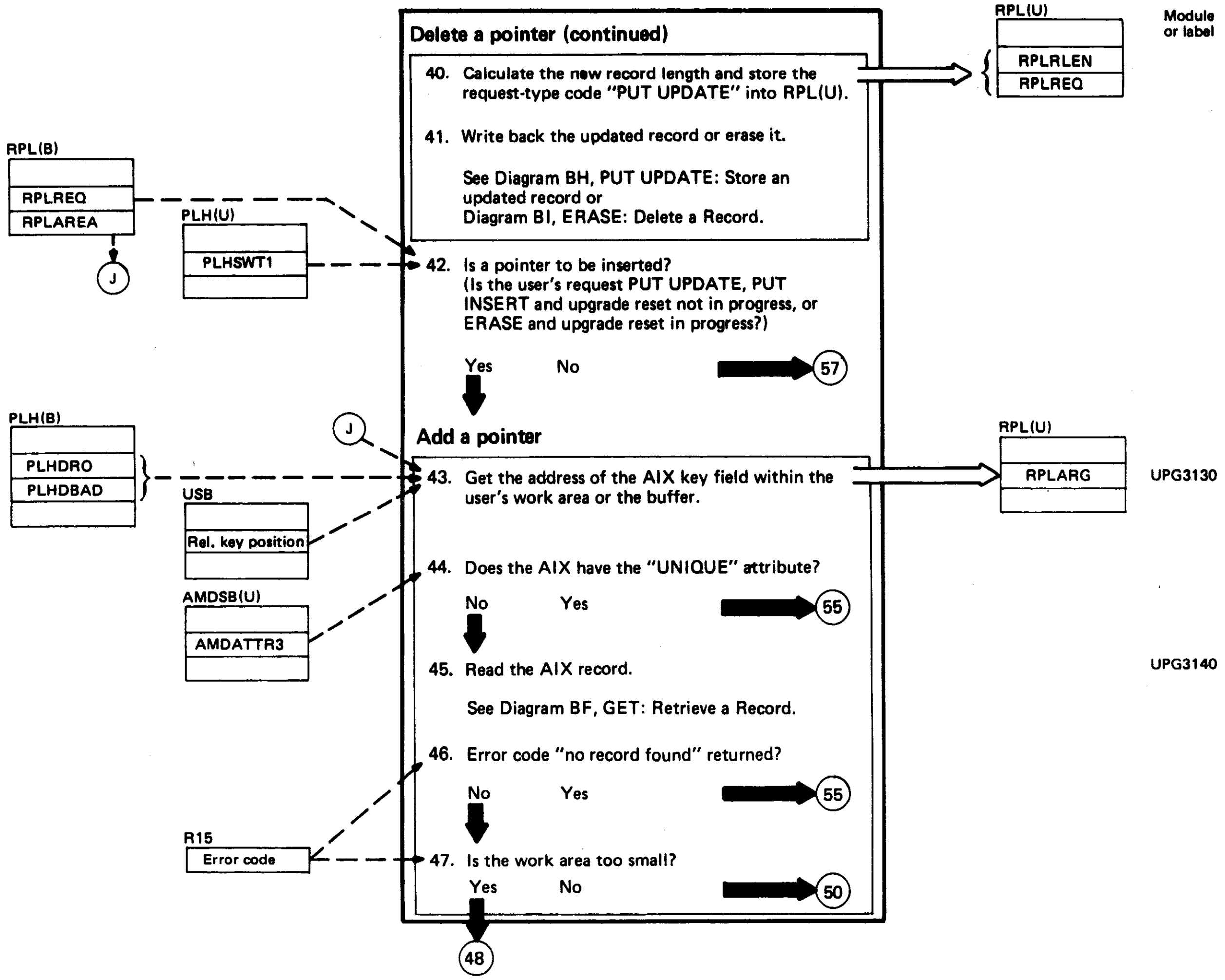
36. Scan the record for the BCR pointer to be deleted. — UPG3060
    Pointer found?
    Yes No → 42

37. Is it the only BCR pointer in this AIX record? — UPG3070
    Yes No → 39

38. Store the request-type code "ERASE" into RPL(U).

39. Delete the BCR pointer.

Diagram BD6. Alternate Index Upgrade



Delete a pointer (continued)

40. Calculate the new record length and store the request-type code "PUT UPDATE" into RPL(U).

41. Write back the updated record or erase it.

See Diagram BH, PUT UPDATE: Store an updated record or Diagram BI, ERASE: Delete a Record.

42. Is a pointer to be inserted? (Is the user's request PUT UPDATE, PUT INSERT and upgrade reset not in progress, or ERASE and upgrade reset in progress?)

Yes No 57

Add a pointer

43. Get the address of the AIX key field within the user's work area or the buffer.

44. Does the AIX have the "UNIQUE" attribute?

No Yes 55

45. Read the AIX record.

See Diagram BF, GET: Retrieve a Record.

46. Error code "no record found" returned?

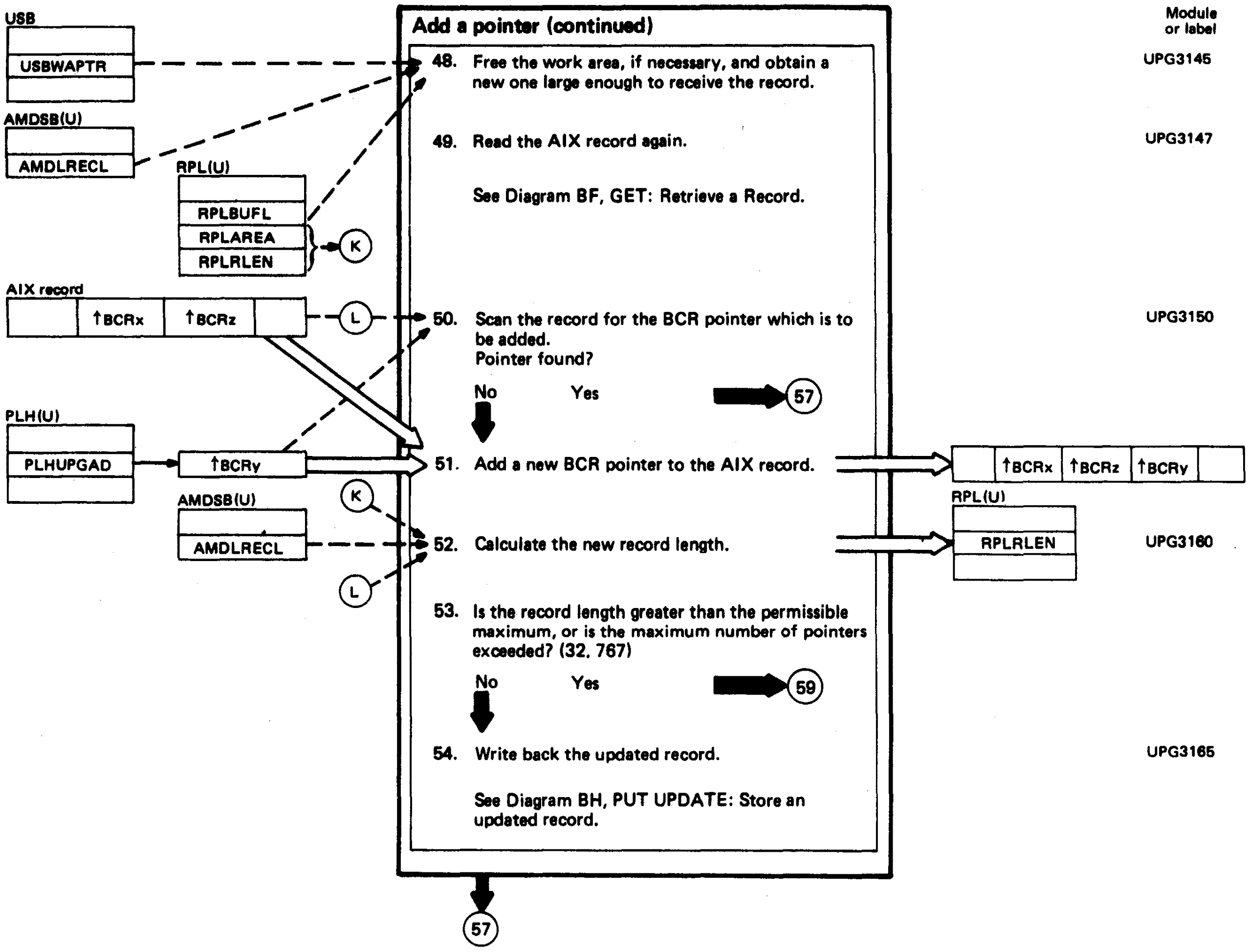
No Yes 55

47. Is the work area too small?

Yes No 50

48

Diagram BD7. Alternate Index Upgrade



Add a pointer (continued)

48. Free the work area, if necessary, and obtain a new one large enough to receive the record.

49. Read the AIX record again.

See Diagram BF, GET: Retrieve a Record.

50. Scan the record for the BCR pointer which is to be added.
Pointer found?
No Yes → 57

51. Add a new BCR pointer to the AIX record.

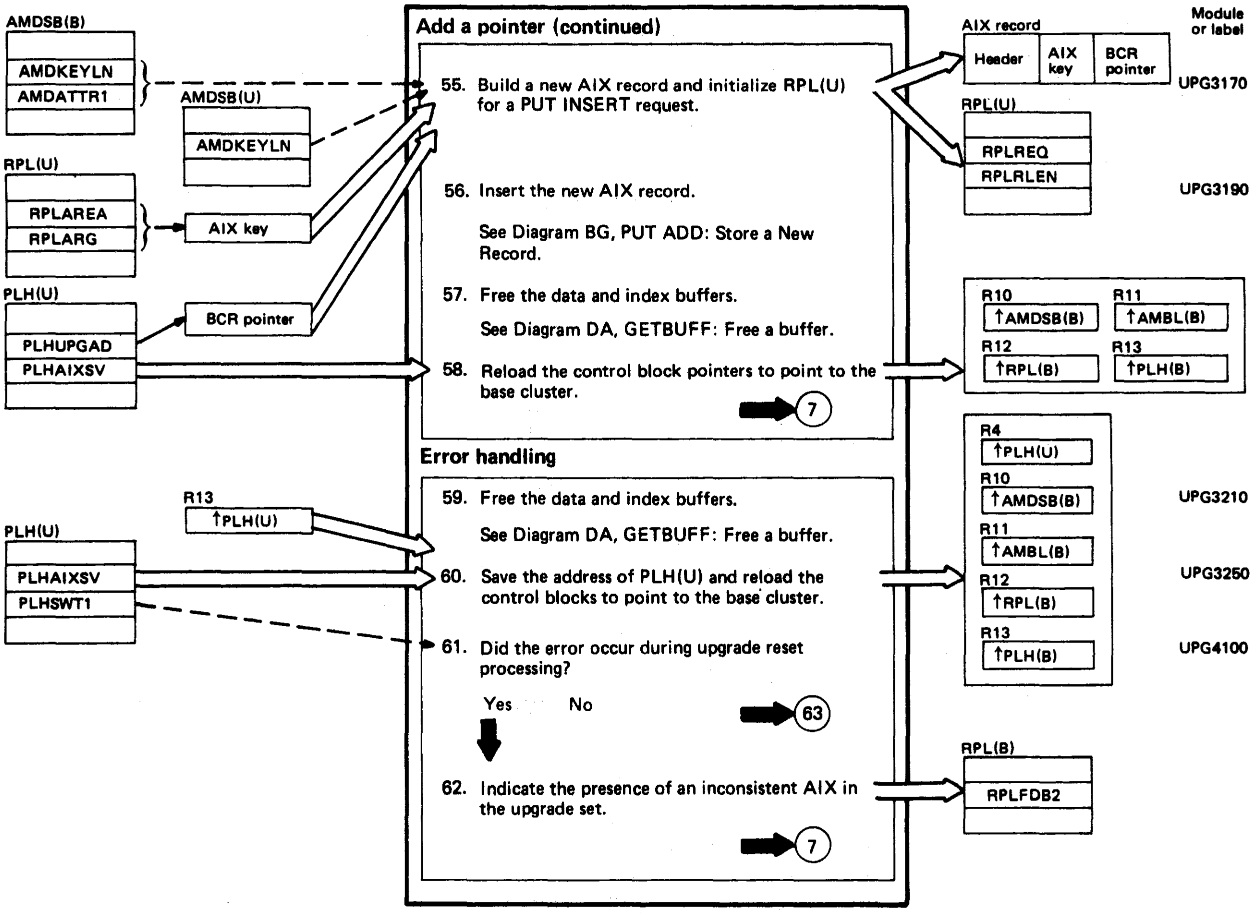
52. Calculate the new record length.

53. Is the record length greater than the permissible maximum, or is the maximum number of pointers exceeded? (32, 767)
No Yes → 59

54. Write back the updated record.

See Diagram BH, PUT UPDATE: Store an updated record.

Diagram BD8. Alternate Index Upgrade



Add a pointer (continued)

55. Build a new AIX record and initialize RPL(U) for a PUT INSERT request.

56. Insert the new AIX record.

    See Diagram BG, PUT ADD: Store a New Record.

57. Free the data and index buffers.

    See Diagram DA, GETBUFF: Free a buffer.

58. Reload the control block pointers to point to the base cluster. → 7

Error handling

59. Free the data and index buffers.

    See Diagram DA, GETBUFF: Free a buffer.

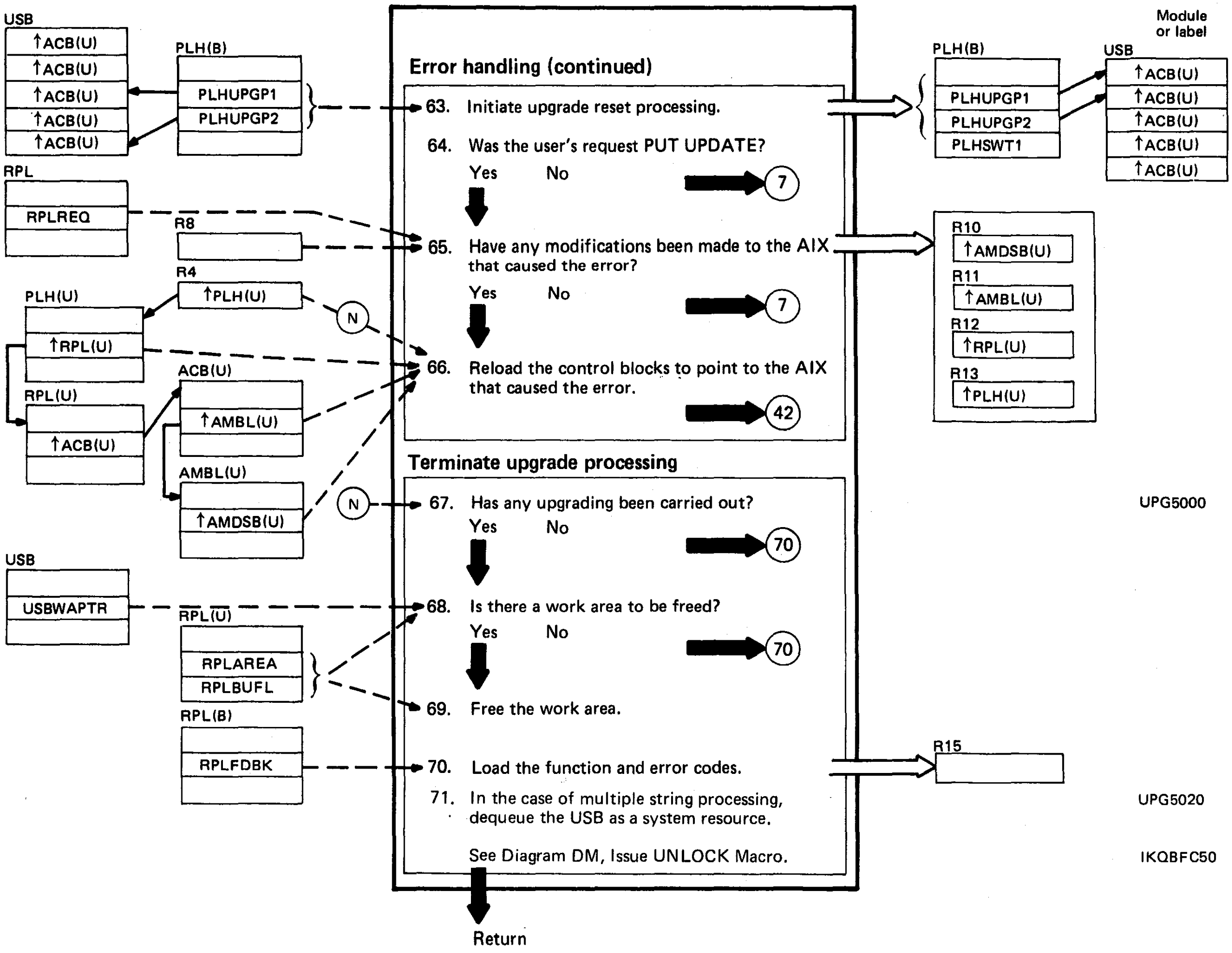
60. Save the address of PLH(U) and reload the control blocks to point to the base cluster.

61. Did the error occur during upgrade reset processing?

    Yes ↓ No → 63

62. Indicate the presence of an inconsistent AIX in the upgrade set. → 7

Diagram BD9. Alternate Index Upgrade



Error handling (continued)

63. Initiate upgrade reset processing.

64. Was the user's request PUT UPDATE?
Yes No → 7

65. Have any modifications been made to the AIX that caused the error?
Yes No → 7

66. Reload the control blocks to point to the AIX that caused the error. → 42

Terminate upgrade processing

67. Has any upgrading been carried out?
Yes No → 70

68. Is there a work area to be freed?
Yes No → 70

69. Free the work area.

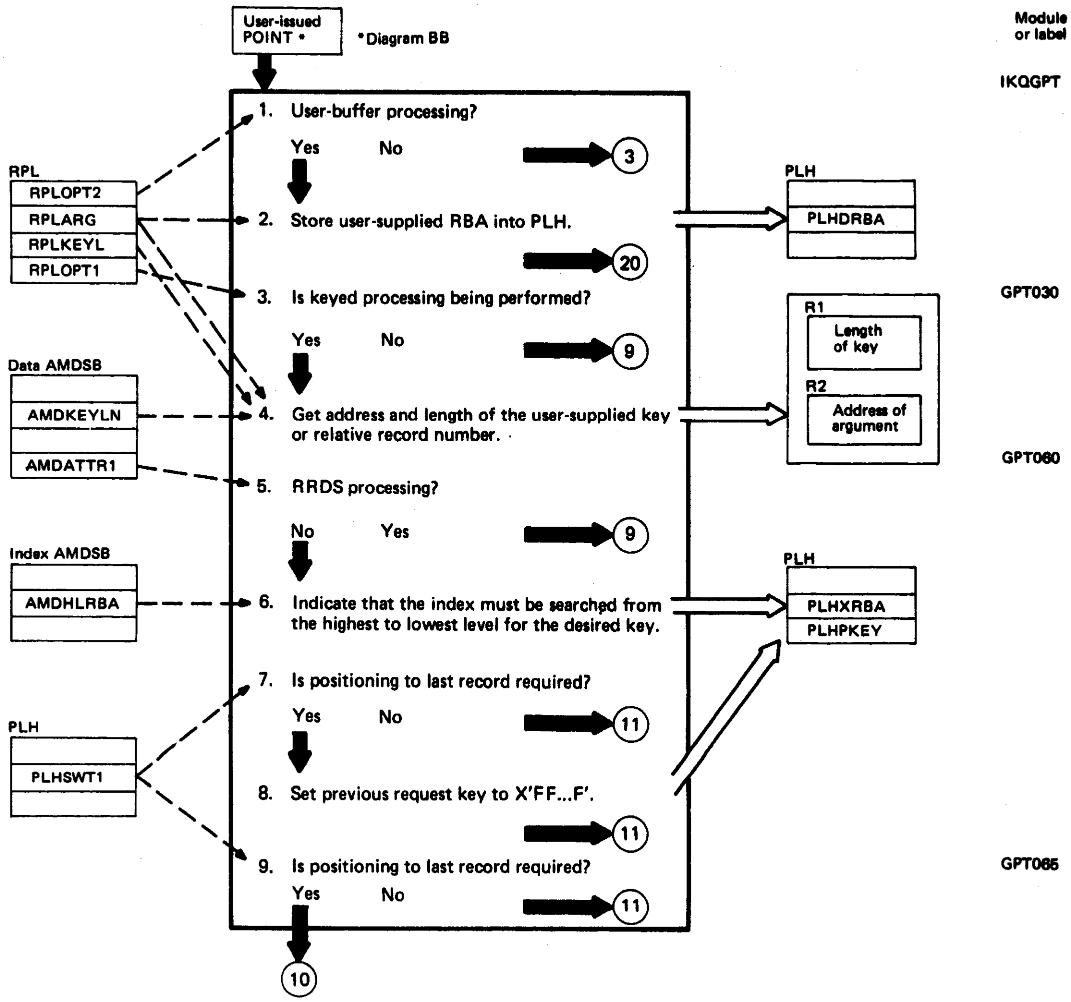
70. Load the function and error codes.

71. In the case of multiple string processing, dequeue the USB as a system resource.

See Diagram DM, Issue UNLOCK Macro.

Return

Diagram BE1. POINT: Position VSAM Data Record



1. User-buffer processing?
2. Store user-supplied RBA into PLH.
3. Is keyed processing being performed?
4. Get address and length of the user-supplied key or relative record number.
5. RRDS processing?
6. Indicate that the index must be searchpd from the highest to lowest level for the desired key.
7. Is positioning to last record required?
8. Set previous request key to X'FF...F'.
9. Is positioning to last record required?

Notes for Diagram BE1

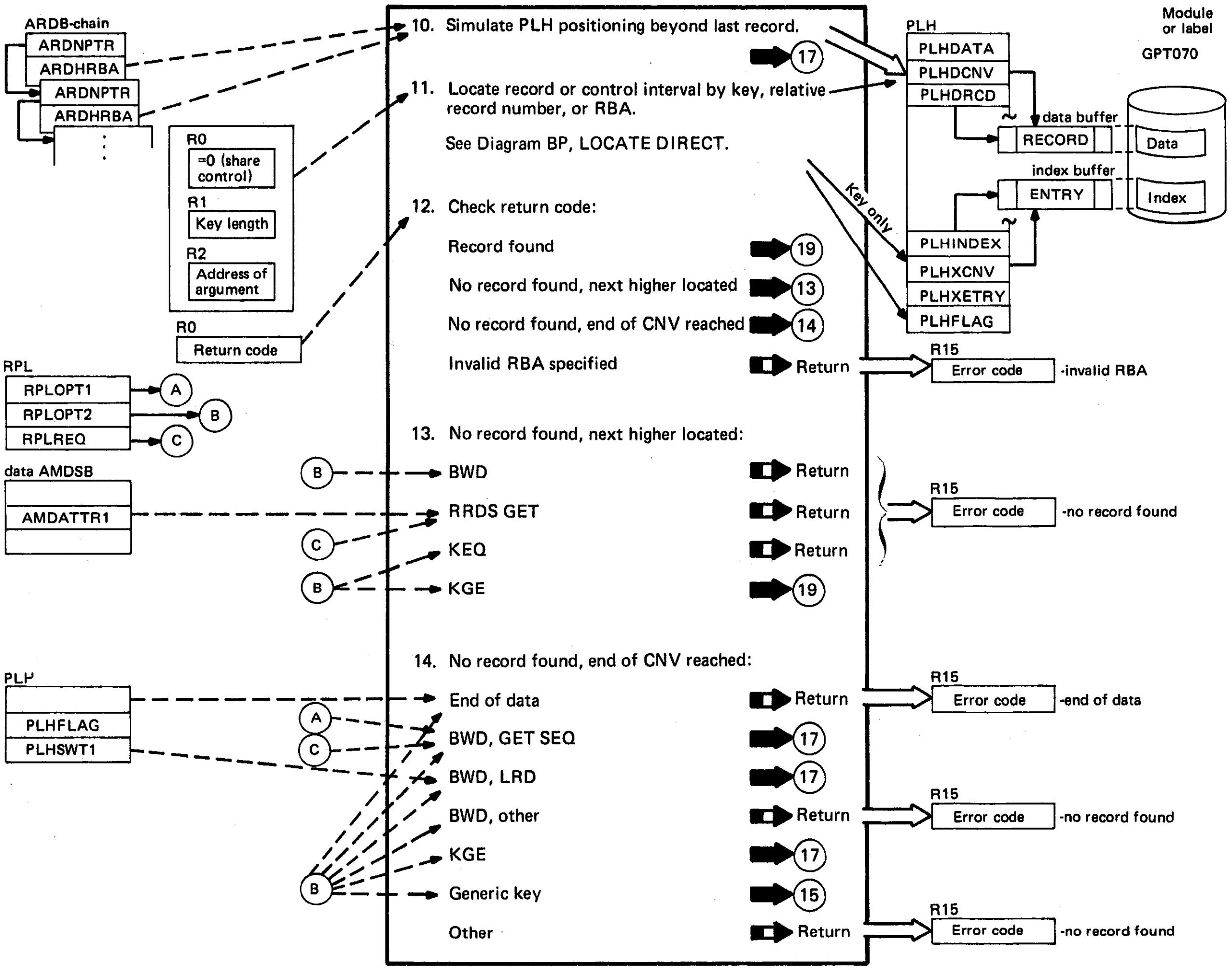
4. The user-supplied argument is a key for keyed processing or a 4-byte relative record number for relative record processing.

7.-9. Positioning to last record is required for backward processing with the option LRD.

8. For keyed processing of a KSDS, positioning to the last record is achieved by searching the index for the maximum possible key (X'FF...FF').

9.-10. For addressed or relative record processing, positioning past the last record is first carried out, using the high water mark of the highest key range as an RBA. The last record is then found by using LOCATE PREVIOUS.

Diagram BE2. POINT: Position VSAM Data Record



10. Simulate PLH positioning beyond last record. 17

11. Locate record or control interval by key, relative record number, or RBA.

See Diagram BP, LOCATE DIRECT.

12. Check return code:

Record found 19

No record found, next higher located 13

No record found, end of CNV reached 14

Invalid RBA specified Return

13. No record found, next higher located:

BWD Return

RRDS GET Return

KEQ Return

KGE 19

14. No record found, end of CNV reached:

End of data Return

BWD, GET SEQ 17

BWD, LRD 17

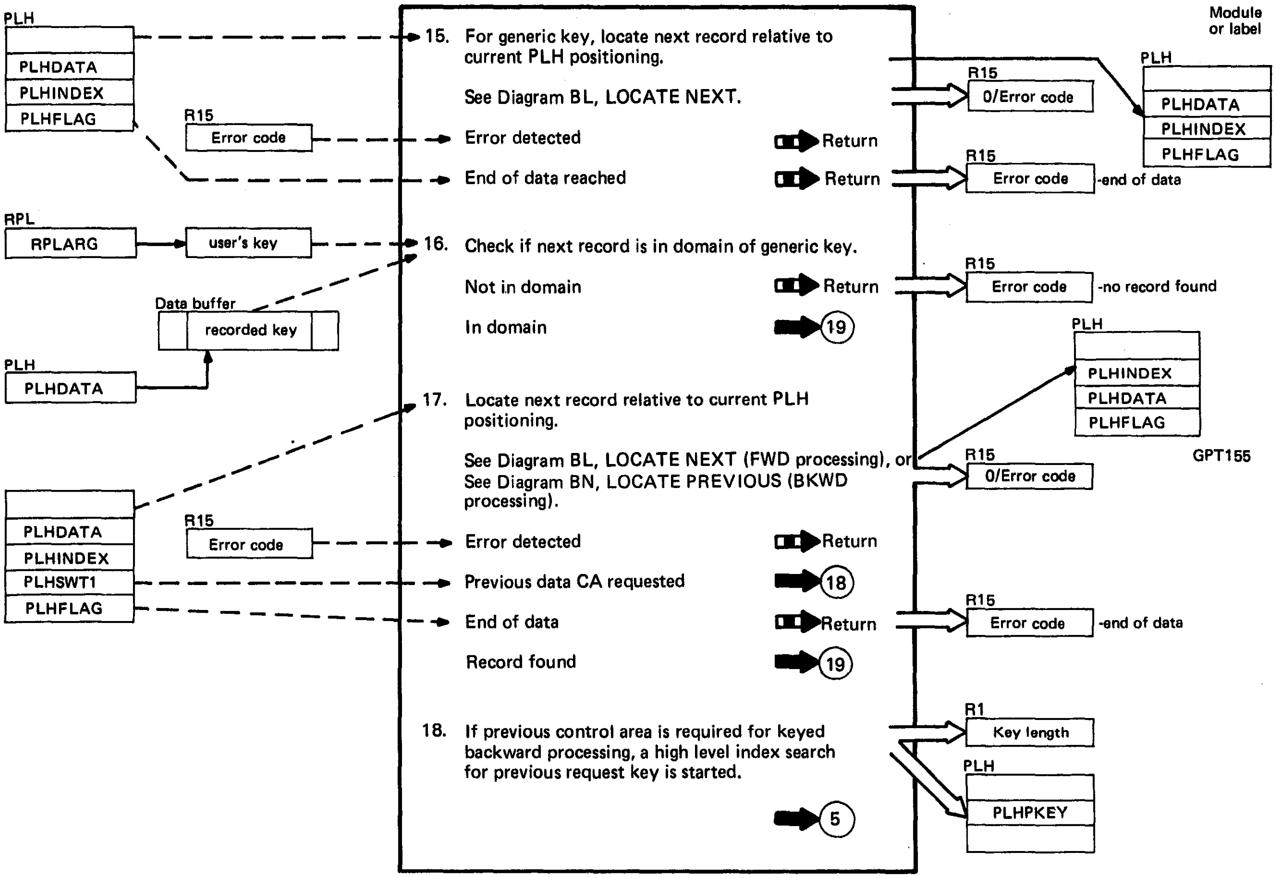
BWD, other Return

KGE 17

Generic key 15

Other Return

Diagram BE3. POINT: Position VSAM Data Record



15. For generic key, locate next record relative to current PLH positioning.

See Diagram BL, LOCATE NEXT.

Error detected — Return

End of data reached — Return

16. Check if next record is in domain of generic key.

Not in domain — Return

In domain — 19

17. Locate next record relative to current PLH positioning.

See Diagram BL, LOCATE NEXT (FWD processing), or See Diagram BN, LOCATE PREVIOUS (BKWD processing).

Error detected — Return

Previous data CA requested — 18

End of data — Return

Record found — 19

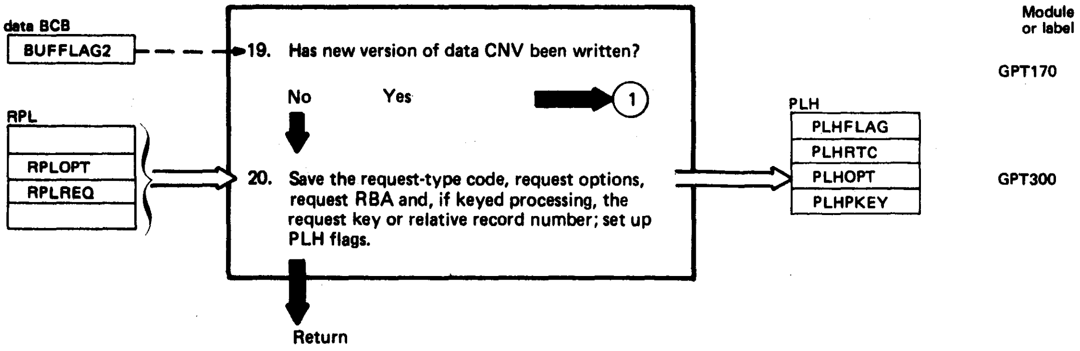
18. If previous control area is required for keyed backward processing, a high level index search for previous request key is started.

5

Notes for Diagram BE3

17.-18. During keyed backward processing of a KSDS, LOCATE PREVIOUS can step backwards only until it reached the beginning of a sequence set record, which corresponds to the start of a data CA. LOCATE PREVIOUS then returns a "previous data CA required" condition, and stores the lowest key of the current CA in the previous request key field. LOCATE DIRECT and INDEX SEARCH will then carry out a top-down search and transition to the previous sequence set record.

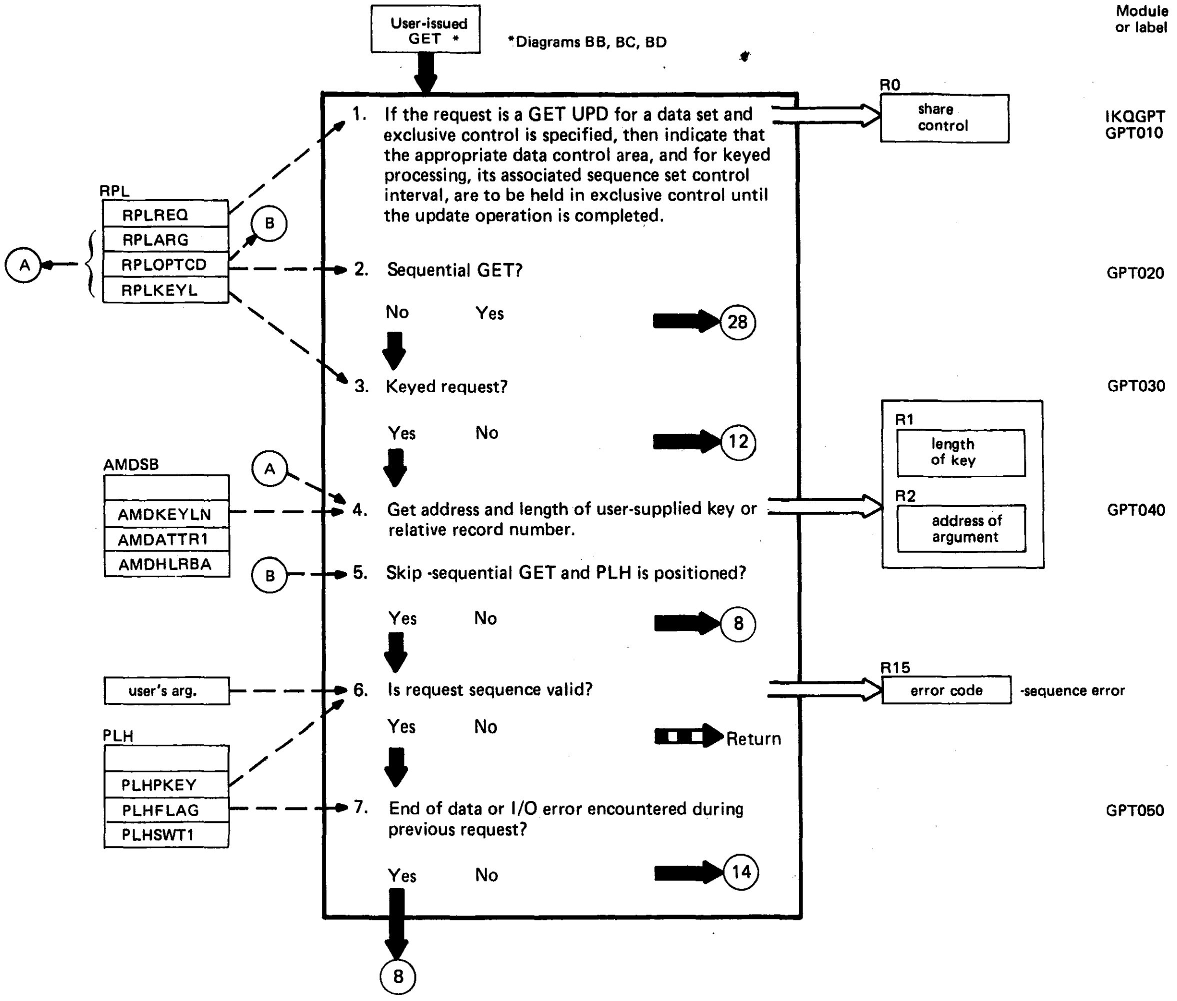
Diagram BE4. POINT: Position VSAM Data Record



19. Has new version of data CNV been written?

20. Save the request-type code, request options, request RBA and, if keyed processing, the request key or relative record number; set up PLH flags.

Diagram BF1. GET: Retrieve a Record



1. If the request is a GET UPD for a data set and exclusive control is specified, then indicate that the appropriate data control area, and for keyed processing, its associated sequence set control interval, are to be held in exclusive control until the update operation is completed.
2. Sequential GET?
3. Keyed request?
4. Get address and length of user-supplied key or relative record number.
5. Skip -sequential GET and PLH is positioned?
6. Is request sequence valid?
7. End of data or I/O error encountered during previous request?

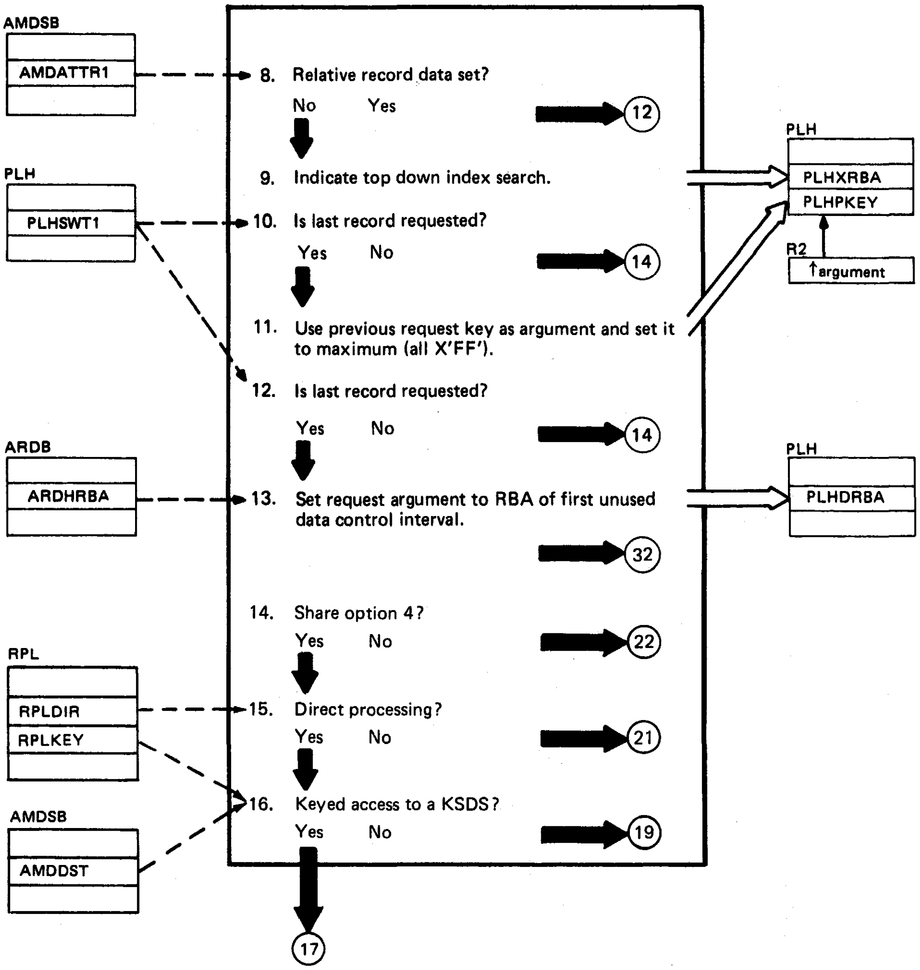
Notes for Diagram BF1

2. A direct or skip sequential GET does not require prepositioning; it implicitly positions to the correct record or control interval before transferring it to the user's area.

4. The key is supplied as argument. The length is the full key length (in AMDSB) for FKS and the KEYLEN supplied by the user for a generic key.

6. For a positioned SKP request, the given key has to be in ascending key sequence with respect to the previous request.

Diagram BF2. GET: Retrieve a Record



8. Relative record data set?

9. Indicate top down index search.

10. Is last record requested?

11. Use previous request key as argument and set it to maximum (all X'FF').

12. Is last record requested?

13. Set request argument to RBA of first unused data control interval.

14. Share option 4?

15. Direct processing?

16. Keyed access to a KSDS?

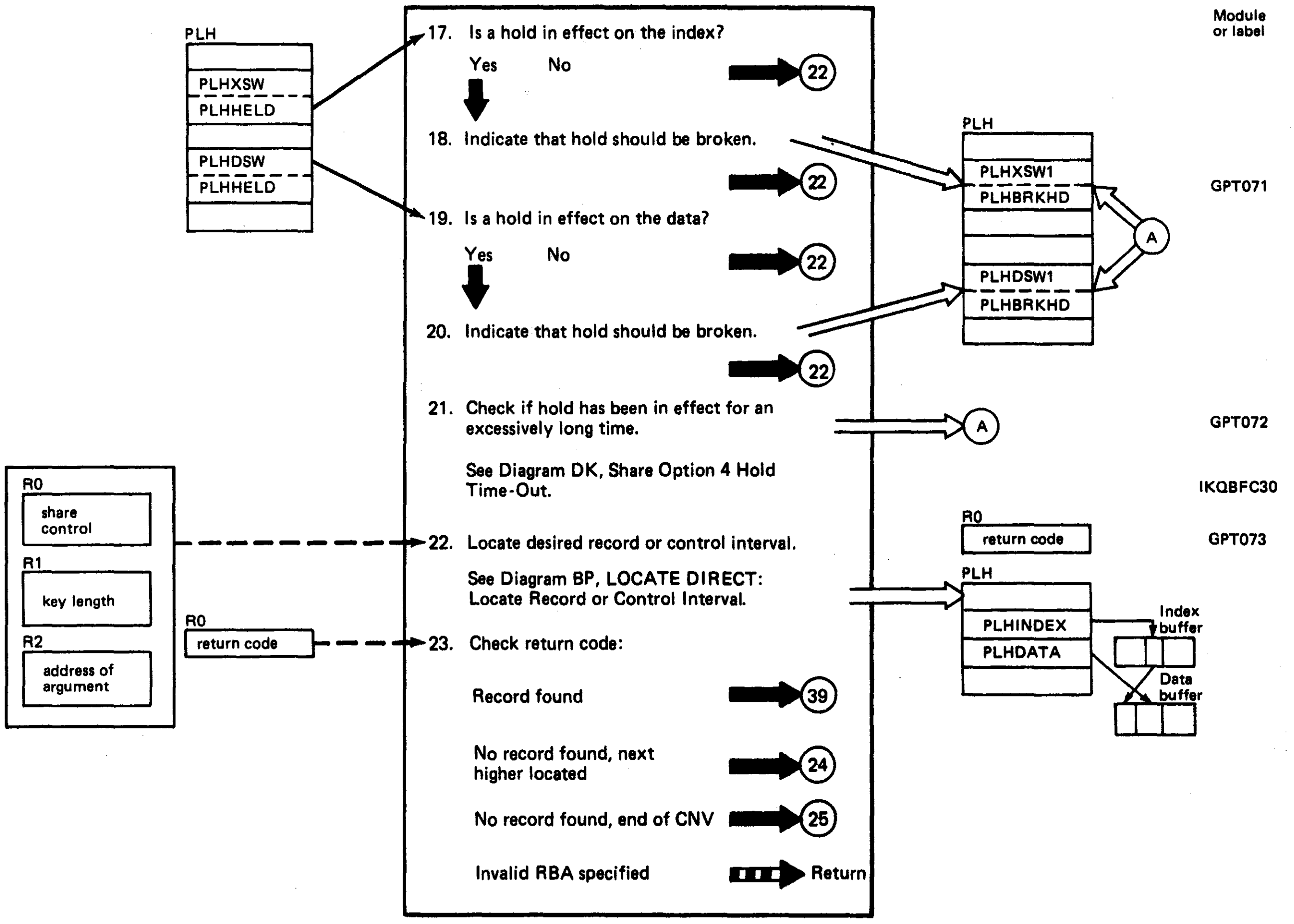
Module or label

GPT060

GPT065

GPT070

Diagram BF3. GET: Retrieve a Record



17. Is a hold in effect on the index?

18. Indicate that hold should be broken.

19. Is a hold in effect on the data?

20. Indicate that hold should be broken.

21. Check if hold has been in effect for an excessively long time.

See Diagram DK, Share Option 4 Hold Time-Out.

22. Locate desired record or control interval.

See Diagram BP, LOCATE DIRECT: Locate Record or Control Interval.

23. Check return code:

Record found

No record found, next higher located

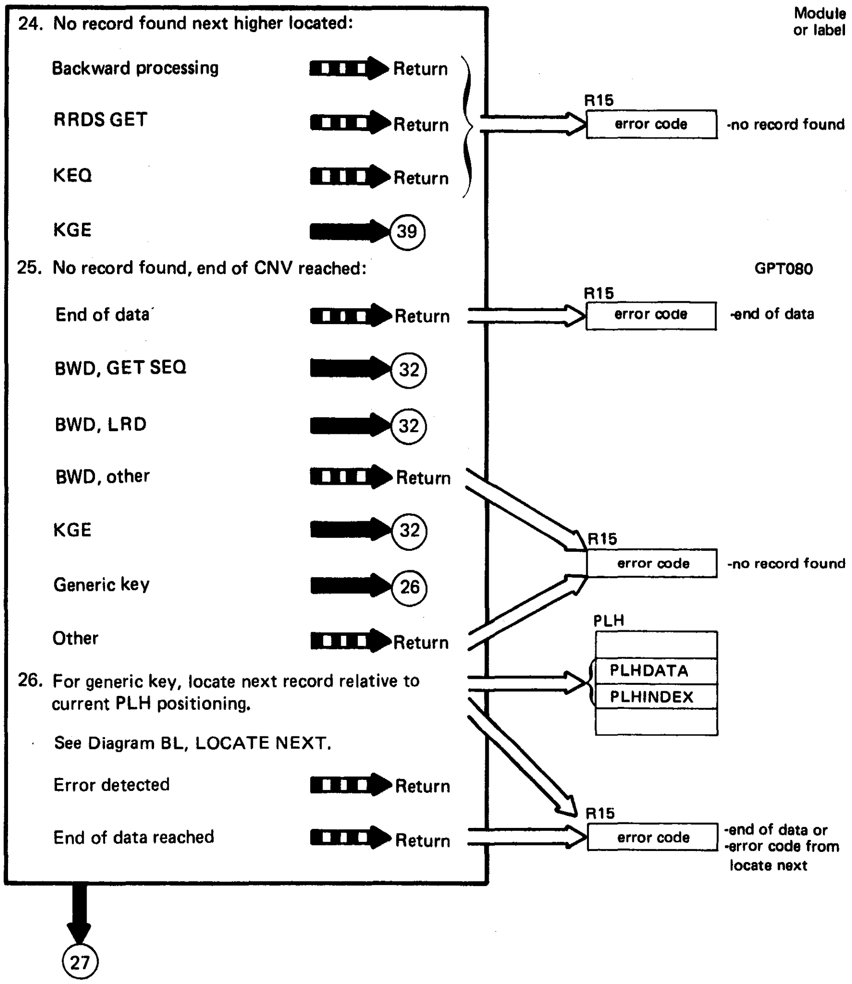
No record found, end of CNV

Invalid RBA specified

Notes for Diagram BF3

22. For a keyed request for KSDS other than a positioned SKP request, the search is top-down; for a positioned SKP request, the searcg is generally horizontal, starting with the sequence set control interval for the previous request. If the previous request had an I/O error or an end-of-data condition, however, the index search is top-down.

Diagram BF4. GET: Retrieve a Record



24. No record found next higher located:

Backward processing — Return

RRDS GET — Return

KEQ — Return

KGE — 39

25. No record found, end of CNV reached:

End of data — Return

BWD, GET SEQ — 32

BWD, LRD — 32

BWD, other — Return

KGE — 32

Generic key — 26

Other — Return

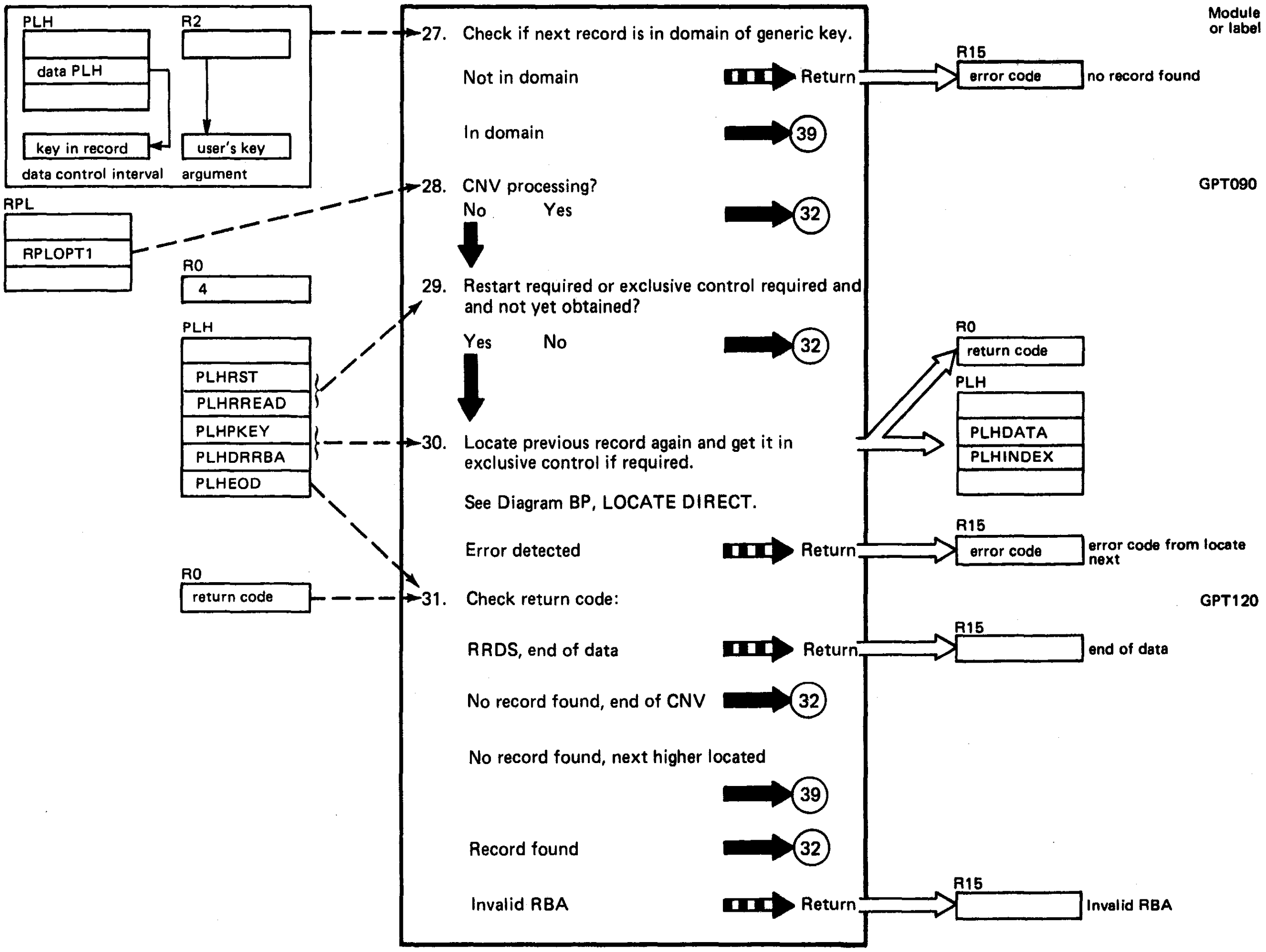
26. For generic key, locate next record relative to current PLH positioning.

· See Diagram BL, LOCATE NEXT.

Error detected — Return

End of data reached — Return

Diagram BF5. GET: Retrieve a Record



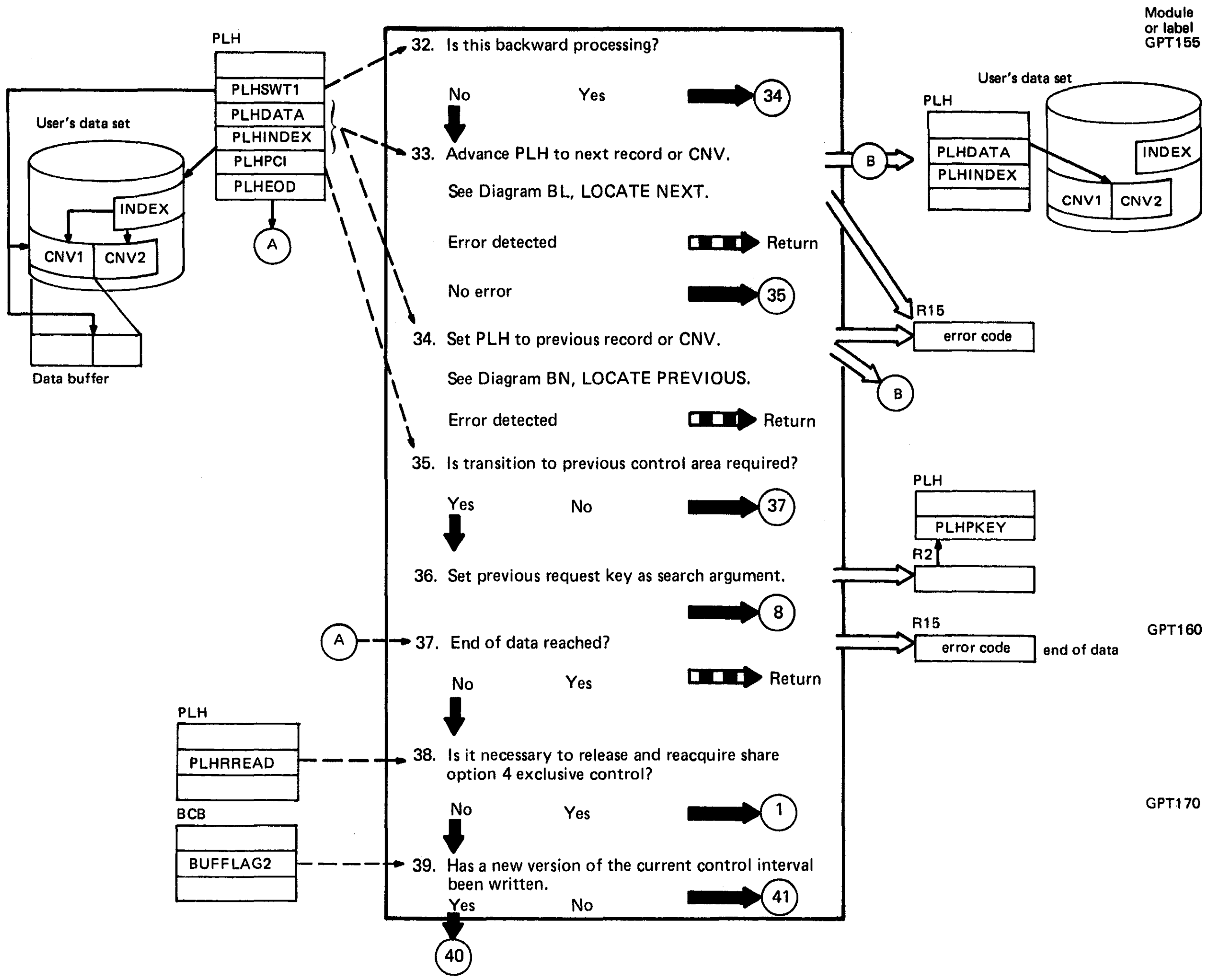
Notes for Diagram BF5

29. The restart flag (PLHRST) is set if an error was detected on the preceding request, or if another string has written either the data or index CI used to satisfy the previous request.

    The reread flag (PLHRREAD) is set if another string has written the data CI used to satisfy the request, or if a share option 4 hold is to be released and reacquired during backward processing. If PLHRREAD is set, a top-down index search is not required.

    (Repositioning can be accomplished by doing a search only at the sequence set level.)

Diagram BF6. GET: Retrieve a Record



32. Is this backward processing?

No — Yes → 34

33. Advance PLH to next record or CNV.

See Diagram BL, LOCATE NEXT.

Error detected → Return

No error → 35

34. Set PLH to previous record or CNV.

See Diagram BN, LOCATE PREVIOUS.

Error detected → Return

35. Is transition to previous control area required?

Yes — No → 37

36. Set previous request key as search argument.

→ 8

37. End of data reached?

No — Yes → Return

38. Is it necessary to release and reacquire share option 4 exclusive control?

No — Yes → 1

39. Has a new version of the current control interval been written.

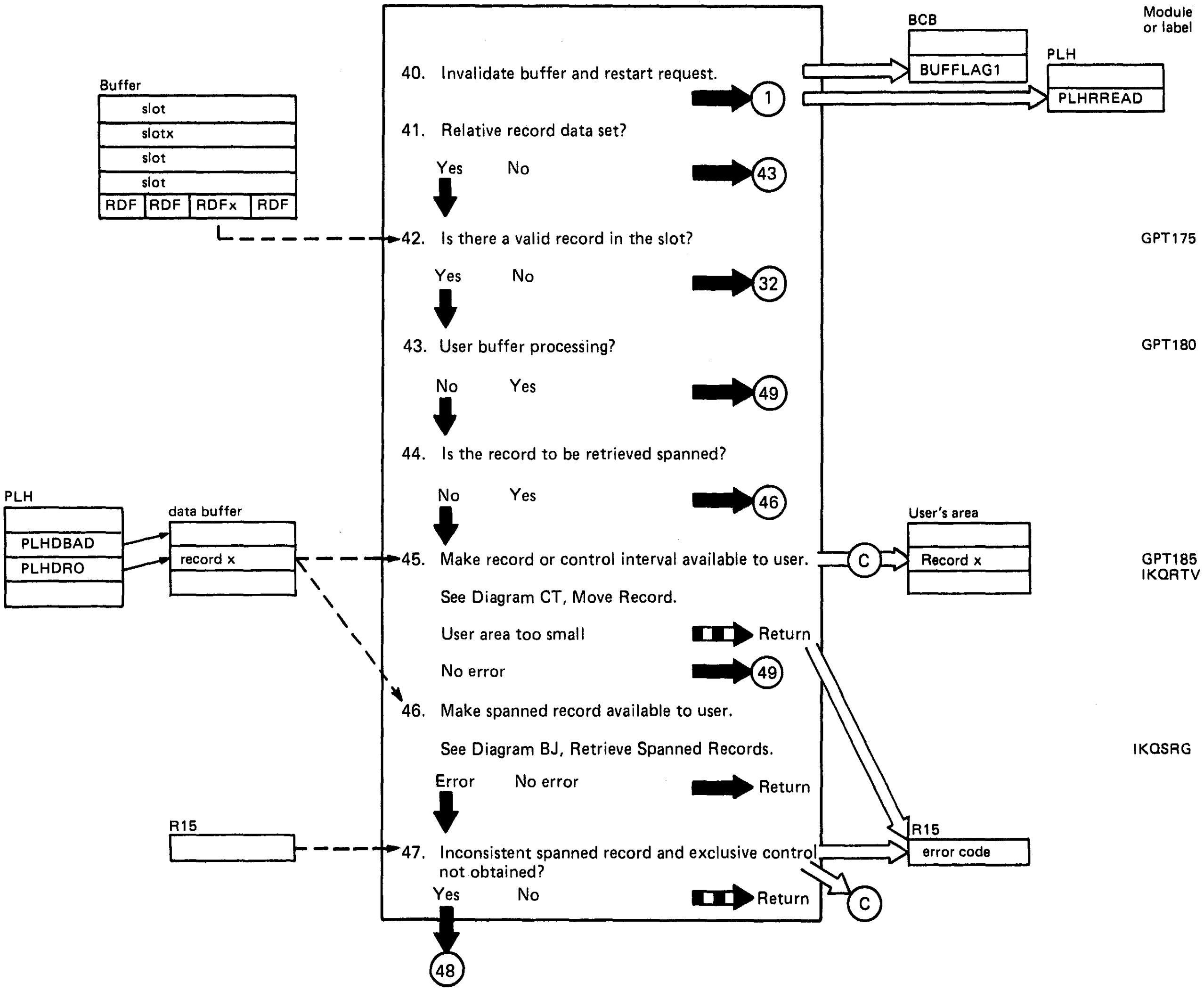
Yes → 40 — No → 41

Notes for Diagram BF6

35.-36. During keyed backward processing of a KSDS, LOCATE PREVIOUS can step backwards only until it reaches the beginning of a sequence set record, which corresponds to the start of a data CA. LOCATE PREVIOUS then references a previous data CA required condition, and stores the lowest key of the current CA in the previous request key field. LOCATE DIRECT and INDEX SEARCH will then carry out a top-down search and transition to the previous sequence set record.

38. The PLHRREAD may have been set by IKQLCP to indicate that share option 4 exclusive control has been held for an excessively long time and should be released and reacquired.

Diagram BF7. GET: Retrieve a Record



Module or label
BCB
BUFFLAG1
PLH
PLHRREAD
Buffer
slot
slotx
slot
slot
RDF
RDF
RDFx
RDF
40. Invalidate buffer and restart request.
1
41. Relative record data set?
Yes
No
43
42. Is there a valid record in the slot?
Yes
No
32
GPT175
43. User buffer processing?
No
Yes
49
GPT180
44. Is the record to be retrieved spanned?
No
Yes
46
PLH
PLHDBAD
PLHDRO
data buffer
record x
45. Make record or control interval available to user.
See Diagram CT, Move Record.
User area too small
Return
No error
49
C
User's area
Record x
GPT185
IKQRTV
46. Make spanned record available to user.
See Diagram BJ, Retrieve Spanned Records.
Error
No error
Return
IKQSRG
R15
47. Inconsistent spanned record and exclusive control not obtained?
Yes
No
Return
R15
error code
C
48

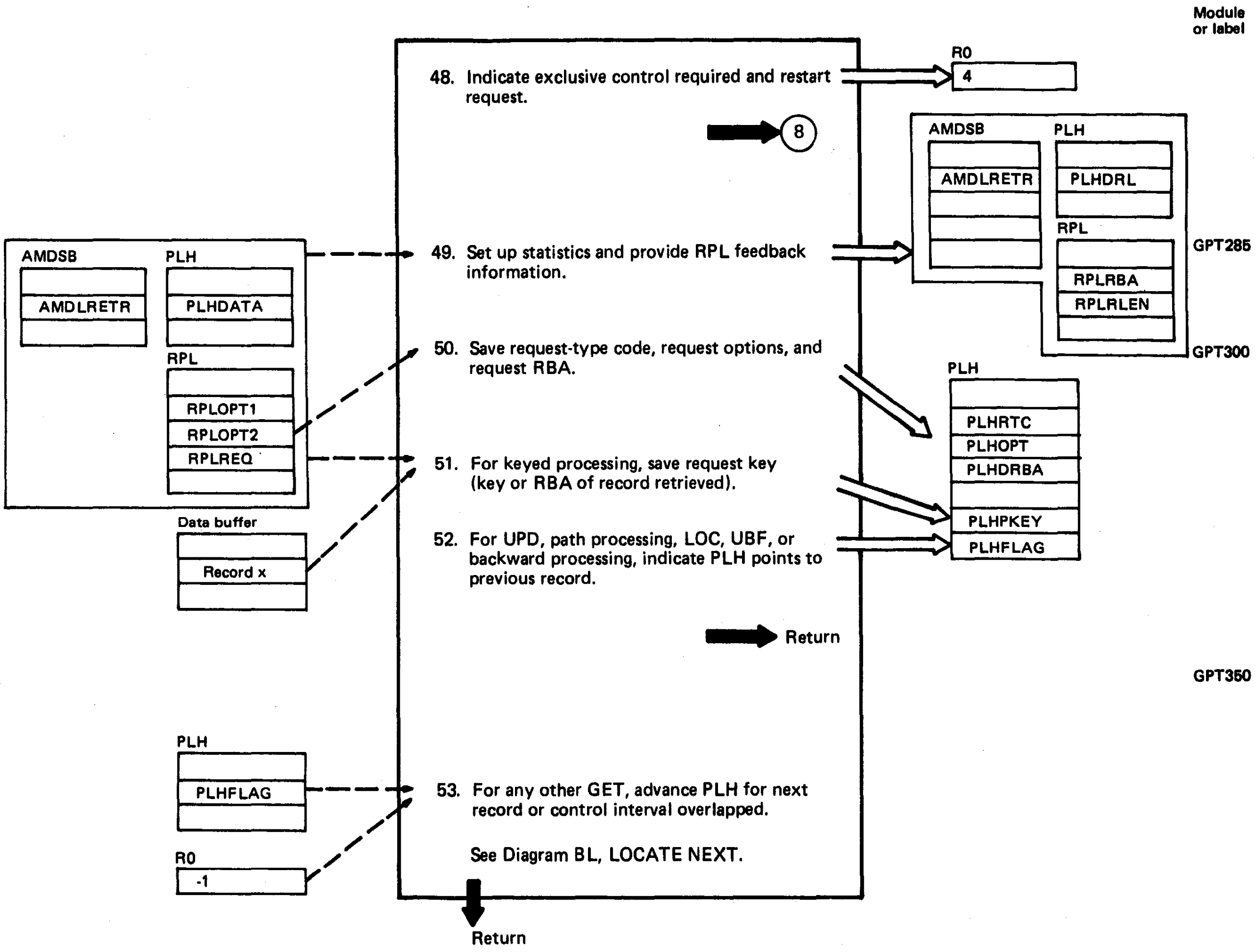
Notes for Diagram BF7

41. Control is also given to step 43 if CNV access is being used.

45. If move mode is specified and not user buffer processing, the data record or control interval is transferred from the VSAM buffer to the user's area.

46. Spanned record GET moves the first segment into the user's area, then reads the next control interval and moves the next segment into the user's area and so on until all segments have been moved.

Diagram BF8. GET: Retrieve a Record



48. Indicate exclusive control required and restart request.

49. Set up statistics and provide RPL feedback information.

50. Save request-type code, request options, and request RBA.

51. For keyed processing, save request key (key or RBA of record retrieved).

52. For UPD, path processing, LOC, UBF, or backward processing, indicate PLH points to previous record.

53. For any other GET, advance PLH for next record or control interval overlapped.

See Diagram BL, LOCATE NEXT.

Notes for Diagram BF8

49. The RBA of the transferred data record or control interval is provided in the RPL as well as the length of the record or control interval retrieved. The number of records retrieved is the statistics updated in the AMDSB.

53. VSAM will be positioned to the next control interval in an overlapped manner, that is, I/O operations (read ahead) are started but completion is not-waited for, in order to overlap the I/O operation with the user's processing of the record or control interval just retrieved, so as to improve throughout.

Diagram BG1. PUT ADD: Store a New Record

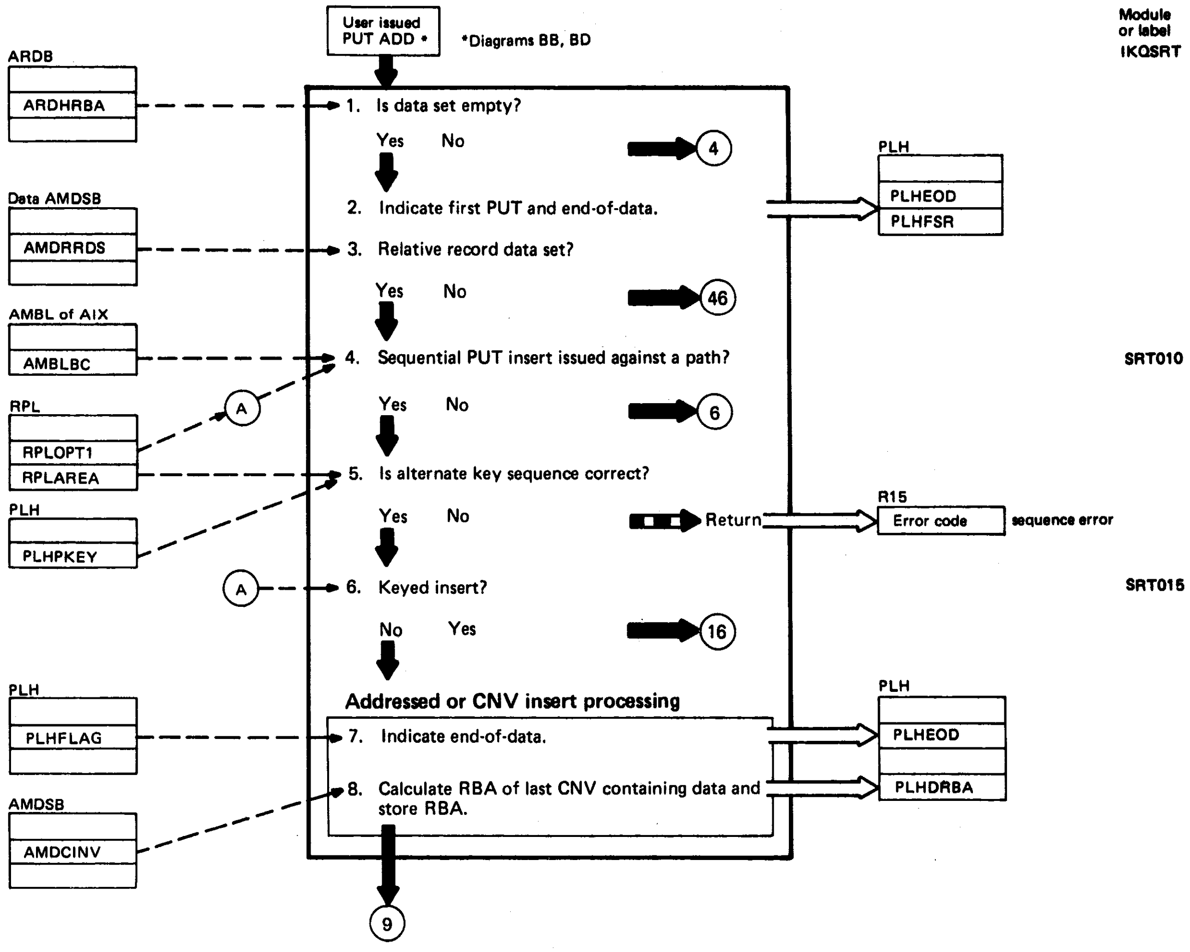


Diagram BG2. PUT ADD: Store a New Record

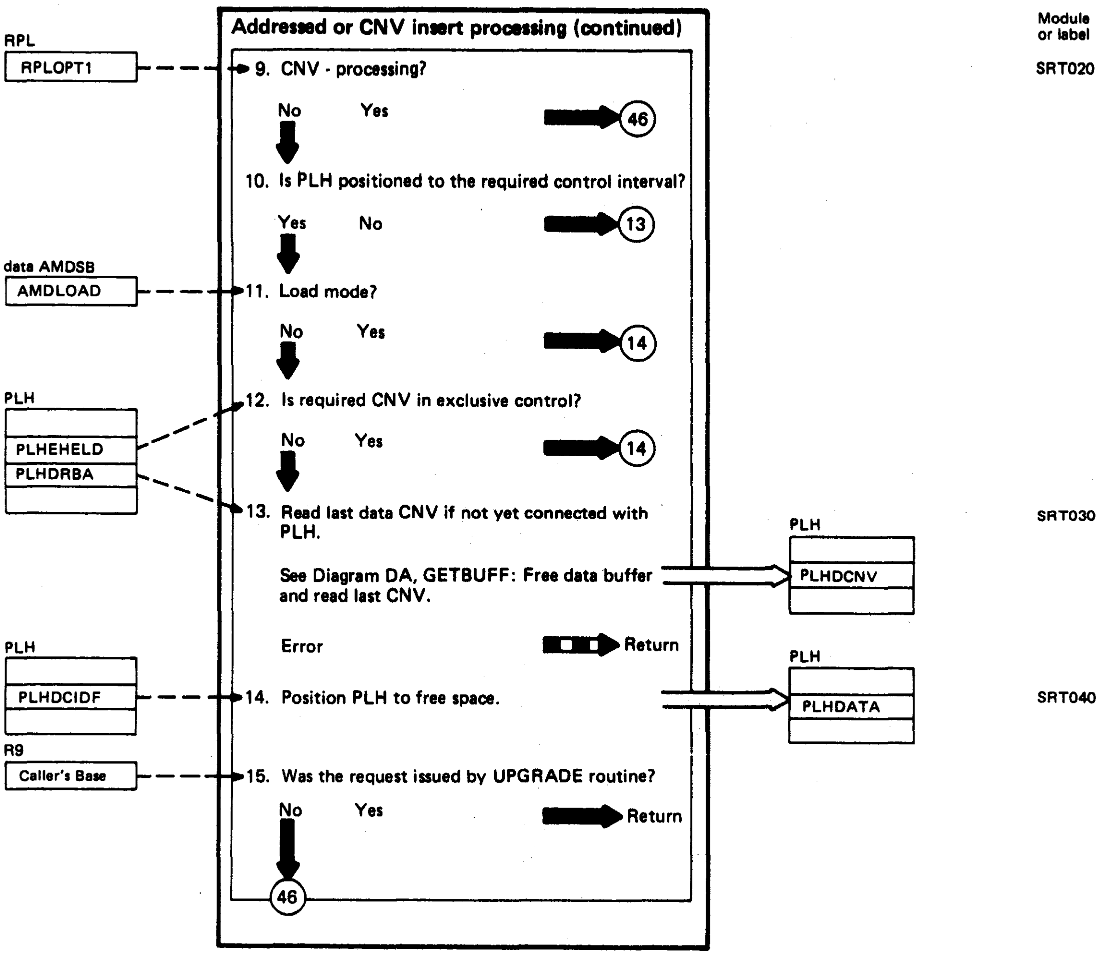
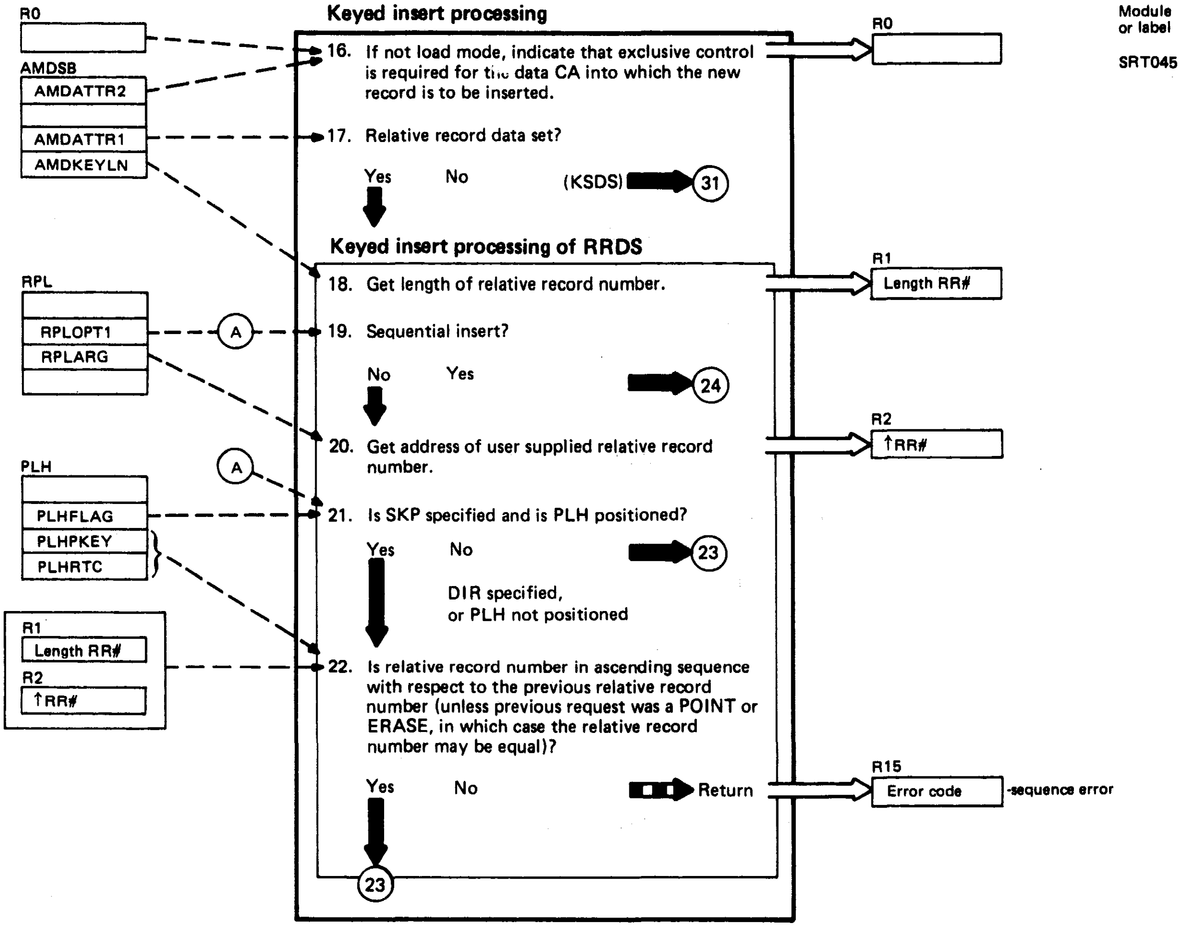

Addressed or CNV insert processing (continued)
Module or label
RPL
RPLOPT1
9. CNV - processing?
No
Yes
46
SRT020
10. Is PLH positioned to the required control interval?
Yes
No
13
data AMDSB
AMDLOAD
11. Load mode?
No
Yes
14
PLH
PLHEHELD
PLHDRBA
12. Is required CNV in exclusive control?
No
Yes
14
13. Read last data CNV if not yet connected with PLH.
SRT030
See Diagram DA, GETBUFF: Free data buffer and read last CNV.
PLH
PLHDCNV
Error
Return
PLH
PLHDCIDF
14. Position PLH to free space.
PLH
PLHDATA
SRT040
R9
Caller's Base
15. Was the request issued by UPGRADE routine?
No
Yes
Return
46

Diagram BG3. PUT ADD: Store a New Record



Keyed insert processing

16. If not load mode, indicate that exclusive control is required for the data CA into which the new record is to be inserted.

17. Relative record data set?

Keyed insert processing of RRDS

18. Get length of relative record number.

19. Sequential insert?

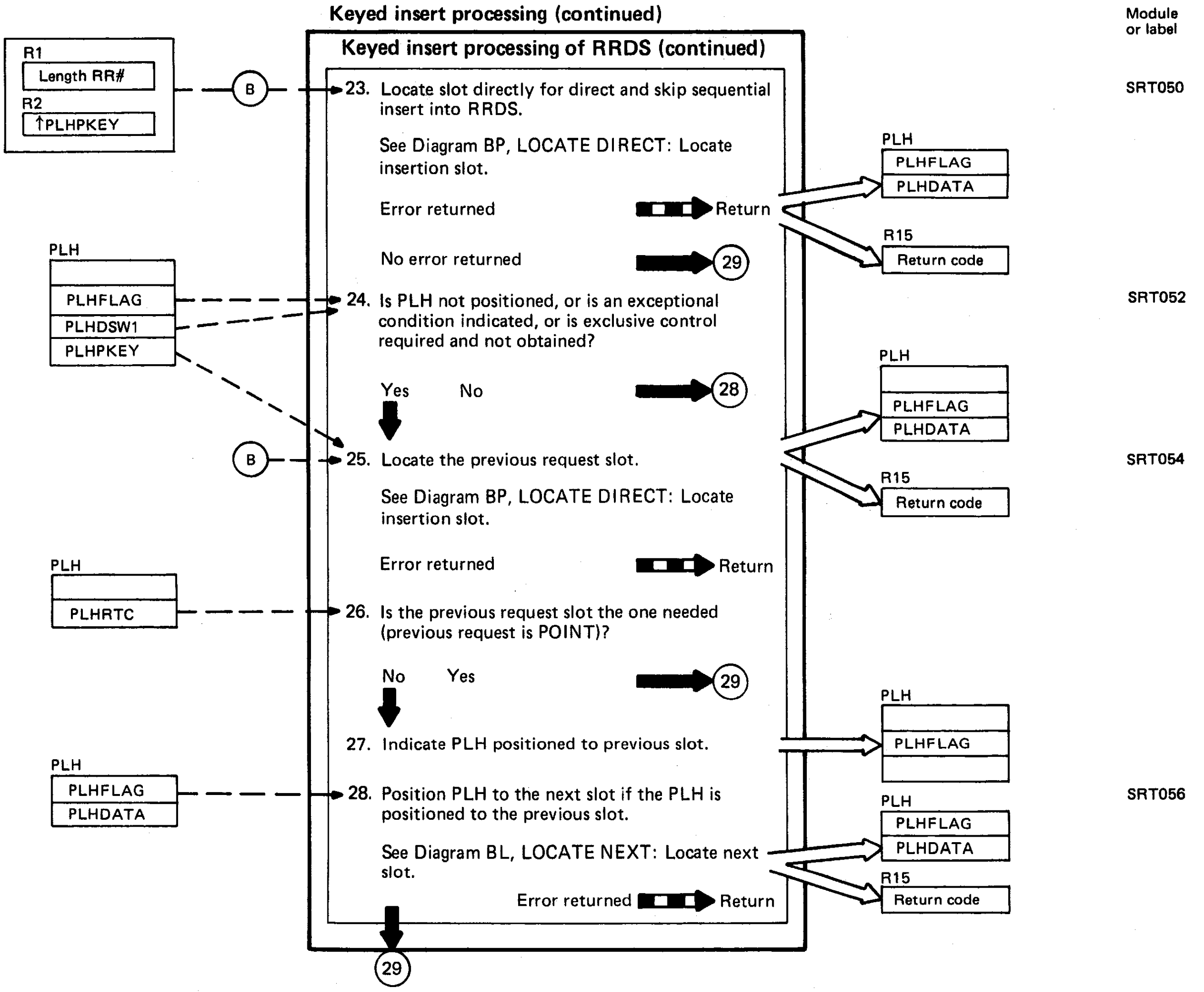
20. Get address of user supplied relative record number.

21. Is SKP specified and is PLH positioned?

DIR specified,
or PLH not positioned

22. Is relative record number in ascending sequence with respect to the previous relative record number (unless previous request was a POINT or ERASE, in which case the relative record number may be equal)?

Diagram BG4. PUT ADD: Store a New Record



Keyed insert processing (continued)

Keyed insert processing of RRDS (continued)

23. Locate slot directly for direct and skip sequential insert into RRDS.

See Diagram BP, LOCATE DIRECT: Locate insertion slot.

Error returned → Return

No error returned → 29

24. Is PLH not positioned, or is an exceptional condition indicated, or is exclusive control required and not obtained?

Yes No → 28

25. Locate the previous request slot.

See Diagram BP, LOCATE DIRECT: Locate insertion slot.

Error returned → Return

26. Is the previous request slot the one needed (previous request is POINT)?

No Yes → 29

27. Indicate PLH positioned to previous slot.

28. Position PLH to the next slot if the PLH is positioned to the previous slot.

See Diagram BL, LOCATE NEXT: Locate next slot.

Error returned → Return

29

Diagram BG5. PUT ADD: Store a New Record

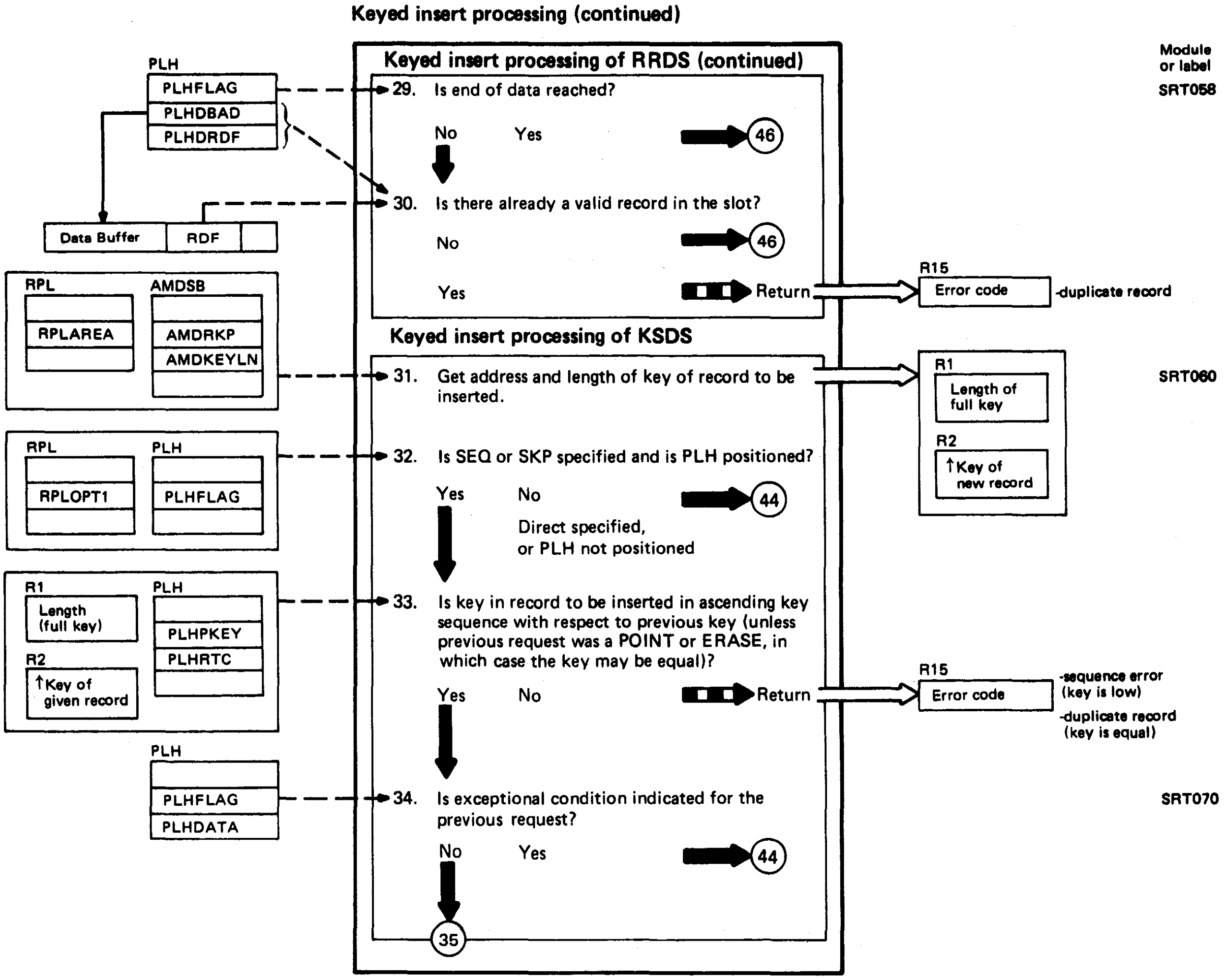
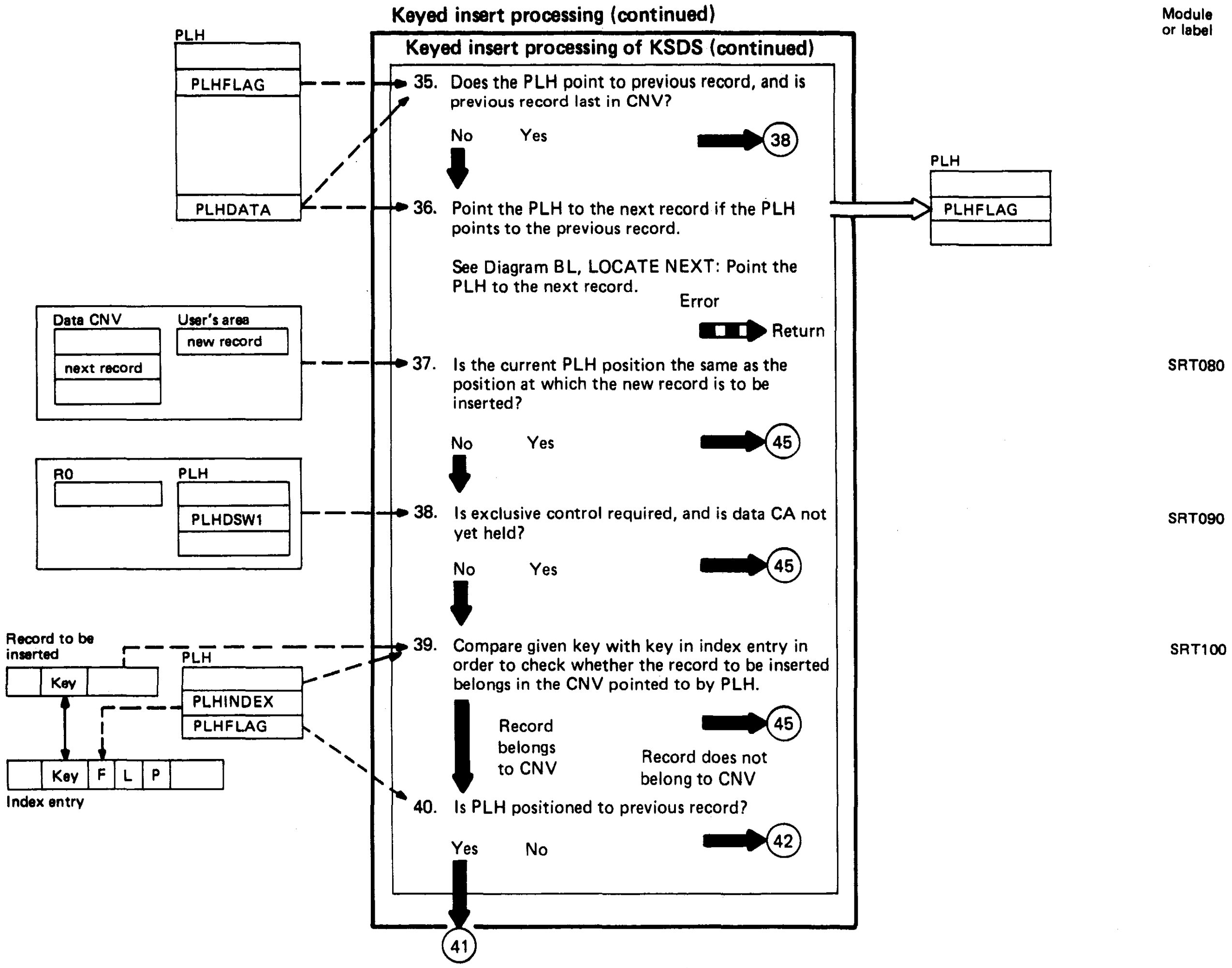

Keyed insert processing (continued)
Module or label
PLH
PLHFLAG
PLHDBAD
PLHDRDF
Keyed insert processing of RRDS (continued)
29. Is end of data reached?
No
Yes
46
SRT058
Data Buffer
RDF
30. Is there already a valid record in the slot?
No
46
Yes
Return
R15
Error code
-duplicate record
RPL
AMDSB
RPLAREA
AMDRKP
AMDKEYLN
Keyed insert processing of KSDS
31. Get address and length of key of record to be inserted.
R1
Length of full key
R2
↑Key of new record
SRT060
RPL
PLH
RPLOPT1
PLHFLAG
32. Is SEQ or SKP specified and is PLH positioned?
Yes
No
44
Direct specified, or PLH not positioned
R1
PLH
Length (full key)
PLHPKEY
PLHRTC
R2
↑Key of given record
33. Is key in record to be inserted in ascending key sequence with respect to previous key (unless previous request was a POINT or ERASE, in which case the key may be equal)?
Yes
No
Return
R15
Error code
-sequence error (key is low)
-duplicate record (key is equal)
PLH
PLHFLAG
PLHDATA
34. Is exceptional condition indicated for the previous request?
No
Yes
44
SRT070
35

Diagram BG6. PUT ADD: Store a New Record



Keyed insert processing (continued)

Keyed insert processing of KSDS (continued)

35. Does the PLH point to previous record, and is previous record last in CNV?

No Yes → 38

36. Point the PLH to the next record if the PLH points to the previous record.

See Diagram BL, LOCATE NEXT: Point the PLH to the next record.

Error → Return

37. Is the current PLH position the same as the position at which the new record is to be inserted?

No Yes → 45

38. Is exclusive control required, and is data CA not yet held?

No Yes → 45

39. Compare given key with key in index entry in order to check whether the record to be inserted belongs in the CNV pointed to by PLH.

Record belongs to CNV

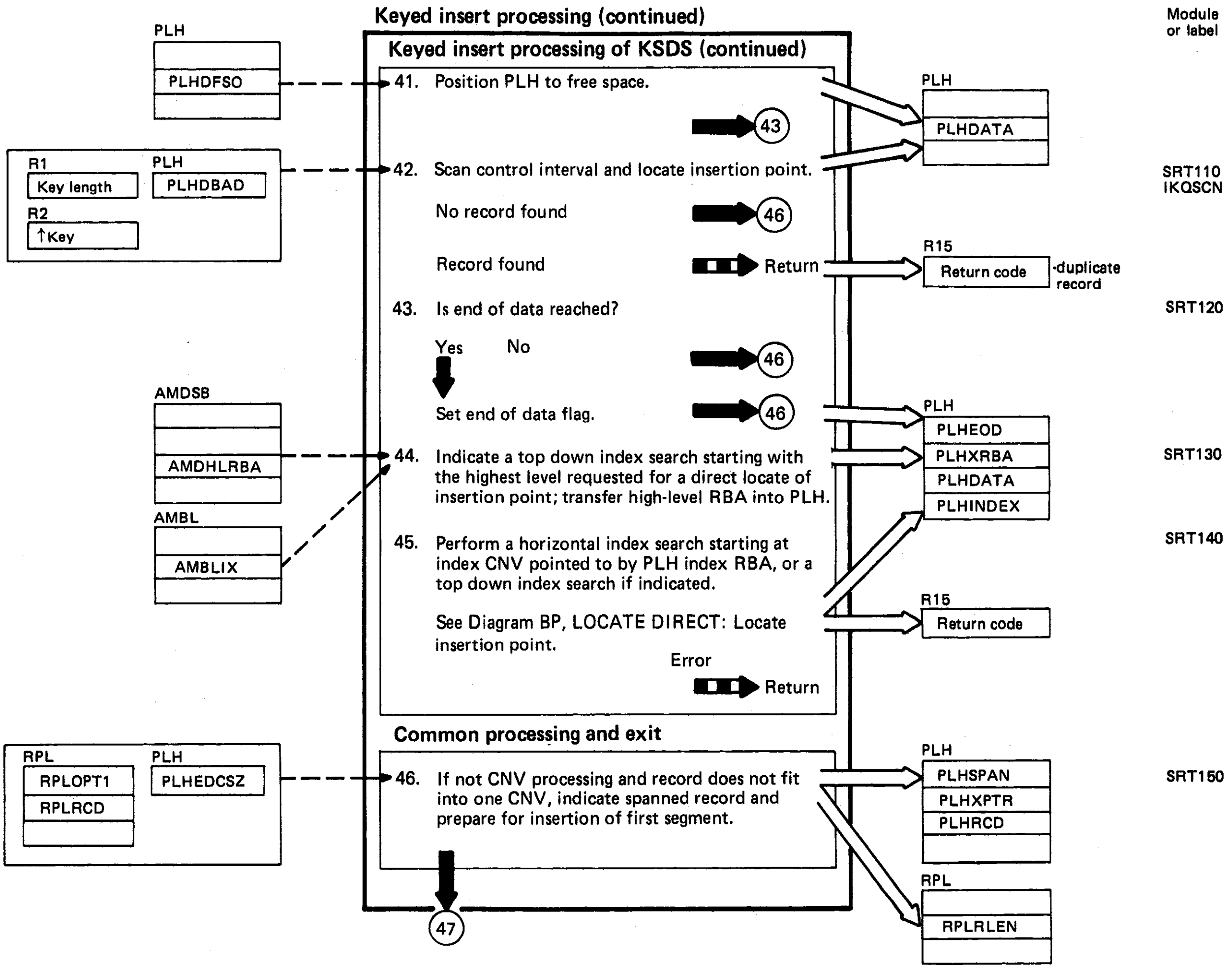
Record does not belong to CNV → 45

40. Is PLH positioned to previous record?

Yes No → 42

41

Diagram BG7. PUT ADD: Store a New Record



Keyed insert processing (continued)

Keyed insert processing of KSDS (continued)

41. Position PLH to free space.

42. Scan control interval and locate insertion point.

No record found

Record found — Return

43. Is end of data reached?

Yes No

Set end of data flag.

44. Indicate a top down index search starting with the highest level requested for a direct locate of insertion point; transfer high-level RBA into PLH.

45. Perform a horizontal index search starting at index CNV pointed to by PLH index RBA, or a top down index search if indicated.

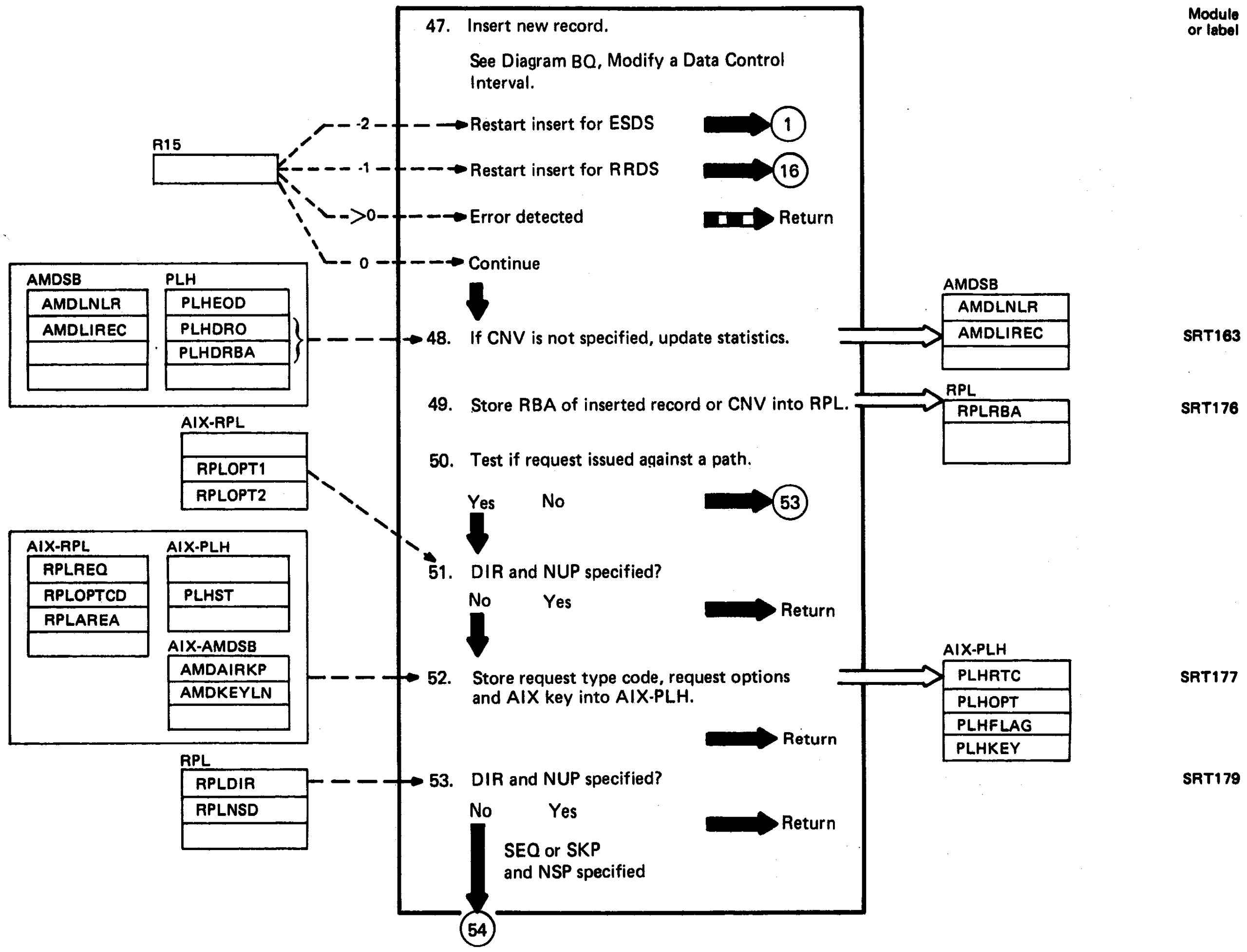
See Diagram BP, LOCATE DIRECT: Locate insertion point.

Error — Return

Common processing and exit

46. If not CNV processing and record does not fit into one CNV, indicate spanned record and prepare for insertion of first segment.

Diagram BG8. PUT ADD: Store a New Record

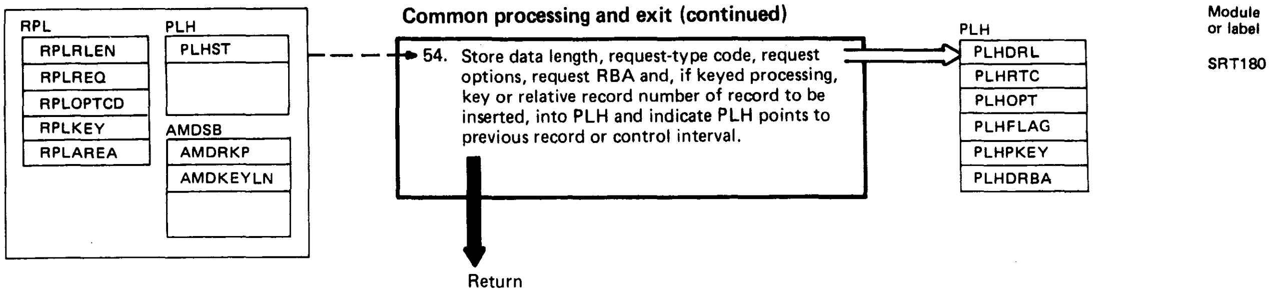

Module or label
47. Insert new record.
See Diagram BQ, Modify a Data Control Interval.
R15
-2 Restart insert for ESDS
1
-1 Restart insert for RRDS
16
>0 Error detected
Return
0 Continue
AMDSB
AMDLNLR
AMDLIREC
PLH
PLHEOD
PLHDRO
PLHDRBA
48. If CNV is not specified, update statistics.
AMDSB
AMDLNLR
AMDLIREC
SRT163
49. Store RBA of inserted record or CNV into RPL.
RPL
RPLRBA
SRT176
AIX-RPL
RPLOPT1
RPLOPT2
50. Test if request issued against a path.
Yes
No
53
AIX-RPL
RPLREQ
RPLOPTCD
RPLAREA
AIX-PLH
PLHST
51. DIR and NUP specified?
No
Yes
Return
AIX-AMDSB
AMDAIRKP
AMDKEYLN
52. Store request type code, request options and AIX key into AIX-PLH.
AIX-PLH
PLHRTC
PLHOPT
PLHFLAG
PLHKEY
SRT177
Return
RPL
RPLDIR
RPLNSD
53. DIR and NUP specified?
SRT179
No
Yes
Return
SEQ or SKP and NSP specified
54

Notes for Diagram BG8

47. For relative record processing, the Modify Data CNV routine may indicate that PUT ADD processing has to be restarted.

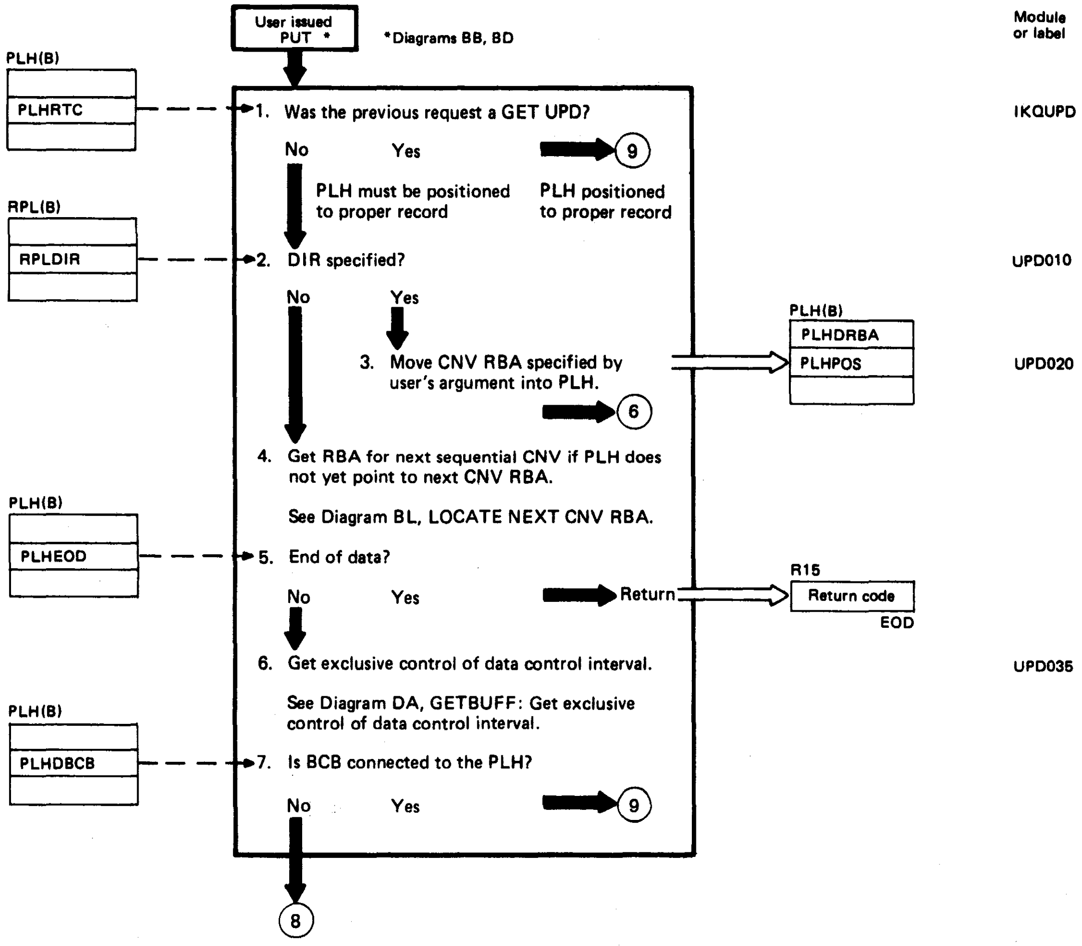
    This occurs when larger portions of the data space had to be preformatted, and the connection between the PLH and the target data CI was lost.

Diagram BG9. PUT ADD: Store a New Record



54. Store data length, request-type code, request options, request RBA and, if keyed processing, key or relative record number of record to be inserted, into PLH and indicate PLH points to previous record or control interval.

Diagram BH1. PUT UPDATE or ISAM-Issued PUT in LOCATE Mode: Store an Updated Record



1. Was the previous request a GET UPD?

No — PLH must be positioned to proper record

Yes — PLH positioned to proper record

2. DIR specified?

3. Move CNV RBA specified by user's argument into PLH.

4. Get RBA for next sequential CNV if PLH does not yet point to next CNV RBA.

See Diagram BL, LOCATE NEXT CNV RBA.

5. End of data?

6. Get exclusive control of data control interval.

See Diagram DA, GETBUFF: Get exclusive control of data control interval.

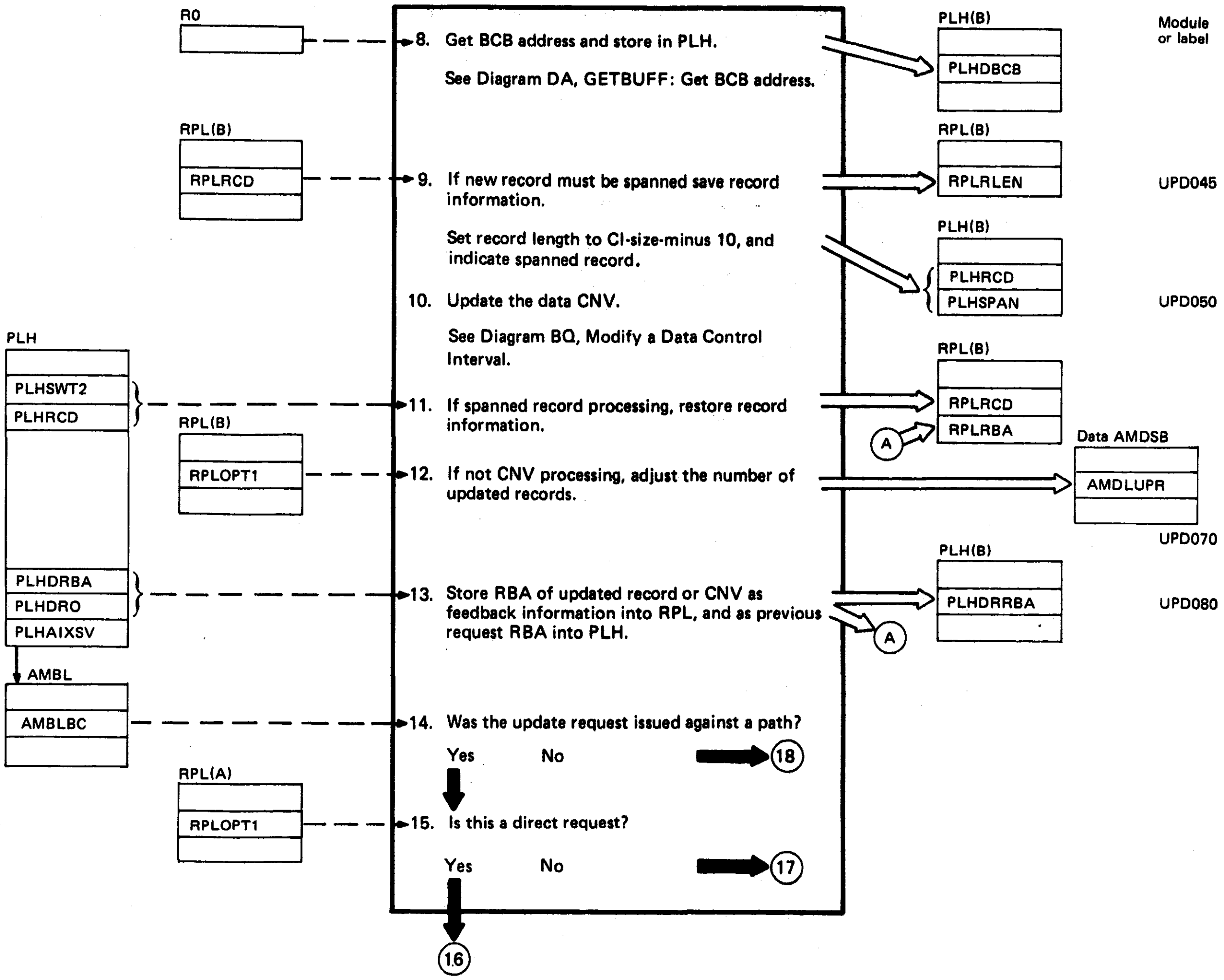
7. Is BCB connected to the PLH?

Notes for Diagram BH1

Notation:
PLH(A), etc.: Control block associated with AIX
PLH(B), etc.: Control block associated with base cluster.

1. Stand-alone updates (updates without previous GETs) are allowed for user buffer processing.

Diagram BH2. PUT UPDATE or ISAM-Issued PUT in LOCATE Mode: Store an Updated Record



8. Get BCB address and store in PLH.

   See Diagram DA, GETBUFF: Get BCB address.

9. If new record must be spanned save record information.

   Set record length to CI-size-minus 10, and indicate spanned record.

10. Update the data CNV.

    See Diagram BQ, Modify a Data Control Interval.

11. If spanned record processing, restore record information.

12. If not CNV processing, adjust the number of updated records.

13. Store RBA of updated record or CNV as feedback information into RPL, and as previous request RBA into PLH.

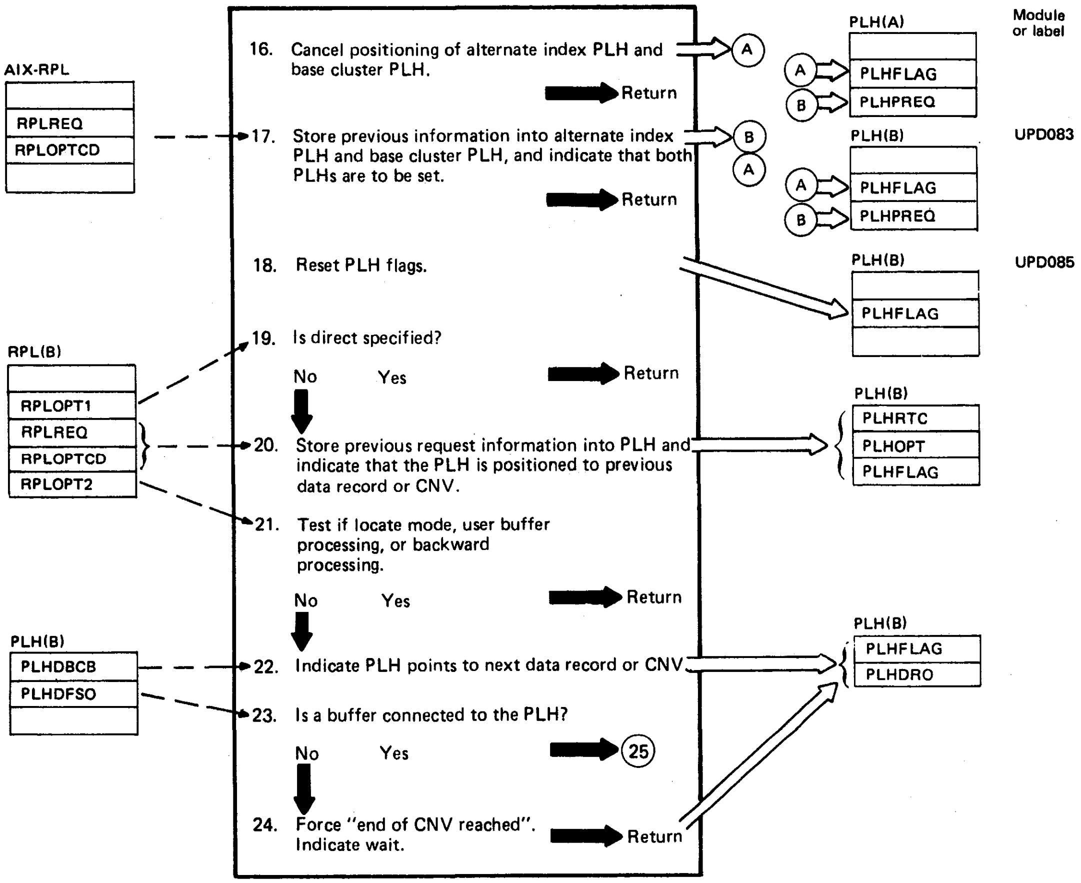
14. Was the update request issued against a path?

    Yes ↓ No → 18

15. Is this a direct request?

    Yes → 16 No → 17

Diagram BH3. PUT UPDATE or ISAM-Issued PUT in LOCATE Mode: Store an Updated Record



16. Cancel positioning of alternate index PLH and base cluster PLH. → Return

17. Store previous information into alternate index PLH and base cluster PLH, and indicate that both PLHs are to be set. → Return

18. Reset PLH flags.

19. Is direct specified?
No Yes → Return

20. Store previous request information into PLH and indicate that the PLH is positioned to previous data record or CNV.

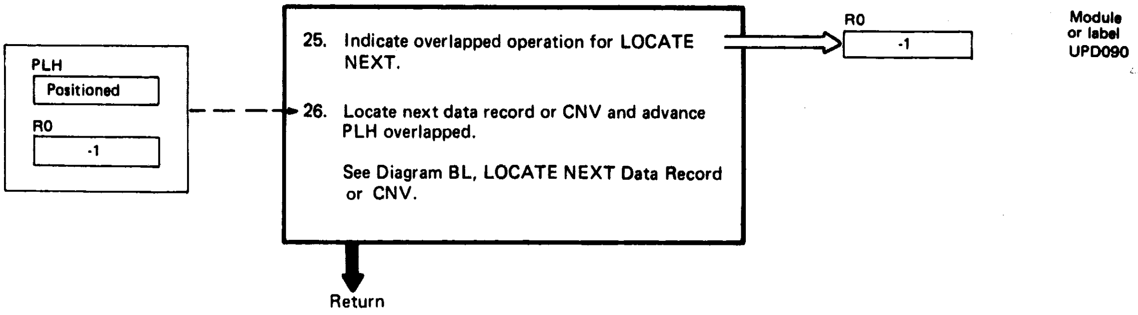
21. Test if locate mode, user buffer processing, or backward processing.
No Yes → Return

22. Indicate PLH points to next data record or CNV.

23. Is a buffer connected to the PLH?
No Yes → 25

24. Force "end of CNV reached". Indicate wait. → Return

Diagram BH4. PUT UPDATE or ISAM-Issued PUT in LOCATE Mode: Store an Updated Record



25. Indicate overlapped operation for LOCATE NEXT.

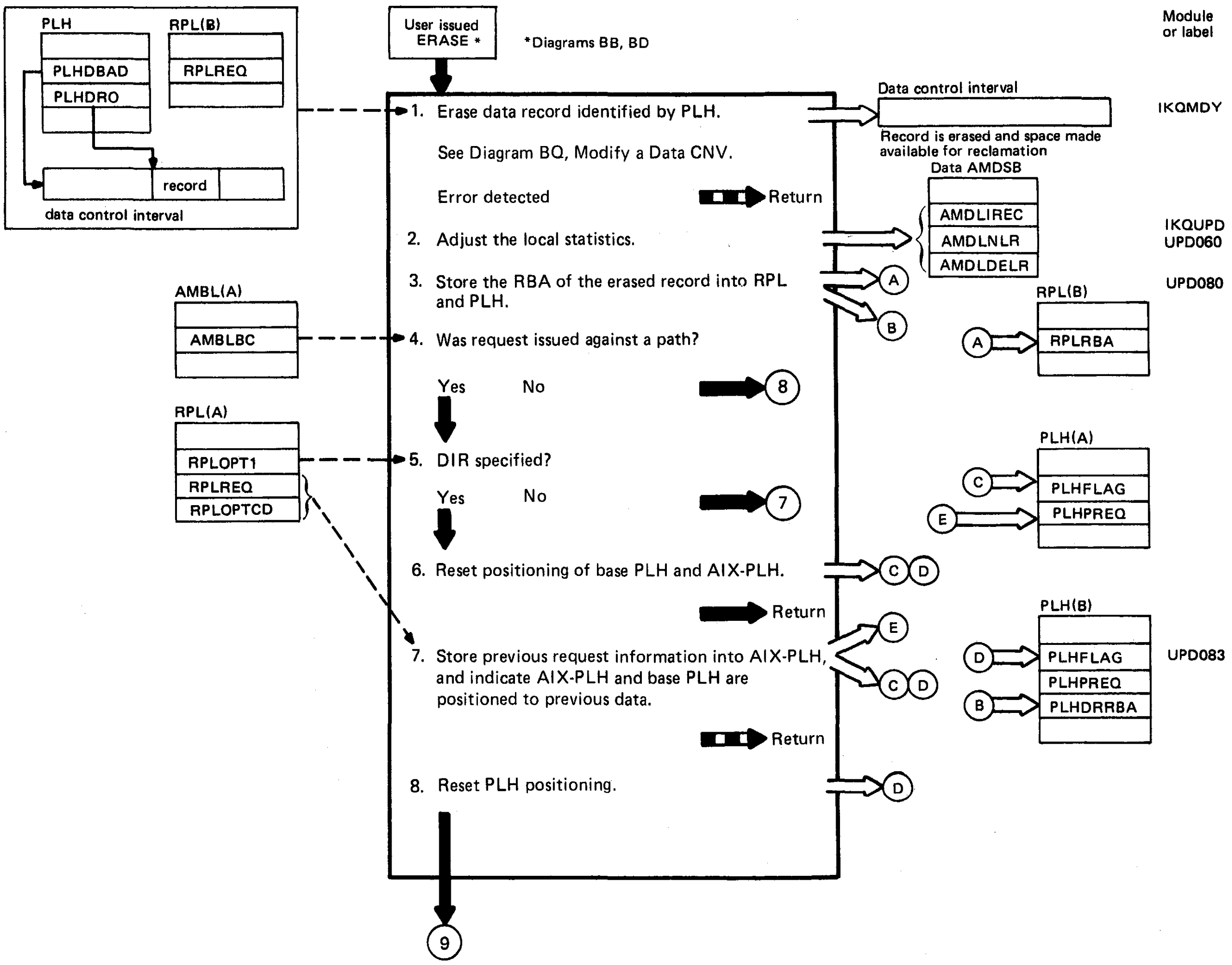
26. Locate next data record or CNV and advance PLH overlapped.

    See Diagram BL, LOCATE NEXT Data Record or CNV.

Notes for Diagram BH4

26. PLH will be positioned to the next CI in an overlapped manner. This means that I/O operations (read-ahead) are started, but their completion is not waited for, in order to overlap the I/O operations with user processing.

Diagram BI1. ERASE: Delete a Record



User issued ERASE *

*Diagrams BB, BD

1. Erase data record identified by PLH.

   See Diagram BQ, Modify a Data CNV.

   Error detected → Return

2. Adjust the local statistics.

3. Store the RBA of the erased record into RPL and PLH.

4. Was request issued against a path?

   Yes → 5; No → 8

5. DIR specified?

   Yes → 6; No → 7

6. Reset positioning of base PLH and AIX-PLH.

   Return

7. Store previous request information into AIX-PLH, and indicate AIX-PLH and base PLH are positioned to previous data.

   Return

8. Reset PLH positioning.

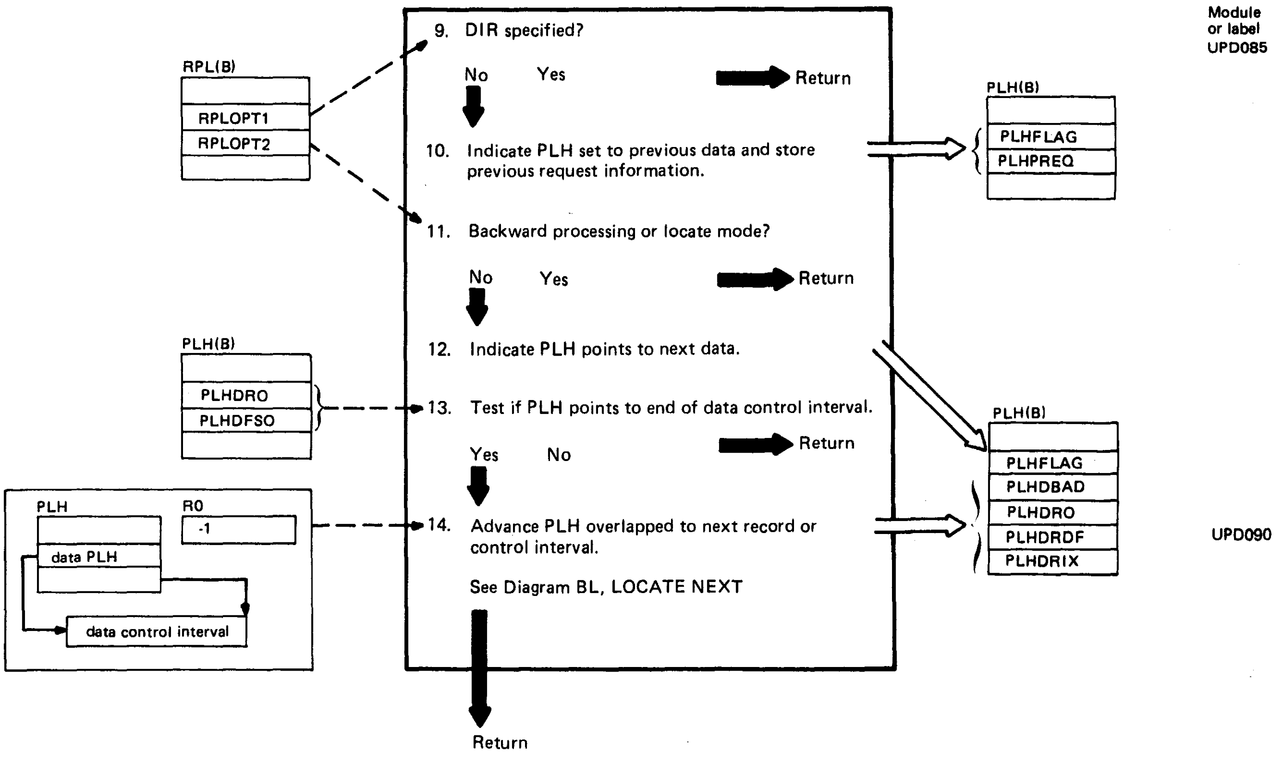
→ 9

Notes for Diagram BI1

Erase requests are allowed only for keyed or addressed processing of key-sequenced data sets or for relative record data sets. An ERASE must be preceded by a GET for update, which positions the PLH to the record to be erased.

Notation:
PLH(B), etc.: Control block associated with base cluster
PLH(A), etc.: Control block associated with AIX

Diagram BI2. ERASE: Delete a Record



9. DIR specified?

No Yes → Return

10. Indicate PLH set to previous data and store previous request information.

11. Backward processing or locate mode?

No Yes → Return

12. Indicate PLH points to next data.

13. Test if PLH points to end of data control interval.

Yes No → Return

14. Advance PLH overlapped to next record or control interval.

See Diagram BL, LOCATE NEXT

Return

Diagram BJ1. Retrieve Spanned Records

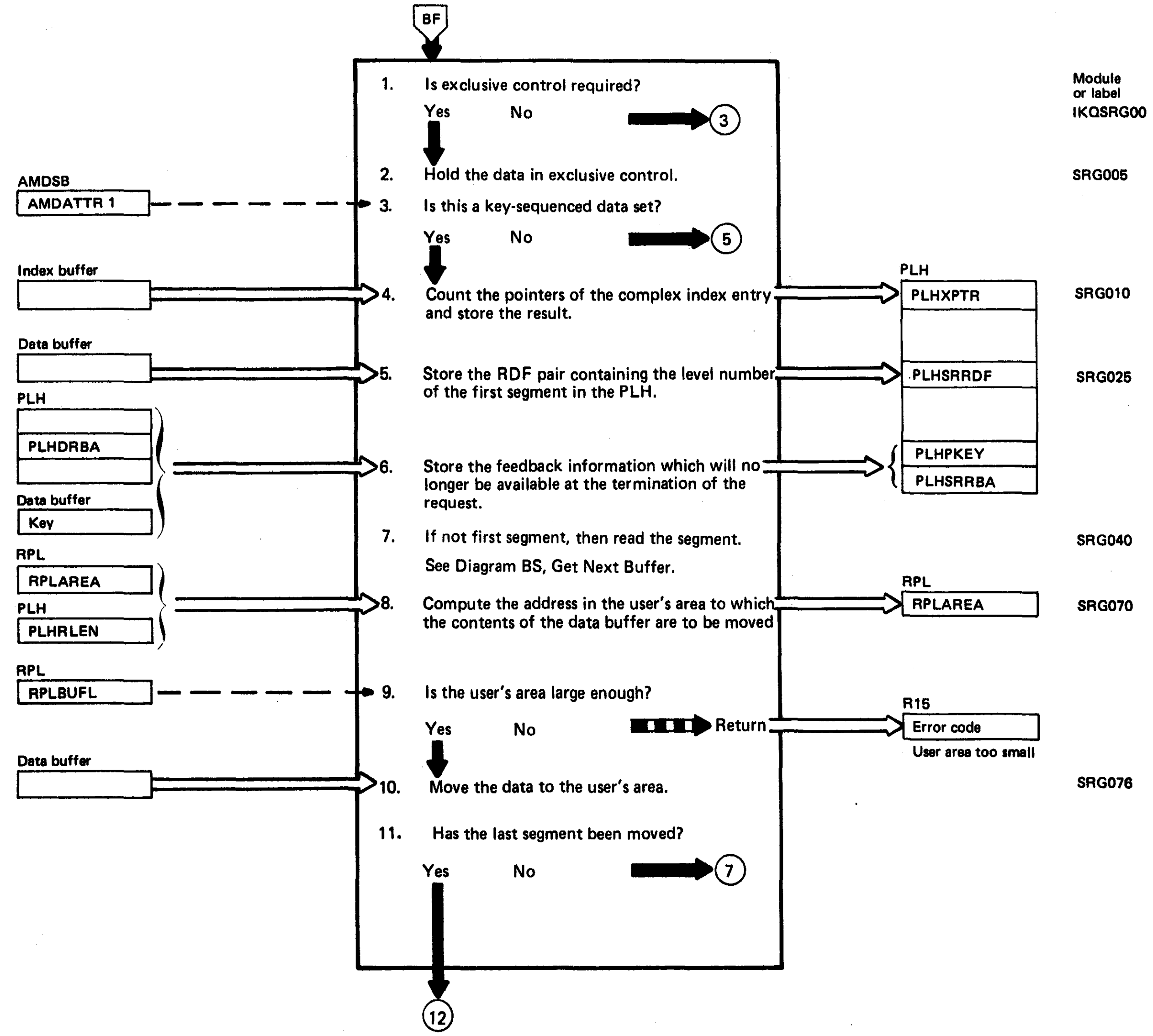
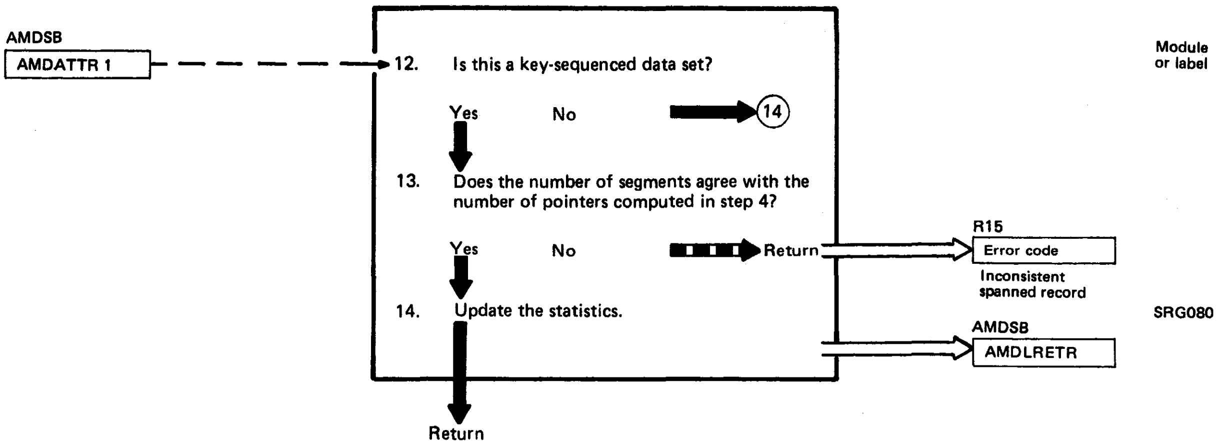

BF
Module or label
IKQSRG00
1. Is exclusive control required?
Yes No
3
AMDSB
AMDATTR 1
2. Hold the data in exclusive control.
SRG005
3. Is this a key-sequenced data set?
Yes No
5
Index buffer
4. Count the pointers of the complex index entry and store the result.
PLH
PLHXPTR
SRG010
Data buffer
5. Store the RDF pair containing the level number of the first segment in the PLH.
PLHSRRDF
SRG025
PLH
PLHDRBA
Data buffer
Key
6. Store the feedback information which will no longer be available at the termination of the request.
PLHPKEY
PLHSRRBA
7. If not first segment, then read the segment.
See Diagram BS, Get Next Buffer.
SRG040
RPL
RPLAREA
PLH
PLHRLEN
8. Compute the address in the user's area to which the contents of the data buffer are to be moved
RPL
RPLAREA
SRG070
RPL
RPLBUFL
9. Is the user's area large enough?
Yes No
Return
R15
Error code
User area too small
Data buffer
10. Move the data to the user's area.
SRG076
11. Has the last segment been moved?
Yes No
7
12

Diagram BJ2. Retrieve Spanned Records

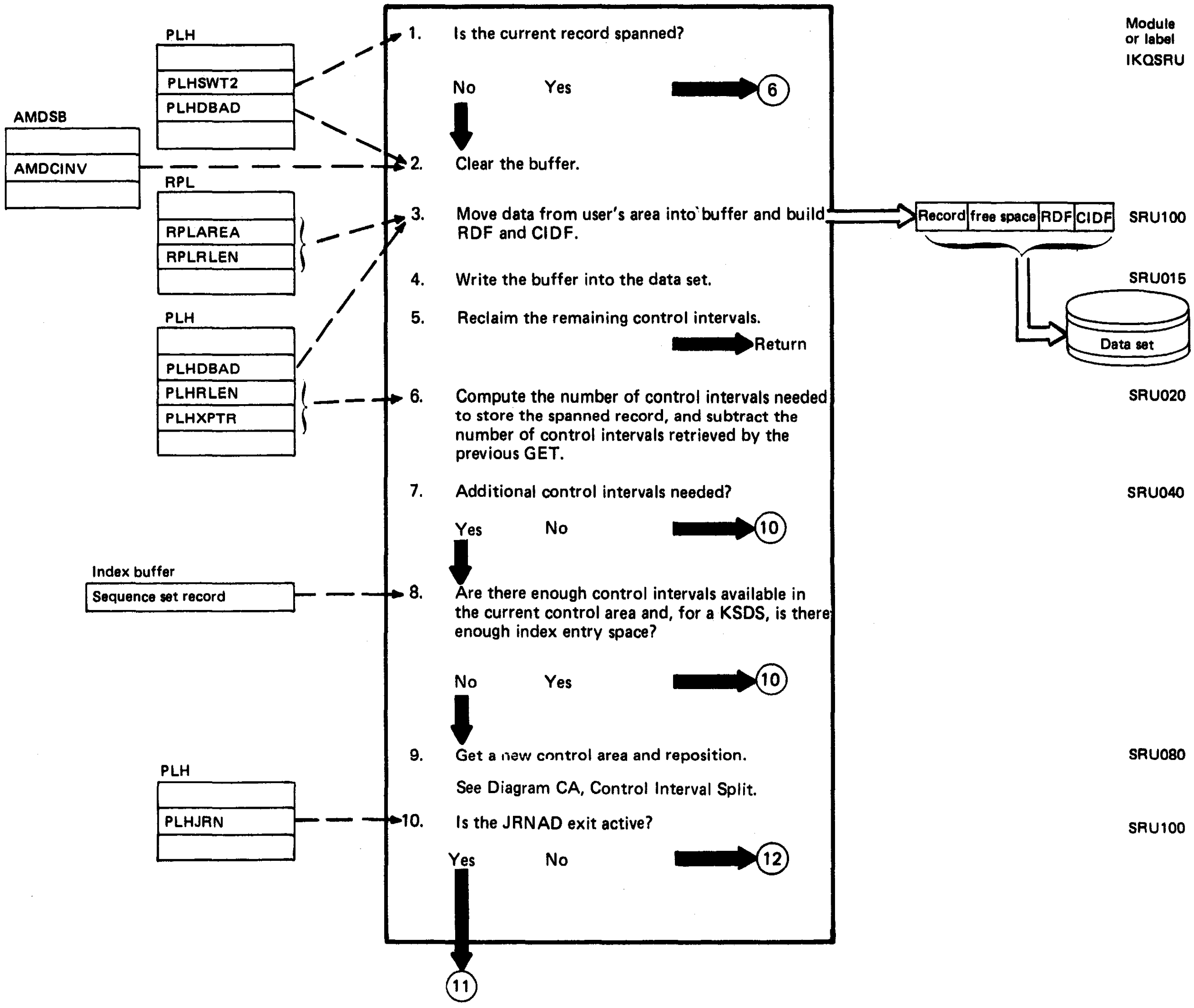


12. Is this a key-sequenced data set?

13. Does the number of segments agree with the number of pointers computed in step 4?

14. Update the statistics.

Diagram BK1. Store Spanned Records



1. Is the current record spanned?

   No Yes → 6

2. Clear the buffer.

3. Move data from user's area into buffer and build RDF and CIDF.

4. Write the buffer into the data set.

5. Reclaim the remaining control intervals. → Return

6. Compute the number of control intervals needed to store the spanned record, and subtract the number of control intervals retrieved by the previous GET.

7. Additional control intervals needed?

   Yes No → 10

8. Are there enough control intervals available in the current control area and, for a KSDS, is there enough index entry space?

   No Yes → 10

9. Get a new control area and reposition.

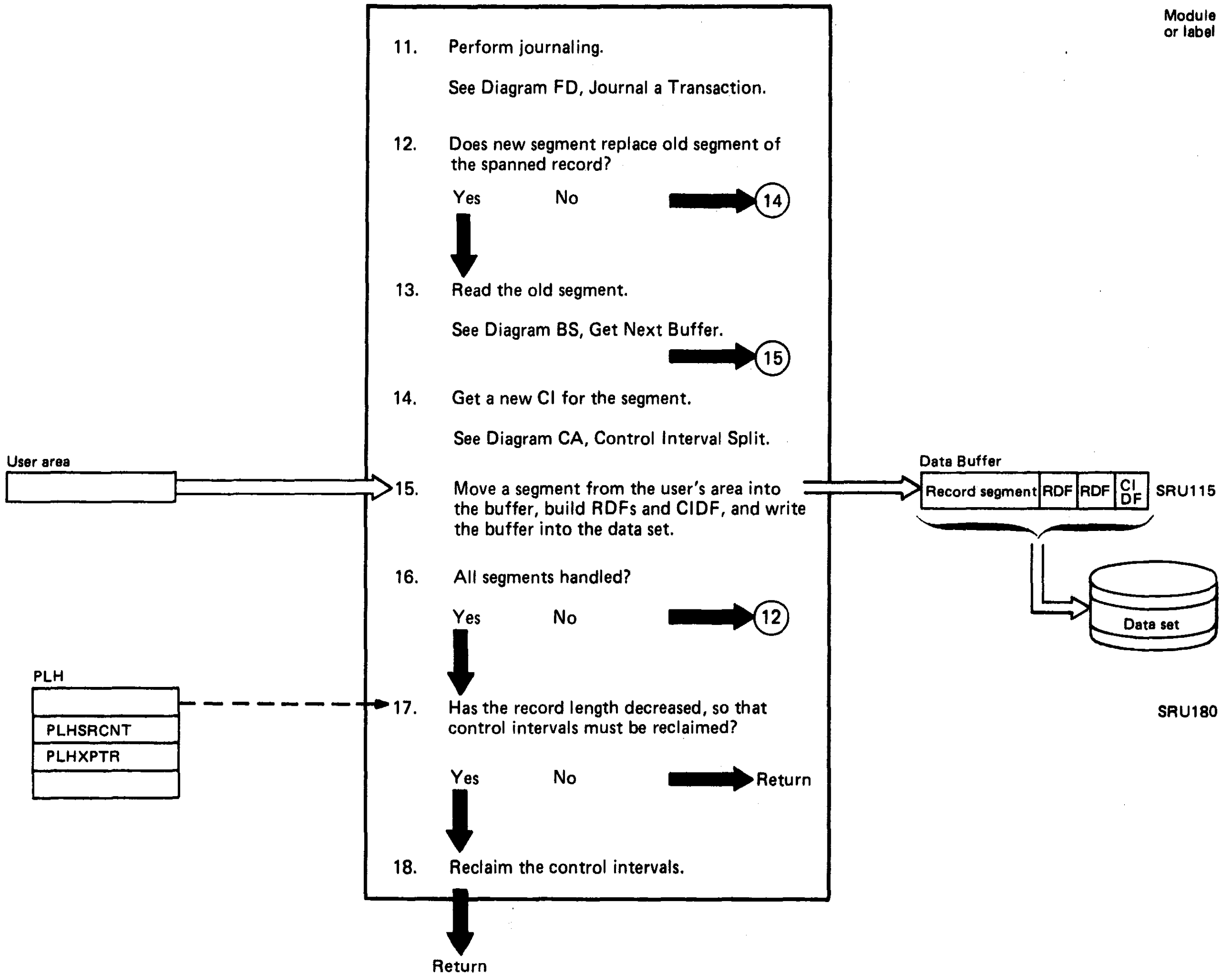
   See Diagram CA, Control Interval Split.

10. Is the JRNAD exit active?

    Yes No → 12

    Yes → 11

Diagram BK2. Store Spanned Records



11. Perform journaling.

See Diagram FD, Journal a Transaction.

12. Does new segment replace old segment of the spanned record?

Yes No → 14

13. Read the old segment.

See Diagram BS, Get Next Buffer. → 15

14. Get a new CI for the segment.

See Diagram CA, Control Interval Split.

15. Move a segment from the user's area into the buffer, build RDFs and CIDF, and write the buffer into the data set.

16. All segments handled?

Yes No → 12

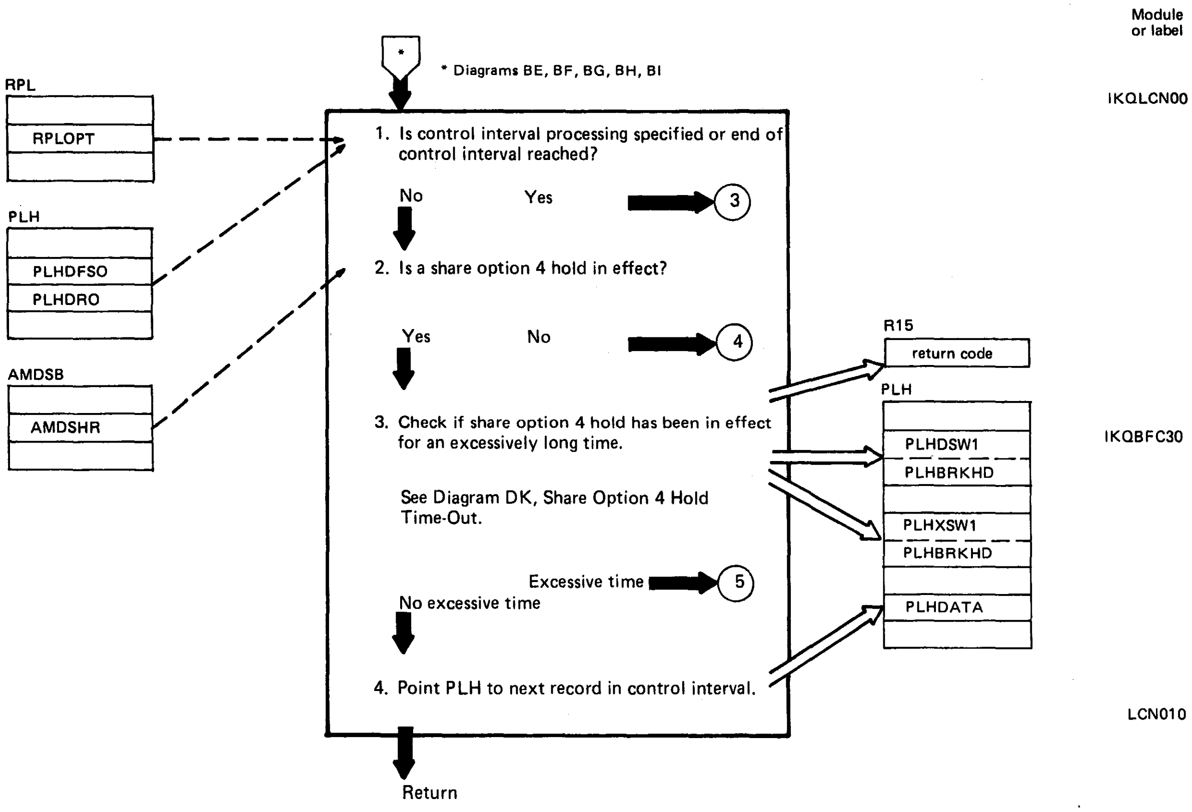
17. Has the record length decreased, so that control intervals must be reclaimed?

Yes No → Return

18. Reclaim the control intervals.

Return

Diagram BL1. LOCATE NEXT: Locate Next Data Record or Control Interval

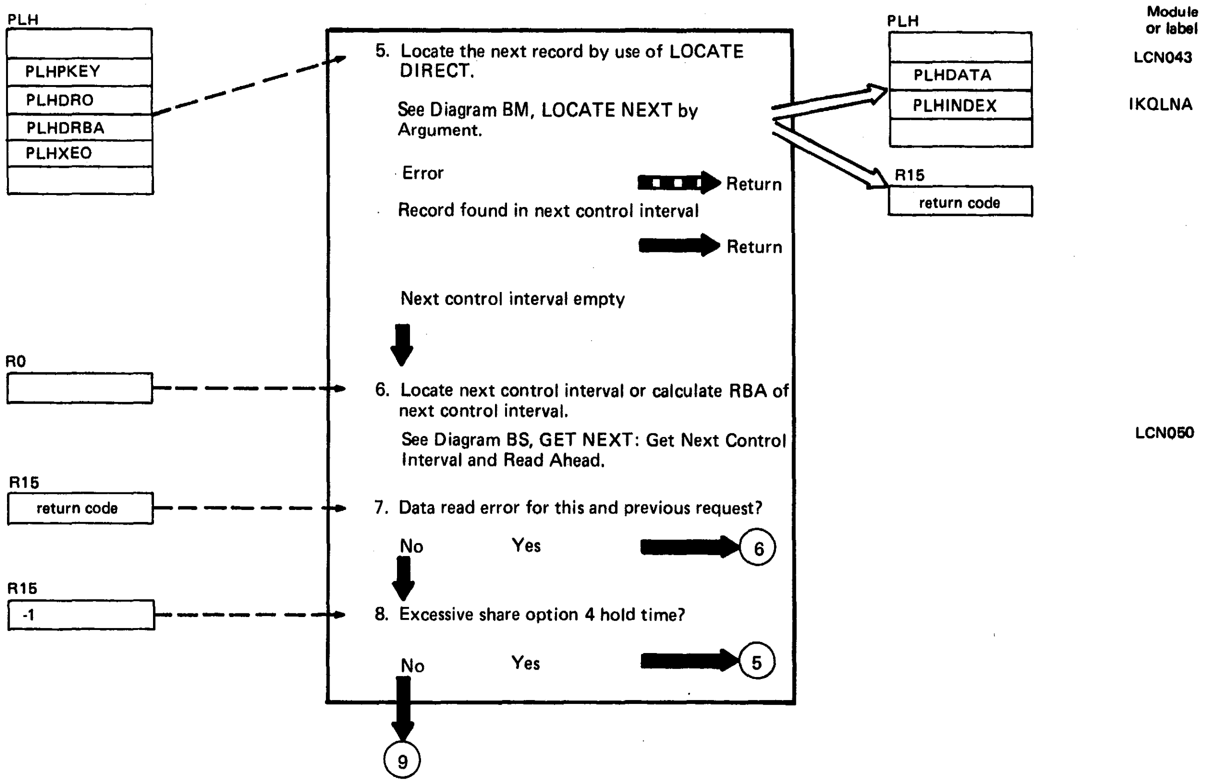


Module
or label
* Diagrams BE, BF, BG, BH, BI
RPL
RPLOPT
PLH
PLHDFSO
PLHDRO
AMDSB
AMDSHR
IKQLCN00
1. Is control interval processing specified or end of control interval reached?
No
Yes
3
2. Is a share option 4 hold in effect?
Yes
No
4
R15
return code
PLH
3. Check if share option 4 hold has been in effect for an excessively long time.
PLHDSW1
PLHBRKHD
IKQBFC30
See Diagram DK, Share Option 4 Hold Time-Out.
PLHXSW1
PLHBRKHD
Excessive time
5
No excessive time
PLHDATA
4. Point PLH to next record in control interval.
LCN010
Return

Notes for Diagram BL1

1. A transition to a new control interval must take place if control interval processing is specified (CNV option in RPL specified), or if the end of a control interval is reached or forced by a previous read error.
3. A negative return code in register 15 indicates excessive hold time.
4. If a new control interval is not needed, the PLH is advanced to the next record within the current control interval, and control is returned to the caller.

Diagram BL2. LOCATE NEXT: Locate Next Data Record or Control Interval



5. Locate the next record by use of LOCATE DIRECT.

See Diagram BM, LOCATE NEXT by Argument.

Error

Record found in next control interval

Next control interval empty

6. Locate next control interval or calculate RBA of next control interval.

See Diagram BS, GET NEXT: Get Next Control Interval and Read Ahead.

7. Data read error for this and previous request?

8. Excessive share option 4 hold time?

Notes for Diagram BL2

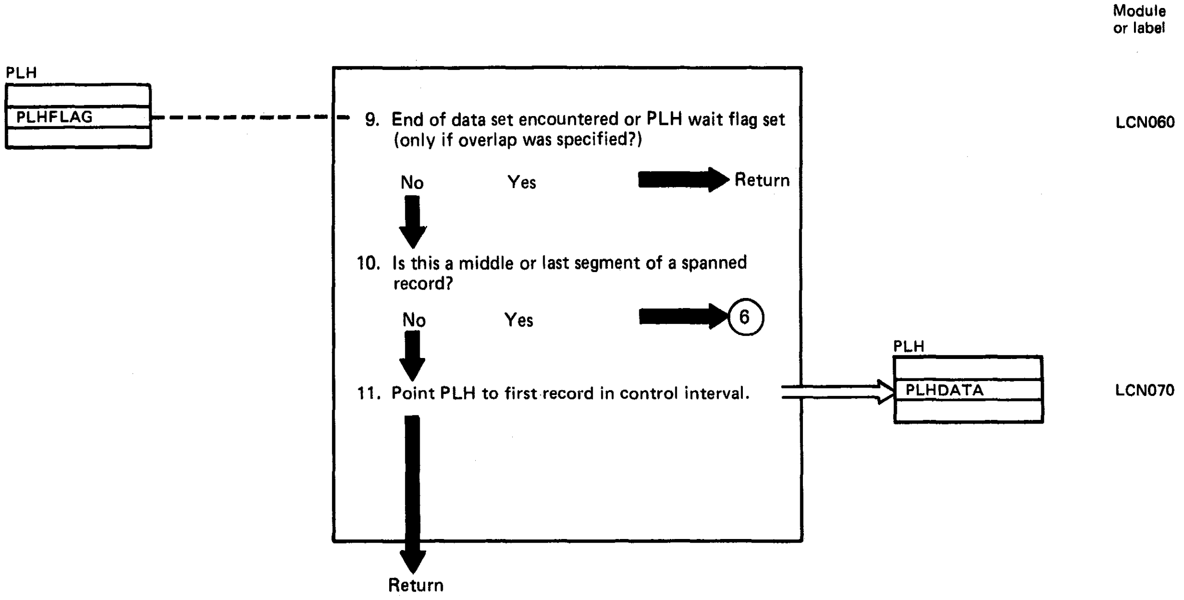
5. A negative return code in register 15 indicates the next control interval is empty.

6. GET NEXT performs two types of operations:

   If register 0 = 0 or 4, an I/O operation is started and finished before further processing is done. If register 0 = 8 (CNV processing only), the RBA is advanced to the physically or logically next control interval, but no reading is done.

   If register 0 contains a negative value, an overlap operation is indicated, causing read ahead to be started while processing continues. A second Locate Next operation with WAIT specified (register 0 = 0 or 4) will ensure that the data is read. This second Locate Next connects the desired data buffer with the PLH. GET NEXT overlap frees the data buffer and initiates read ahead (the index buffer is, however, retained).

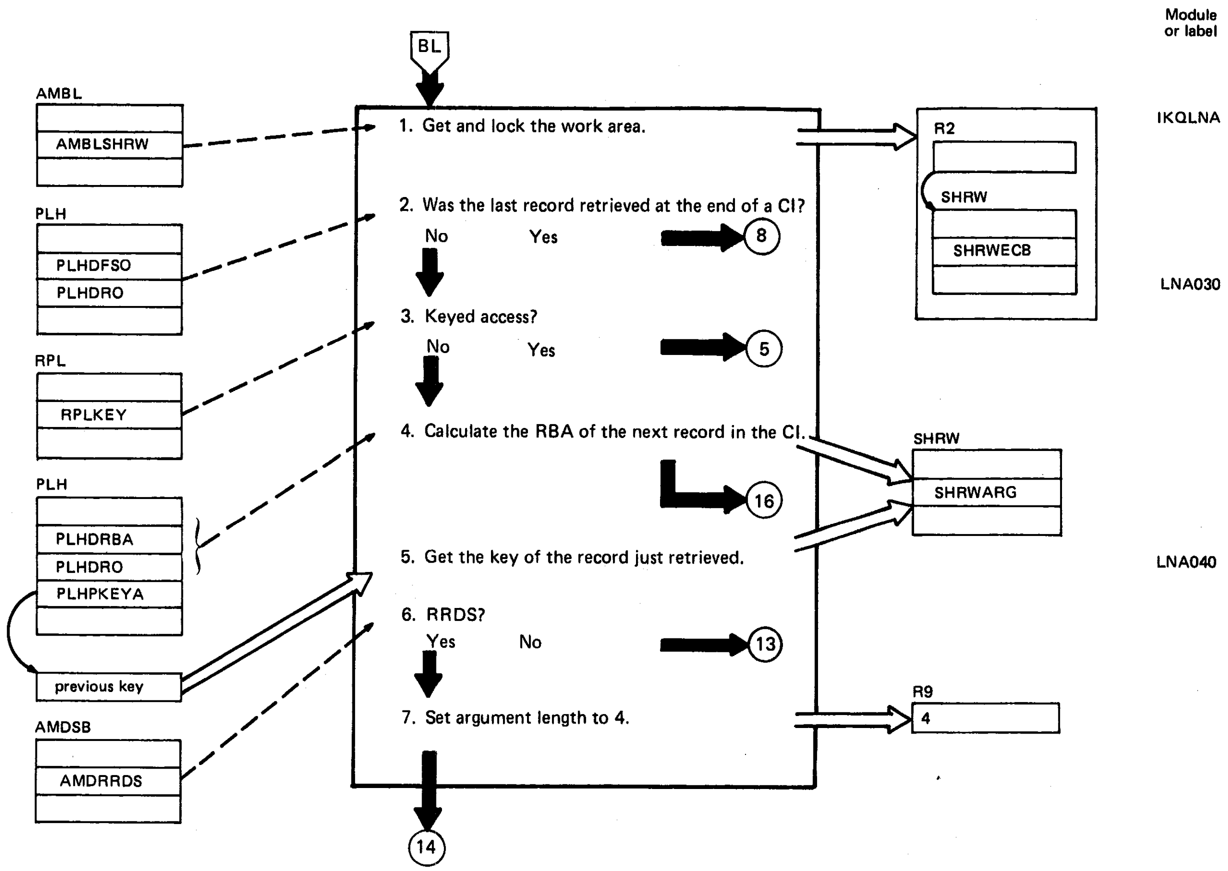
Diagram BL3. LOCATE NEXT: Locate Next Data Record or Control Interval


Module or label
PLH
PLHFLAG
9. End of data set encountered or PLH wait flag set (only if overlap was specified?)
No
Yes
Return
LCN060
10. Is this a middle or last segment of a spanned record?
No
Yes
6
PLH
11. Point PLH to first record in control interval.
PLHDATA
LCN070
Return

Notes for Diagram BL3

10. As the start of the next record is to be located, the middle and last segments of spanned records are skipped, unless CNV access is being used.

Diagram BM1. LOCATE NEXT by Argument


Module or label
BL
AMBL
AMBLSHRW
PLH
PLHDFSO
PLHDRO
RPL
RPLKEY
PLH
PLHDRBA
PLHDRO
PLHPKEYA
previous key
AMDSB
AMDRRDS
1. Get and lock the work area.
2. Was the last record retrieved at the end of a CI?
No
Yes
8
3. Keyed access?
No
Yes
5
4. Calculate the RBA of the next record in the CI.
16
5. Get the key of the record just retrieved.
6. RRDS?
Yes
No
13
7. Set argument length to 4.
14
R2
SHRW
SHRWECB
SHRW
SHRWARG
R9
4
IKQLNA
LNA030
LNA040

Notes for Diagram BM1

1. IKQLNA is called to construct an argument (an RBA or a key) that can be used by Locate Direct (IKQLCD) to find the next record after the one that was just retrieved. IKQLNA then calls IKQLCD.

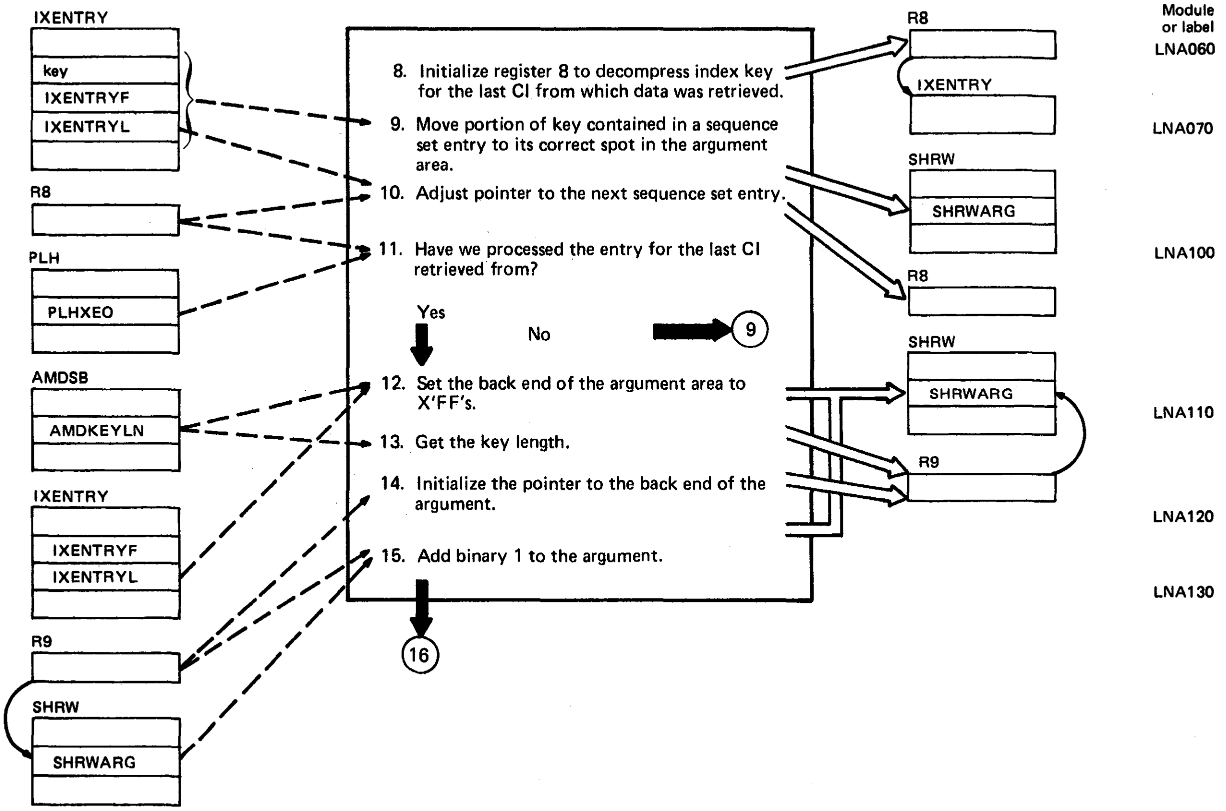
   IKQLNA does *not* construct an argument if the record just retrieved is at the end of a CI, except for keyed access to a KSDS. IKQGNX handles the records at the end of a CI.

   For sequential retrieval, when the CA has been held in share option 4 exclusive control for an excessively long time, IKQLCD releases and reacquires the share option 4 exclusive control.

2. If the last record retrieved was at the end of a CI, the processing is assumed to be keyed access to a KSDS.

4. IKQCLN has already updated PLHDRO to point to the next record.

Diagram BM2. LOCATE NEXT by Argument



8. Initialize register 8 to decompress index key for the last CI from which data was retrieved.

9. Move portion of key contained in a sequence set entry to its correct spot in the argument area.

10. Adjust pointer to the next sequence set entry.

11. Have we processed the entry for the last CI retrieved from?

12. Set the back end of the argument area to X'FF's.

13. Get the key length.

14. Initialize the pointer to the back end of the argument.

15. Add binary 1 to the argument.

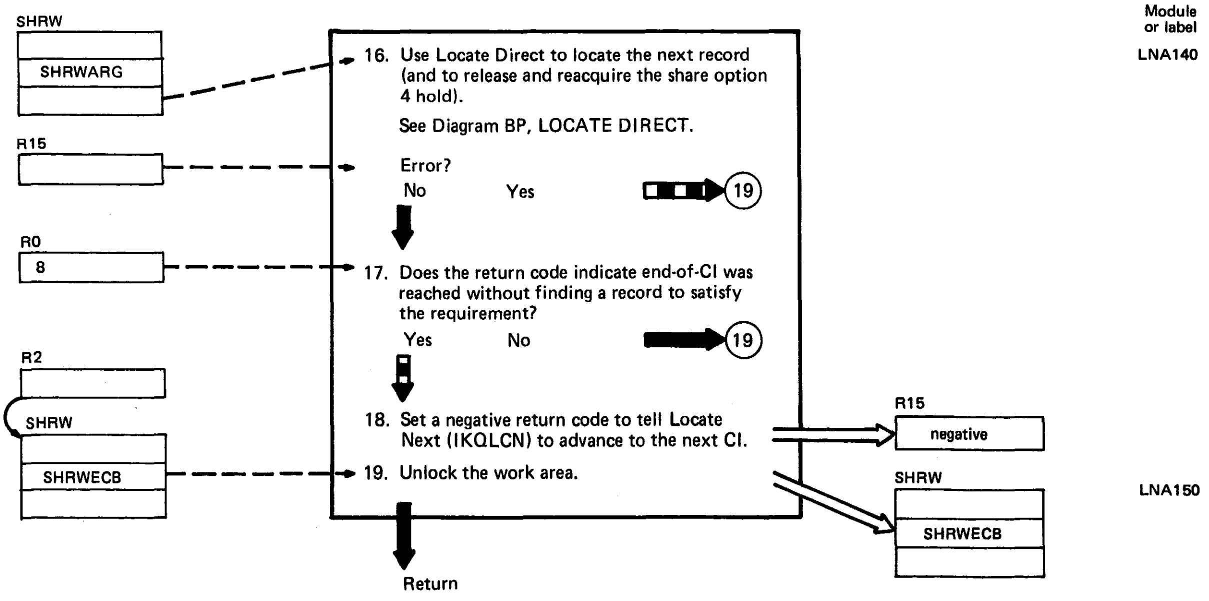
Notes for Diagram BM2

8.-11. The key in the sequence set entry for the CI last processed is decompressed in order to give the high key assigned by the index to that CI. See notes for Diagram BU for an example of key compression.

12. Set to X'FF's the part of the back end of the argument area that was not set from the index entry key (due to rear compression of the index entry key).

15. Adding binary 1 to the argument gives the lowest possible key that the next record could have. This step handles the relative record number for an RRDS, as well as the key for a KSDS.

Diagram BM3. LOCATE NEXT by Argument



16. Use Locate Direct to locate the next record (and to release and reacquire the share option 4 hold).

    See Diagram BP, LOCATE DIRECT.

    Error?

    No Yes ⟶ 19

17. Does the return code indicate end-of-CI was reached without finding a record to satisfy the requirement?

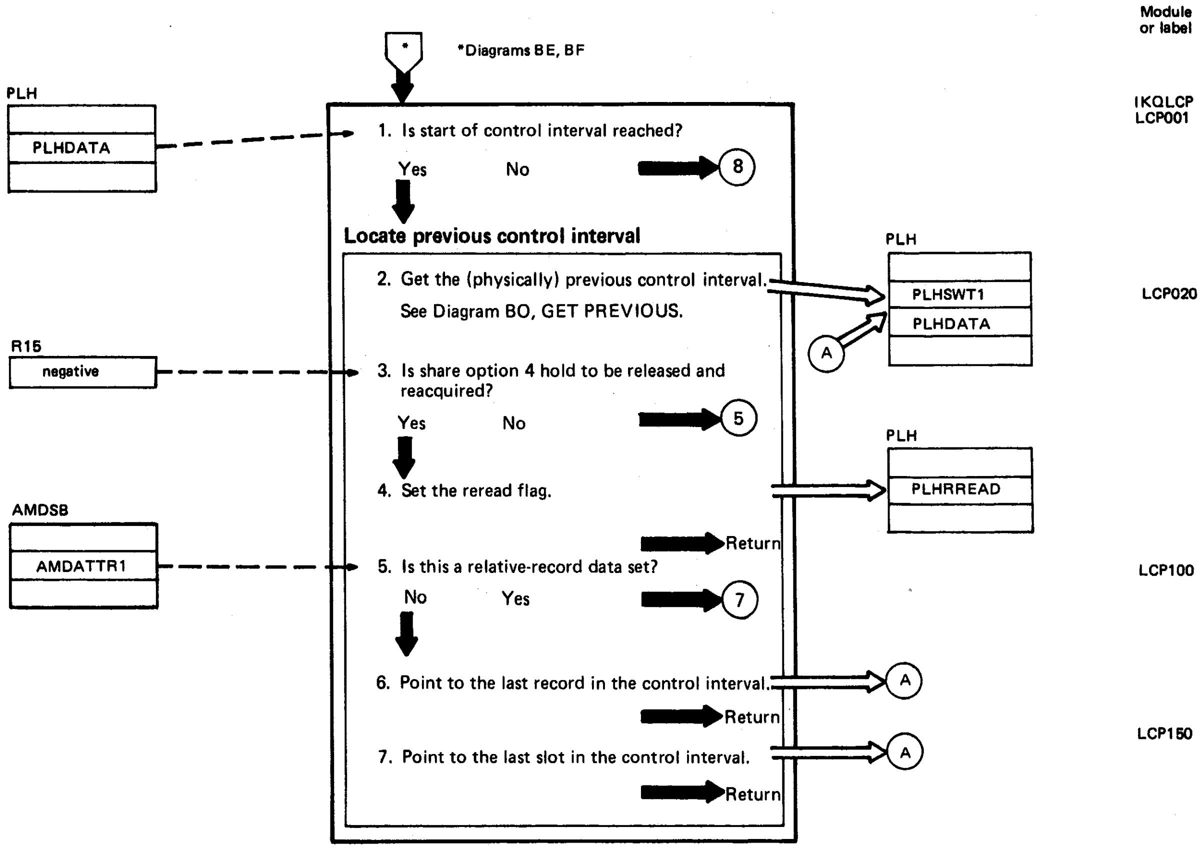
    Yes No ⟶ 19

18. Set a negative return code to tell Locate Next (IKQLCN) to advance to the next CI.

19. Unlock the work area.

Return

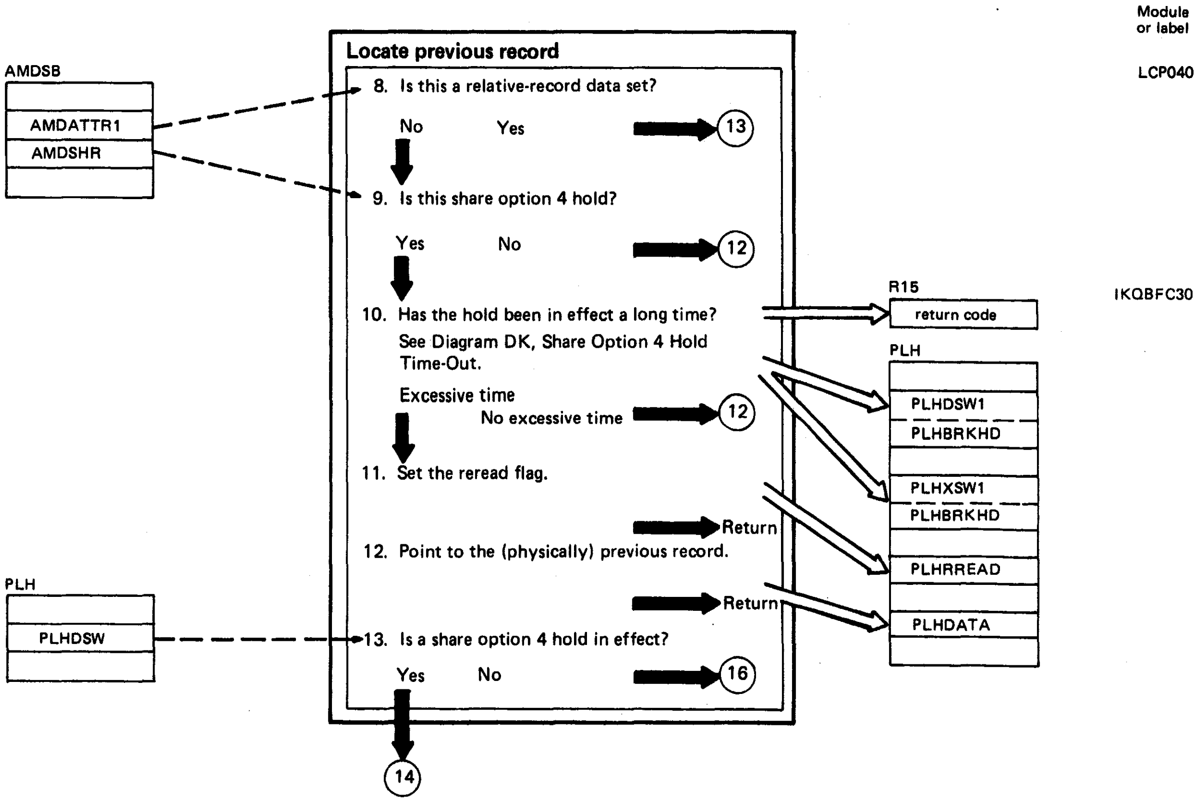
Diagram BN1. LOCATE PREVIOUS: Locate Previous Data Record or Control Interval


Module or label
*Diagrams BE, BF
PLH
PLHDATA
IKQLCP
LCP001
1. Is start of control interval reached?
Yes
No
8
Locate previous control interval
2. Get the (physically) previous control interval. See Diagram BO, GET PREVIOUS.
PLH
PLHSWT1
PLHDATA
A
LCP020
R15
negative
3. Is share option 4 hold to be released and reacquired?
Yes
No
5
PLH
4. Set the reread flag.
PLHRREAD
AMDSB
AMDATTR1
Return
5. Is this a relative-record data set?
No
Yes
7
LCP100
6. Point to the last record in the control interval.
A
Return
7. Point to the last slot in the control interval.
A
Return
LCP150

Notes for Diagram BN1

4. The reread flag causes the PLH to be positioned to the previously retrieved record (with release and reacquisition of the share option 4 hold), after which IKQLCP is called again.

Diagram BN2. LOCATE PREVIOUS: Locate Previous Data Record or Control Interval



Locate previous record

8. Is this a relative-record data set?

9. Is this share option 4 hold?

10. Has the hold been in effect a long time? See Diagram DK, Share Option 4 Hold Time-Out.

11. Set the reread flag.

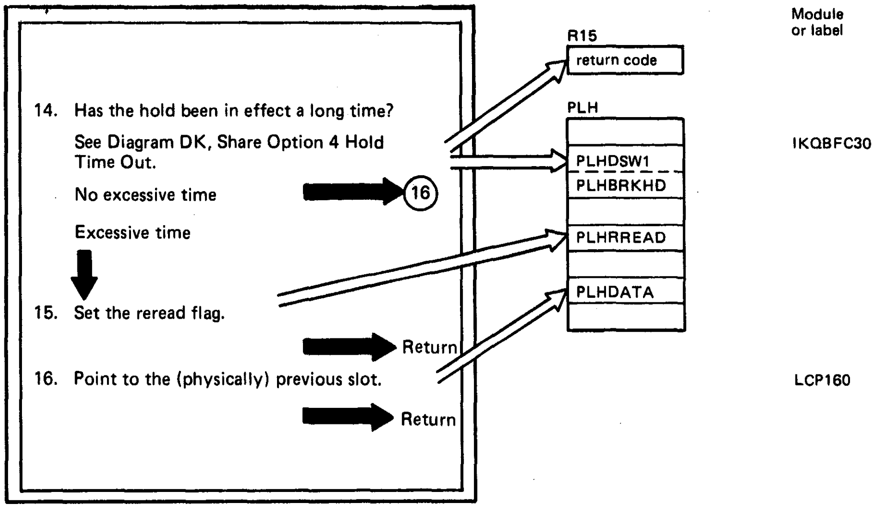
12. Point to the (physically) previous record.

13. Is a share option 4 hold in effect?

Notes for Diagram BN2

11. See note for step 4.

Diagram BN3. LOCATE PREVIOUS: Locate Previous Data Record or Control Interval



14. Has the hold been in effect a long time?

See Diagram DK, Share Option 4 Hold Time Out.

No excessive time

Excessive time

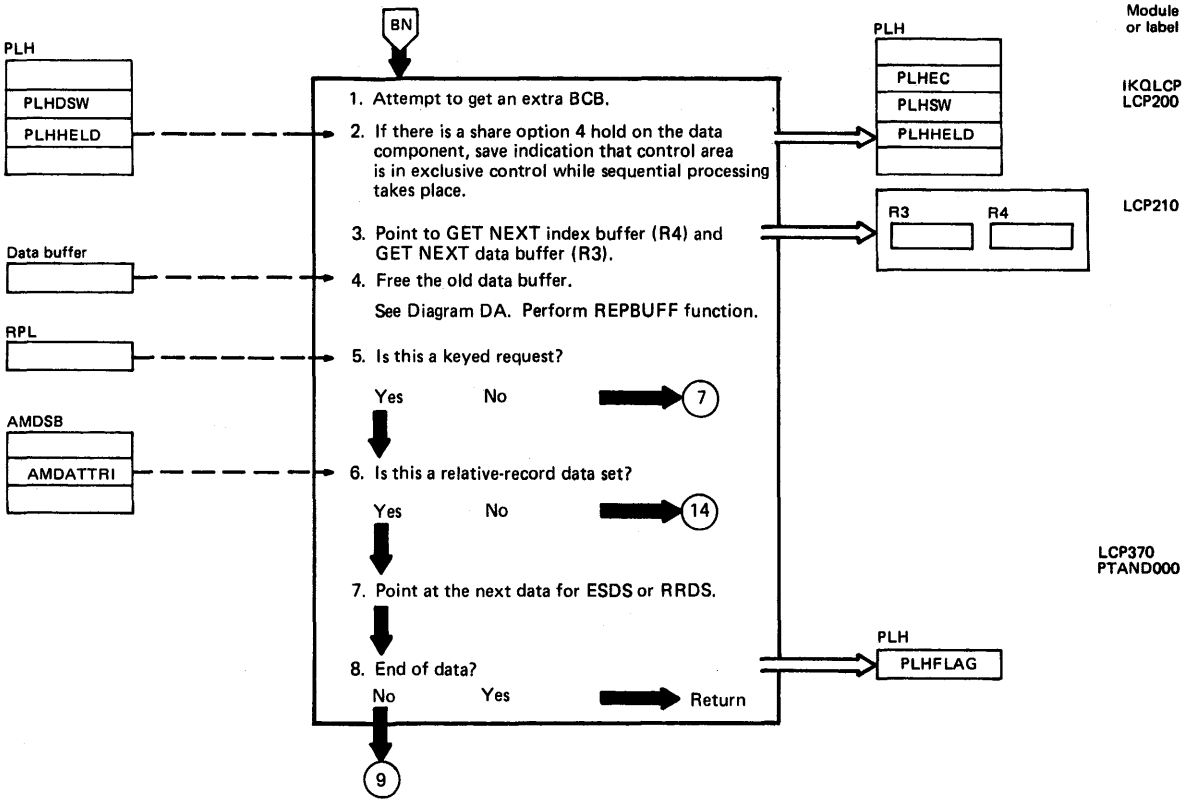
15. Set the reread flag.

16. Point to the (physically) previous slot.

Notes for Diagram BN3

15. See note for step 4.

Diagram BO1. GET PREVIOUS: Retrieve Previous Record


1. Attempt to get an extra BCB.
2. If there is a share option 4 hold on the data component, save indication that control area is in exclusive control while sequential processing takes place.
3. Point to GET NEXT index buffer (R4) and GET NEXT data buffer (R3).
4. Free the old data buffer.

   See Diagram DA. Perform REPBUFF function.
5. Is this a keyed request?

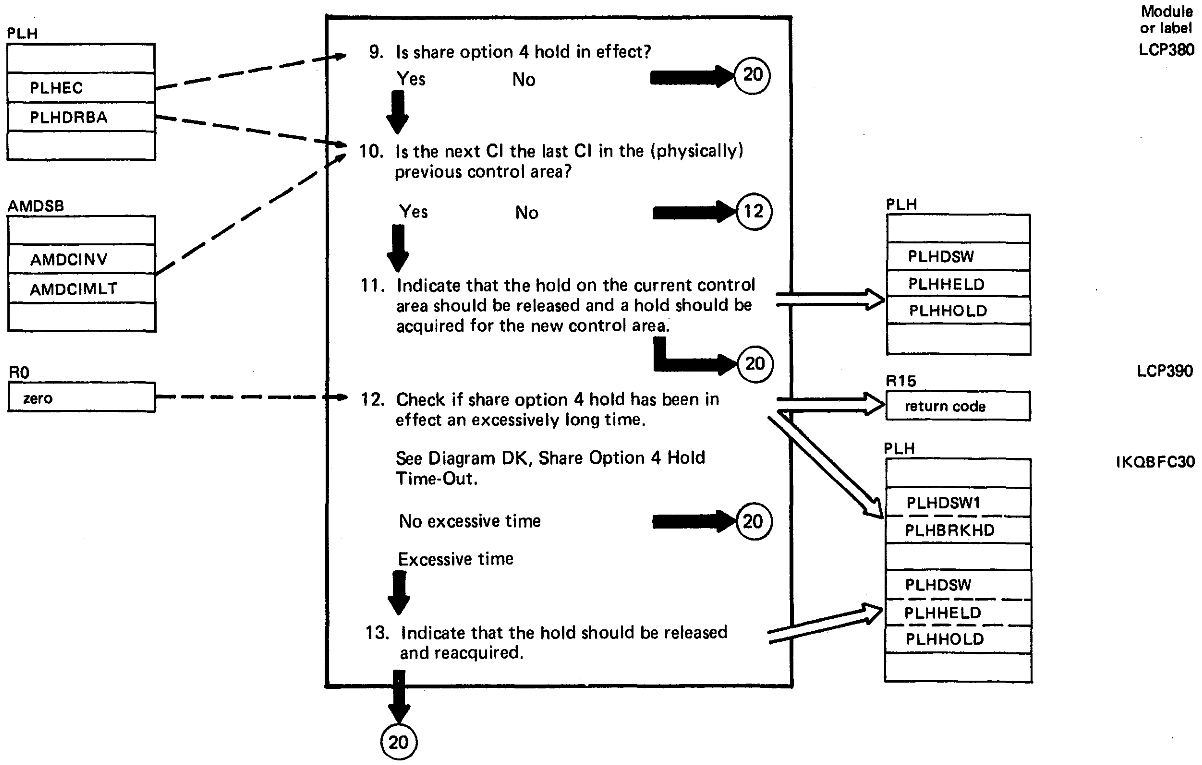
   Yes No → 7
6. Is this a relative-record data set?

   Yes No → 14
7. Point at the next data for ESDS or RRDS.
8. End of data?

   No Yes → Return

→ 9

Diagram BO2. GET PREVIOUS: Retrieve Previous Record



9. Is share option 4 hold in effect?
Yes No → 20

10. Is the next CI the last CI in the (physically) previous control area?
Yes No → 12

11. Indicate that the hold on the current control area should be released and a hold should be acquired for the new control area. → 20

12. Check if share option 4 hold has been in effect an excessively long time.

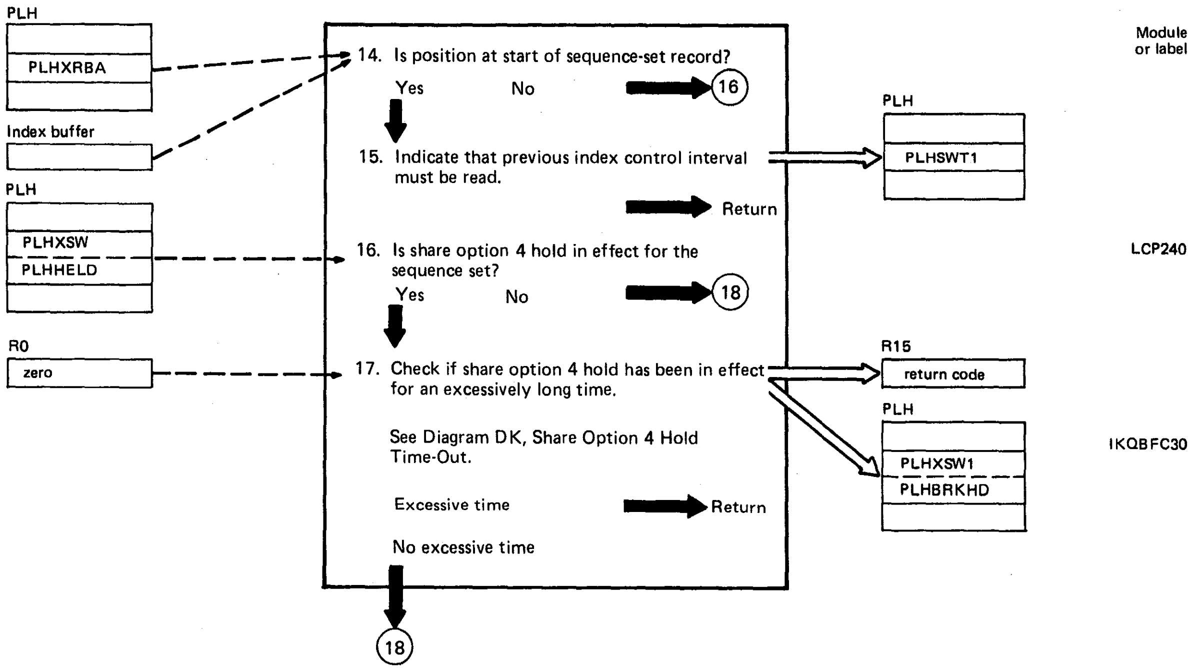
See Diagram DK, Share Option 4 Hold Time-Out.

No excessive time → 20

Excessive time

13. Indicate that the hold should be released and reacquired.

Diagram BO3. GET PREVIOUS: Retrieve Previous Record



14. Is position at start of sequence-set record?

Yes — No ➡ 16

15. Indicate that previous index control interval must be read.

➡ Return

16. Is share option 4 hold in effect for the sequence set?

Yes — No ➡ 18

17. Check if share option 4 hold has been in effect for an excessively long time.

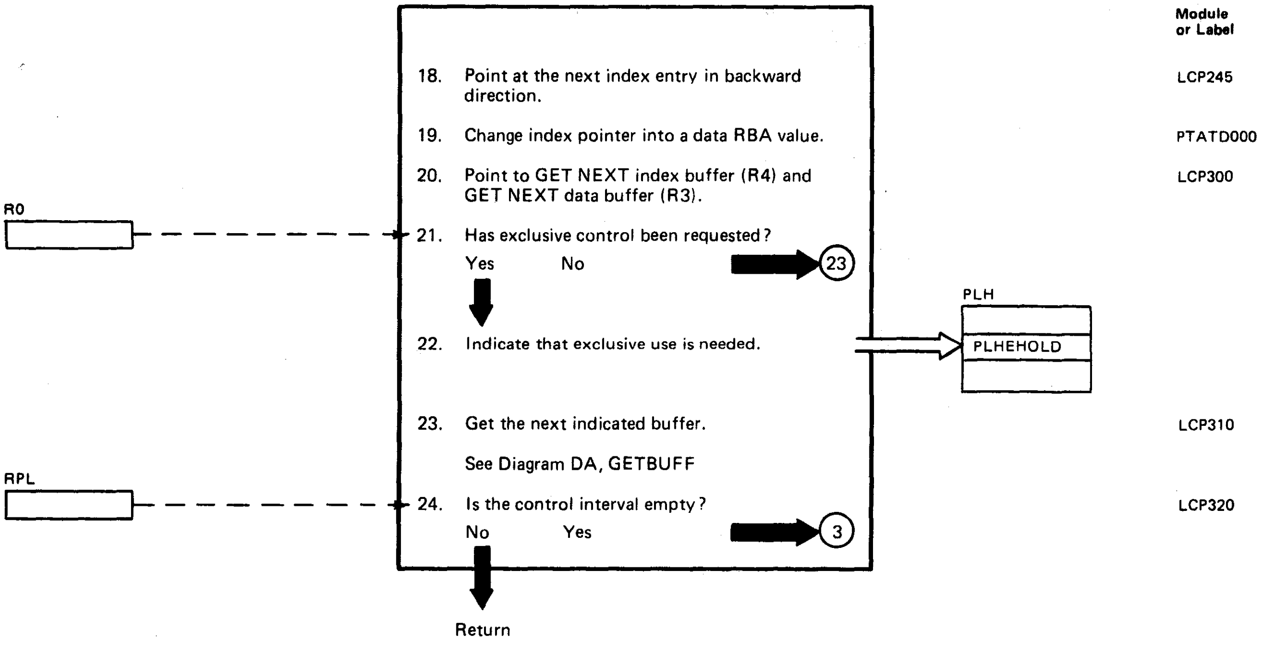
See Diagram DK, Share Option 4 Hold Time-Out.

Excessive time ➡ Return

No excessive time

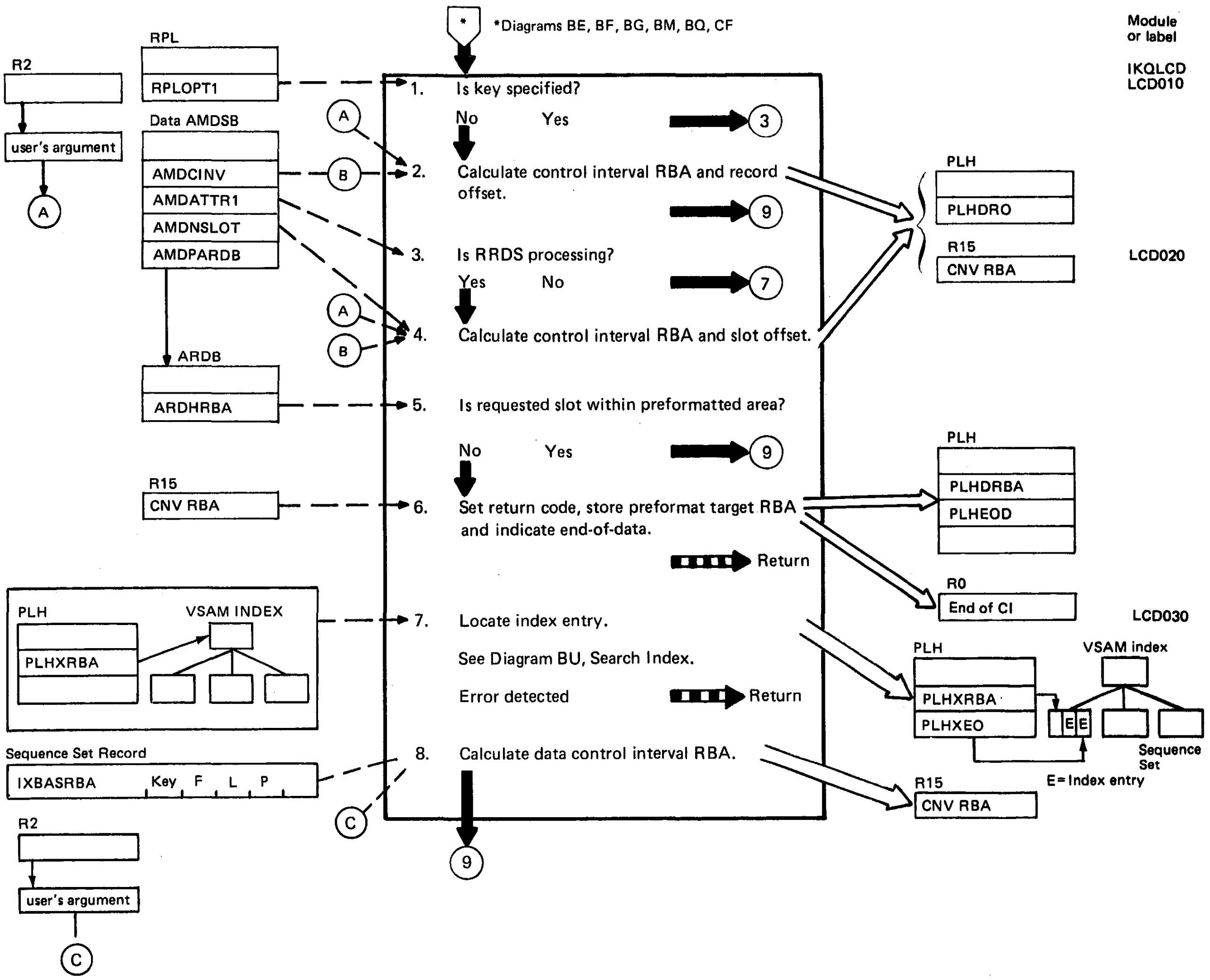
18

Diagram BO4. GET PREVIOUS: Retrieve Previous Record



	Module or Label
18. Point at the next index entry in backward direction.	LCP245
19. Change index pointer into a data RBA value.	PTATD000
20. Point to GET NEXT index buffer (R4) and GET NEXT data buffer (R3).	LCP300
21. Has exclusive control been requested? Yes No	
22. Indicate that exclusive use is needed.	
23. Get the next indicated buffer. See Diagram DA, GETBUFF	LCP310
24. Is the control interval empty? No Yes	LCP320

Diagram BP1. LOCATE DIRECT: Locate Data Record or Control Interval by Key or RBA



*Diagrams BE, BF, BG, BM, BQ, CF

1. Is key specified?

   No Yes → 3

2. Calculate control interval RBA and record offset.

   → 9

3. Is RRDS processing?

   Yes No → 7

4. Calculate control interval RBA and slot offset.

5. Is requested slot within preformatted area?

   No Yes → 9

6. Set return code, store preformat target RBA and indicate end-of-data.

   → Return

7. Locate index entry.

   See Diagram BU, Search Index.

   Error detected → Return

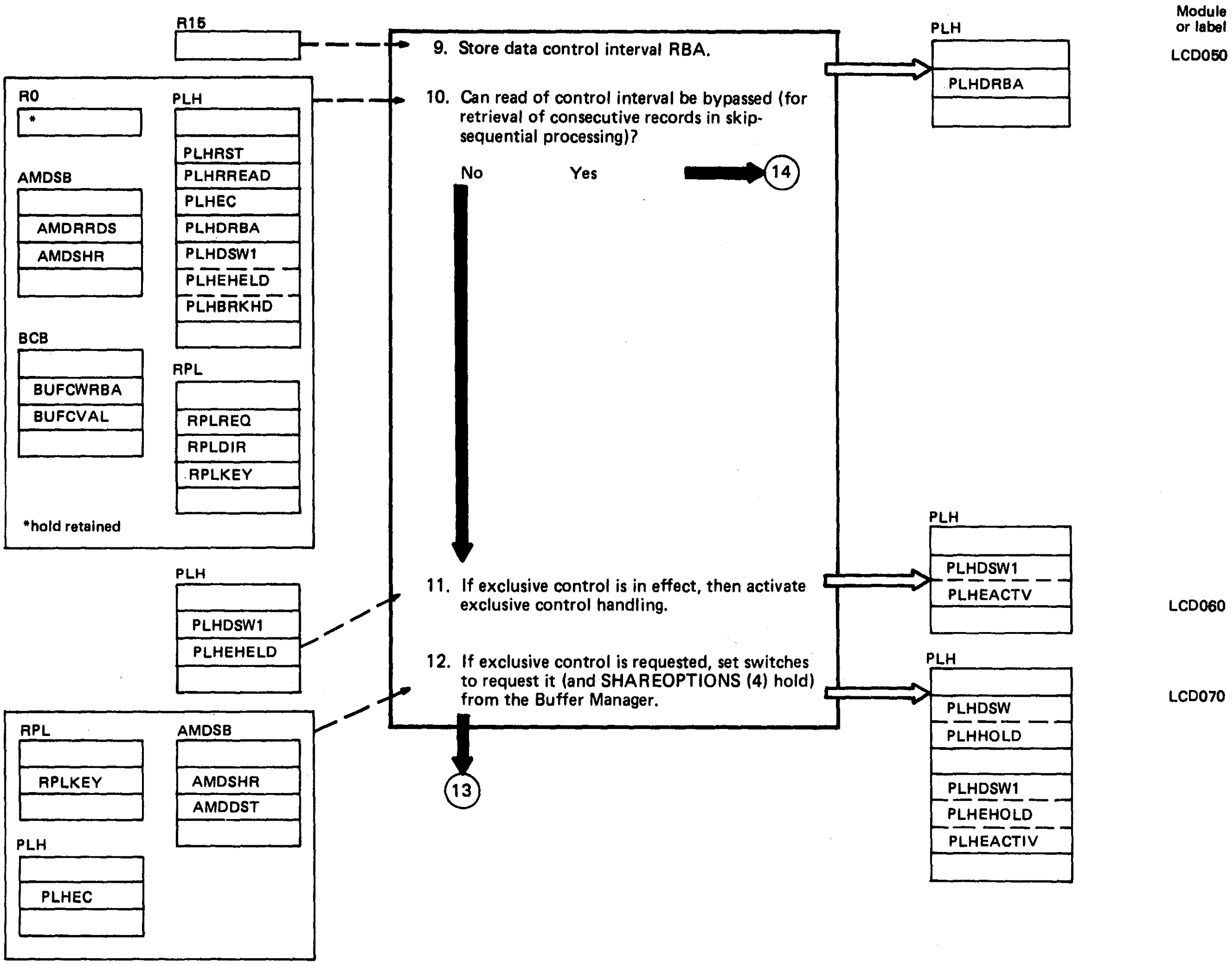
8. Calculate data control interval RBA.

   → 9

Notes for Diagram BP1

7. If exclusive control is requested for a SHAREOPTIONS (4) hold file, the index search routine requests a SHAREOPTIONS (4) hold on the sequence set.

Diagram BP2. LOCATE DIRECT: Locate Data Record or Control Interval by Key or RBA



9. Store data control interval RBA.

10. Can read of control interval be bypassed (for retrieval of consecutive records in skip-sequential processing)?

11. If exclusive control is in effect, then activate exclusive control handling.

12. If exclusive control is requested, set switches to request it (and SHAREOPTIONS (4) hold) from the Buffer Manager.

Notes for Diagram BP2

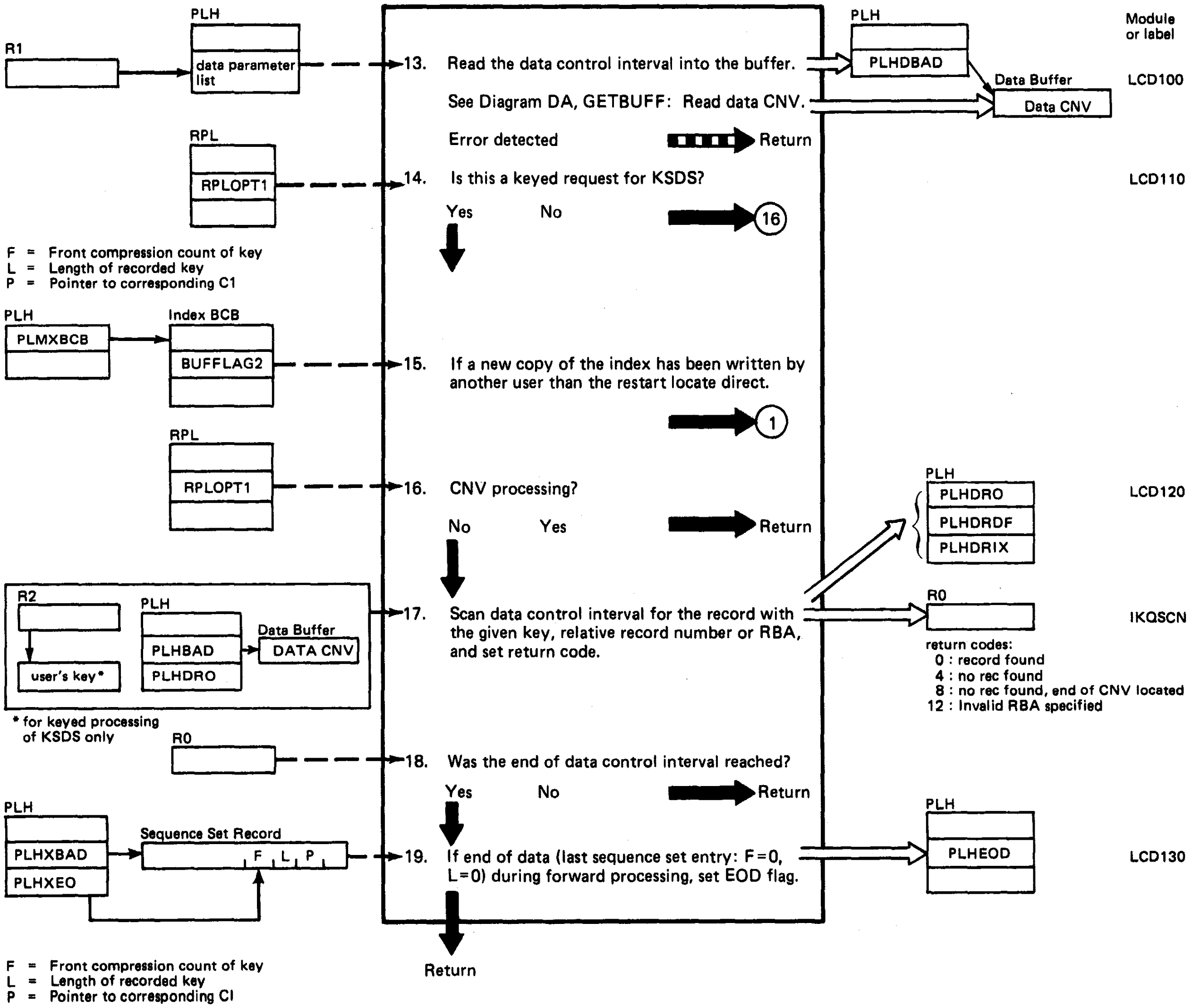
10. The call to the Buffer Manager to read the control interval is bypassed if:

- The desired control interval is already in the buffer connected to the PLH, and
- No change of exclusive control is required.

This bypass can serve two purposes during skip-sequential processing:

- It prevents multiple writes to the file if multiple records are updated in the same control interval.
- If the file is SHAREOPTIONS (4), it prevents multiple reads of the same control interval.

Diagram BP3. LOCATE DIRECT: Locate Data Record or Control Interval by Key or RBA



13. Read the data control interval into the buffer.

See Diagram DA, GETBUFF: Read data CNV.

Error detected → Return

14. Is this a keyed request for KSDS?

Yes No → 16

15. If a new copy of the index has been written by another user than the restart locate direct. → 1

16. CNV processing?

No Yes → Return

17. Scan data control interval for the record with the given key, relative record number or RBA, and set return code.

18. Was the end of data control interval reached?

Yes No → Return

19. If end of data (last sequence set entry: F=0, L=0) during forward processing, set EOD flag.

Return

F = Front compression count of key
L = Length of recorded key
P = Pointer to corresponding C1

F = Front compression count of key
L = Length of recorded key
P = Pointer to corresponding CI

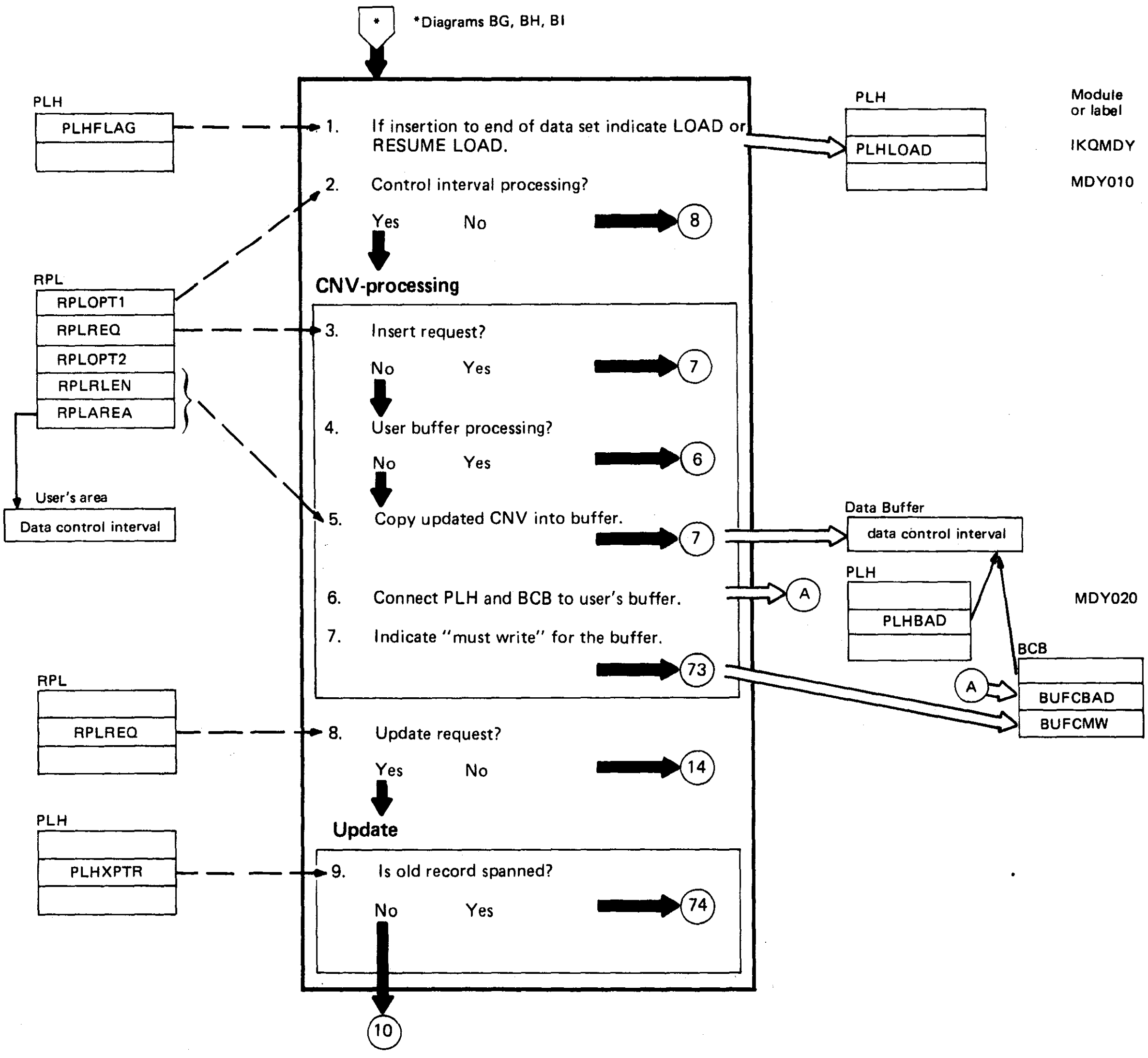
Notes for Diagram BP3

17. Possible return codes for addressing processing:
- record found
- invalid RBA

Possible return codes for RRDS processing:
- record found
- no record found
- no record found, end of CNV located
  (if requested slot is beyond preformatted area)

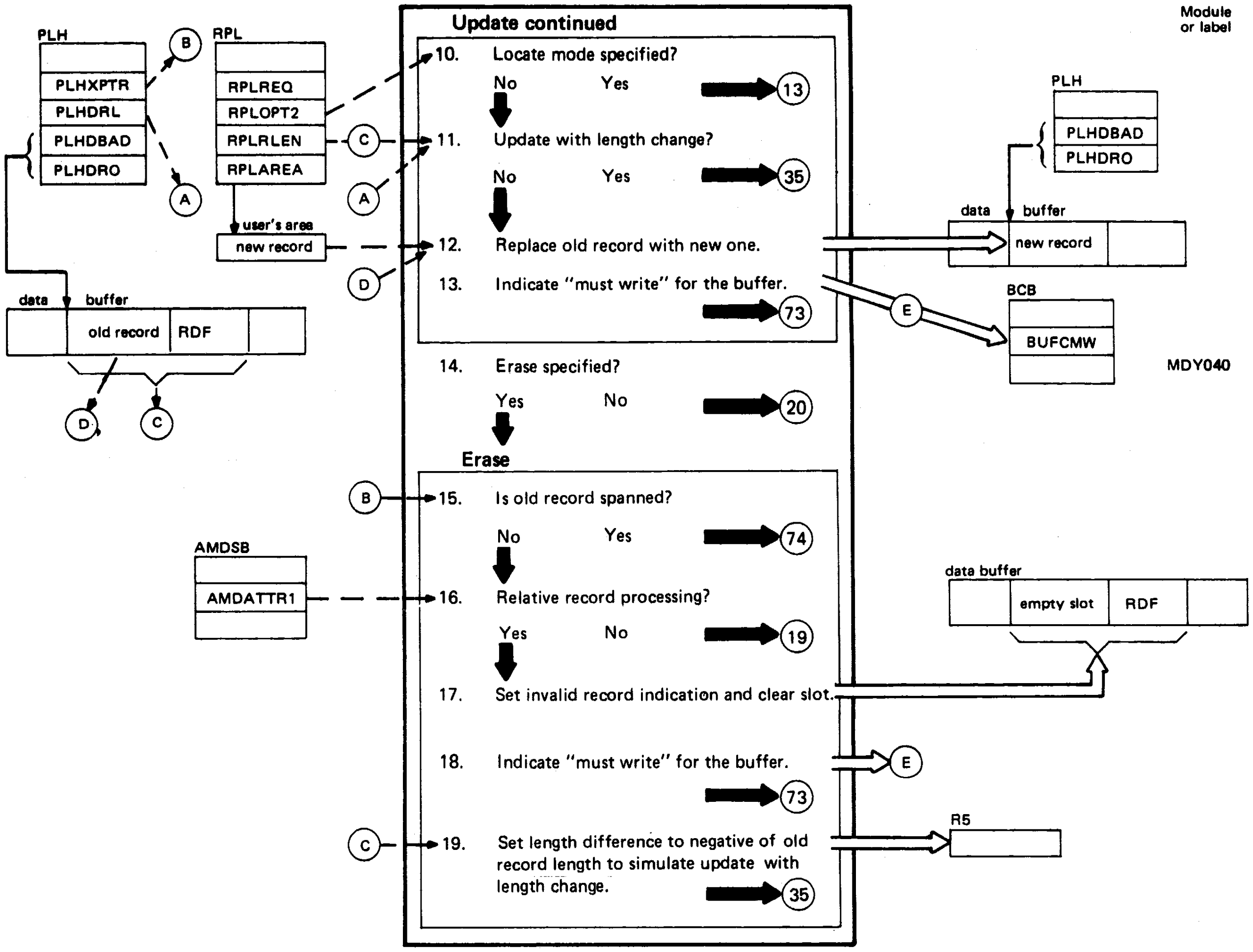
Possible return codes for keyed processing of KSDS:
- record found
- no record found
- no record found, end of CNV located

Diagram BQ1. Modify a Data Control Interval



1. If insertion to end of data set indicate LOAD or RESUME LOAD.
2. Control interval processing?
3. Insert request?
4. User buffer processing?
5. Copy updated CNV into buffer.
6. Connect PLH and BCB to user's buffer.
7. Indicate "must write" for the buffer.
8. Update request?
9. Is old record spanned?

Diagram BQ2. Modify a Data Control Interval



Update continued

10. Locate mode specified?
No Yes → 13

11. Update with length change?
No Yes → 35

12. Replace old record with new one.

13. Indicate "must write" for the buffer. → 73

14. Erase specified?
Yes No → 20

Erase

15. Is old record spanned?
No Yes → 74

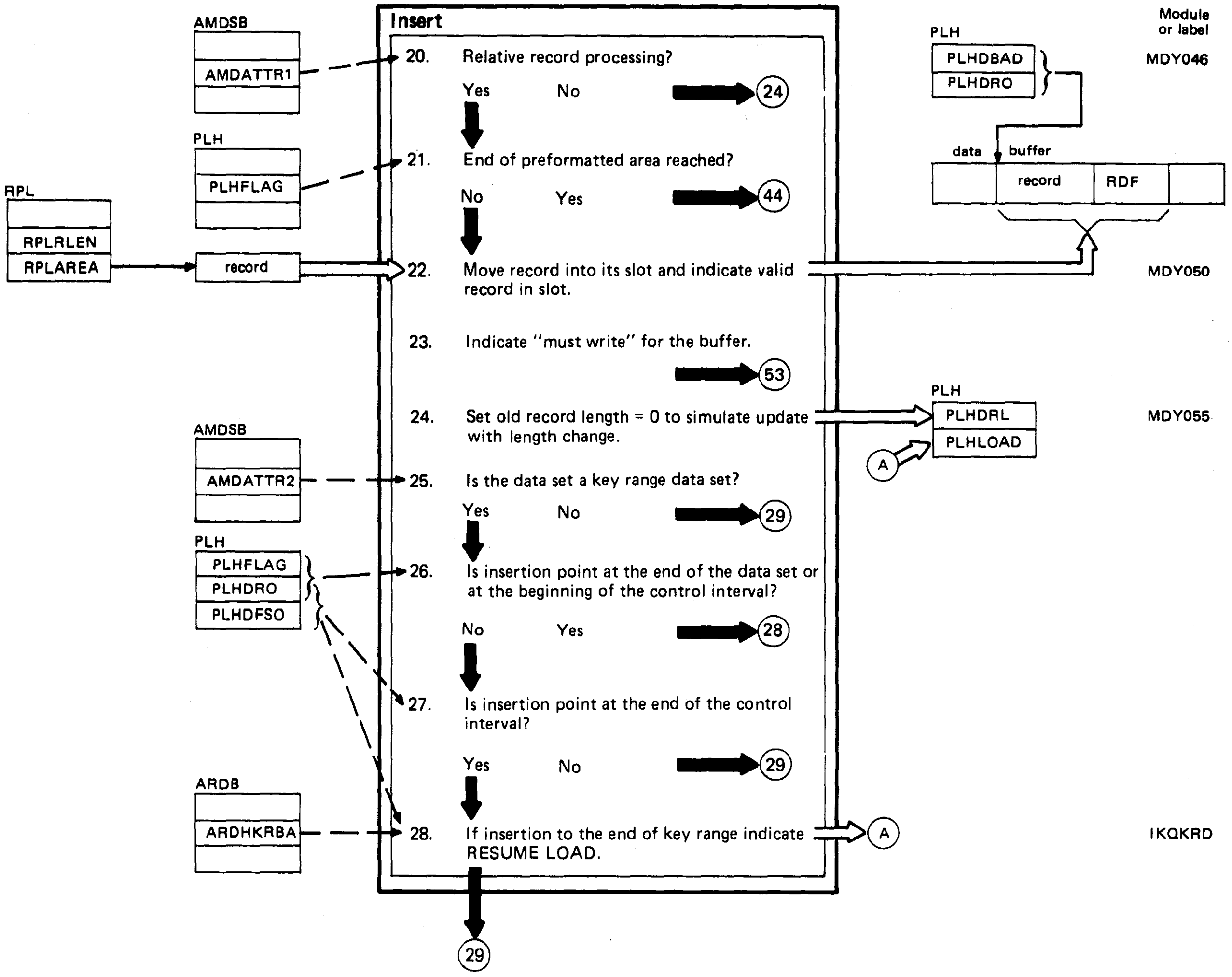
16. Relative record processing?
Yes No → 19

17. Set invalid record indication and clear slot.

18. Indicate "must write" for the buffer. → 73

19. Set length difference to negative of old record length to simulate update with length change. → 35

Diagram BQ3. Modify a Data Control Interval



20. Relative record processing?

21. End of preformatted area reached?

22. Move record into its slot and indicate valid record in slot.

23. Indicate "must write" for the buffer.

24. Set old record length = 0 to simulate update with length change.

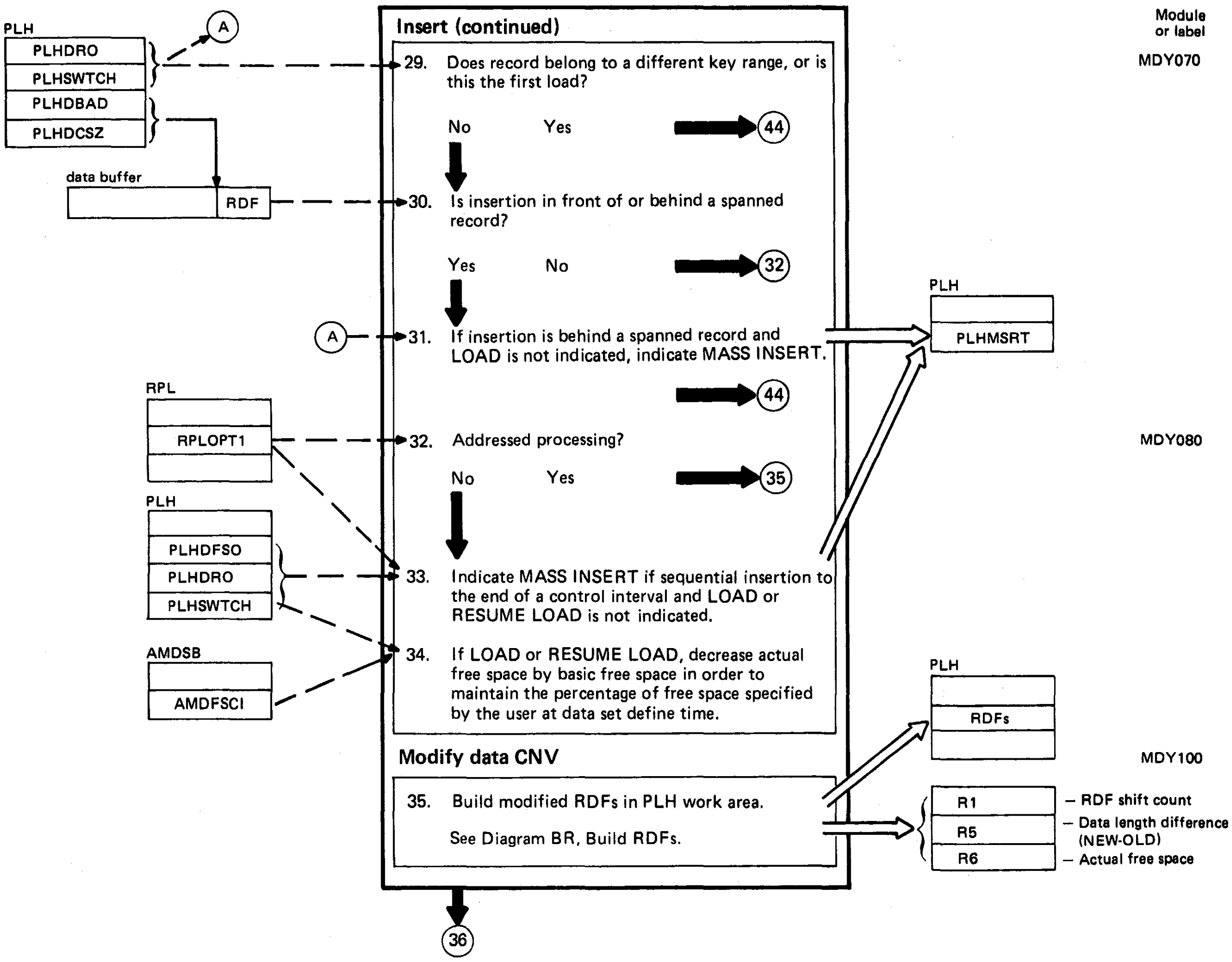
25. Is the data set a key range data set?

26. Is insertion point at the end of the data set or at the beginning of the control interval?

27. Is insertion point at the end of the control interval?

28. If insertion to the end of key range indicate RESUME LOAD.

Diagram BQ4. Modify a Data Control Interval



Insert (continued)

29. Does record belong to a different key range, or is this the first load?

No Yes → 44

30. Is insertion in front of or behind a spanned record?

Yes No → 32

31. If insertion is behind a spanned record and LOAD is not indicated, indicate MASS INSERT.

→ 44

32. Addressed processing?

No Yes → 35

33. Indicate MASS INSERT if sequential insertion to the end of a control interval and LOAD or RESUME LOAD is not indicated.

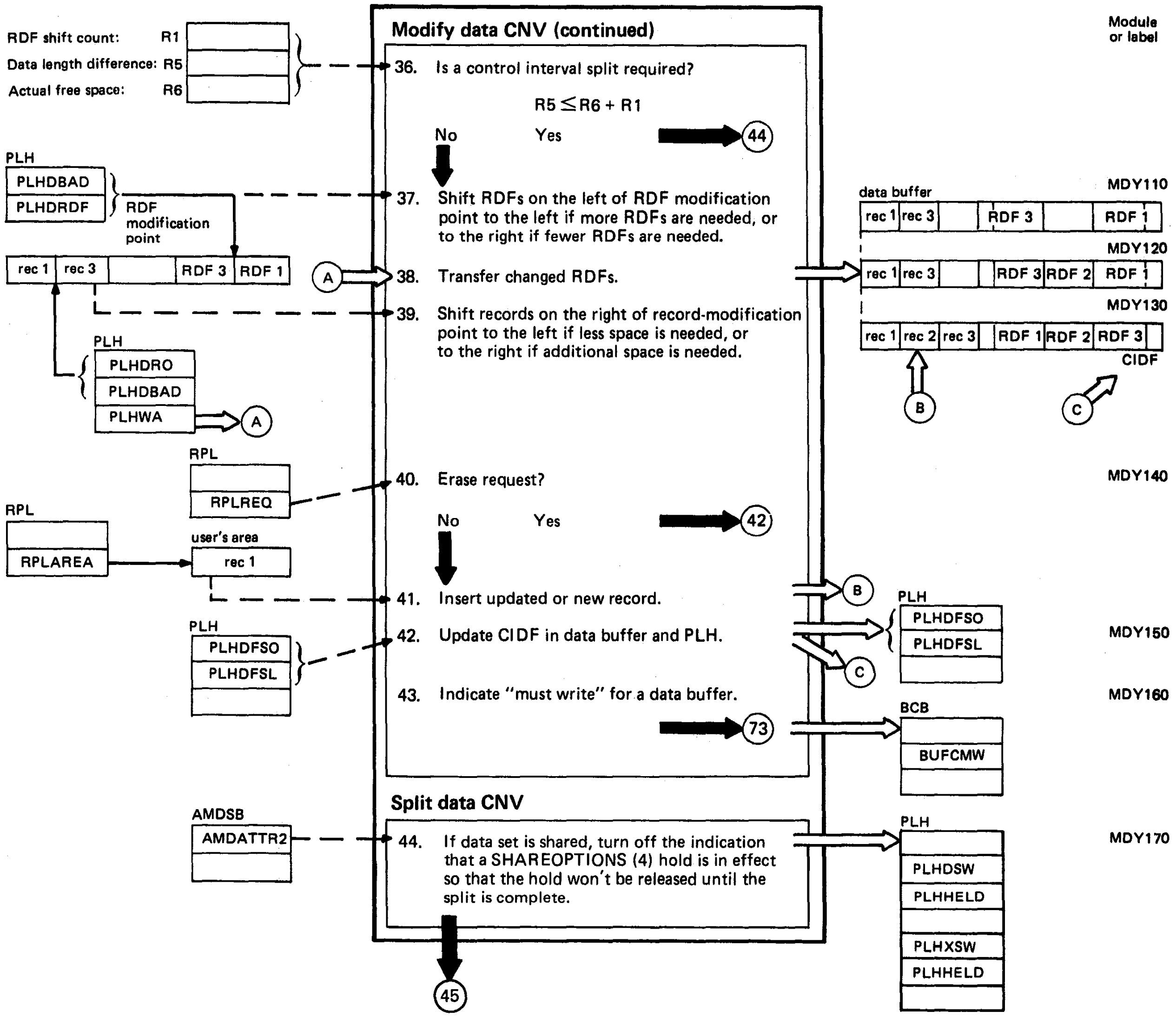
34. If LOAD or RESUME LOAD, decrease actual free space by basic free space in order to maintain the percentage of free space specified by the user at data set define time.

Modify data CNV

35. Build modified RDFs in PLH work area.

See Diagram BR, Build RDFs.

Diagram BQ5. Modify a Data Control Interval



Modify data CNV (continued)

36. Is a control interval split required?

R5 ≤ R6 + R1

No Yes → 44

37. Shift RDFs on the left of RDF modification point to the left if more RDFs are needed, or to the right if fewer RDFs are needed.

38. Transfer changed RDFs.

39. Shift records on the right of record-modification point to the left if less space is needed, or to the right if additional space is needed.

40. Erase request?

No Yes → 42

41. Insert updated or new record.

42. Update CIDF in data buffer and PLH.

43. Indicate "must write" for a data buffer.

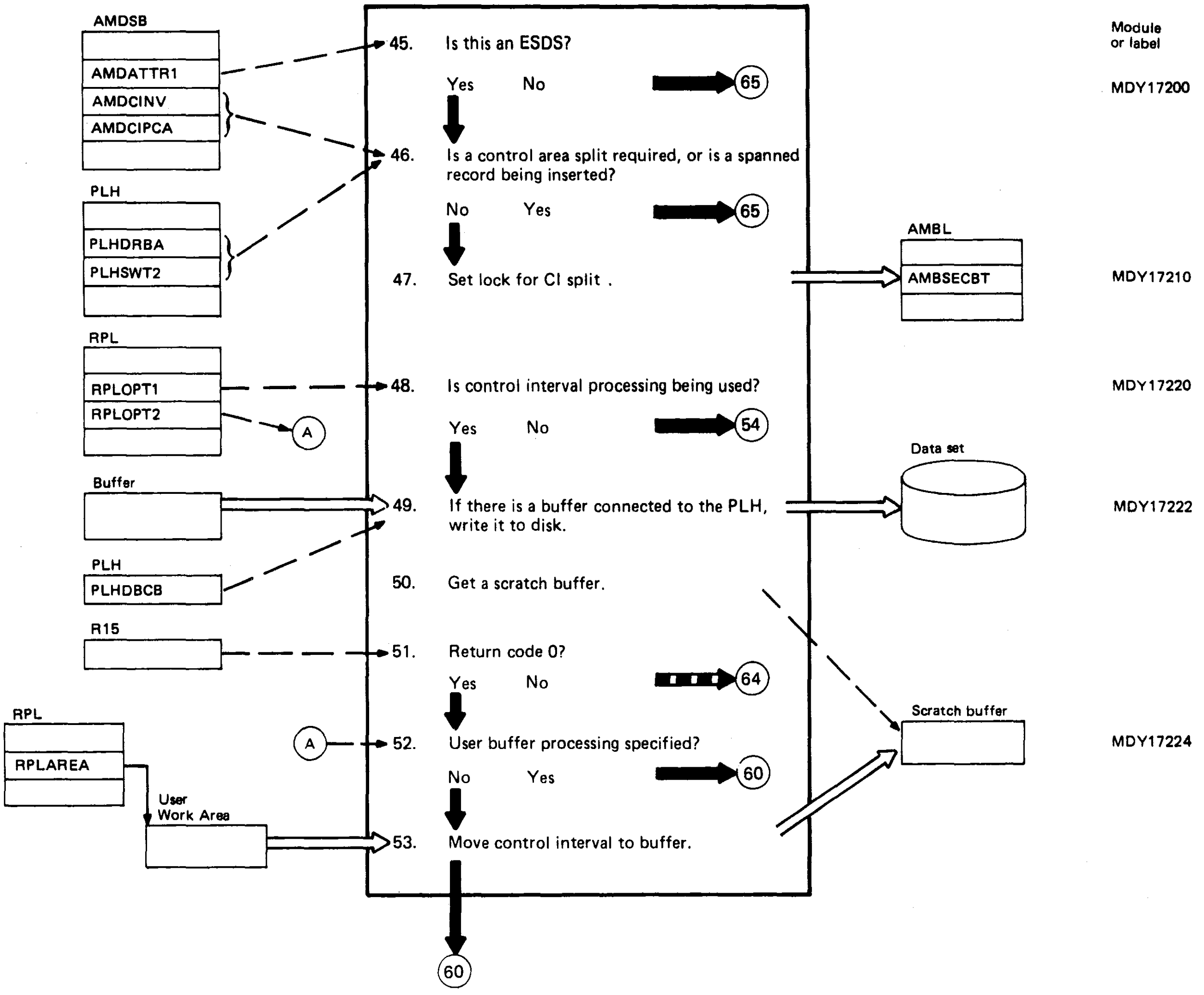
→ 73

Split data CNV

44. If data set is shared, turn off the indication that a SHAREOPTIONS (4) hold is in effect so that the hold won't be released until the split is complete.

→ 45

Diagram BQ6. Modify a Data Control Interval



45. Is this an ESDS?

Yes No

46. Is a control area split required, or is a spanned record being inserted?

No Yes

47. Set lock for CI split .

48. Is control interval processing being used?

Yes No

49. If there is a buffer connected to the PLH, write it to disk.

50. Get a scratch buffer.

51. Return code 0?

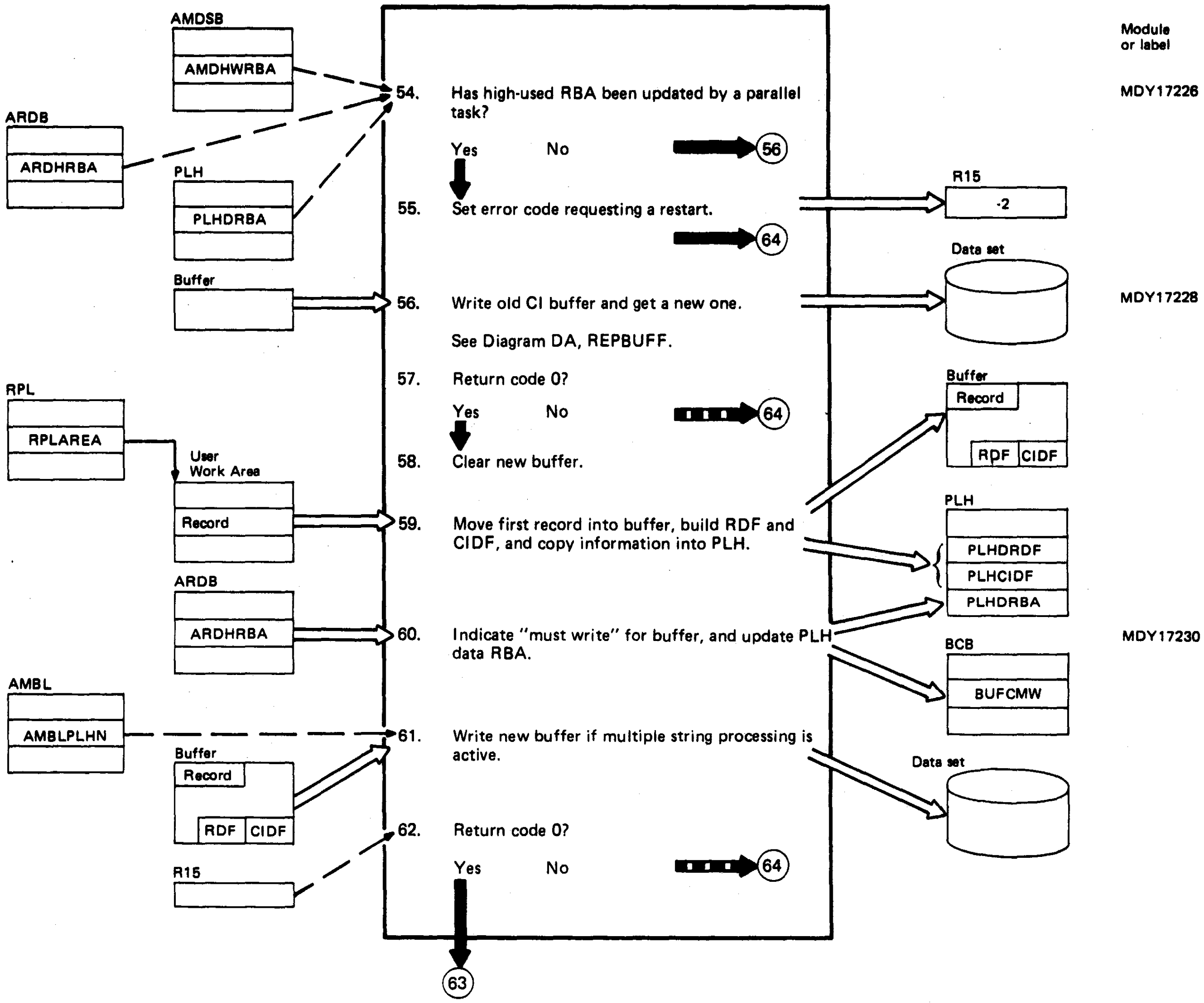
Yes No

52. User buffer processing specified?

No Yes

53. Move control interval to buffer.

Diagram BQ7. Modify a Data Control Interval



54. Has high-used RBA been updated by a parallel task?

Yes No

55. Set error code requesting a restart.

56. Write old CI buffer and get a new one.

See Diagram DA, REPBUFF.

57. Return code 0?

Yes No

58. Clear new buffer.

59. Move first record into buffer, build RDF and CIDF, and copy information into PLH.

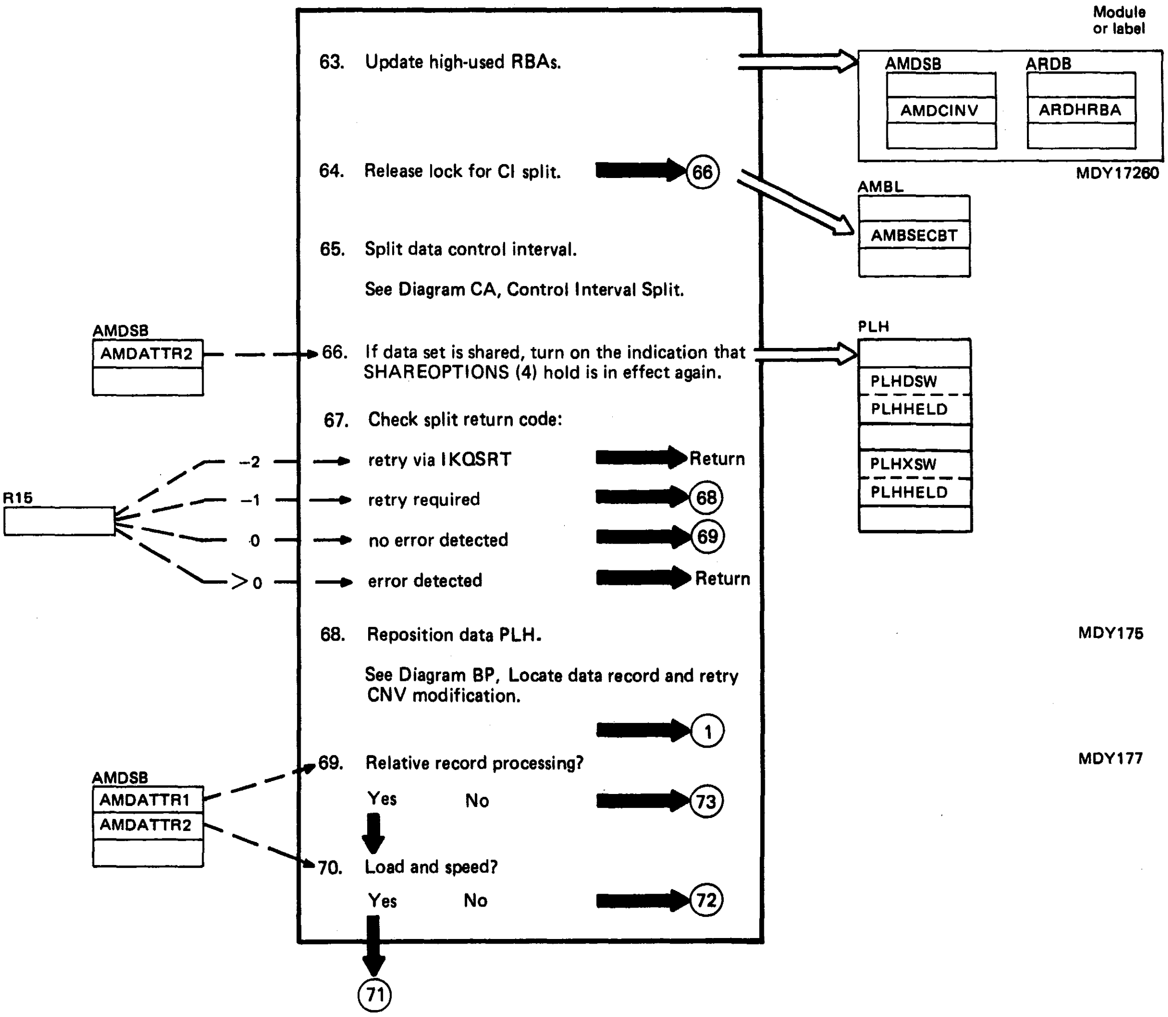
60. Indicate "must write" for buffer, and update PLH data RBA.

61. Write new buffer if multiple string processing is active.

62. Return code 0?

Yes No

Diagram BQ8. Modify a Data Control Interval



63. Update high-used RBAs.

64. Release lock for CI split. → 66

65. Split data control interval.

See Diagram CA, Control Interval Split.

66. If data set is shared, turn on the indication that SHAREOPTIONS (4) hold is in effect again.

67. Check split return code:

-2 → retry via IKQSRT → Return

-1 → retry required → 68

0 → no error detected → 69

>0 → error detected → Return

68. Reposition data PLH.

See Diagram BP, Locate data record and retry CNV modification. → 1

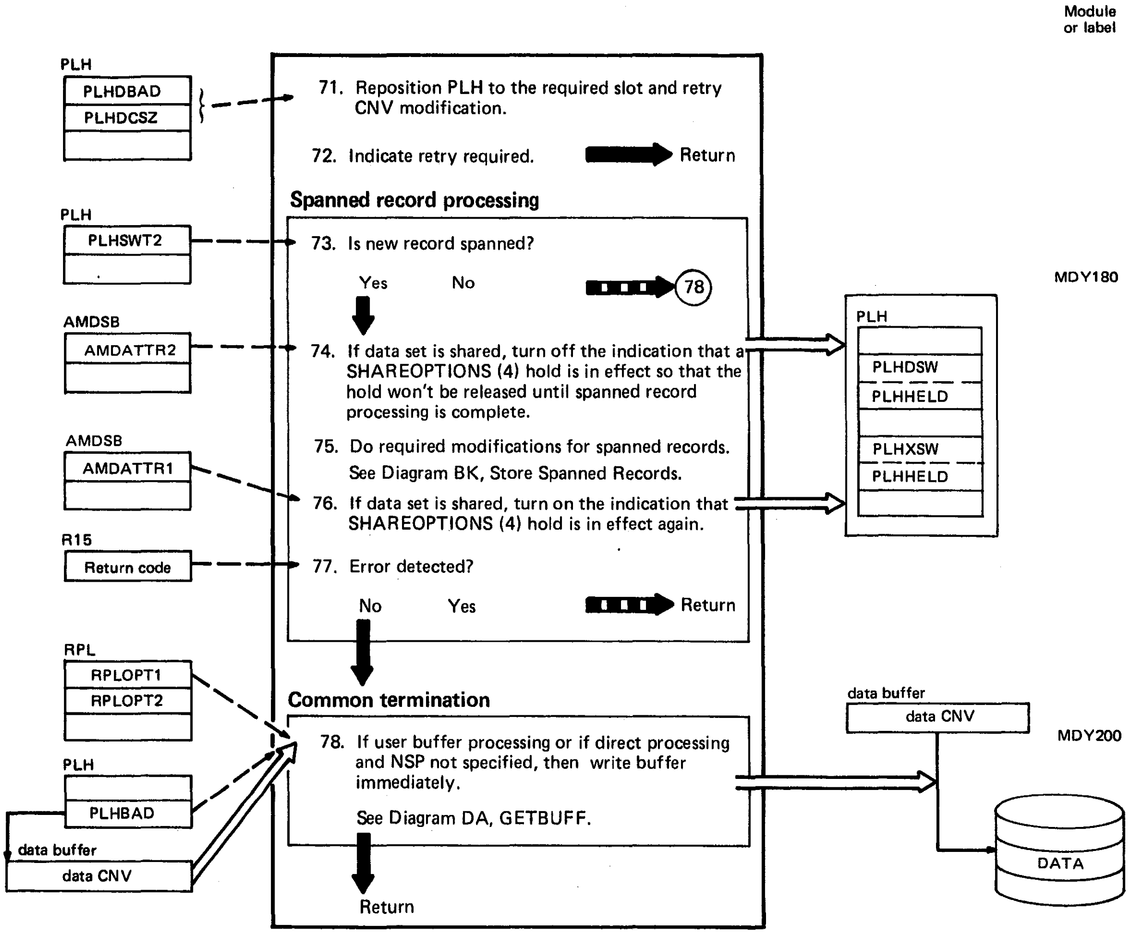
69. Relative record processing?

Yes ↓ No → 73

70. Load and speed?

Yes ↓ 71 No → 72

Diagram BQ9. Modify a Data Control Interval



71. Reposition PLH to the required slot and retry CNV modification.

72. Indicate retry required. → Return

Spanned record processing

73. Is new record spanned?

Yes No → 78

74. If data set is shared, turn off the indication that a SHAREOPTIONS (4) hold is in effect so that the hold won't be released until spanned record processing is complete.

75. Do required modifications for spanned records. See Diagram BK, Store Spanned Records.

76. If data set is shared, turn on the indication that SHAREOPTIONS (4) hold is in effect again.

77. Error detected?

No Yes → Return

Common termination

78. If user buffer processing or if direct processing and NSP not specified, then write buffer immediately.

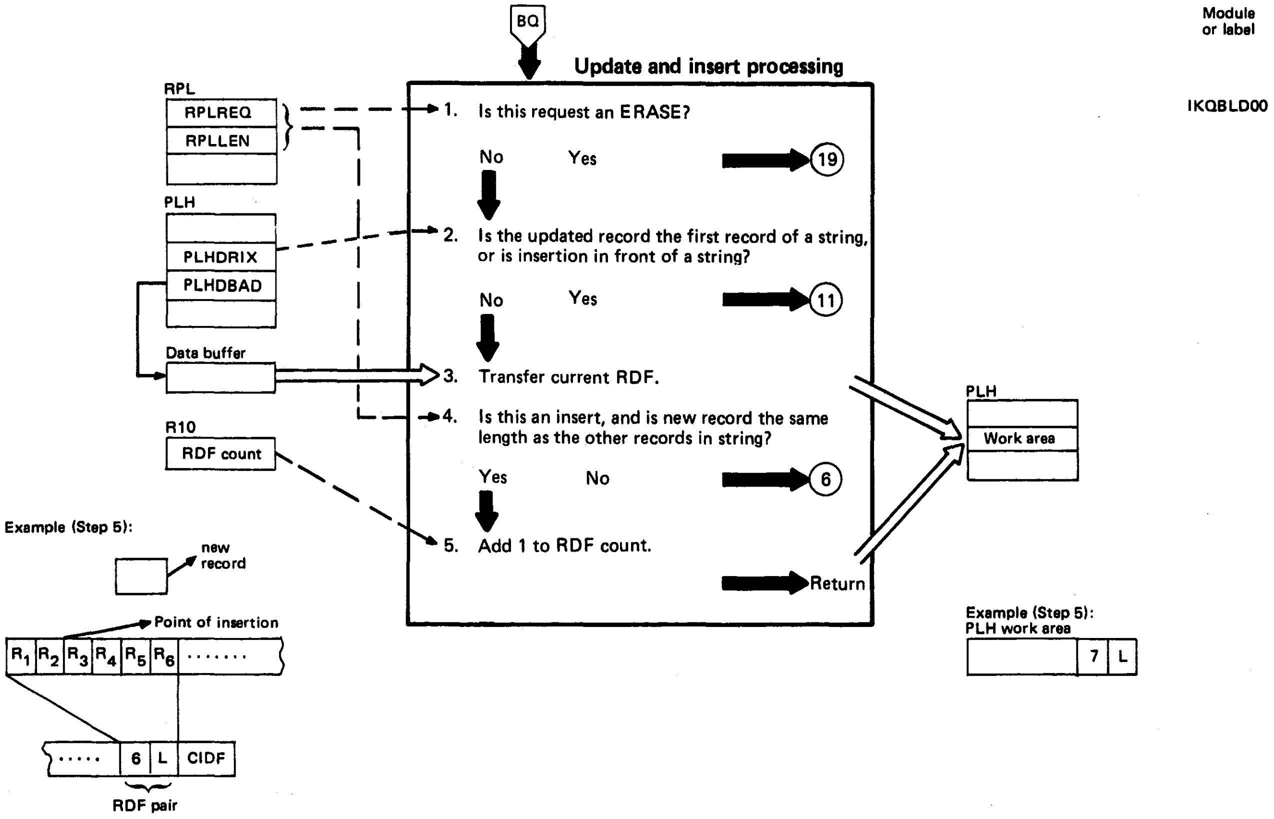
See Diagram DA, GETBUFF.

Return

Notes for Diagram BQ9

78. If the buffer is written immediately and it is for keyed access to a SHAREOPTIONS (4) KSDS, then an additional call is made to the Buffer Manager to release the SHAREOPTIONS (4) hold on the sequence set.

Diagram BR1. Build New and/or Changed RDFs for Nonspanned KSDS and ESDS



Update and insert processing

1. Is this request an ERASE?
2. Is the updated record the first record of a string, or is insertion in front of a string?
3. Transfer current RDF.
4. Is this an insert, and is new record the same length as the other records in string?
5. Add 1 to RDF count.

Notes for Diagram BR1

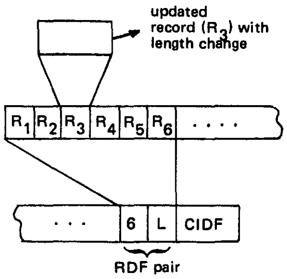
1.-5. RDF processing takes place if the updated record is not the first record of a string or if the insert record is inserted into the middle of a string. In either case, the string must consist of equal-length records.

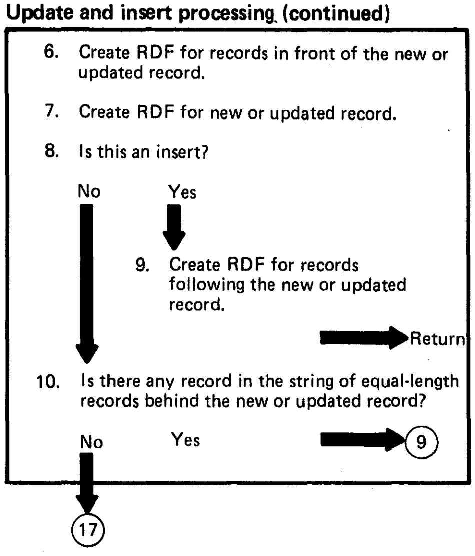
3. If the RDF index is equal to 1, the updated or new record may have the same length as the record(s) described by the previous RDF. For update, the RDF-build routine is entered only if the length of the updated record differs from the length of the original record.

5. If the new record has the same length as the records described by the previous RDF and the RDF index in the PLH is equal to 1, and if the previous RDF does not have an RDF, an RDF count of 2 has to be created for the previous RDF.

Diagram BR2. Build New and/or Changed RDFs for Nonspanned KSDS and ESDS

Example (Steps 6, 7 and 9):


updated record (R_3) with length change
R_1 R_2 R_3 R_4 R_5 R_6 . . . .
. . . 6 L CIDF
RDF pair



Update and insert processing (continued)

6. Create RDF for records in front of the new or updated record.
7. Create RDF for new or updated record.
8. Is this an insert?

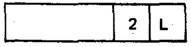
9. Create RDF for records following the new or updated record.

10. Is there any record in the string of equal-length records behind the new or updated record?

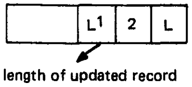
Module or label

BLD010

Example (Step 6):
PLH work area

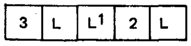


Example (Step 7):
PLH work area



BLD160

Example (Step 9):
PLH work area



Notes for Diagram BR2

6.-10. RDF processing takes place if the record to be updated is in the middle of a string or if the record to be inserted into a string has a length different from the length of the records in the string. In either case, the string must consist of equal-length records and the RDF must be split.

9. RDF processing takes place for records to the right of the new or updated record if the insertion is into the middle of a string of equal-length records and the length of the new record is different from the length of the records in the string or for any update with length change.

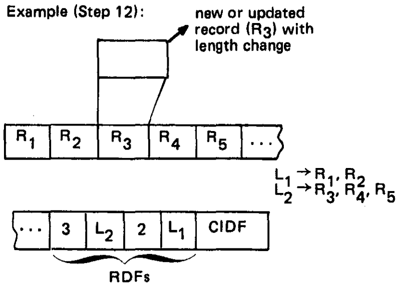
Diagram BR3. Build New and/or Changed RDFs for Nonspanned KSDS and ESDS

Module or label

BLD040

Update and insert processing (continued)

Example (Step 12):



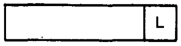
new or updated record (R_3) with length change
R_1
R_2
R_3
R_4
R_5
...
$L_1 \rightarrow R_1, R_2$
$L_2 \rightarrow R_3, R_4, R_5$
...
3
L_2
2
L_1
CIDF
RDFs

11. Can the new or updated record be combined with the preceding string of records (if there is a preceding string of records)?

No ↓ Yes → (14)

12. Create a single-record-length-count RDF for the new or updated record.

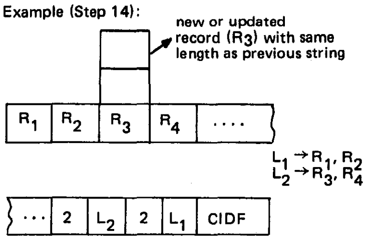
Example (Step 12):
PLH work area



13. Is this an insert or an update?

Insert → (17) Update → (16)

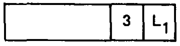
Example (Step 14):



new or updated record (R_3) with same length as previous string
R_1
R_2
R_3
R_4
....
$L_1 \rightarrow R_1, R_2$
$L_2 \rightarrow R_3, R_4$
...
2
L_2
2
L_1
CIDF

14. Attach new or updated record to previous string of equal-length records.

Example (Step 14):
PLH work area



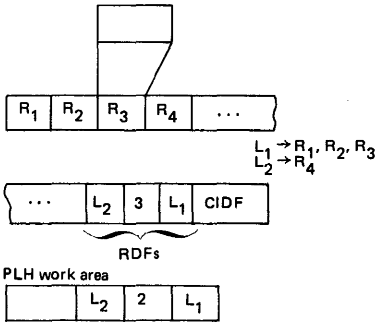
15. Is this an insert or an update?

Update ↓ Insert → Return

16. Does the current RDF have an RDF count?

No ↓ Yes → (9)

Example (Step 18):



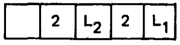
17. Can the last RDF in the work area be combined with the next RDF (if there is a next RDF)?

Yes ↓ No → Return

18. Combine last RDF in work area with next RDF.

↓ Return

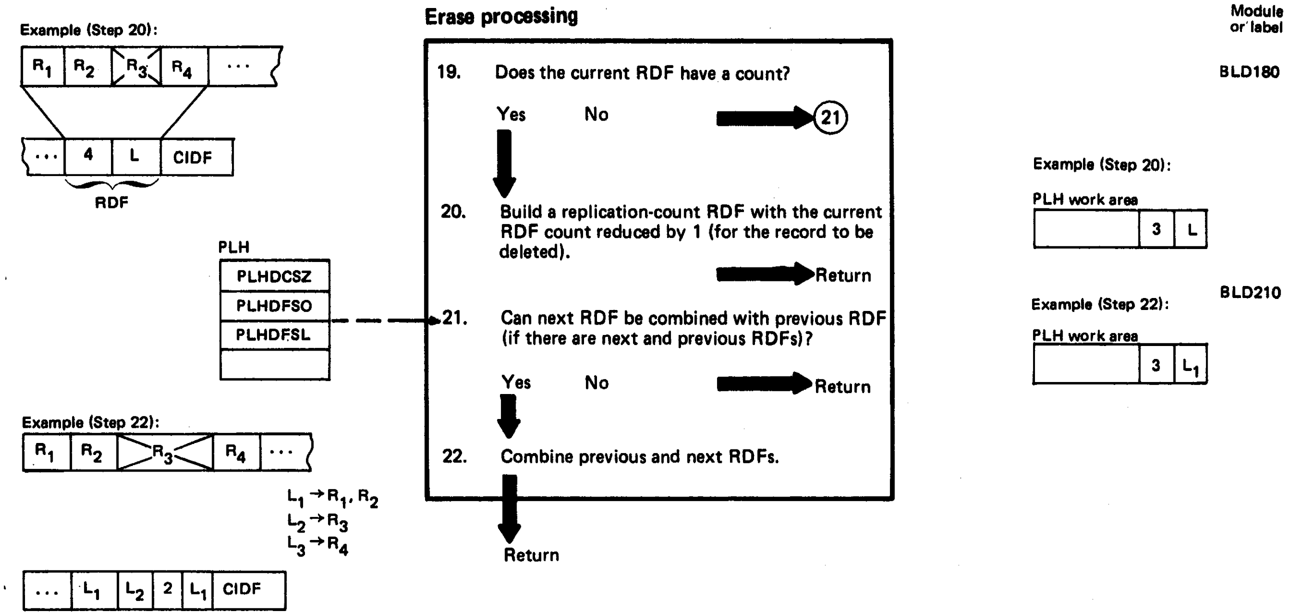
Example (Step 18):
PLH work area



Notes for Diagram BR3

11.-18. RDF processing takes place if the first and possibly only record in a string is updated or if a record has to be inserted in front of the string. In either case, the string must consist of equal-length records.

Diagram BR4. Build New and/or Changed RDFs for Nonspanned KSDS and ESDS



Erase processing

19. Does the current RDF have a count?

20. Build a replication-count RDF with the current RDF count reduced by 1 (for the record to be deleted).

21. Can next RDF be combined with previous RDF (if there are next and previous RDFs)?

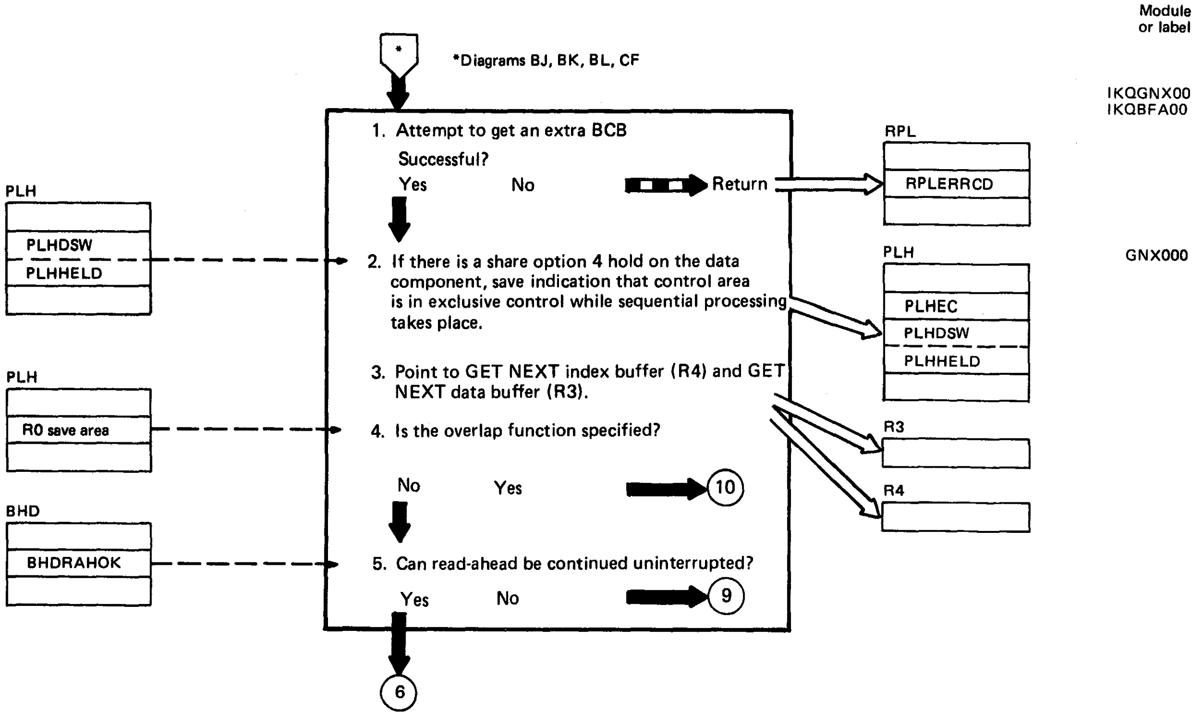
22. Combine previous and next RDFs.

Notes for Diagram BR4

19.-20. For an ERASE request, the RDF-count of the RDF describing the record to be erased is reduced by 1 if the count was previously greater than 1. If the count was 2, the new RDF will not have a replication RDF.

21.-22. If the record to be erased was described by an RDF without count (the record being the only one of that length), and if the neighboring RDFs have the same length count (that is, represent records of the same length), the RDFs can be combined to a single RDF with replication count.

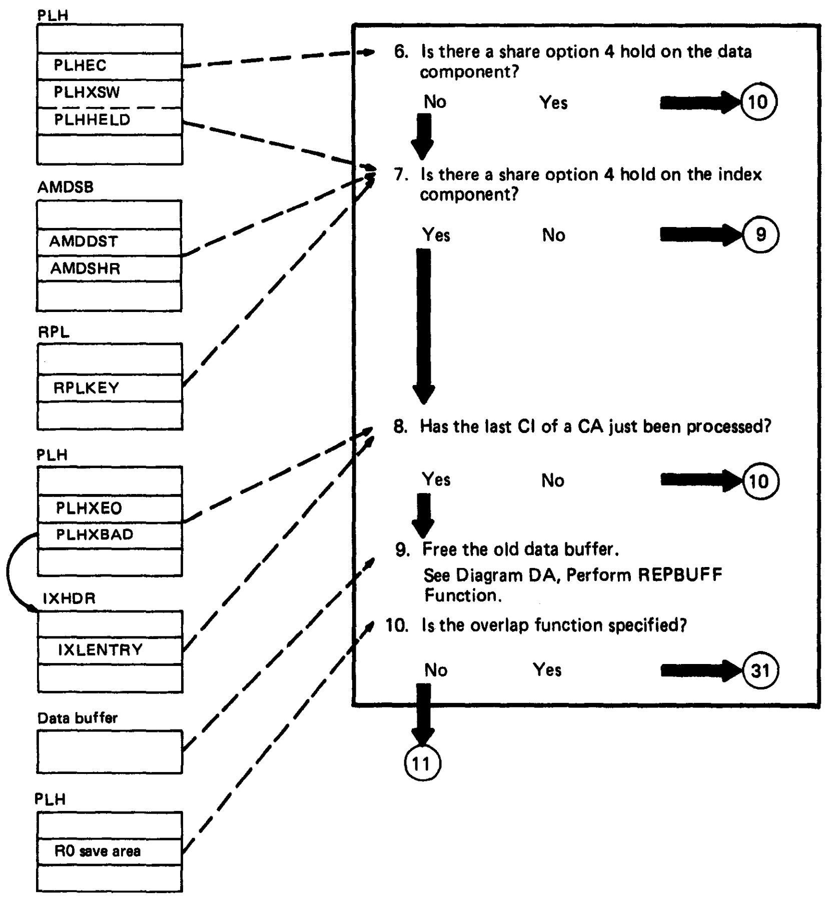
Diagram BS1. GET NEXT: Get Next Buffer and Read Ahead



*Diagrams BJ, BK, BL, CF

1. Attempt to get an extra BCB
   Successful?
   Yes — No: Return
2. If there is a share option 4 hold on the data component, save indication that control area is in exclusive control while sequential processing takes place.
3. Point to GET NEXT index buffer (R4) and GET NEXT data buffer (R3).
4. Is the overlap function specified?
   No — Yes: 10
5. Can read-ahead be continued uninterrupted?
   Yes: 6 — No: 9

Diagram BS2. GET NEXT: Get Next Buffer and Read Ahead



Module or label

6. Is there a share option 4 hold on the data component?
   No Yes → 10

7. Is there a share option 4 hold on the index component?
   Yes No → 9

8. Has the last CI of a CA just been processed?
   Yes No → 10 — GNX005

9. Free the old data buffer. — IKQBFA10
   See Diagram DA, Perform REPBUFF Function.

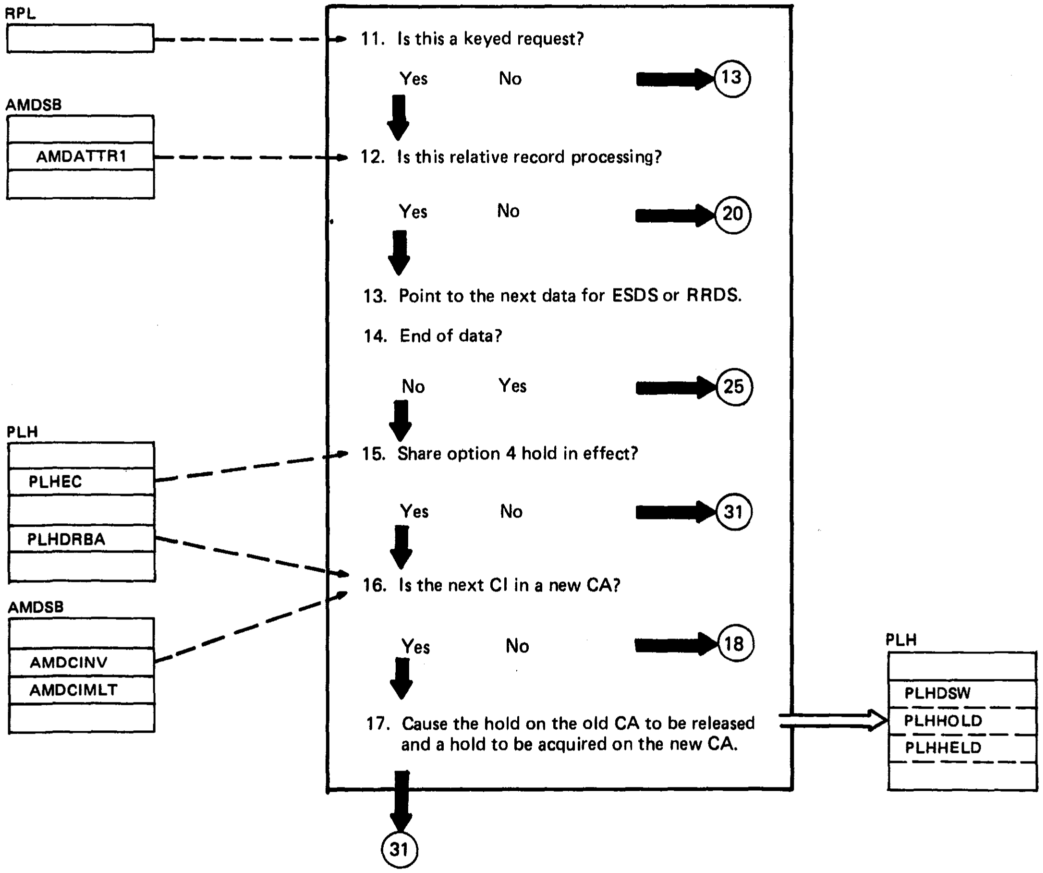
10. Is the overlap function specified? — GNX007
    No Yes → 31

Notes for Diagram BS2

6. If there is a share option 4 hold in effect, the old data buffer will not be freed until the next data CI is read. If an update write is specified for the old data buffer, the write will be combined with the read of the next CI (as happens anyway for non-share option 4).

7.-8. See note for step 6.

Diagram BS3. GET NEXT: Get Next Buffer and Read Ahead



11. Is this a keyed request?

12. Is this relative record processing?

13. Point to the next data for ESDS or RRDS.

14. End of data?

15. Share option 4 hold in effect?

16. Is the next CI in a new CA?

17. Cause the hold on the old CA to be released and a hold to be acquired on the new CA.

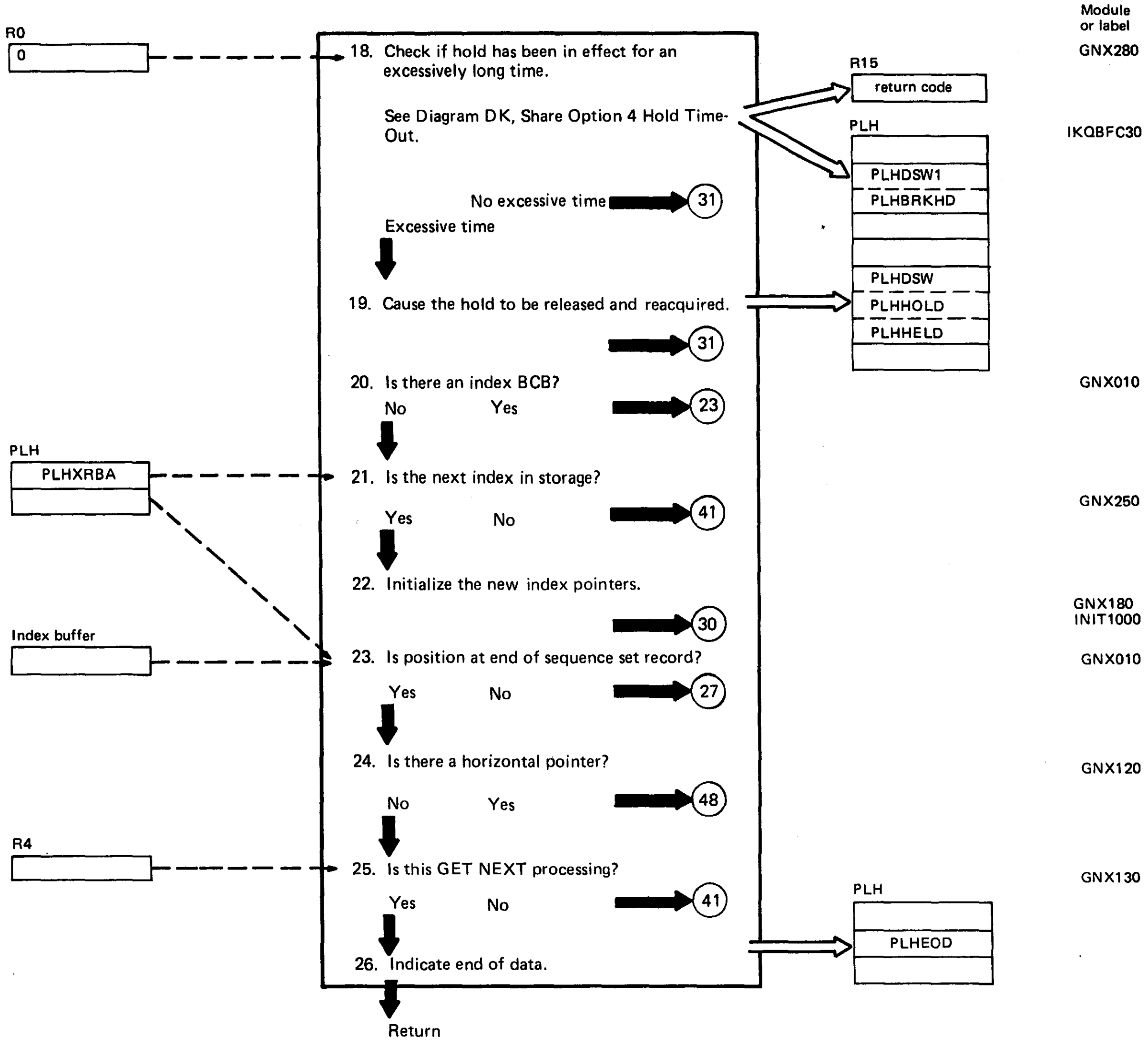
Module or label

GNX010

GNX260
PTAND000

GNX270

Diagram BS4. GET NEXT: Get Next Buffer and Read Ahead



18. Check if hold has been in effect for an excessively long time.

See Diagram DK, Share Option 4 Hold Time-Out.

No excessive time → 31

Excessive time

19. Cause the hold to be released and reacquired. → 31

20. Is there an index BCB?

No Yes → 23

21. Is the next index in storage?

Yes No → 41

22. Initialize the new index pointers. → 30

23. Is position at end of sequence set record?

Yes No → 27

24. Is there a horizontal pointer?

No Yes → 48

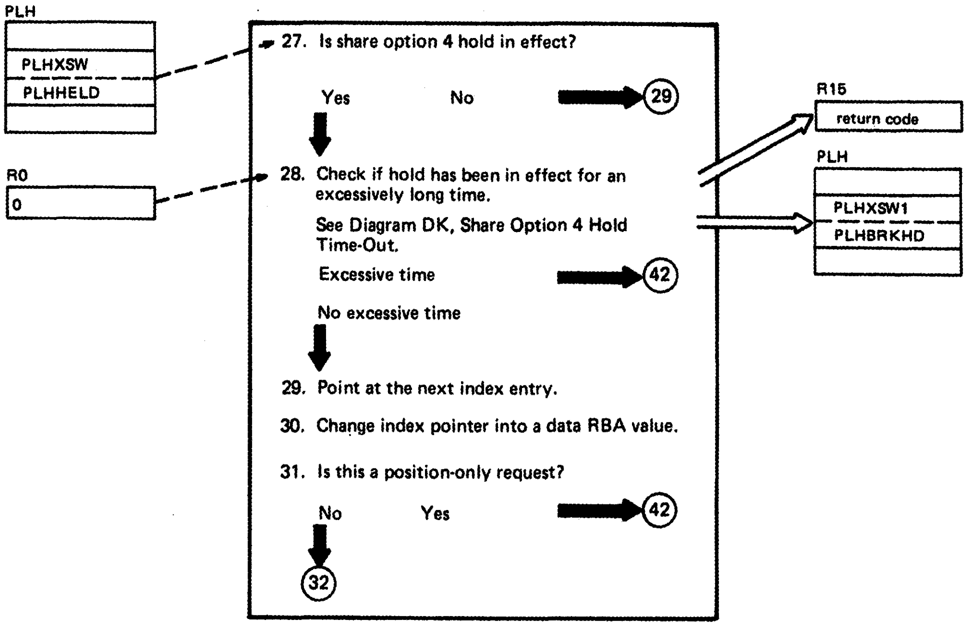
25. Is this GET NEXT processing?

Yes No → 41

26. Indicate end of data.

Return

Diagram BS5. GET NEXT: Get Next Buffer and Read Ahead



27. Is share option 4 hold in effect?

Yes No → 29

28. Check if hold has been in effect for an excessively long time.

See Diagram DK, Share Option 4 Hold Time-Out.

Excessive time → 42

No excessive time

29. Point at the next index entry.

30. Change index pointer into a data RBA value.

31. Is this a position-only request?

No Yes → 42

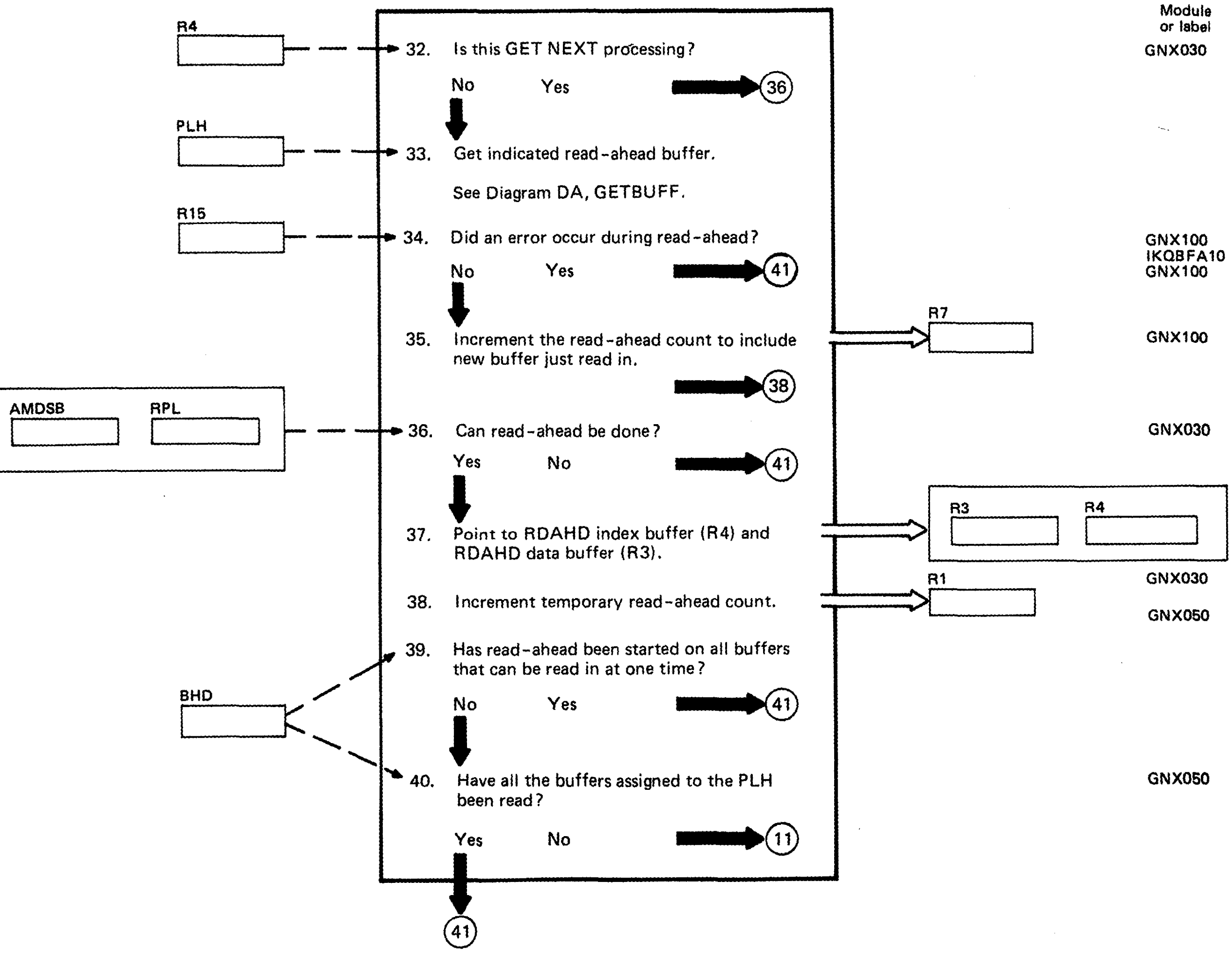
No → 32

M
or

IKQ

ADV
GNX
PTA
GNX

iagram BS6. GET NEXT: Get Next Buffer and Read Ahead



32. Is this GET NEXT processing? No / Yes → 36

33. Get indicated read-ahead buffer. See Diagram DA, GETBUFF.

34. Did an error occur during read-ahead? No / Yes → 41

35. Increment the read-ahead count to include new buffer just read in. → 38

36. Can read-ahead be done? Yes / No → 41

37. Point to RDAHD index buffer (R4) and RDAHD data buffer (R3).

38. Increment temporary read-ahead count.

39. Has read-ahead been started on all buffers that can be read in at one time? No / Yes → 41

40. Have all the buffers assigned to the PLH been read? Yes → 41 / No → 11

for Diagram BS6

d-ahead is prohibited under any one of the following conditions:

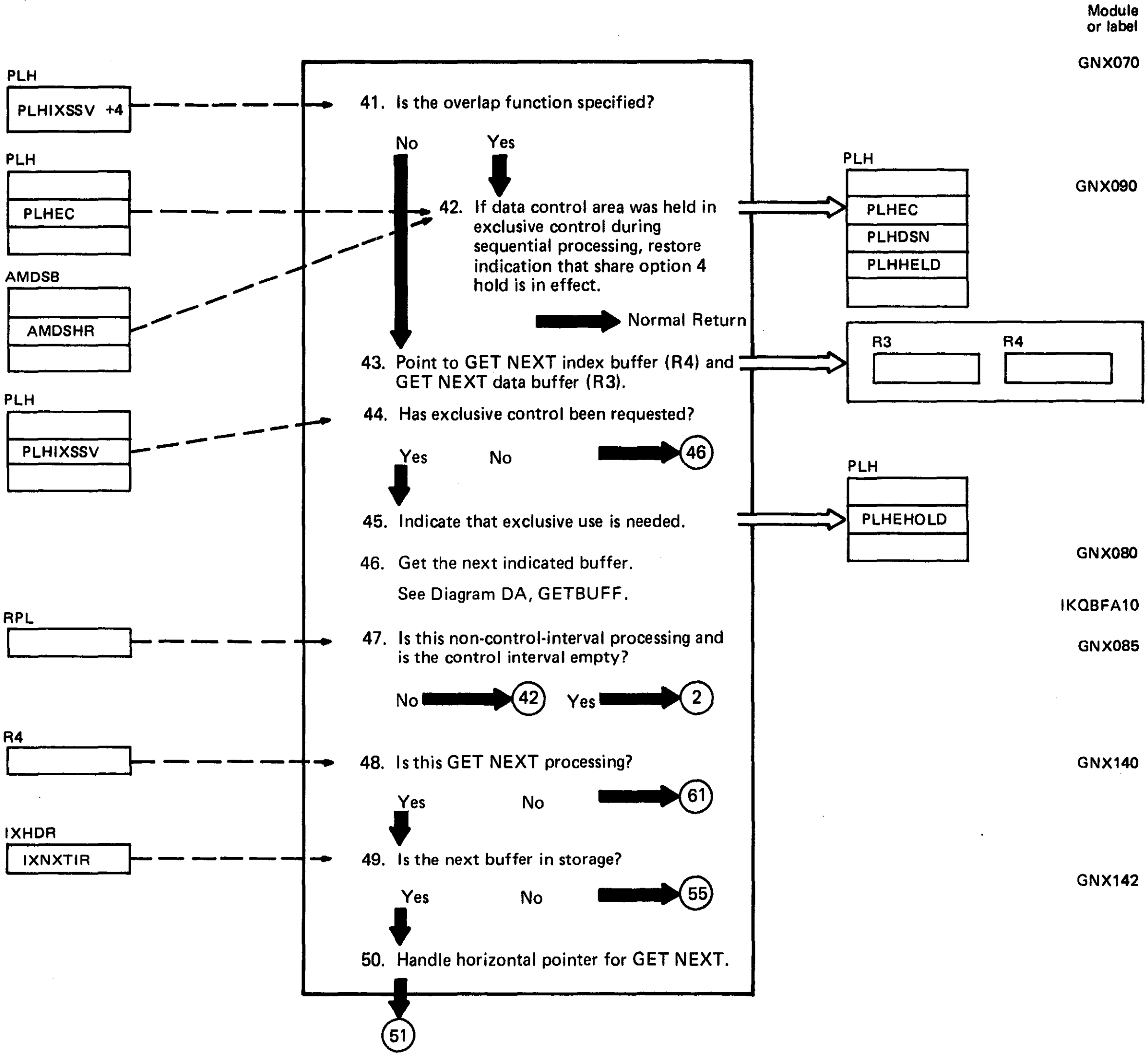
User buffers

Data set is a catalog

Share option 4

Data set is an index component opened by itself, and it includes both fixed block and CKD volumes.

Diagram BS7. GET NEXT: Get Next Buffer and Read Ahead



41. Is the overlap function specified?

42. If data control area was held in exclusive control during sequential processing, restore indication that share option 4 hold is in effect.

43. Point to GET NEXT index buffer (R4) and GET NEXT data buffer (R3).

44. Has exclusive control been requested?

45. Indicate that exclusive use is needed.

46. Get the next indicated buffer.

See Diagram DA, GETBUFF.

47. Is this non-control-interval processing and is the control interval empty?

48. Is this GET NEXT processing?

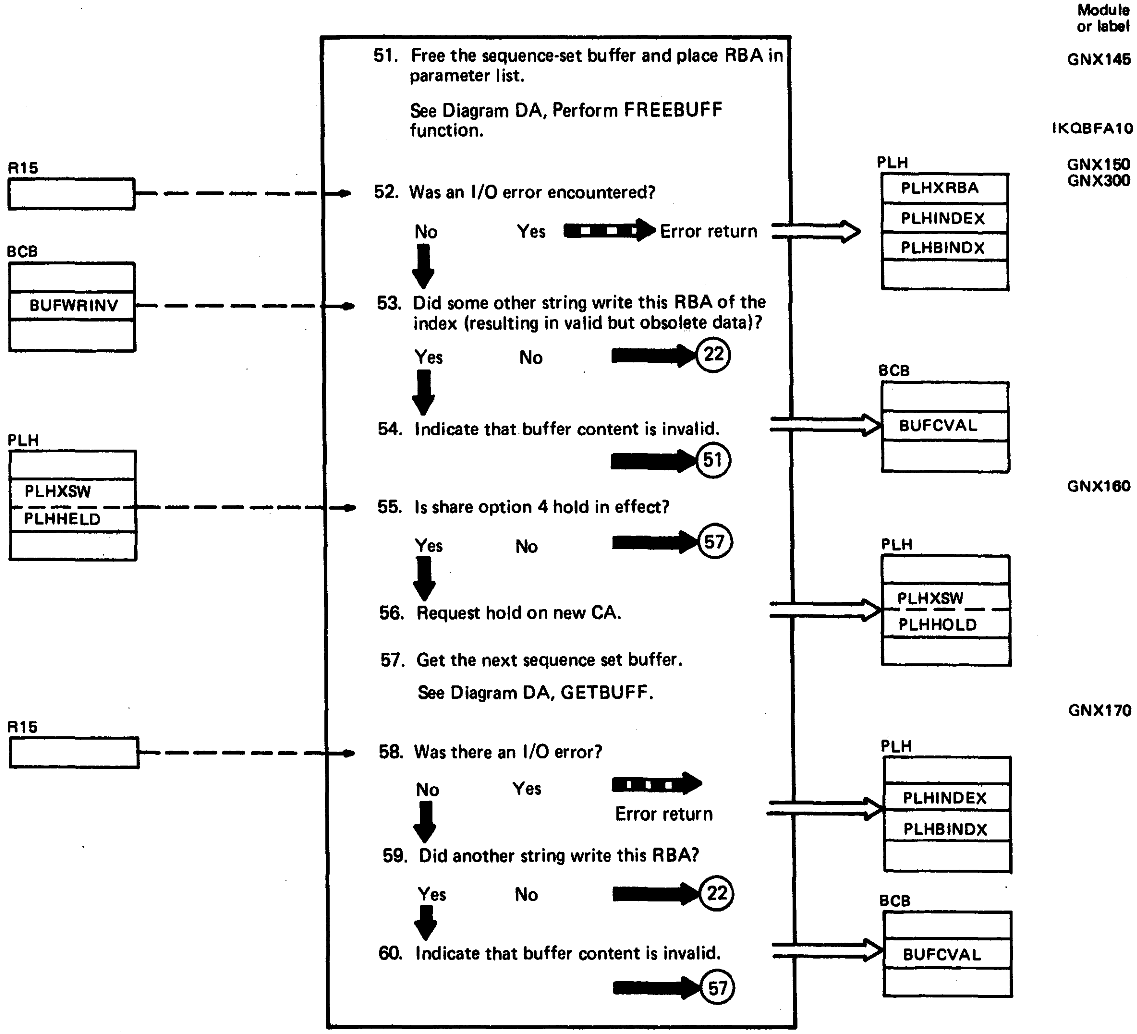
49. Is the next buffer in storage?

50. Handle horizontal pointer for GET NEXT.

Notes for Diagram BS7

42. Flag PLHKGEHD in PLHXSW1 (that could have been set on by IKQGPT under keyed access to a KSDS) is always cleared if the data set is share option 4.

Diagram BS8. GET NEXT: Get Next Buffer and Read Ahead



51. Free the sequence-set buffer and place RBA in parameter list.

    See Diagram DA, Perform FREEBUFF function.

52. Was an I/O error encountered?

    No Yes → Error return

53. Did some other string write this RBA of the index (resulting in valid but obsolete data)?

    Yes No → 22

54. Indicate that buffer content is invalid. → 51

55. Is share option 4 hold in effect?

    Yes No → 57

56. Request hold on new CA.

57. Get the next sequence set buffer.

    See Diagram DA, GETBUFF.

58. Was there an I/O error?

    No Yes → Error return

59. Did another string write this RBA?

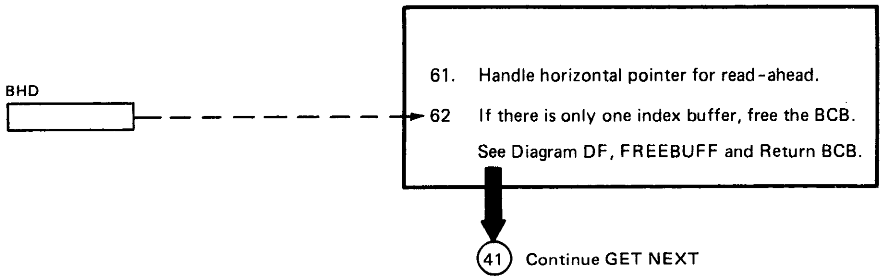
    Yes No → 22

60. Indicate that buffer content is invalid. → 57

Notes for Diagram BS8

56. The hold on the old CA will be released when the hold on the new CA is acquired.

Diagram BS9. GET NEXT: Get Next Buffer and Read Ahead



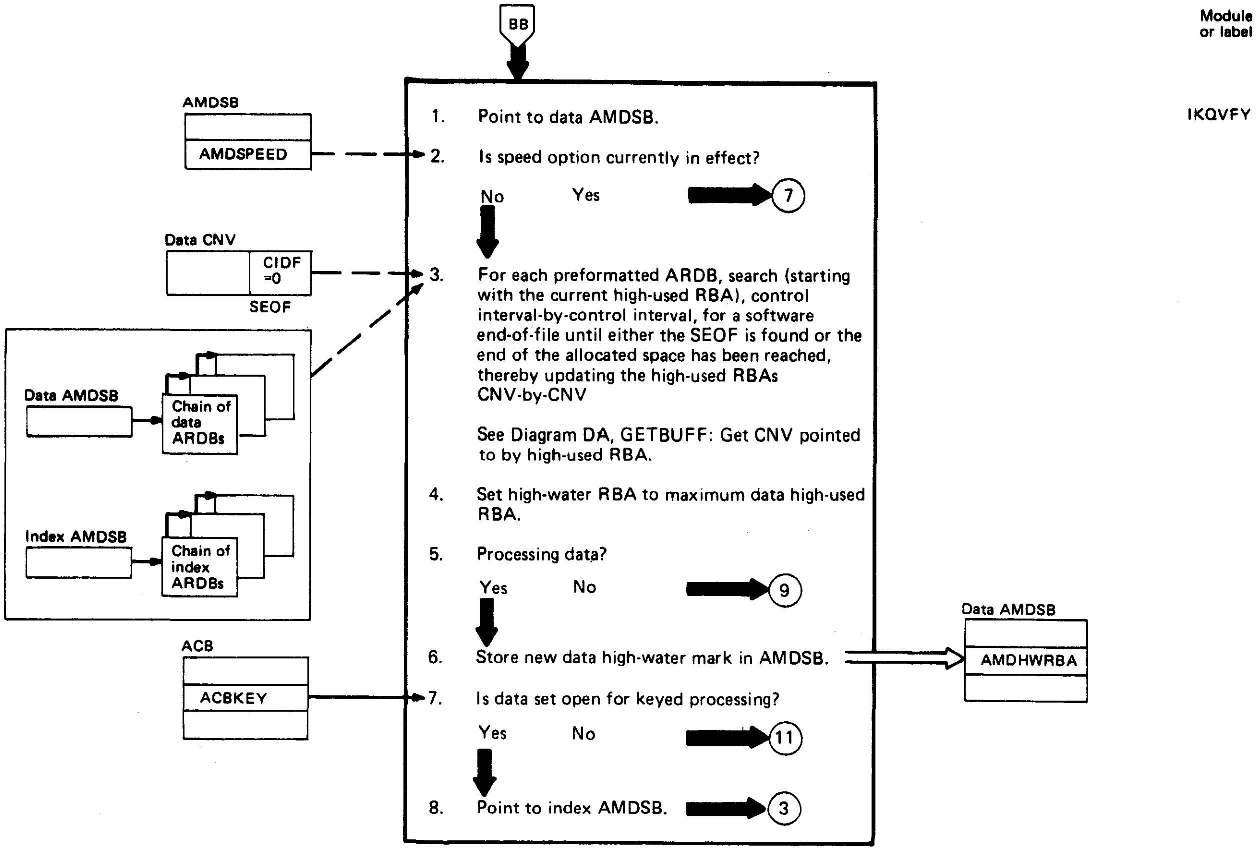
Module or label

61. Handle horizontal pointer for read-ahead. GNX200

62 If there is only one index buffer, free the BCB.

See Diagram DF, FREEBUFF and Return BCB. GNX240 IKQBFA20

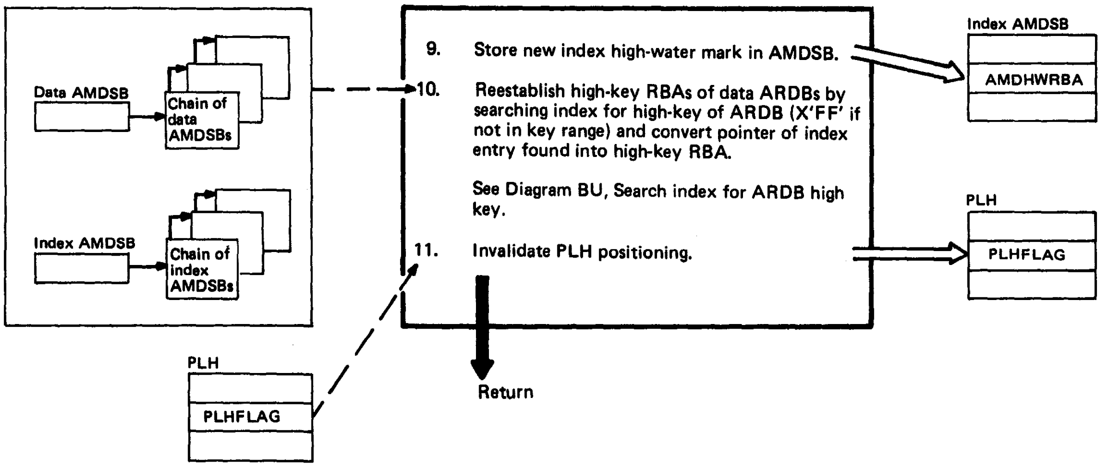
Diagram BT1. VERIFY: Reestablish High-used and High-key RBAs



1. Point to data AMDSB.
2. Is speed option currently in effect?
3. For each preformatted ARDB, search (starting with the current high-used RBA), control interval-by-control interval, for a software end-of-file until either the SEOF is found or the end of the allocated space has been reached, thereby updating the high-used RBAs CNV-by-CNV

   See Diagram DA, GETBUFF: Get CNV pointed to by high-used RBA.
4. Set high-water RBA to maximum data high-used RBA.
5. Processing data?
6. Store new data high-water mark in AMDSB.
7. Is data set open for keyed processing?
8. Point to index AMDSB.

Diagram BT2. VERIFY: Reestablish High-used and High-key RBAs



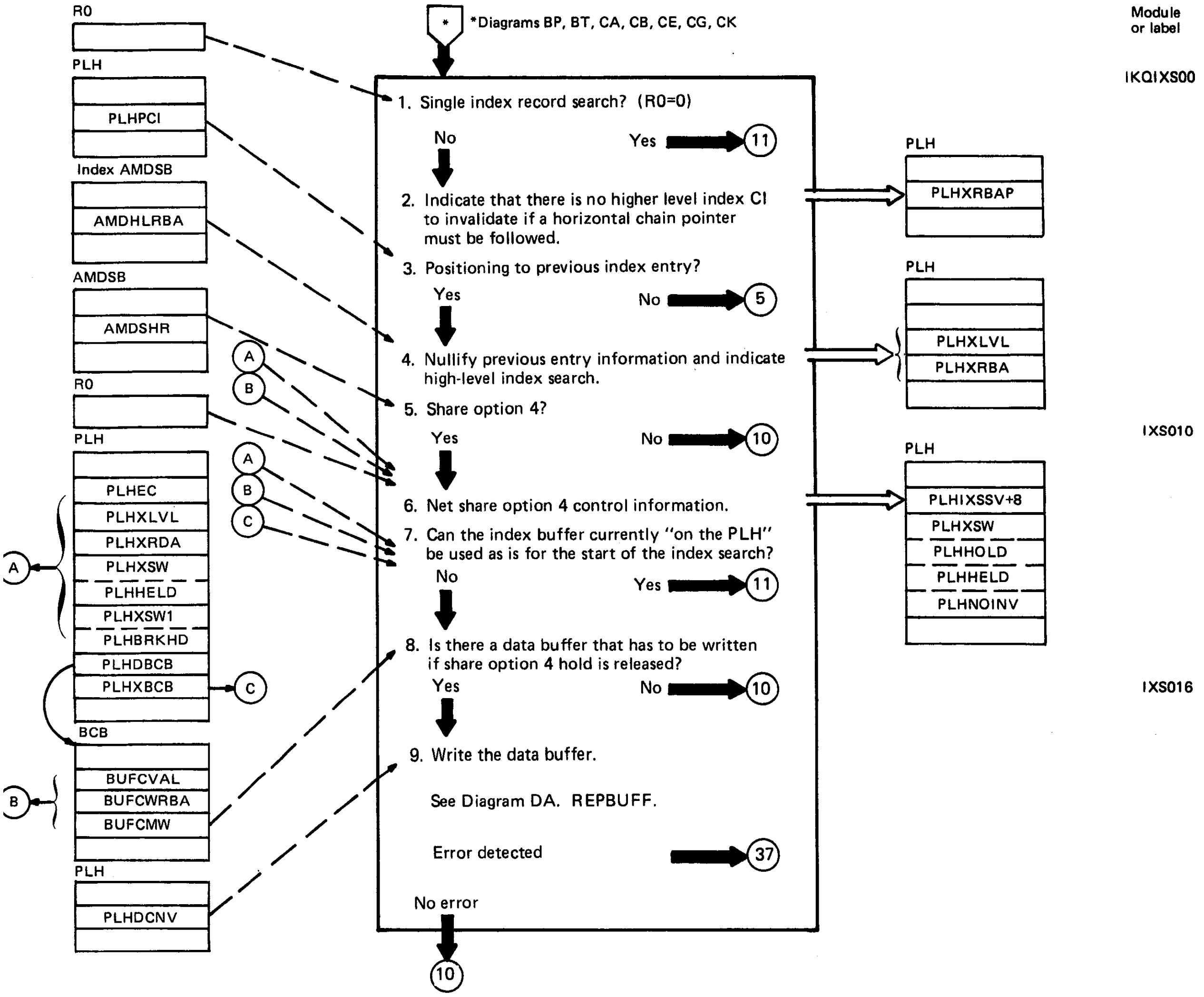
9. Store new index high-water mark in AMDSB.

10. Reestablish high-key RBAs of data ARDBs by searching index for high-key of ARDB (X'FF' if not in key range) and convert pointer of index entry found into high-key RBA.

See Diagram BU, Search index for ARDB high key.

11. Invalidate PLH positioning.

Diagram BU1. Search Index



*Diagrams BP, BT, CA, CB, CE, CG, CK

1. Single index record search? (R0=0)
   No / Yes → 11
2. Indicate that there is no higher level index CI to invalidate if a horizontal chain pointer must be followed.
3. Positioning to previous index entry?
   Yes / No → 5
4. Nullify previous entry information and indicate high-level index search.
5. Share option 4?
   Yes / No → 10
6. Net share option 4 control information.
7. Can the index buffer currently "on the PLH" be used as is for the start of the index search?
   No / Yes → 11
8. Is there a data buffer that has to be written if share option 4 hold is released?
   Yes / No → 10
9. Write the data buffer.

   See Diagram DA. REPBUFF.

   Error detected → 37

   No error → 10

Diagram BU2. Search Index

Notes for Diagram BU1

Normal index search:
An index record is searched for the entry containing a key that is not low compared with the search argument. The search starts with the low-key entry and ends when the required entry is found. A search is done in two steps: first the section containing the desired entry is located (steps 11-15), then the entry within the section is identified (steps 21-25).

If R0=0 only the index record attached to the PLH (PLHXBAD) is searched. No I/O is done.

If R0 > 0 its value identifies the index level to stop on, and PLHXRBA contains the RBA of the index record to start with. The "desired index entry" in high-level index records contains a relative pointer to the index record one level deeper, which must be searched next. If R0=1 the index hierarchy is searched down to the sequence set level, which is the deepest index level. The "desired entry" in the sequence set contains a relative pointer to the data control interval that contains (if inserted) the data record with the desired key.

The high-key entry is identified by a key with length 0.

Output of index search is the PLH positioned to the "desired index entry" on the desired level. If the "desired entry" is complex, (i.e. for a spanned record) the PLH is positioned to the leftmost subentry.

Index search for the previous index entry:
During sequential backward processing, whenever the low-key entry in a sequence set record was reached, the previous index entry in the next (low-key direction) sequence set record must be located.

This is achieved by a normal top-down index search for the previous request key, during which previous entry information is saved whenever the inspected index entry is not the "desired entry" (steps 13 and 23).

Normally the saved previous index entry at the end of the normal top-down index search for the previous request key will lie in a higher level index; so a secondary index search is started (steps 32 and 33), with the index record identified by the previous entry information. The secondary search locates the high-key index entry in the next lower sequence set record, which is the previous key.

The end of data set (in backward direction) is reached when no previous entry information was stored during the normal index search for the previous request key.

2. The PLHXREAP field is set to minus one to indicate no significant contents. At any stage in the search, PLHXREAP is used to remember which higher-level index record pointed to the one that is currently being searched. If the one currently searched does not include the desired key, then the higher-level pointer is invalid, and any buffer still containing the higher-level index CI will be invalidated. This use of PLHXRBAP does not apply to backward sequential processing (PLHPCI is on).

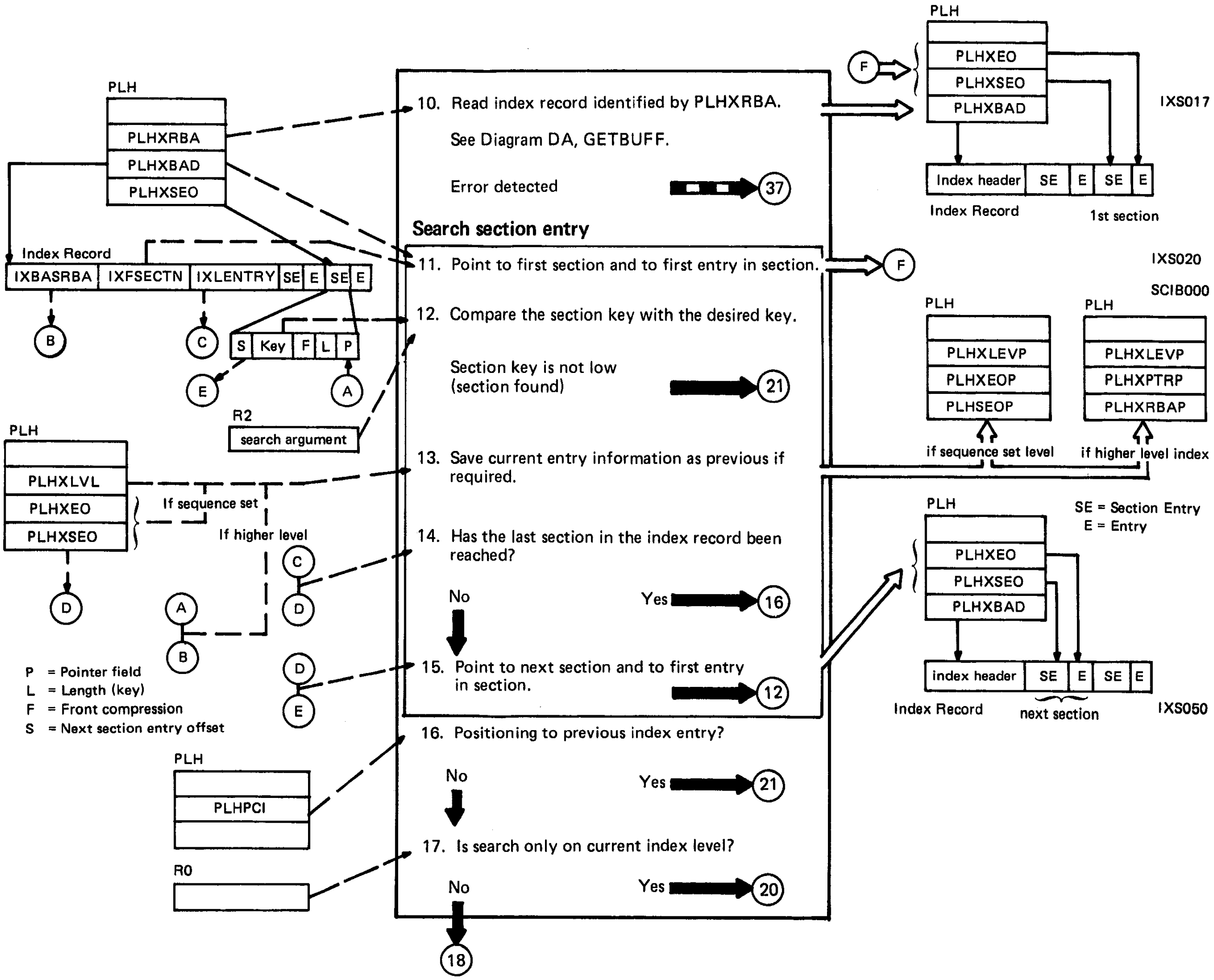
6. The setting of the share option 4 control information provides the following functions:
   - Allows share option 4 buffer invalidation to be suppressed on all retrievals of index CIs, except at the level-to-stop-on (usually the sequence set).
   - Acquires a share option 4 hold on the sequence set when required and releases a share option 4 hold when necessary.
   - Allows a share option 4 hold to be (optionally) retained without interruption if the search terminates at (one of) the same sequence set CI(s) that was held when IKQIXS was entered.

7. This decision allows I/O to be completely bypassed under the following conditions:
   - The search starts at the sequence set level and terminates in the same sequence set CI that is already on the PLH; and
   - There is an active hold that is to be retained without interruption.

   In this case, skip-sequential GET-for-update does not usually require any more I/O under share option 4 than under other share options.

   Minus one is set into the caller's return register 0 to indicate that the hold was retained without release.

Diagram BU3. Search Index



10. Read index record identified by PLHXRBA.

See Diagram DA, GETBUFF.

Error detected → 37

Search section entry

11. Point to first section and to first entry in section.

12. Compare the section key with the desired key.

Section key is not low (section found) → 21

13. Save current entry information as previous if required.

14. Has the last section in the index record been reached?

No / Yes → 16

15. Point to next section and to first entry in section. → 12

16. Positioning to previous index entry?

No / Yes → 21

17. Is search only on current index level?

No → 18 / Yes → 20

Notes for Diagram BU3

12. Front key compression:

It is not necessary to expand the compressed key in the index entry to a full key before doing the comparison in this step. Instead, a "cumulative compression count" is initialized to zero before the first iteration through this step, and the following procedure performed.

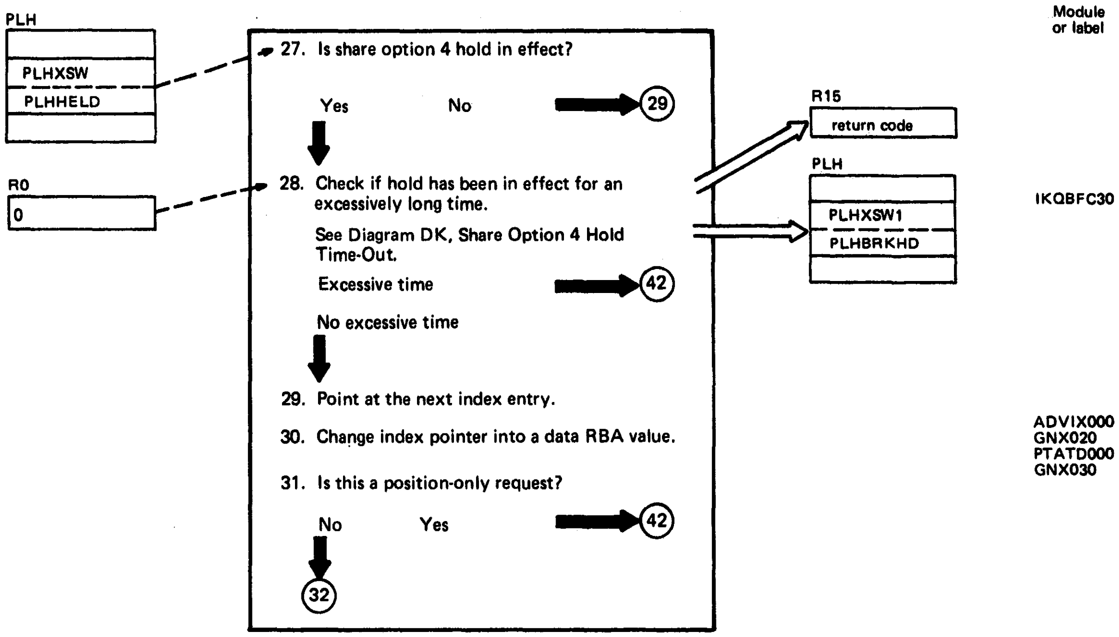
a. The cumulative compression count is compared to the front compression count in the index entry. (On the first iteration these counts are equal because the low key in an index record is not front-compressed.) If the front compression count is high, then the desired key is high compared to this index entry, and the cumulative compression count is left unchanged from its current value.

b. If the front compression count is low, the compressed key in the index entry is compared to the corresponding portion of the desired key. If the desired key is low or equal, the comparison is complete and the index entry has been found. (The cumulative compression count is left unchanged from its current value.)

c. If the key is high (compared to this index entry), the cumulative compression count is updated to indicate the total number of bytes that are equal between the desired key and this index entry key (including both the front compression count plus the number of bytes that were equal when the index entry key was compared).

See the note for step 22 for an example of front compression.

Diagram BS5. GET NEXT: Get Next Buffer and Read Ahead



27. Is share option 4 hold in effect?

Yes No → 29

28. Check if hold has been in effect for an excessively long time.

See Diagram DK, Share Option 4 Hold Time-Out.

Excessive time → 42

No excessive time

29. Point at the next index entry.

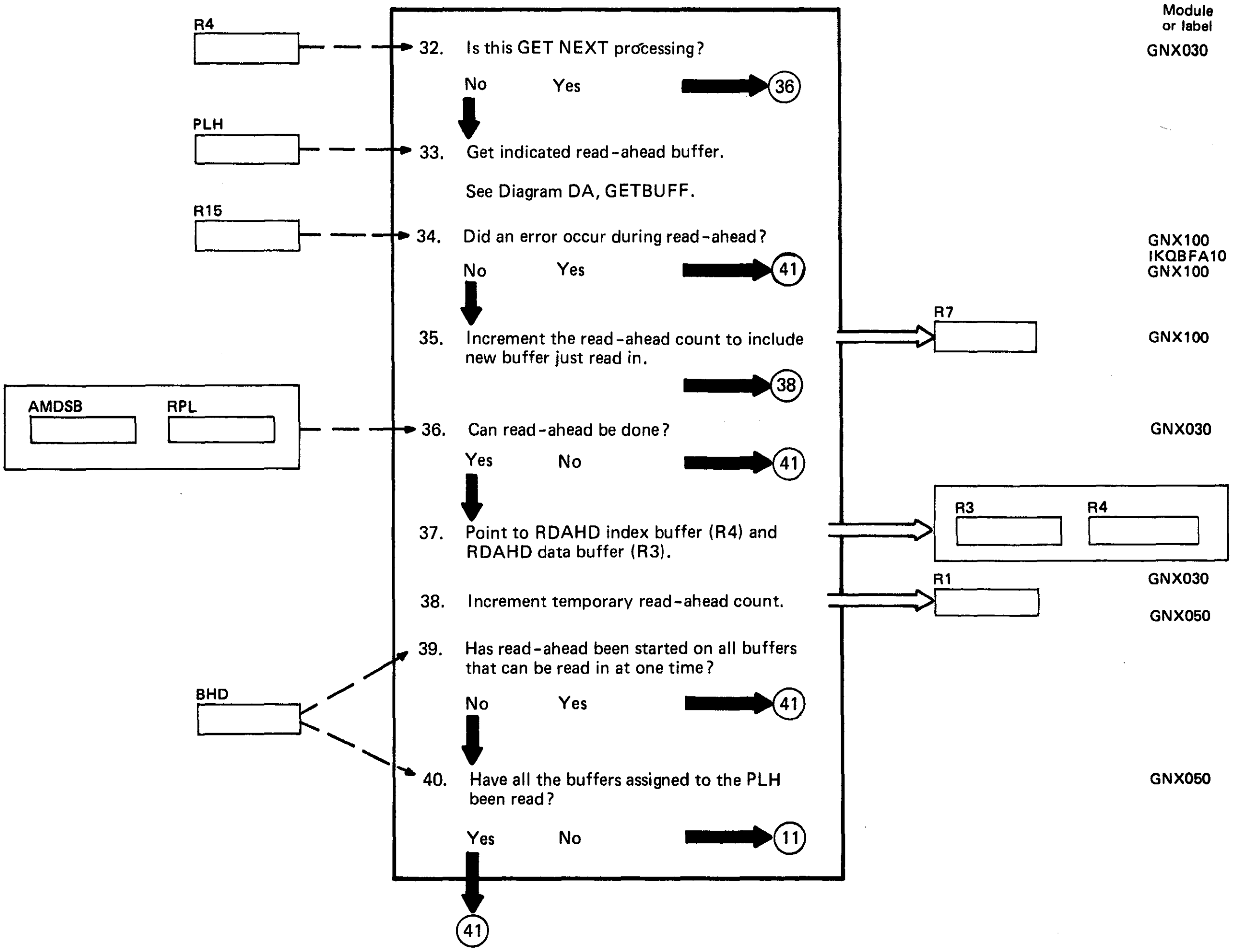
30. Change index pointer into a data RBA value.

31. Is this a position-only request?

No Yes → 42

32

Diagram BS6. GET NEXT: Get Next Buffer and Read Ahead



32. Is this GET NEXT processing?

33. Get indicated read-ahead buffer.

See Diagram DA, GETBUFF.

34. Did an error occur during read-ahead?

35. Increment the read-ahead count to include new buffer just read in.

36. Can read-ahead be done?

37. Point to RDAHD index buffer (R4) and RDAHD data buffer (R3).

38. Increment temporary read-ahead count.

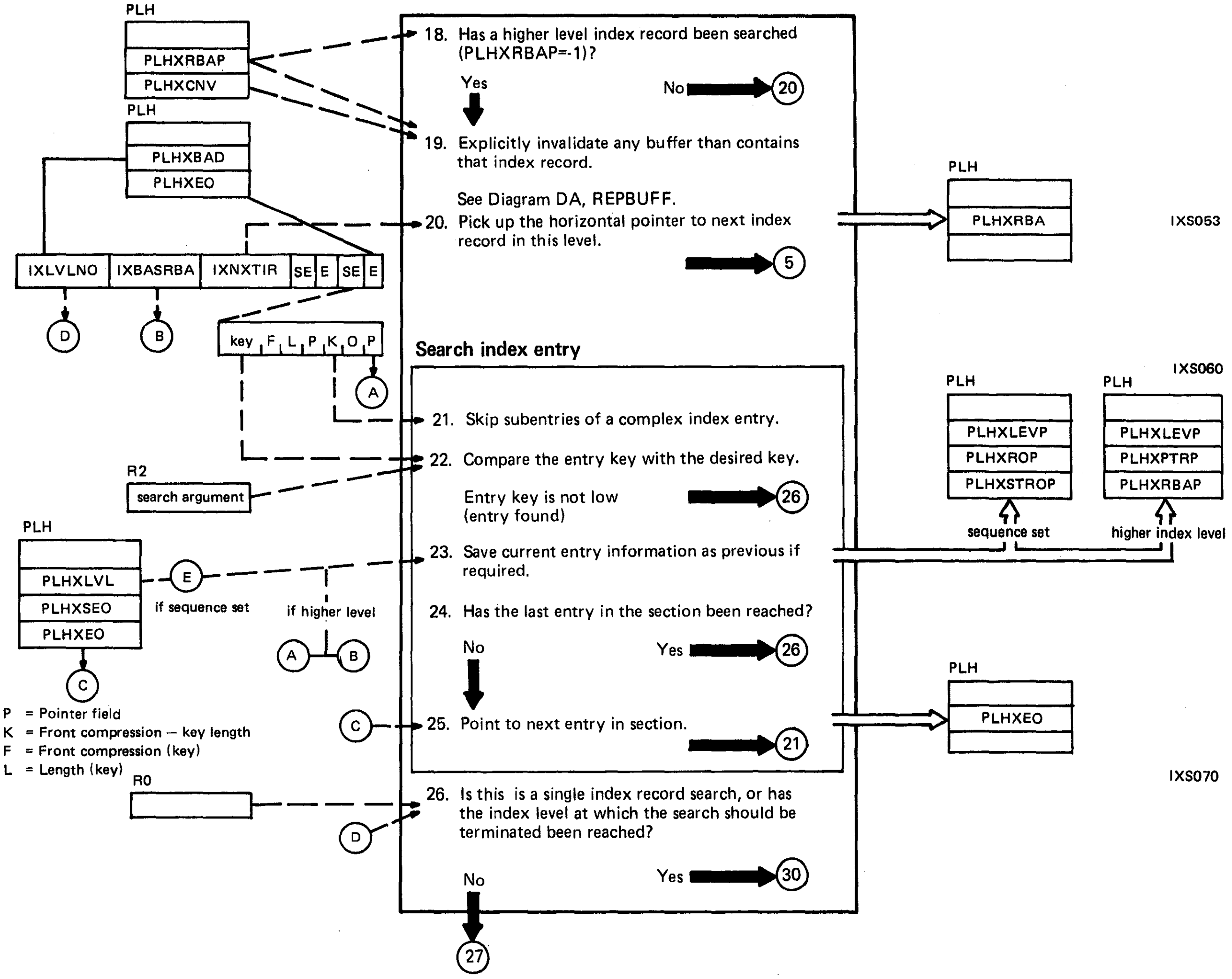
39. Has read-ahead been started on all buffers that can be read in at one time?

40. Have all the buffers assigned to the PLH been read?

Notes for Diagram BS6

36. Read-ahead is prohibited under any one of the following conditions:
    - User buffers
    - Data set is a catalog
    - Share option 4
    - Data set is an index component opened by itself, and it includes both fixed block and CKD volumes.

Diagram BU4. Search Index



P = Pointer field
K = Front compression — key length
F = Front compression (key)
L = Length (key)

18. Has a higher level index record been searched (PLHXRBAP=-1)?

Yes — No → 20

19. Explicitly invalidate any buffer than contains that index record.

See Diagram DA, REPBUFF.

20. Pick up the horizontal pointer to next index record in this level. → 5

Search index entry

21. Skip subentries of a complex index entry.

22. Compare the entry key with the desired key.

Entry key is not low (entry found) → 26

23. Save current entry information as previous if required.

24. Has the last entry in the section been reached?

No — Yes → 26

25. Point to next entry in section. → 21

26. Is this is a single index record search, or has the index level at which the search should be terminated been reached?

No → 27 — Yes → 30

Diagram BU5. Search Index

Notes for Diagram BU4

19. The higher-level index record that was searched was out of date (because it positioned us to the current record, which no longer includes the desired key). The higher-level record should be invalidated so it is not used again without rereading from the data set. Bit PLHNORDD is set in PLHXSW to tell the Buffer Manager that this is a request for explicit invalidation.

    An out of date higher-level index record occurs if another program is updating the data set and has caused a CA split since the higher-level index record was read.

22. The note for step 12 describes the procedure used in front compression. Note that the cumulative compression count left after the last iteration of step 12 becomes the cumulative compression count for the first iteration of step 22. After the last iteration of step 22, the cumulative compression count is equal to the front compression count that would be used if the desired key were entered into this index record as a new entry (see Diagram CK).

    Example of front key compression:
    The highest key within a CI used as the basis of the index entry for that CI. For front compression, this key is compared to the highest key of the logically preceding CI. All leading characters that are repeated in this key are eliminated.

    In the example, the first three characters (100) of the highest key in CI 2 are equal to the first three characters of the highest key in CI 1; therefore they can be eliminated.

    Front key compression discards the leading characters repeated from the previous key. The full key can be reconstructed by working back through the lower key entries in the index record, as can be seen by studying the following chart.

Complete key	F	L	Key after Compression
10008	0	5	10008
10080	3	2	- - - 80
10333	2	3	- - 333
14000	1	4	- 4000
14028	3	2	- - - 28
23630	0	5	23630

F = number of high-order characters deleted
L = length of compressed key
P = pointer field (relative CI number)

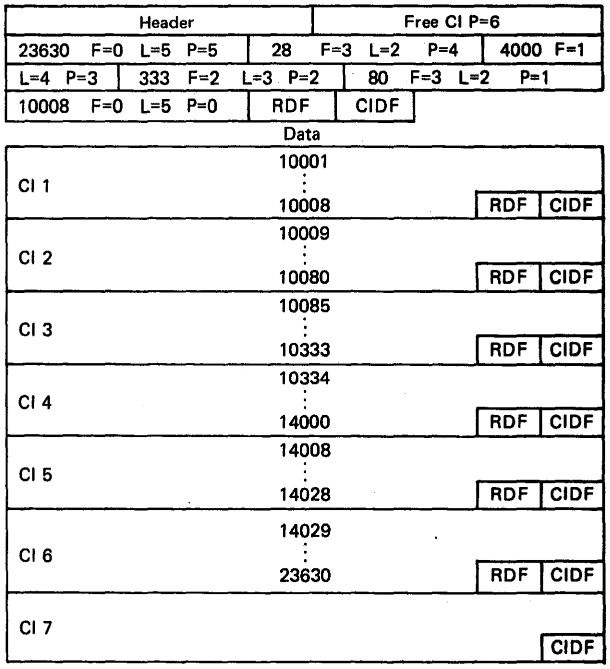
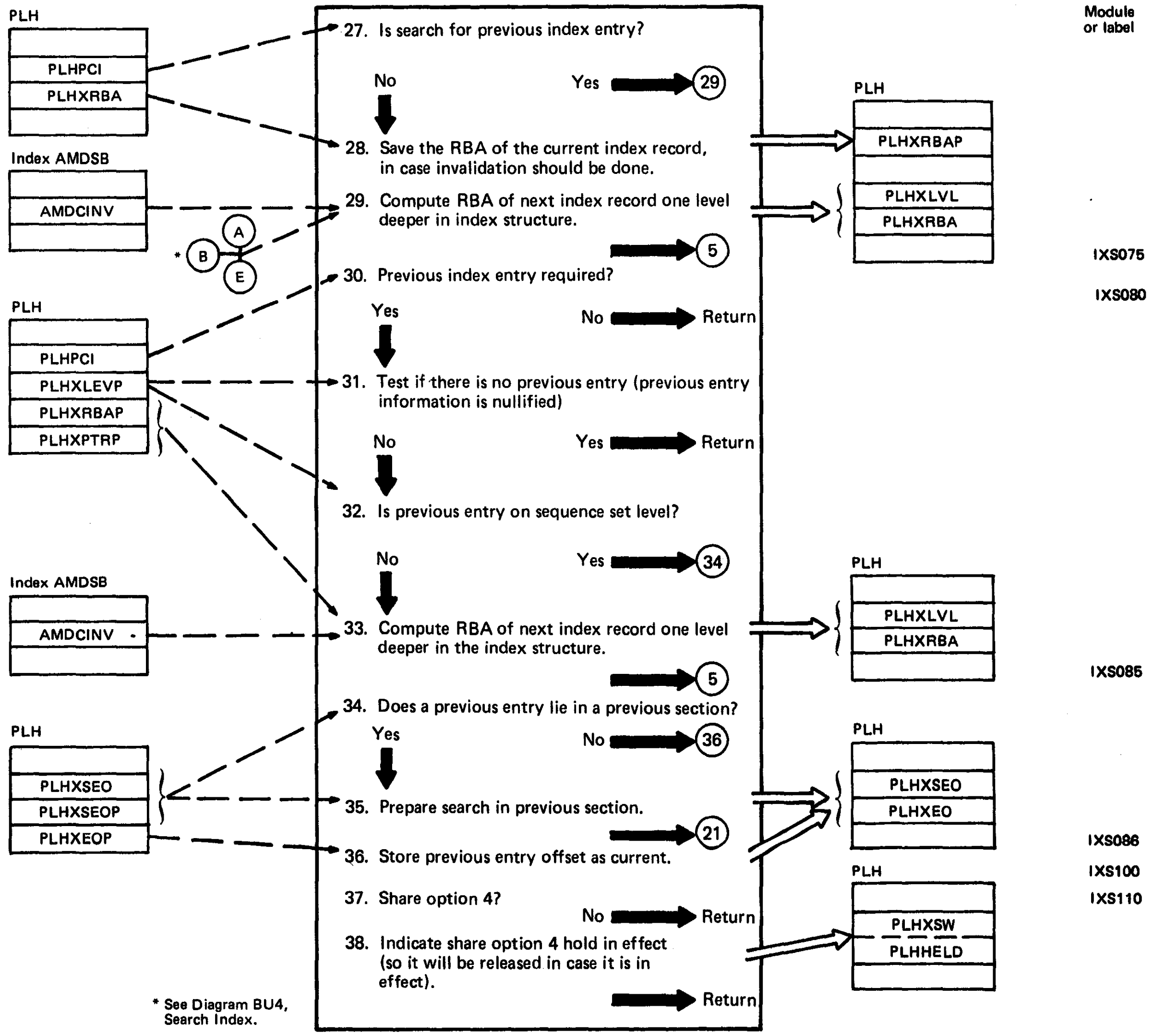


Diagram BU6. Search Index



27. Is search for previous index entry?
No
Yes → 29

28. Save the RBA of the current index record, in case invalidation should be done.

29. Compute RBA of next index record one level deeper in index structure.
→ 5

30. Previous index entry required?
Yes
No → Return

31. Test if there is no previous entry (previous entry information is nullified)
No
Yes → Return

32. Is previous entry on sequence set level?
No
Yes → 34

33. Compute RBA of next index record one level deeper in the index structure.
→ 5

34. Does a previous entry lie in a previous section?
Yes
No → 36

35. Prepare search in previous section.
→ 21

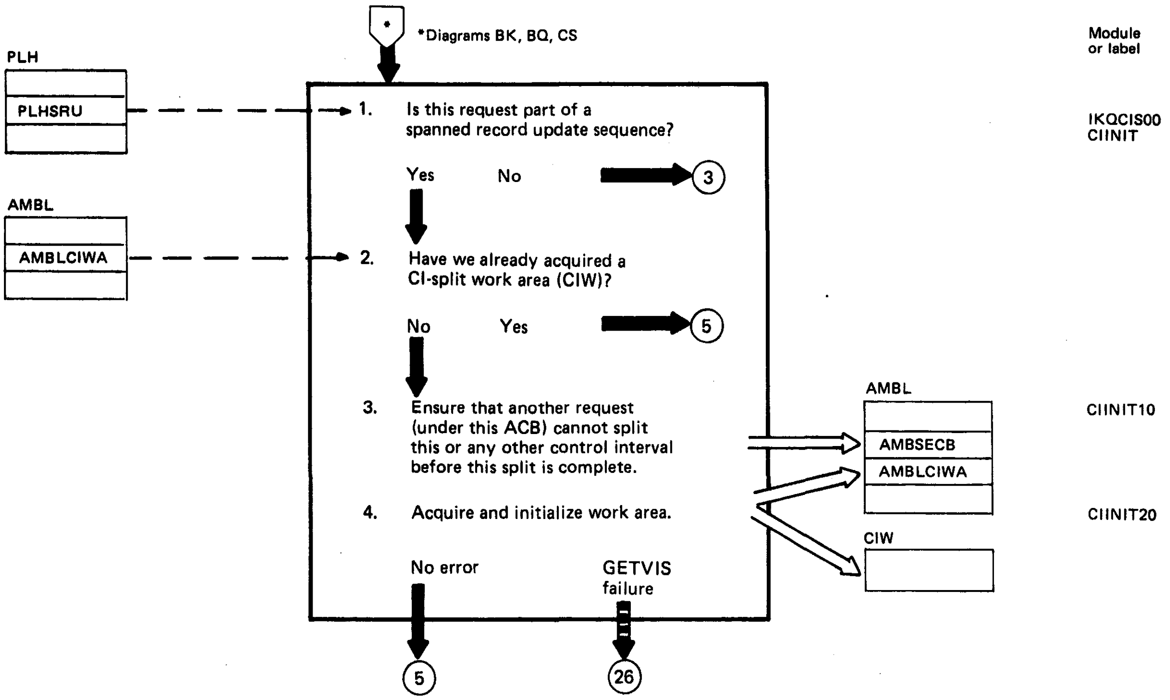
36. Store previous entry offset as current.

37. Share option 4?
No → Return

38. Indicate share option 4 hold in effect (so it will be released in case it is in effect).
→ Return

* See Diagram BU4, Search Index.

Diagram CA1. Control Interval Split



1. Is this request part of a spanned record update sequence?
2. Have we already acquired a CI-split work area (CIW)?
3. Ensure that another request (under this ACB) cannot split this or any other control interval before this split is complete.
4. Acquire and initialize work area.

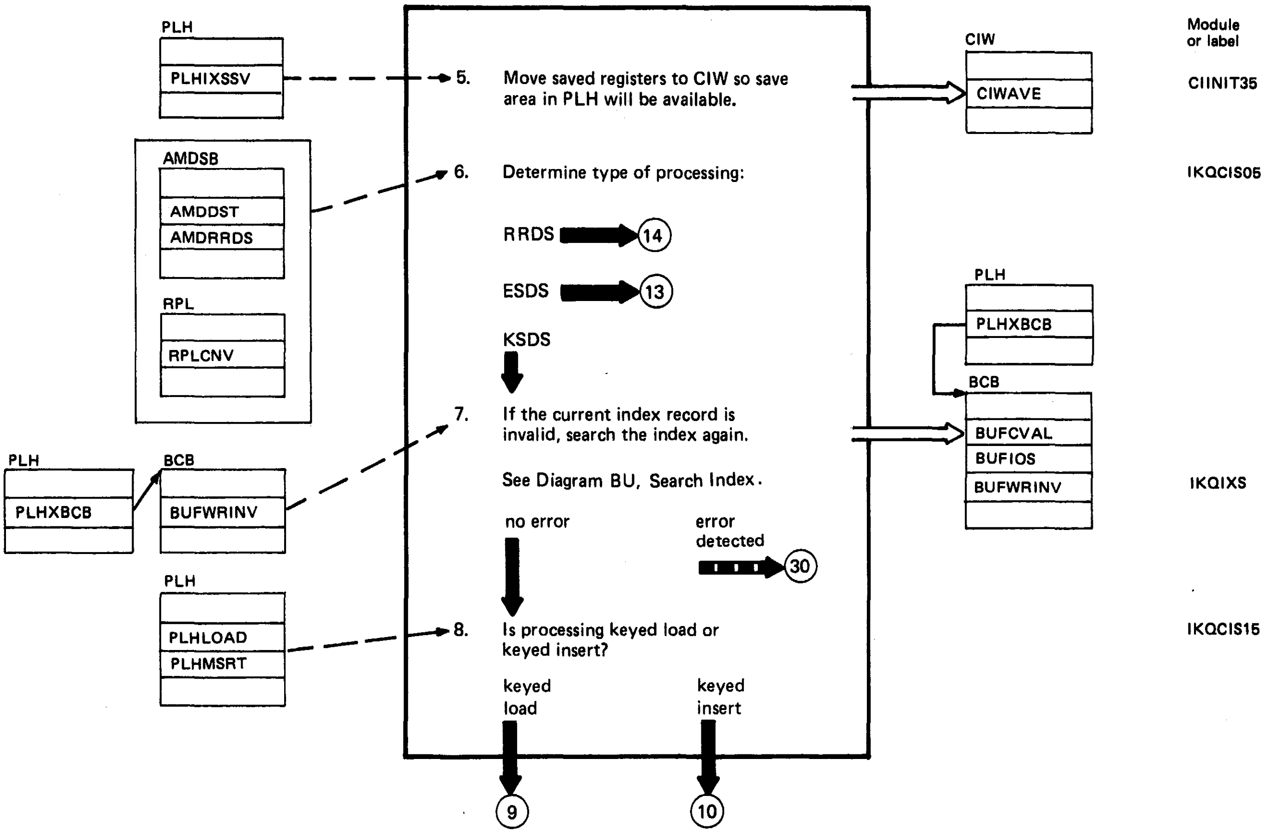
Notes for Diagram CA1

1. IKQCIS00 is called whenever a new control interval is required, except for the mainline entry-sequenced case. This mainline case applies to ESDS whenever the following conditions are satisfied:
   - It is not the first PUT request to the file.
   - No spanned record is being processed.
   - No new control area is required.

   For this ESDS, CI-split processing is performed by module IKQMDY.
2. A CI-split work area has already been acquired if this request is part of a spanned record update sequence, and if there have been previous CI-split requests in this sequence.

   Note that a spanned record update sequence starts with the second CI of a spanned record. IKQCIS00 handles the first CI without really being aware that it is part of a spanned record.

Diagram CA2. Control Interval Split



5. Move saved registers to CIW so save area in PLH will be available.

6. Determine type of processing:

7. If the current index record is invalid, search the index again.

See Diagram BU, Search Index.

8. Is processing keyed load or keyed insert?

Notes for Diagram CA2

5. Registers are saved temporarily in the index search save area (PLHIXSSV) when IKQCIS00 is entered. Because the index search routine may be called during the CI-split process, this save area must be made available.

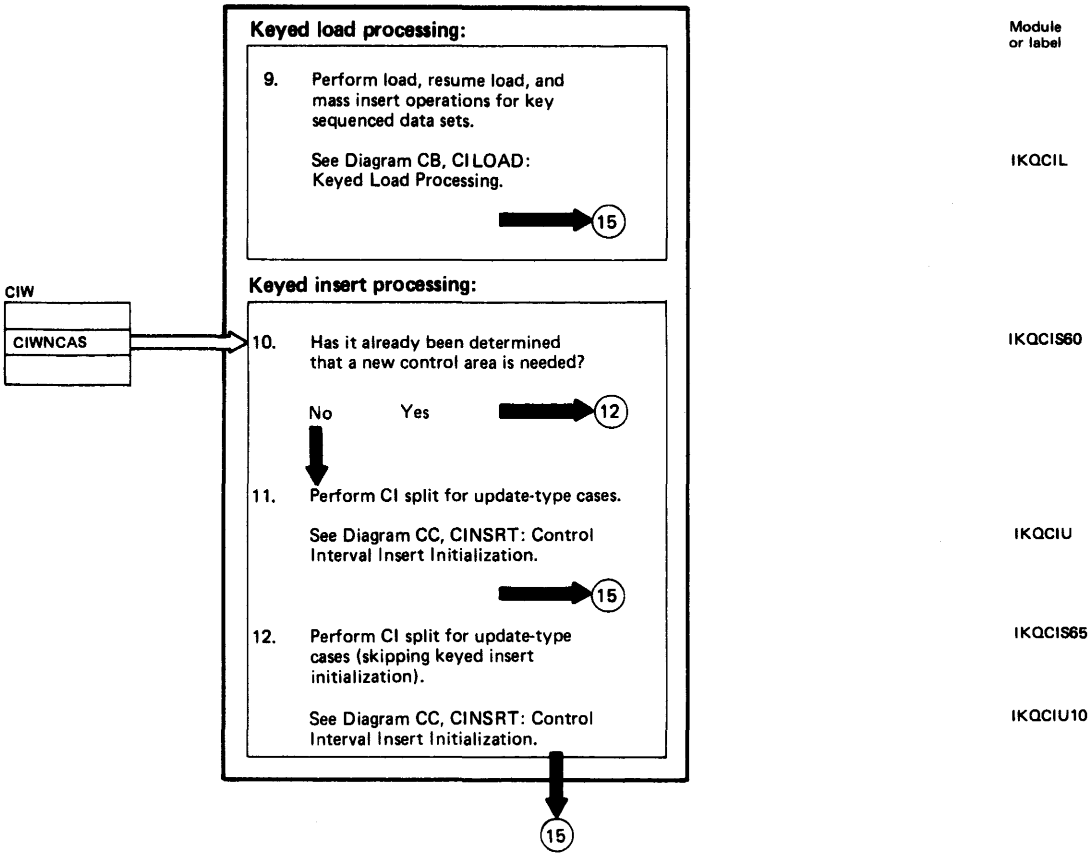
7. This test determines if a new version of the index record may have been written under a different PLH, thereby invalidating the version owned by the current PLH.

8. Keyed load processing includes initial load or resume load (additions to the end of the data set), and mass insert (insertions to the end of a control interval in sequential mode). Initial load or resume load is indicated by the flag PLHLOAD. Mass insert is indicated by the flag PLHMSRT.

   Keyed insert (or update type) CI-split processing includes CI splits caused by any of the following:

   - Update with length change of an existing record.
   - Insert in front of one or more records already in the CI.
   - Any insert in direct mode, other than to the end of the data set.

Diagram CA3. Control Interval Split



Keyed load processing:

9. Perform load, resume load, and mass insert operations for key sequenced data sets.

See Diagram CB, CILOAD: Keyed Load Processing.

Keyed insert processing:

10. Has it already been determined that a new control area is needed?

11. Perform CI split for update-type cases.

See Diagram CC, CINSRT: Control Interval Insert Initialization.

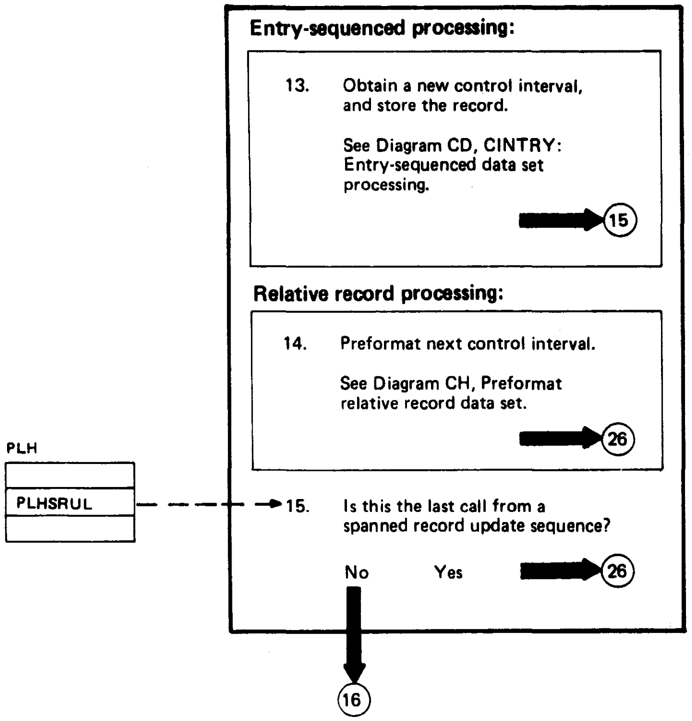
12. Perform CI split for update-type cases (skipping keyed insert initialization).

See Diagram CC, CINSRT: Control Interval Insert Initialization.

Notes for Diagram CA3

10. This step tests for a control area split being caused because there isn't enough room in the sequence set record.

Diagram CA4. Control Interval Split



Module or label

Entry-sequenced processing:

13. Obtain a new control interval, and store the record. — IKQCIS75

See Diagram CD, CINTRY: Entry-sequenced data set processing. — CINTRY

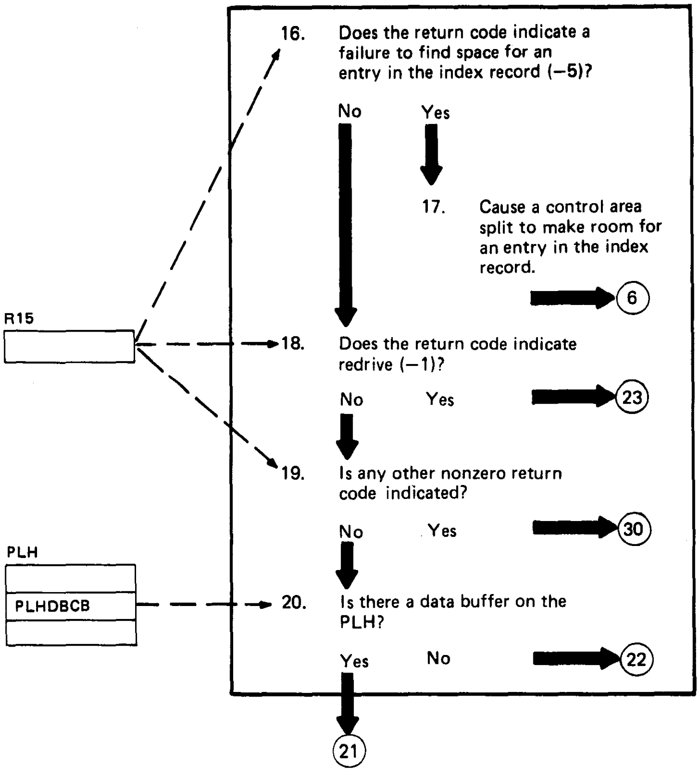
Relative record processing:

14. Preformat next control interval. — IKQCIS85

See Diagram CH, Preformat relative record data set. — IKQRRP

15. Is this the last call from a spanned record update sequence? — IKQCIS90

Diagram CA5. Control Interval Split



16. Does the return code indicate a failure to find space for an entry in the index record (−5)?

17. Cause a control area split to make room for an entry in the index record.

18. Does the return code indicate redrive (−1)?

19. Is any other nonzero return code indicated?

20. Is there a data buffer on the PLH?

Module or label

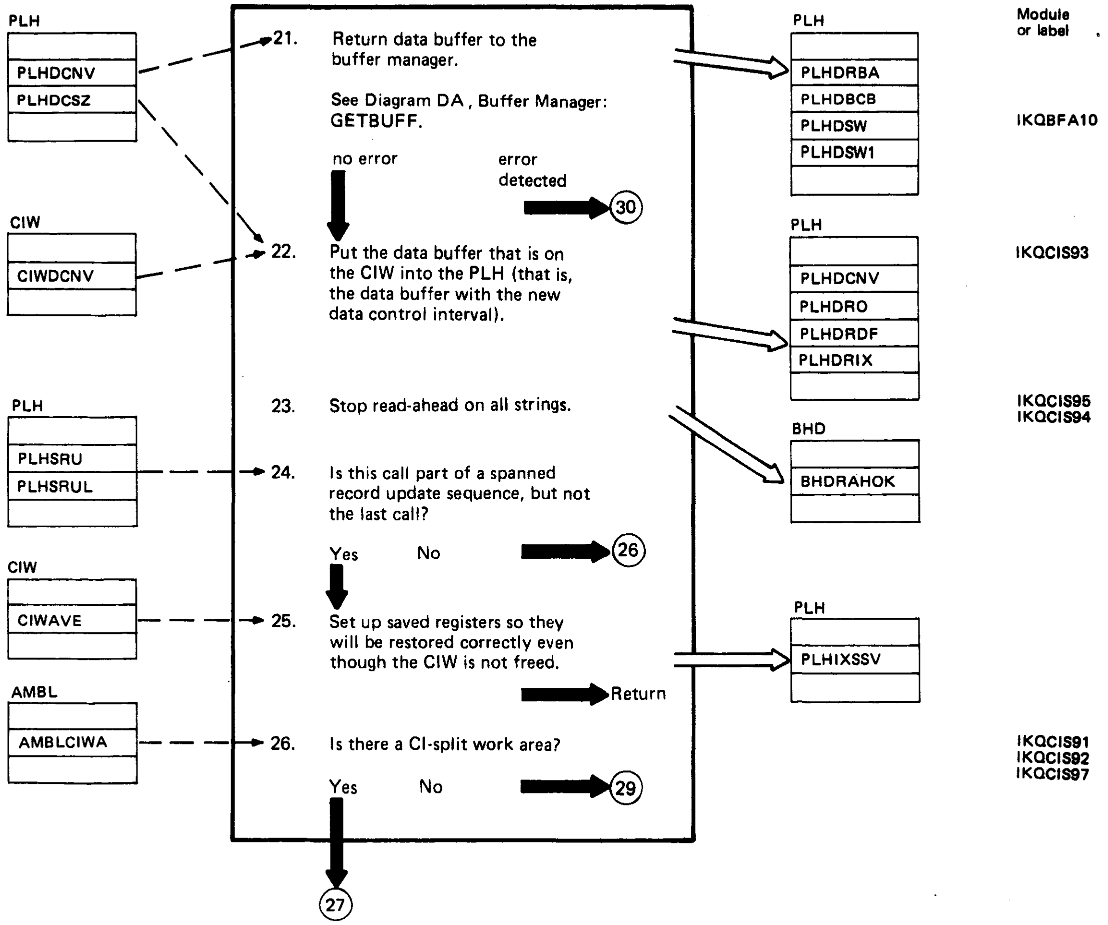
CINXRM

Notes for Diagram CA5

17. This routine (CINXRM) is a recovery routine, called when CI-split processing must be interrupted after partial completion, due to unavailability of index space in the first-level index (sequence set) record. Partially-filled buffers are purged, the control area split switch (CIWNCAS) is turned on, and CI-split is restarted.

18. If the redrive return code (negative 1 in register 15) is indicated, the whole insert or update-with-length-change operation is to be restarted by module IKQMDY.

Diagram CA6. Control Interval Split



21. Return data buffer to the buffer manager.

See Diagram DA, Buffer Manager: GETBUFF.

no error — error detected ➡ 30

22. Put the data buffer that is on the CIW into the PLH (that is, the data buffer with the new data control interval).

23. Stop read-ahead on all strings.

24. Is this call part of a spanned record update sequence, but not the last call?

Yes — No ➡ 26

25. Set up saved registers so they will be restored correctly even though the CIW is not freed.

➡ Return

26. Is there a CI-split work area?

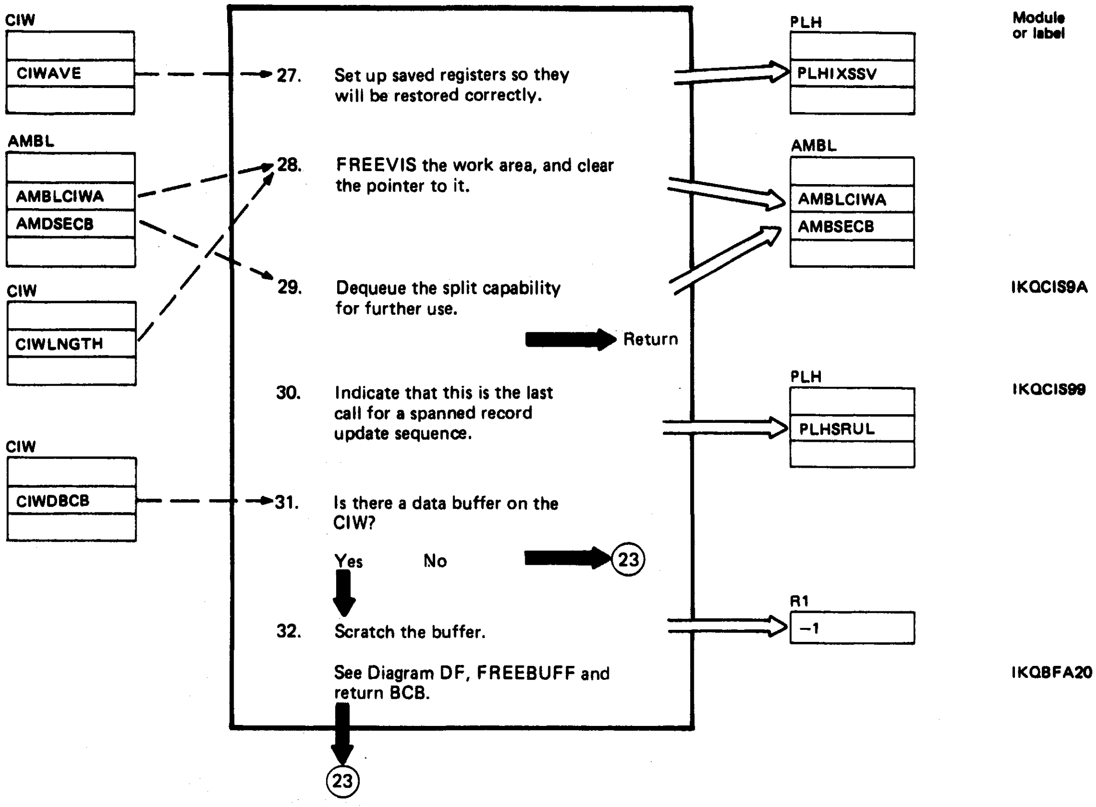
Yes ➡ 27 — No ➡ 29

Notes for Diagram CA6

23. This step loops through all PLHs, turning off the BHDRAHOK flag in the data BHD.

24. If this call is part of a spanned record update sequence but not the last call of the sequence, then the CI work area will not be freed. It will be reused on the next call of the sequence.

Diagram CA7. Control Interval Split



27. Set up saved registers so they will be restored correctly.

28. FREEVIS the work area, and clear the pointer to it.

29. Dequeue the split capability for further use.

Return

30. Indicate that this is the last call for a spanned record update sequence.

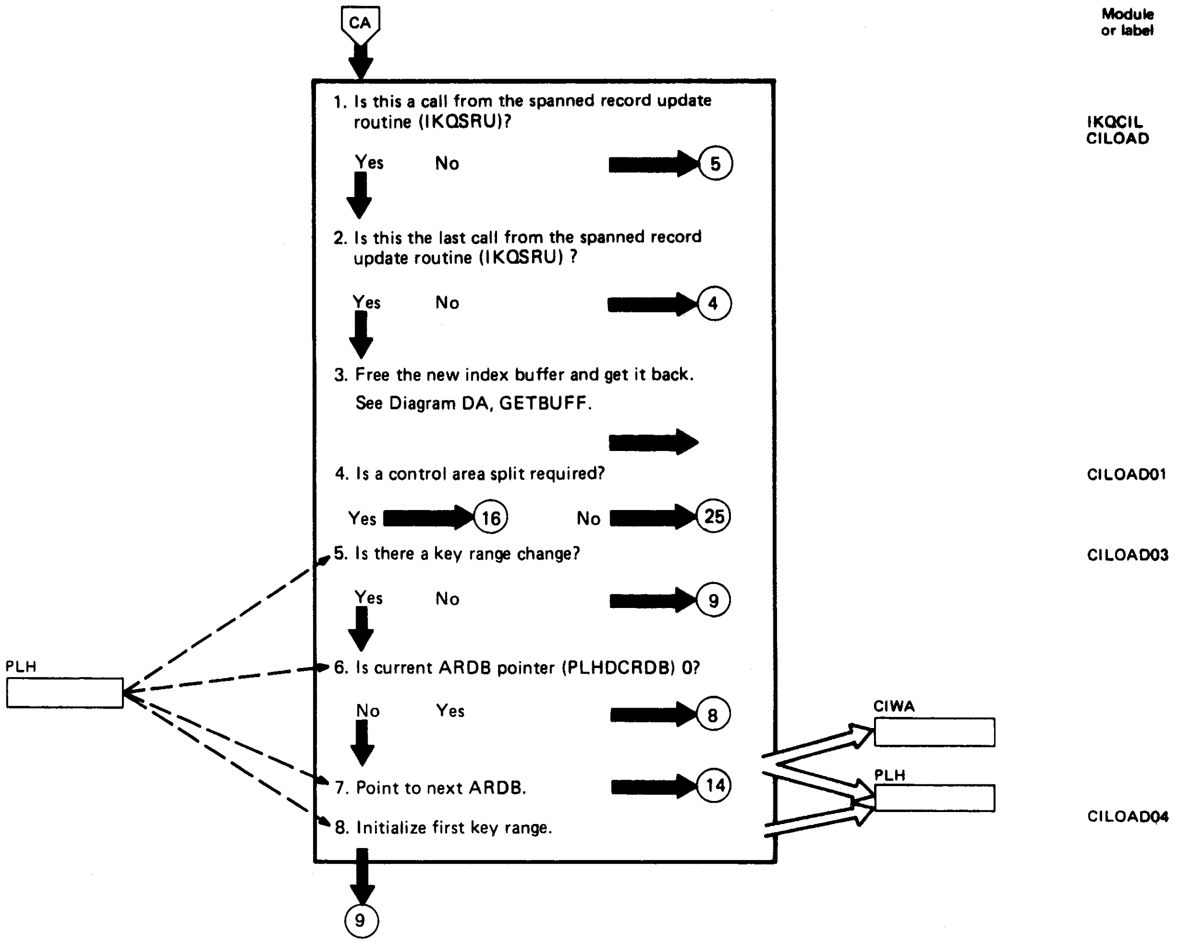
31. Is there a data buffer on the CIW?

Yes No

32. Scratch the buffer.

See Diagram DF, FREEBUFF and return BCB.

Diagram CB1. CILOAD: Keyed Load Processing

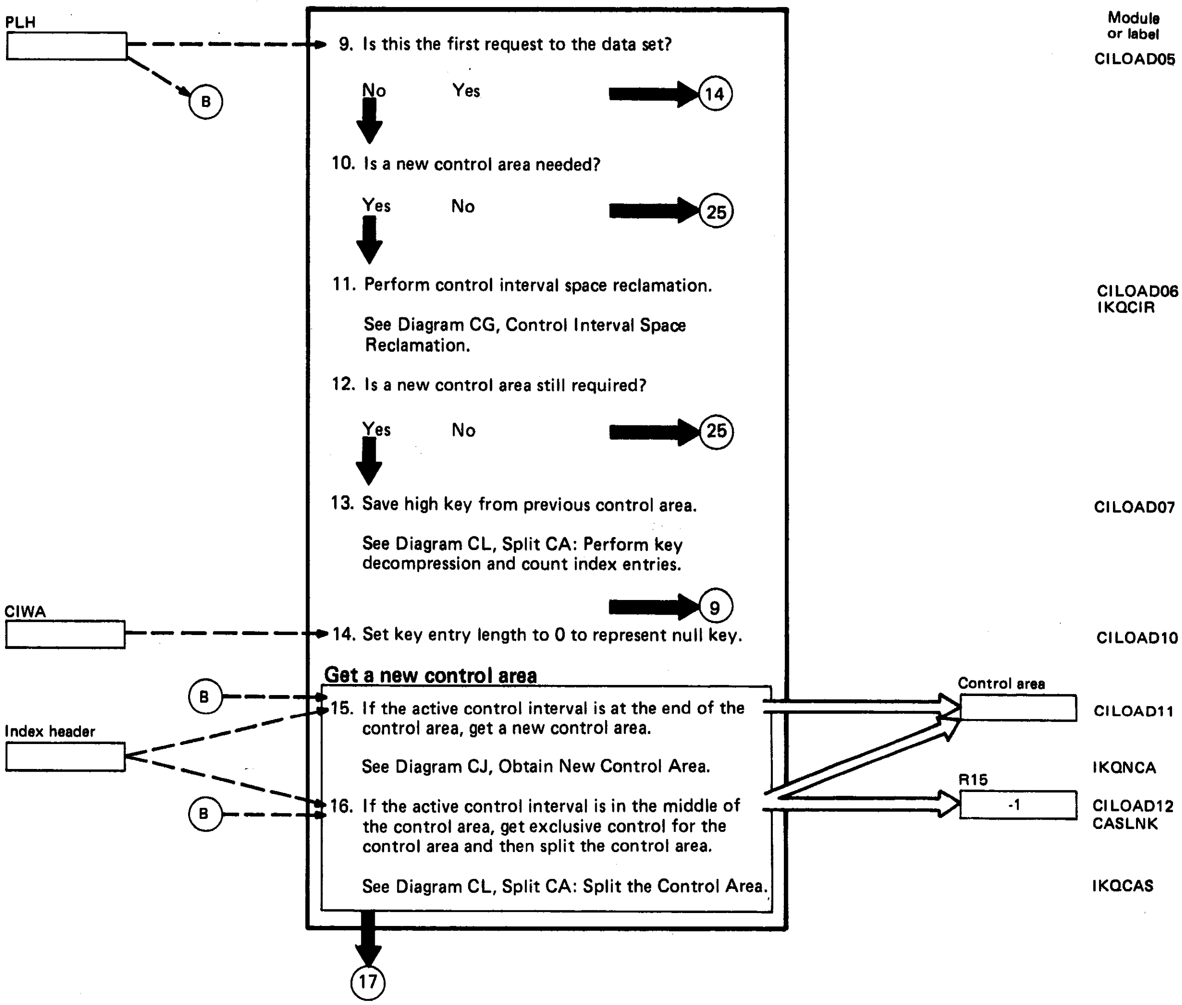

Module or label
CA
1. Is this a call from the spanned record update routine (IKQSRU)?
Yes
No
5
IKQCIL
CILOAD
2. Is this the last call from the spanned record update routine (IKQSRU) ?
Yes
No
4
3. Free the new index buffer and get it back.
See Diagram DA, GETBUFF.
4. Is a control area split required?
CILOAD01
Yes
16
No
25
5. Is there a key range change?
CILOAD03
Yes
No
9
PLH
6. Is current ARDB pointer (PLHDCRDB) 0?
No
Yes
8
CIWA
PLH
7. Point to next ARDB.
14
8. Initialize first key range.
CILOAD04
9

Notes for Diagram CB1

5. For key range processing, pointers in the PLH are set for both the current ARDB and the ARDB for the key range in which the insert activity will take place. If these are not the same, one key range must be cleaned up before the next can be accessed. Any intervening key ranges must also be formatted and have index records built for them.

7. The pointer to the current ARDB is saved in the work area and the pointer to the next ARDB is updated in the PLH.

Diagram CB2. CILOAD: Keyed Load Processing



9. Is this the first request to the data set?

10. Is a new control area needed?

11. Perform control interval space reclamation.

    See Diagram CG, Control Interval Space Reclamation.

12. Is a new control area still required?

13. Save high key from previous control area.

    See Diagram CL, Split CA: Perform key decompression and count index entries.

14. Set key entry length to 0 to represent null key.

Get a new control area

15. If the active control interval is at the end of the control area, get a new control area.

    See Diagram CJ, Obtain New Control Area.

16. If the active control interval is in the middle of the control area, get exclusive control for the control area and then split the control area.

    See Diagram CL, Split CA: Split the Control Area.

Notes for Diagram CB2

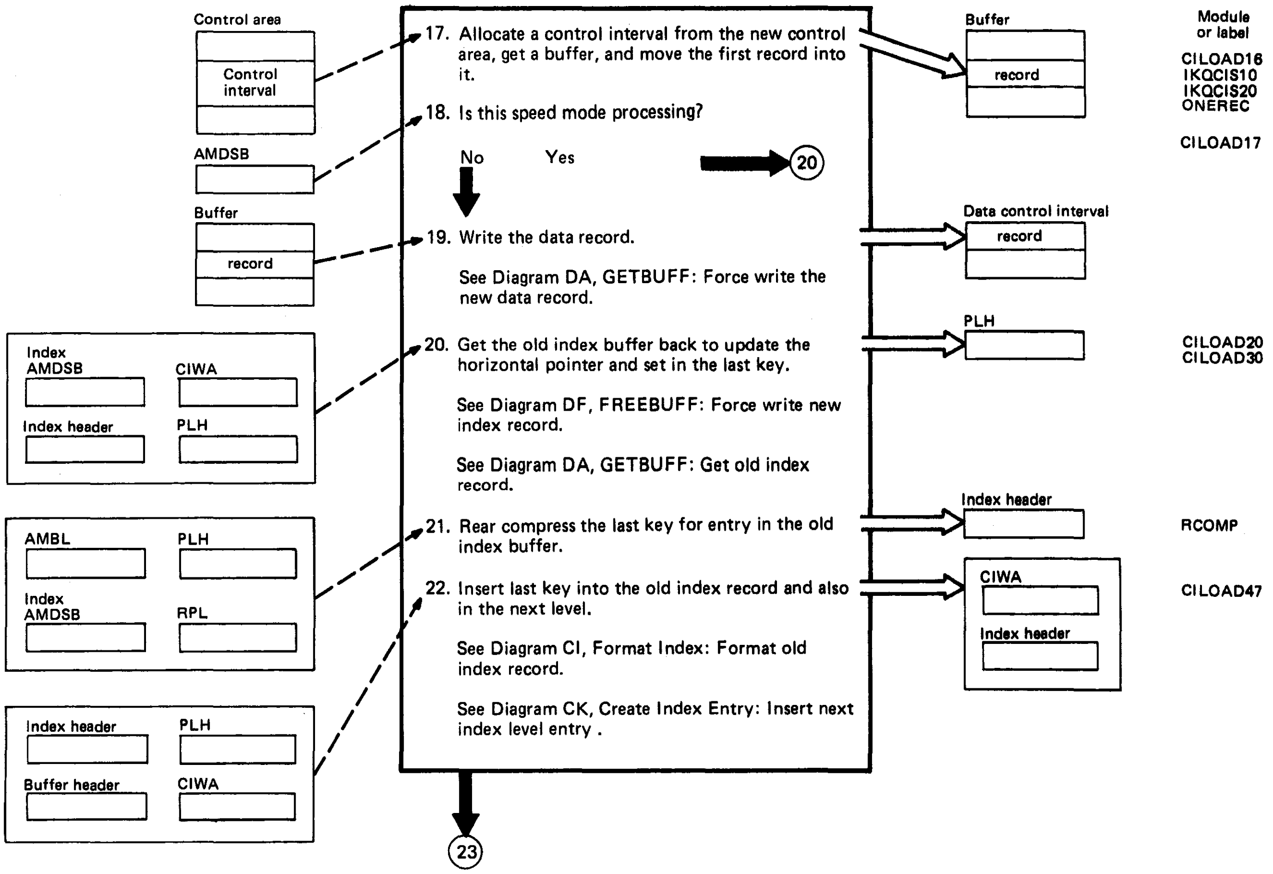
11. If the CI split routine determines that a CA split is necessary (as there are no free CIs available), it first calls IKQCIR, which searches for CIs whose contents have been erased. It reclaims their space and informs IKQCIL, which then attaches a reclaimed CI. If there are no CIs which can be reclaimed, IKQCIR informs IKQCIL, which then continues with a CA split.

15. Key range processing is the same as normal except IKQNCA00 allocates only from the ARDB for the target key range. When a key range is exited, the dummy (F=L=0) index entry is replaced by the high key of the key range. If a key range is skipped, an index record must nevertheless be allocated and a control area formatted.

    For share option 4, the Record Management space allocation lock is acquired for the file before IKQNCA is called. The lock name for this lock is the basic lock name for the file (see note for step 1 of Diagram DG), except that the last two bytes are X'0005'.

16. If a mass insert is being made, the packing factor is ignored, and the control area is split at the end of the active control interval (unless it is the last in the control area — see step 8).

Diagram CB3. CILOAD: Keyed Load Processing



17. Allocate a control interval from the new control area, get a buffer, and move the first record into it.

18. Is this speed mode processing?

19. Write the data record.

    See Diagram DA, GETBUFF: Force write the new data record.

20. Get the old index buffer back to update the horizontal pointer and set in the last key.

    See Diagram DF, FREEBUFF: Force write new index record.

    See Diagram DA, GETBUFF: Get old index record.

21. Rear compress the last key for entry in the old index buffer.

22. Insert last key into the old index record and also in the next level.

    See Diagram CI, Format Index: Format old index record.

    See Diagram CK, Create Index Entry: Insert next index level entry .

Notes for Diagram CB3

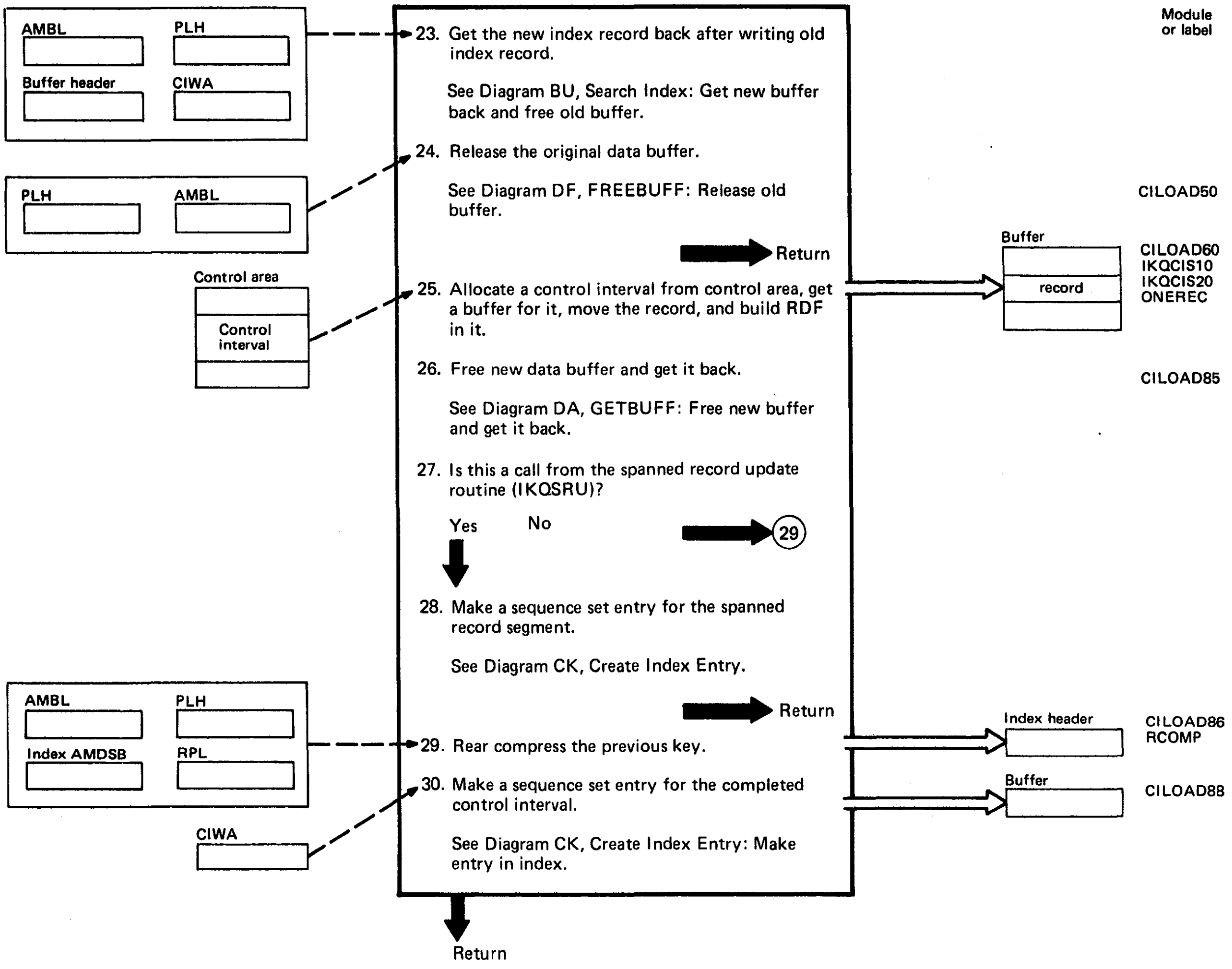
20. If processing is being done in speed mode, the REPBUF step is skipped. If processing is being done in recovery mode, the write is forced by means of the combined FREEBUFF and GETBUFF, implied by REPBUF.

21. If there is not a key range change, the high key from the last control interval processed is picked up and rear compressed.

22. If there is a key range change, the key range high key is picked up from the ARDB and replaces the key of the last entry in the record. The record is formatted and IKQIXE00 is called to make the next index level entry.

    At completion of this step, the Record Management space allocation lock for the file is released if share option 4 is in effect for the file.

Diagram CB4. CILOAD: Keyed Load Processing



23. Get the new index record back after writing old index record.

    See Diagram BU, Search Index: Get new buffer back and free old buffer.

24. Release the original data buffer.

    See Diagram DF, FREEBUFF: Release old buffer.

    Return

25. Allocate a control interval from control area, get a buffer for it, move the record, and build RDF in it.

26. Free new data buffer and get it back.

    See Diagram DA, GETBUFF: Free new buffer and get it back.

27. Is this a call from the spanned record update routine (IKQSRU)?

    Yes No 29

28. Make a sequence set entry for the spanned record segment.

    See Diagram CK, Create Index Entry.

    Return

29. Rear compress the previous key.

30. Make a sequence set entry for the completed control interval.

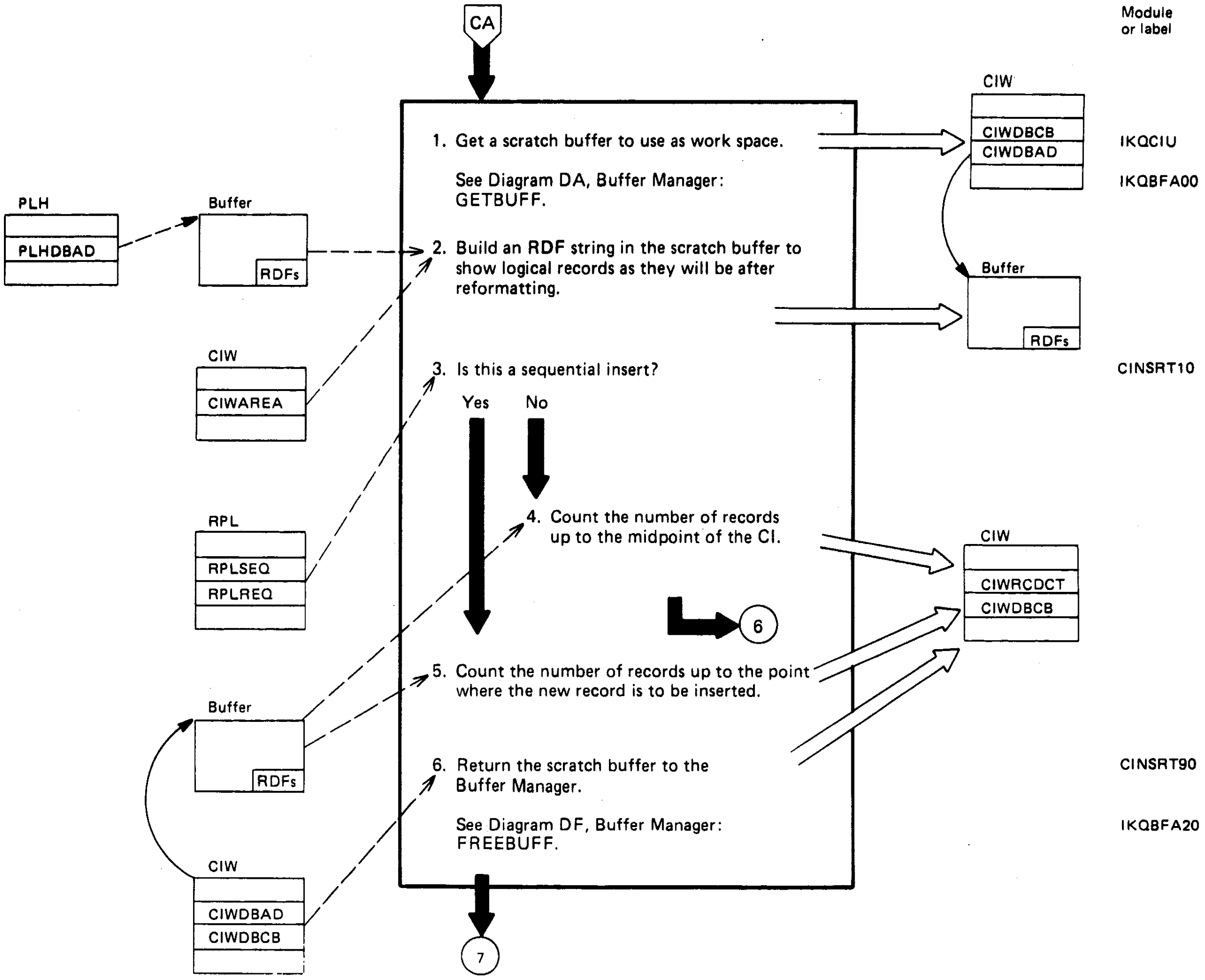
    See Diagram CK, Create Index Entry: Make entry in index.

Return

Notes for Diagram CB4

25. In normal record processing, records are added to the end of a control interval. When the packing factor is reached, a new control interval is obtained. When the packing factor for a control area is reached, a new control area is obtained.

Diagram CC1. CINSRT: Control Interval Insert Initialization



1. Get a scratch buffer to use as work space.

   See Diagram DA, Buffer Manager: GETBUFF.

2. Build an RDF string in the scratch buffer to show logical records as they will be after reformatting.

3. Is this a sequential insert?

4. Count the number of records up to the midpoint of the CI.

5. Count the number of records up to the point where the new record is to be inserted.

6. Return the scratch buffer to the Buffer Manager.

   See Diagram DF, Buffer Manager: FREEBUFF.

Notes for Diagram CC1

1. IKQCIU is called when a CI split is caused by:
   - An update with length change of an existing record.
   - A sequential insert, if the sequence of inserts started in the middle of the CI and this is the first split caused during the sequence. (Other cases of splits during sequential insert are handled by IKQCIL.)
   - Any direct insert, other than to the end of the data set.

   Four cases of CI splits can occur under IKQCIU processing, as illustrated by the following diagrams. Note that the actual insertion of the new record is not done by IKQCIU, but by IKQMDY after control is passed back to it.

   A split caused by an update with length change is handled the same as the splits shown in Examples 1, 2, and 3. IKQCIU treats the point at which the length change is to be made as the beginning of a record whose length is to be changed.

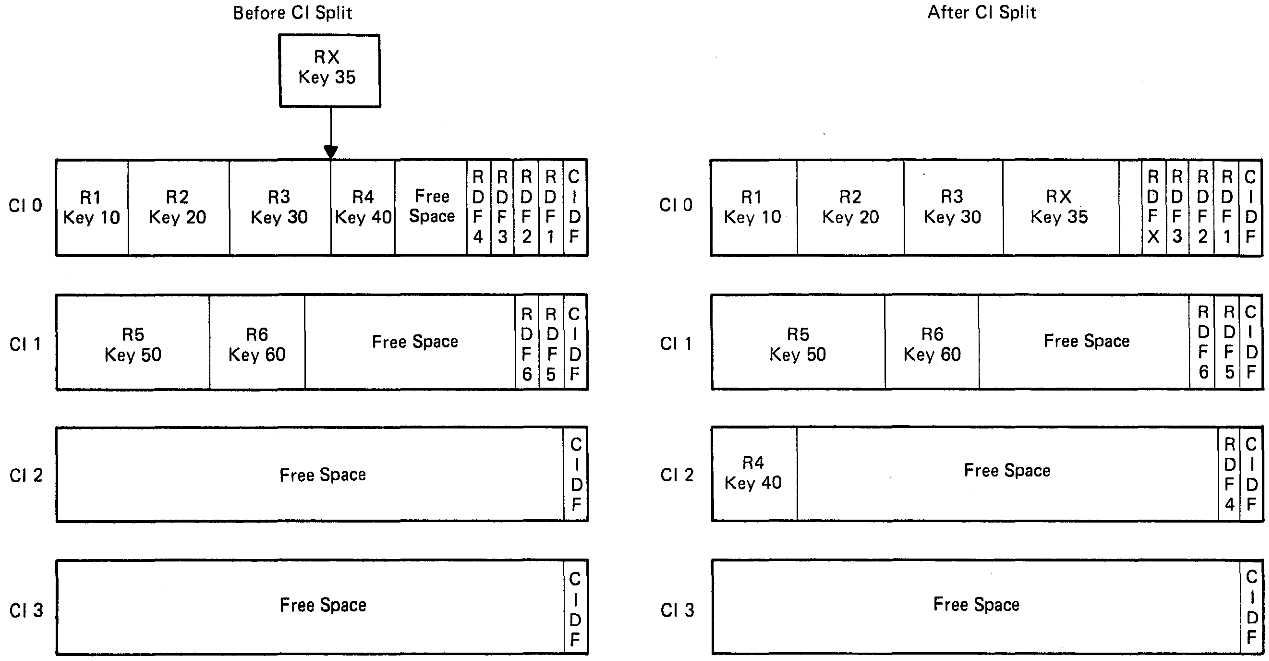
2. A string of RDFs is built as if the insert or update-with-length-change had completed without causing a split. This string is used to determine the optimum point at which to split the CI, as illustrated in the examples. (Modified RDFs were copied into CIWAREA by module IKQCIS00. They were copied from PLHWAREA, where they had been built by module IKQBLD.)

3. The optimum split point is to be determined. Refer to the examples in step 1.

4. The count is rounded so that the record boundary nearest the middle becomes the split point.

Diagram CC2. CINSRT: Control Interval Insert Initialization

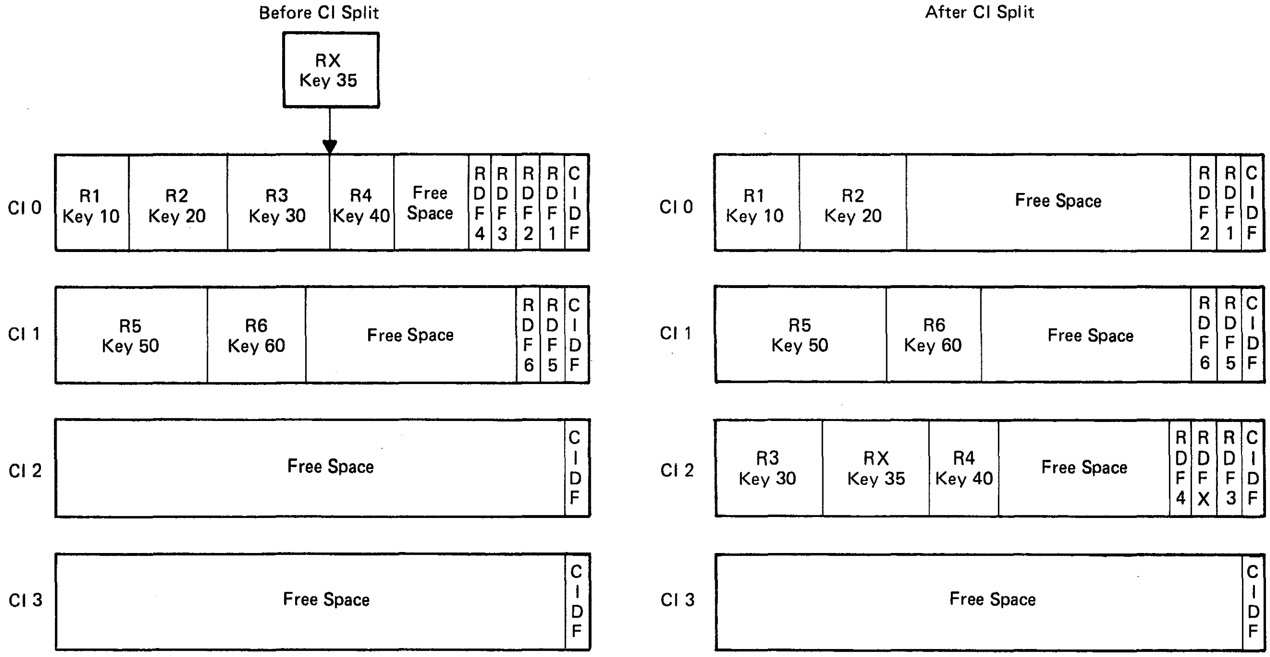
Example 1



Sequential Insert.

The split is done so that the inserted record will become the last record in the "old" CI (CI 0 in this example). If the inserted record won't fit into the old CI, IKQCIU performs a CI split. Control returns to IKQMDY, which again determines that a split is necessary, and causes a second split. IKQCIL handles this second split in the normal manner for mass insert.

Example 2

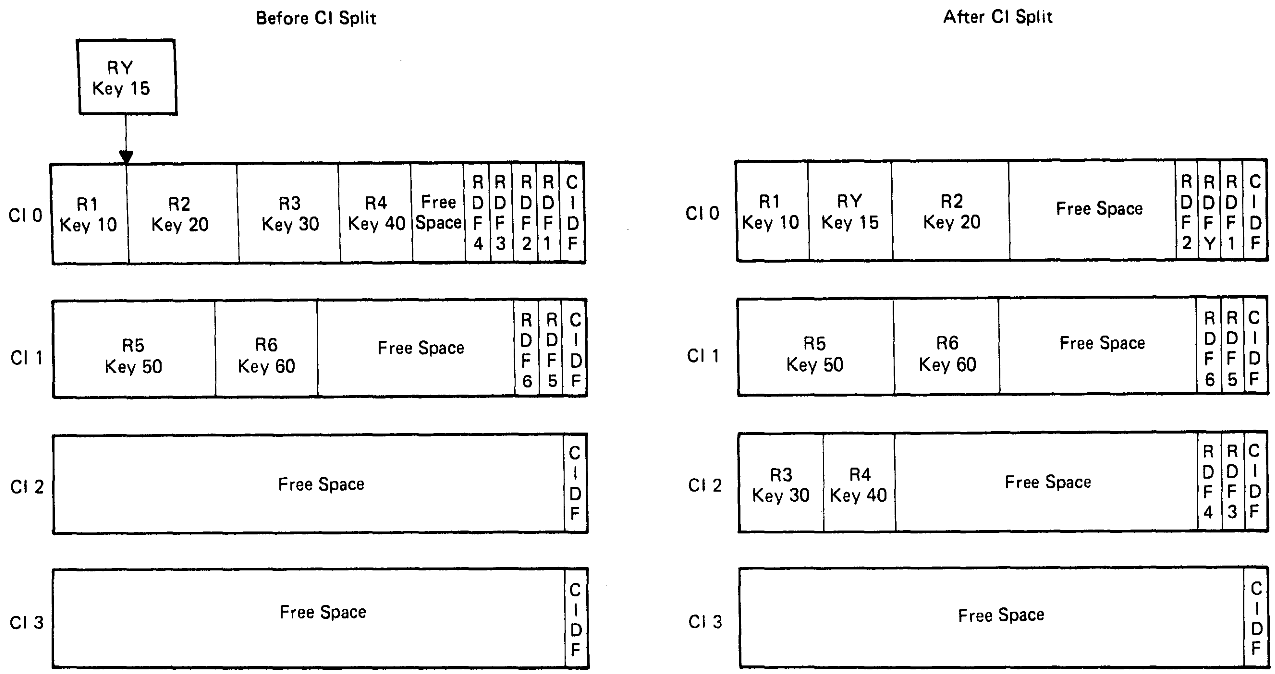


Direct Insert at or Beyond the Middle of a CI.

The CI is split so that after the insert is finished, half the data (or as near half as possible) will be in each CI (CI 0 and CI 2 in this example).

Diagram CC3. CINSRT: Control Interval Insert Initialization

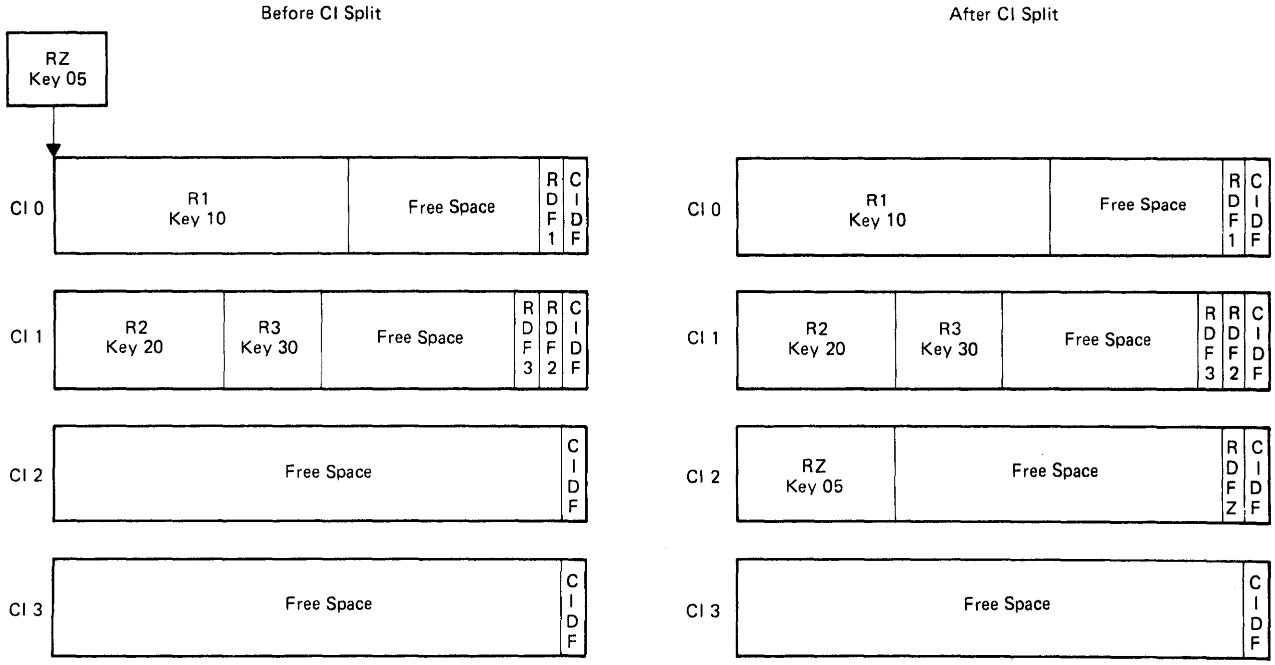
Example 3



Direct Insert Before the Middle of a CI.

The CI is split so that after the insert is finished, half the data (or as near half the data as possible) will be in each CI (CI 0 and CI 2 in this example).

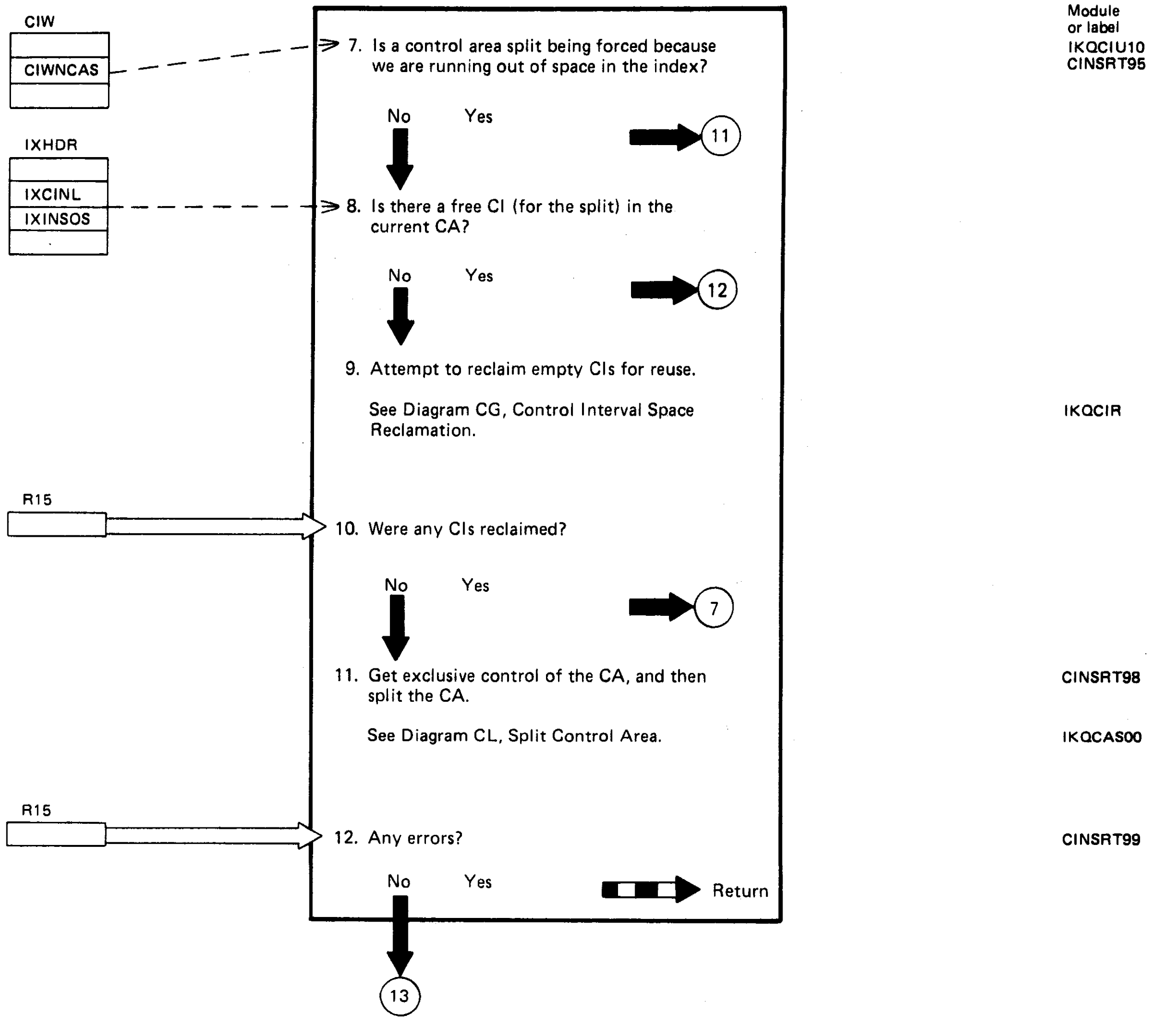
Example 4 ("the special case")



Sequential or Direct Insert at Beginning of CI (not update with length change).

The normal procedure of copying the high-key records of the "old" CI into the "new" CI is not done. Instead, the inserted record is put into the new CI (after control has returned to IKQMDY). This example is referred to as "the special case" in this HIPO.

Diagram CC4. CINSRT: Control Interval Insert Initialization



7. Is a control area split being forced because we are running out of space in the index? — IKQCIU10 CINSRT95

8. Is there a free CI (for the split) in the current CA?

9. Attempt to reclaim empty CIs for reuse.

See Diagram CG, Control Interval Space Reclamation. — IKQCIR

10. Were any CIs reclaimed?

11. Get exclusive control of the CA, and then split the CA. — CINSRT98

See Diagram CL, Split Control Area. — IKQCAS00

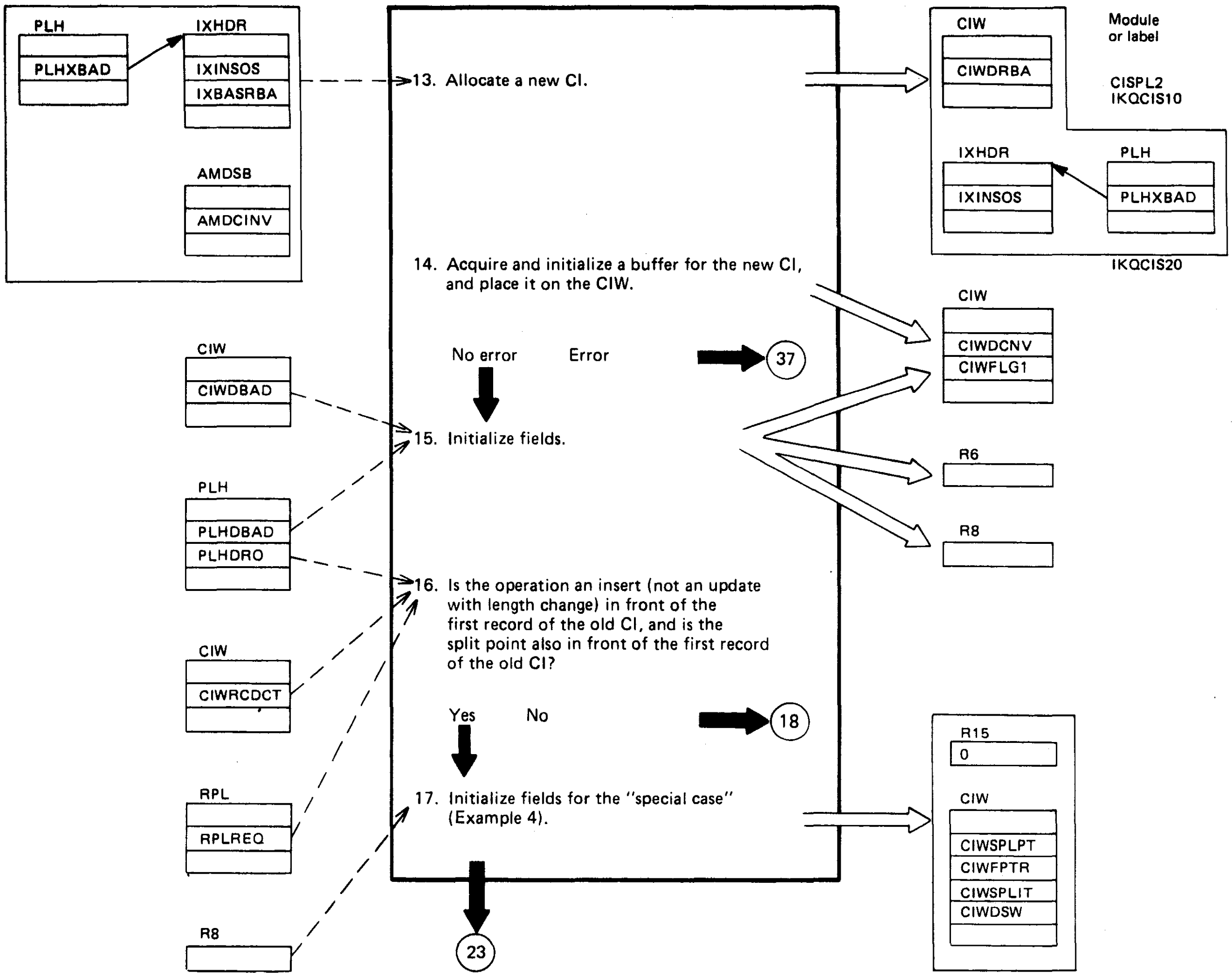
12. Any errors? — CINSRT99

Notes for Diagram CC4

7. The flag CIWNCAS is set by IKQCIS00 (subroutine CINXRM) because a minus 5 was returned from either the index enter (IKQIXE) or index format (IKQIXF) routine, while a sequence set was being processed. (If the condition were detected during processing of a high-level index, IKQIXE would call the CA split routine [IKQCAS] directly.)

10. If register 15 is zero, at least one CI was reclaimed. If register 15 is negative, no CIs were reclaimed.

Diagram CC5. CINSRT: Control Interval Insert Initialization



13. Allocate a new CI.

14. Acquire and initialize a buffer for the new CI, and place it on the CIW.

15. Initialize fields.

16. Is the operation an insert (not an update with length change) in front of the first record of the old CI, and is the split point also in front of the first record of the old CI?

17. Initialize fields for the "special case" (Example 4).

Notes for Diagram CC5

13. A free-CI index entry is located by field IXINSOS (which points to the byte immediately following the free-CI index entry). This pointer is converted to a data RBA.

15. Register 6 is initialized as the current "to" pointer for moving logical records from the old CI to the new CI.

    Register 8 is initialized as the current "from" pointer for moving logical records from the old CI to the new CI.

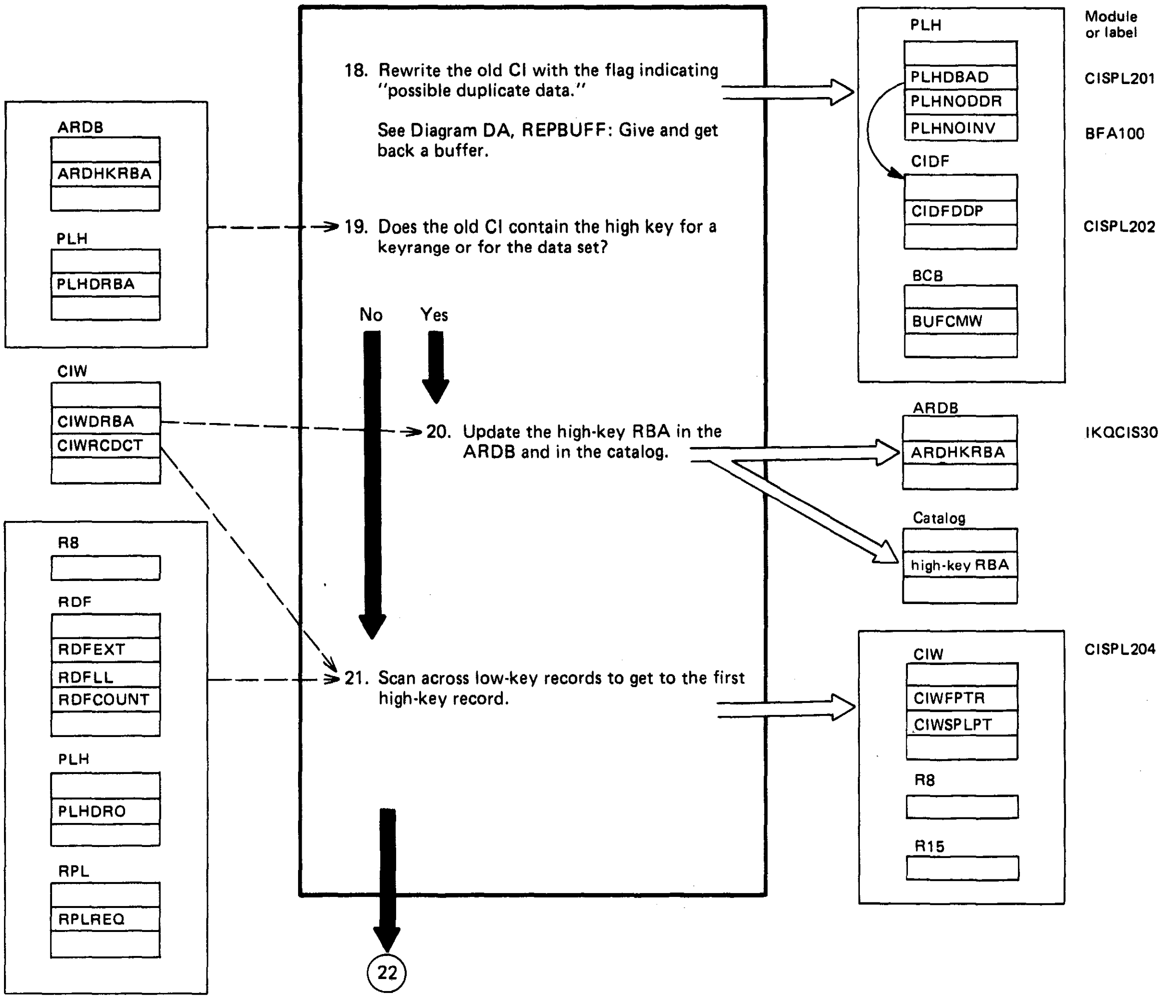
    The CIWFLG1 flags are initialized as required by the index enter (IKQIXE) routine. In particular, the flag CIWSPLIT ("split entry") is set on, as required, for a normal split. It tells IKQIXE to take the new key that will be supplied (for the low-key records) and put it with the pointer to the old CI, and to take the pointer that will be supplied (to the new CI) and put it with the old (high) key.

16. This test provides an optimization so that a newly inserted record will be placed in a CI by itself. Instead of moving the total contents of the old CI to the newly allocated CI, the newly allocated CI will be used for the new low-key record. This case is referred to as "the special case" in Example 4.

    The same procedure applies to a new high-key record so that will end up in a CI by itself.

17. The CIWSPLIT flag is turned off. This causes IKQIXE to make the new entry in the normal manner (not the "split entry" manner). In other words, the entry for the new CI is made with the key of the new record to be inserted. CIWSPLIT is referenced later in IKQCIU for determining whether "the special case" applies (Example 4).

Diagram CC6. CINSRT: Control Interval Insert Initialization



18. Rewrite the old CI with the flag indicating "possible duplicate data."

    See Diagram DA, REPBUFF: Give and get back a buffer.

19. Does the old CI contain the high key for a keyrange or for the data set?

No Yes

20. Update the high-key RBA in the ARDB and in the catalog.

21. Scan across low-key records to get to the first high-key record.

Notes for Diagram CC6

18. The duplicate data feature of the CI-split algorithm protects data if a system crash occurs during a CI split.

    The steps in the CI-split algorithm are:

    A. Copy the high key records to the new CI (on DASD).

    B. Update the sequence set (on DASD).

    C. Erase the high-key records from the old CI (on DASD).

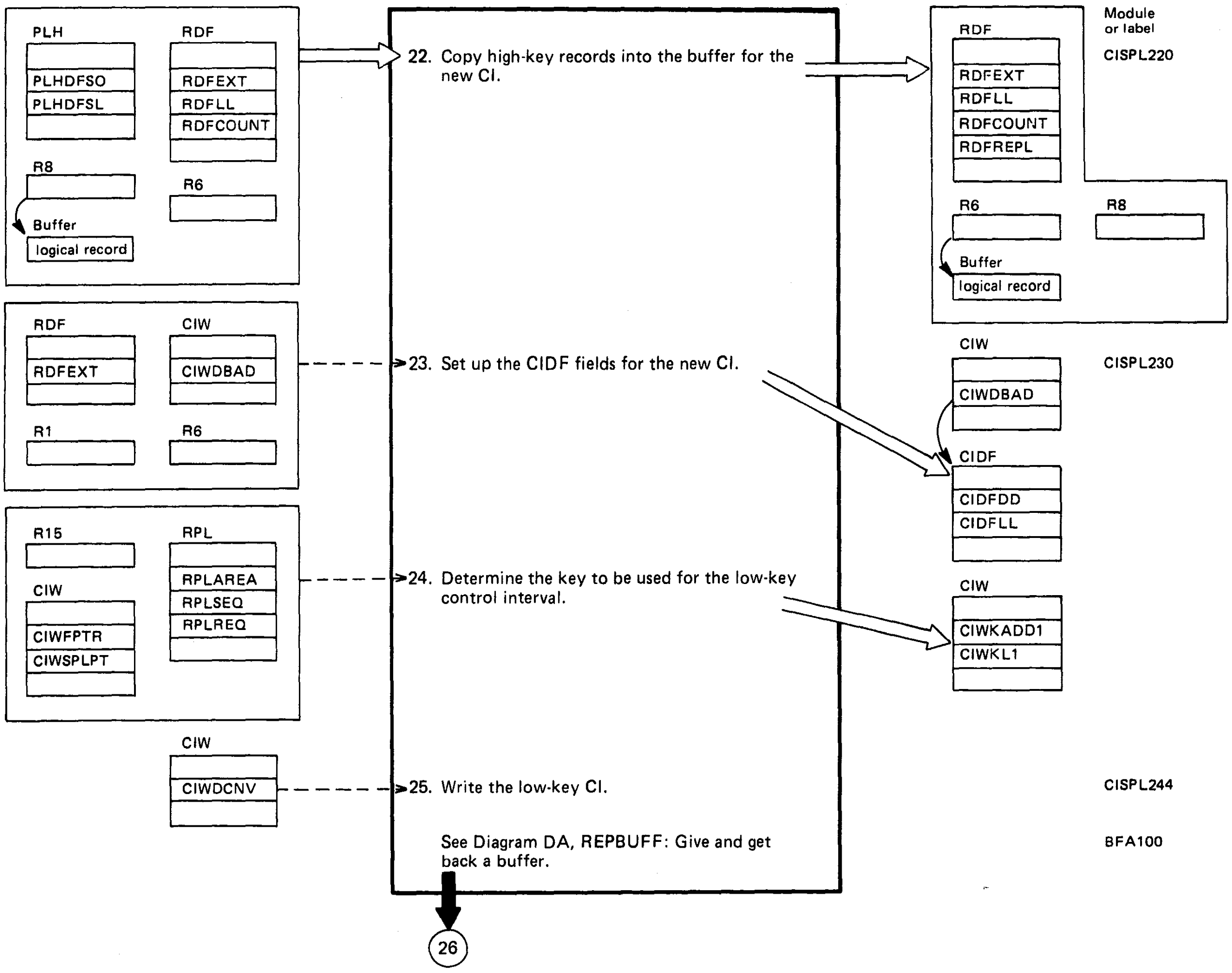
    If a system crash occurs between steps B and C, the split would be essentially complete, but the high-key records would exist in two places. So that this situation can be detected, the flag indicating "possible duplicate data" is turned on in the old CI. This flag serves as a "busy" bit, indicating that this CI is in the process of being split. The flag is turned off when the high-key records are erased from the old CI at the completion of the CI-split process.

    The Buffer Manager writes the buffer (with the flag on) before control is returned to IKQCIU.

    For share option 4, buffer invalidation is suppressed because the lock on the CA protects the buffer.

20. To update the catalog, phase IKQVRBA is called with the code X'08' in the high-order byte of register 0. Register 1 points to field CIWDRBA.

Diagram CC7. CINSRT: Control Interval Insert Initialization



22. Copy high-key records into the buffer for the new CI.

23. Set up the CIDF fields for the new CI.

24. Determine the key to be used for the low-key control interval.

25. Write the low-key CI.

See Diagram DA, REPBUFF: Give and get back a buffer.

Notes for Diagram CC7

22. If journaling is active (AMBJRACT), then journaling is done for each logical record that is moved to the new CI. (See Diagram FD.) Journaling indicates that the RBA of a logical record is being changed.

23. If no records were moved to the new CI, the CI is set up as an empty CI.

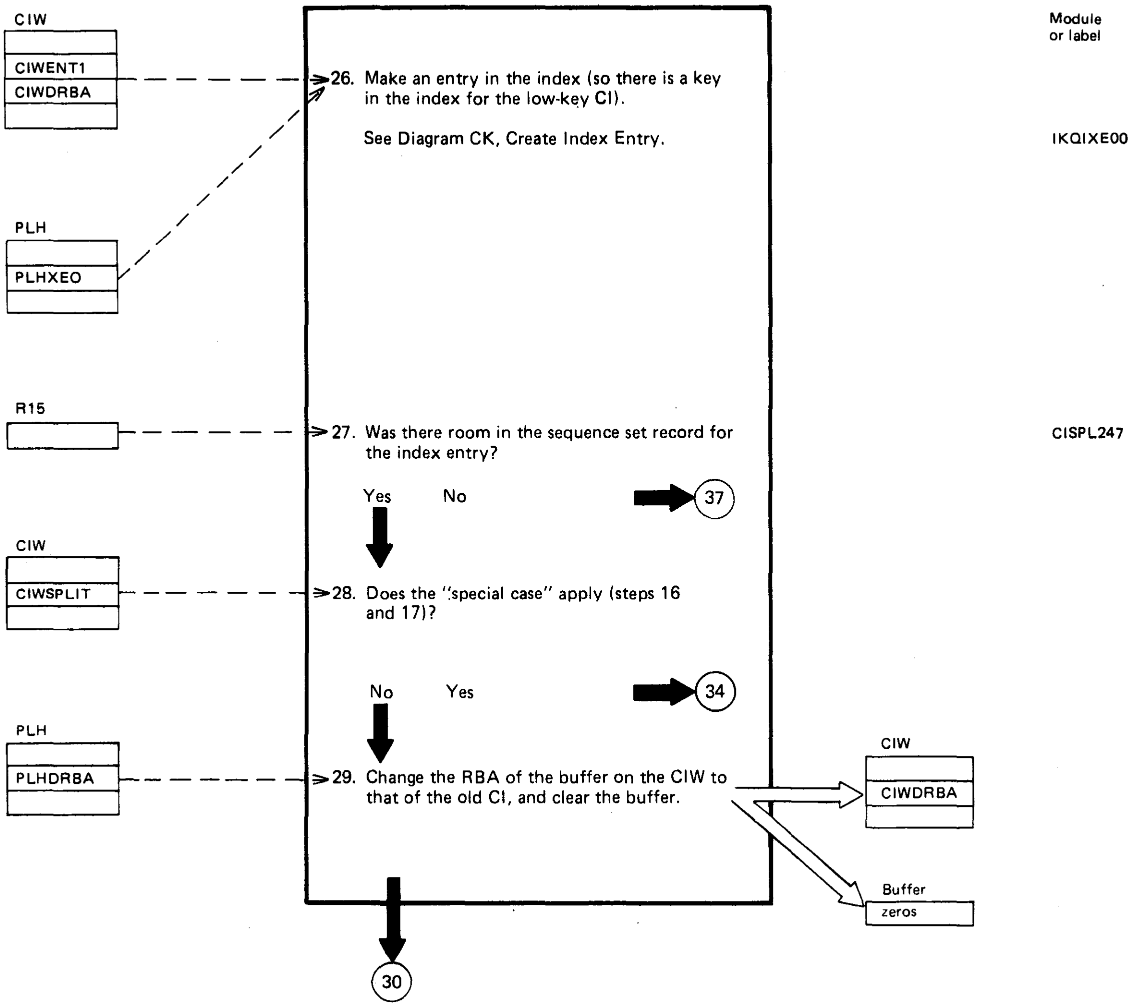
24. In all cases except sequential insert, the key of the record that will be the last record of the low-key CI is the basis for the key of the CI. For sequential insert, the key of the first record of the high-key CI is the basis for the key of the low-key CI. Then, for purposes of index entry, a compression operation is done on this key.

    In all cases except sequential insert, the compression operation is normal rear compression. This consists of finding the first byte necessary to distinguish between the key of the last record of the low-key CI and the key of the first record of the high-key CI. All bytes following this distinguishing byte are discarded for purposes of index entry.

    For sequential insert, the compression operation consists of taking the last nonzero byte of the key of the first record of the high-key CI and decrementing it by binary one. All zero bytes following this nonzero byte are discarded for purposes of index entry. This allows as many key values as possible between the record that is being inserted now (at the end of the low-key CI) and the first record of the next CI.

25. For share option 4, buffer invalidation is suppressed because there is complete protection by the lock facility.

Diagram CC8. CINSRT: Control Interval Insert Initialization



26. Make an entry in the index (so there is a key in the index for the low-key CI).

    See Diagram CK, Create Index Entry.

27. Was there room in the sequence set record for the index entry?

28. Does the "special case" apply (steps 16 and 17)?

29. Change the RBA of the buffer on the CIW to that of the old CI, and clear the buffer.

Notes for Diagram CC8

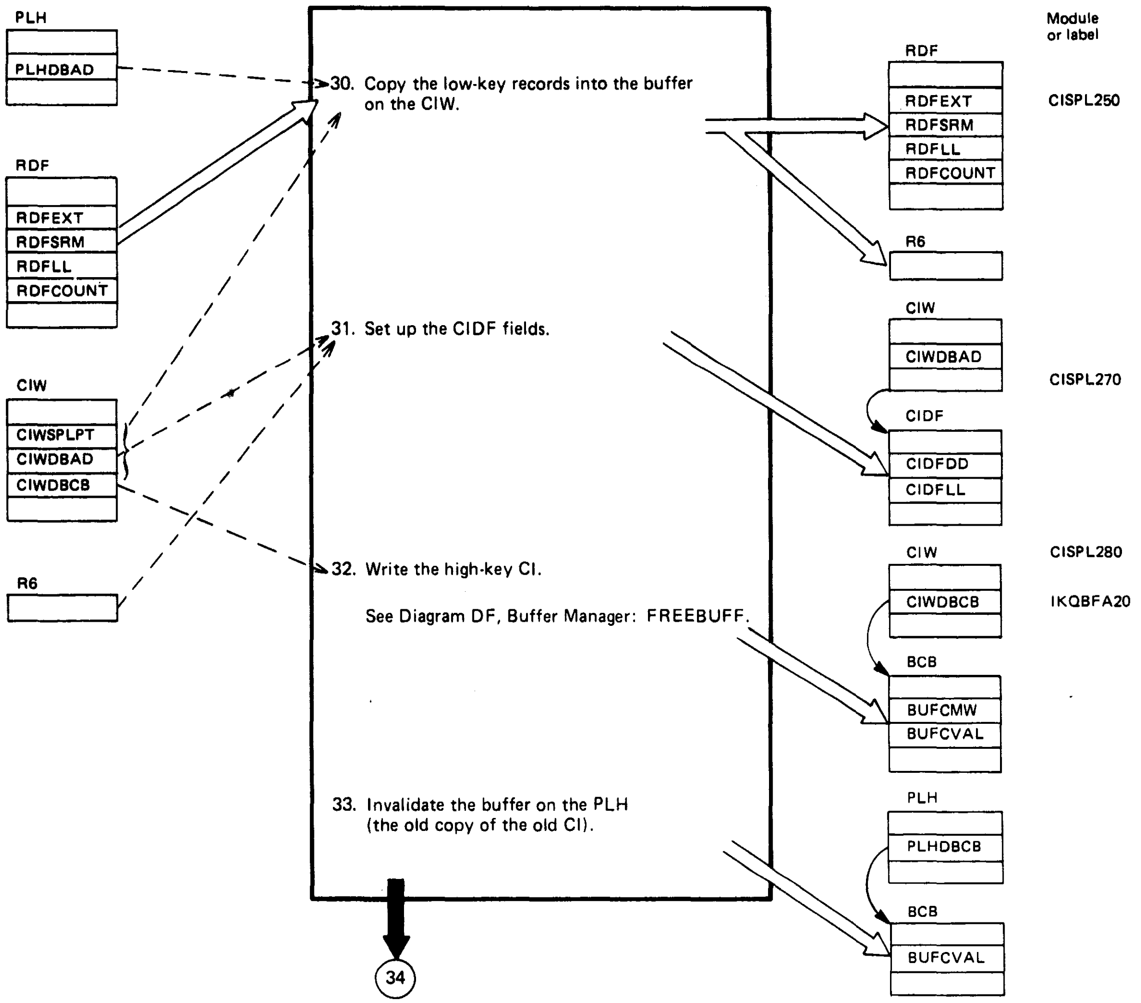
26. Normally the entry is made in the "split entry" manner. The new key is put with the pointer to the old CI, and the pointer to the new CI is put with the old key. For "the special case" (Example 4), the entry is made in the normal manner. (A complete new entry is created.)

27. If there wasn't enough room for the entry, the index enter routine (IKQIXE) will have returned a minus 5 in register 15. Control is also given to step 37 for any other errors (such as an I/O error) detected by IKQIXE.

    For a minus 5 return code, CI-split processing is forced, along with a forced CA split. CI-split processing is re-entered at IKQCIU10.

29. Records are deleted from the old CI by copying the records to be saved into a scratch buffer (on the CIW).

Diagram CC9. CINSRT: Control Interval Insert Initialization



30. Copy the low-key records into the buffer on the CIW.

31. Set up the CIDF fields.

32. Write the high-key CI.

    See Diagram DF, Buffer Manager: FREEBUFF.

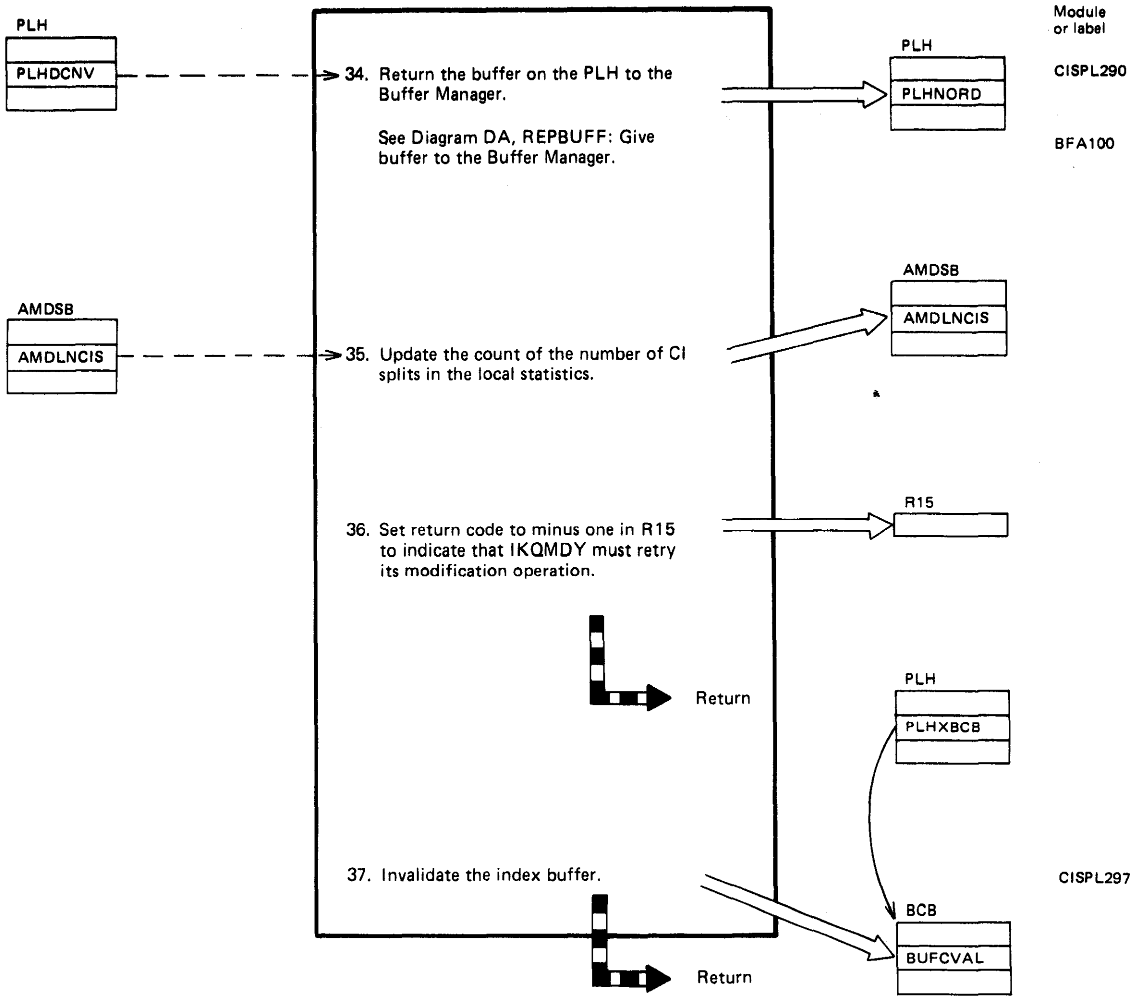
33. Invalidate the buffer on the PLH (the old copy of the old CI).

Notes for Diagram CC9

30. Register 6 points to the first byte of freespace in the scratch buffer at completion of the copy.

31. If no records were moved to the buffer on the CIW, the buffer is set up as an empty CI.

Diagram CC10. CINSRT: Control Interval Insert Initialization



34. Return the buffer on the PLH to the Buffer Manager.

    See Diagram DA, REPBUFF: Give buffer to the Buffer Manager.

35. Update the count of the number of CI splits in the local statistics.

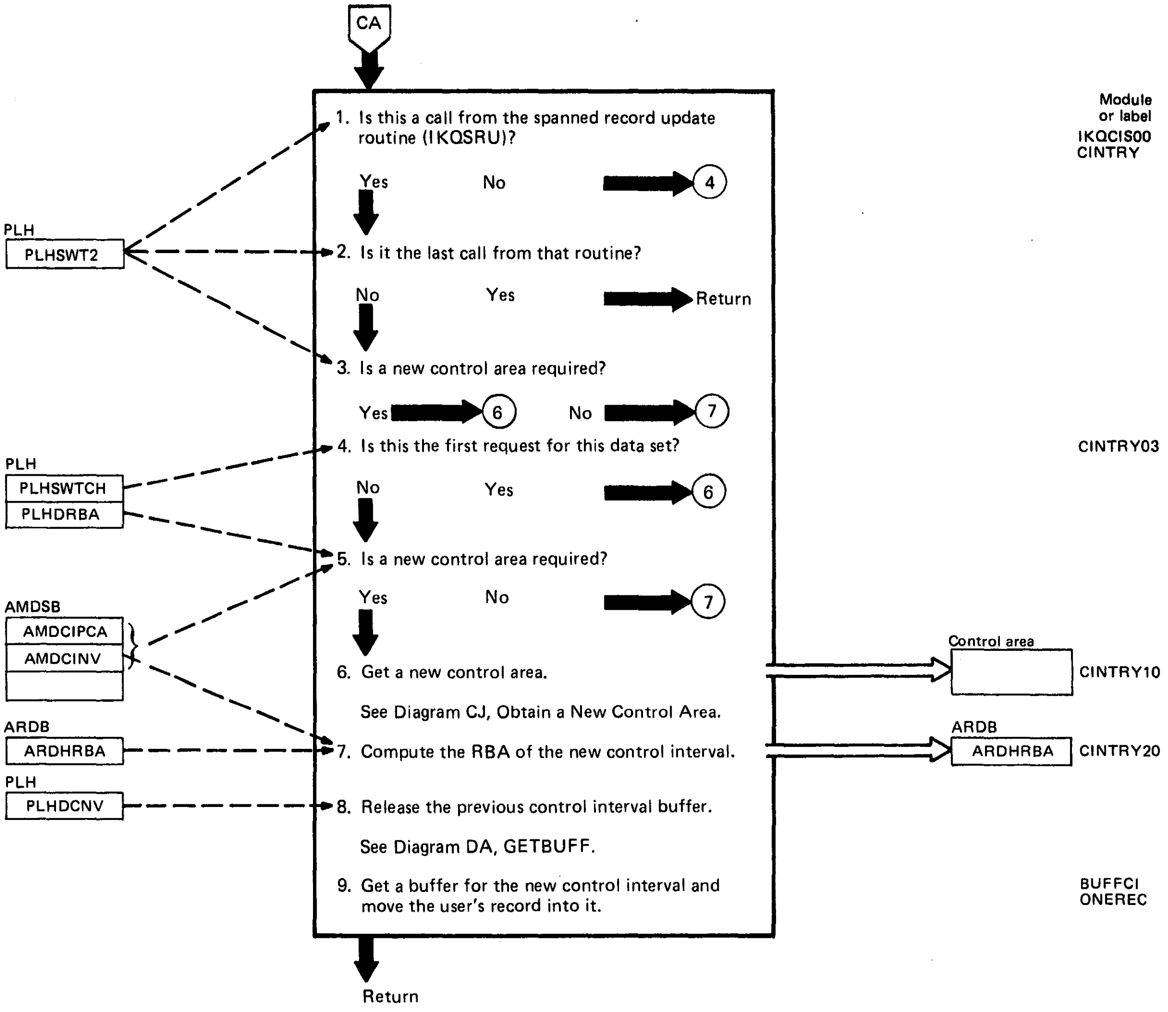
36. Set return code to minus one in R15 to indicate that IKQMDY must retry its modification operation.

37. Invalidate the index buffer.

Notes for Diagram CC10

34. The PLHNORD flag is set on in the PLHDCNV Buffer Manager Parameter List so the Buffer Manager will keep the buffer and not give any buffers back to IKQCIU.

Diagram CD1. CINTRY: Entry-sequenced Data Set Processing



1. Is this a call from the spanned record update routine (IKQSRU)?

2. Is it the last call from that routine?

3. Is a new control area required?

4. Is this the first request for this data set?

5. Is a new control area required?

6. Get a new control area.

   See Diagram CJ, Obtain a New Control Area.

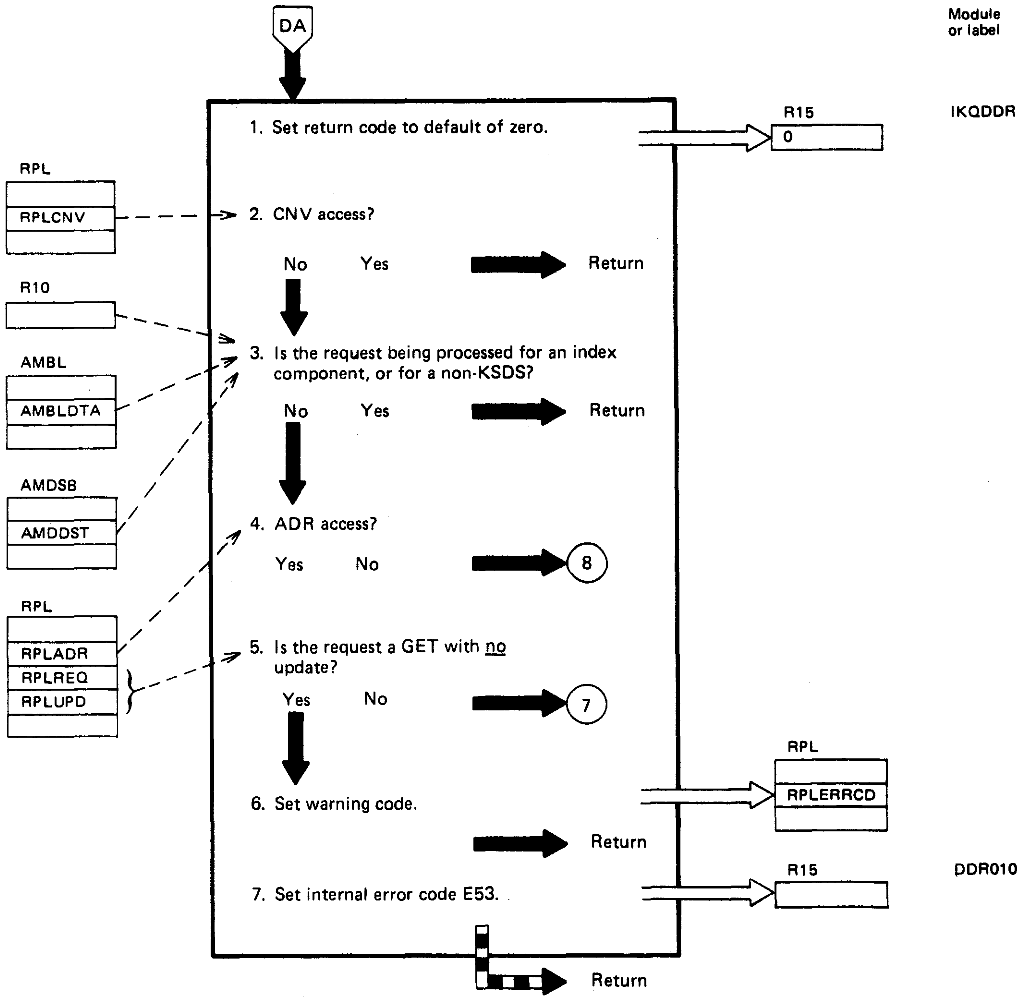
7. Compute the RBA of the new control interval.

8. Release the previous control interval buffer.

   See Diagram DA, GETBUFF.

9. Get a buffer for the new control interval and move the user's record into it.

Diagram CE1. Duplicate Data Recovery



Notes for Diagram CE1

1. IKQDDR is called by the Buffer Manager when a CI is being retrieved and the CIDFDDP ("possible duplicate data") bit is on (except when the Buffer Manager has been explicitly told not to check CIDFDDP).

   CIDFDDP is set on when a copy of records in a KSDS CI is about to be made as part of a CI split (see Diagram CC). If a system crash occurred during the split, two copies of the records would be left on DASD. CIDFDDP prevents any program from accessing the old copy of the records once the new copy has been created. Note that because CIDFDDP is set on before the new copy is actually created, there might not actually be duplicate records in the CI.

   IKQDDR performs recovery when keyed access is being done. IKQDDR determines, using the index, whether an actual duplicate data situation exists. If it does exist, the old copies of the duplicated records are removed from the CI.

   The correct CI is written to DASD if the retrieval is part of an update or insert operation. Otherwise, the correction is only made available in the buffer.

2. If CNV access is being done, no attempt at duplicate data recovery is made, and no diagnostic is issued. No attempt is made to do recovery because the index is not available. No diagnostic is issued because CIDFDDP does not necessarily indicate an abnormal condition. (The user can validly use the entire CI, including the CIDF, as long as he only uses CNV access.)

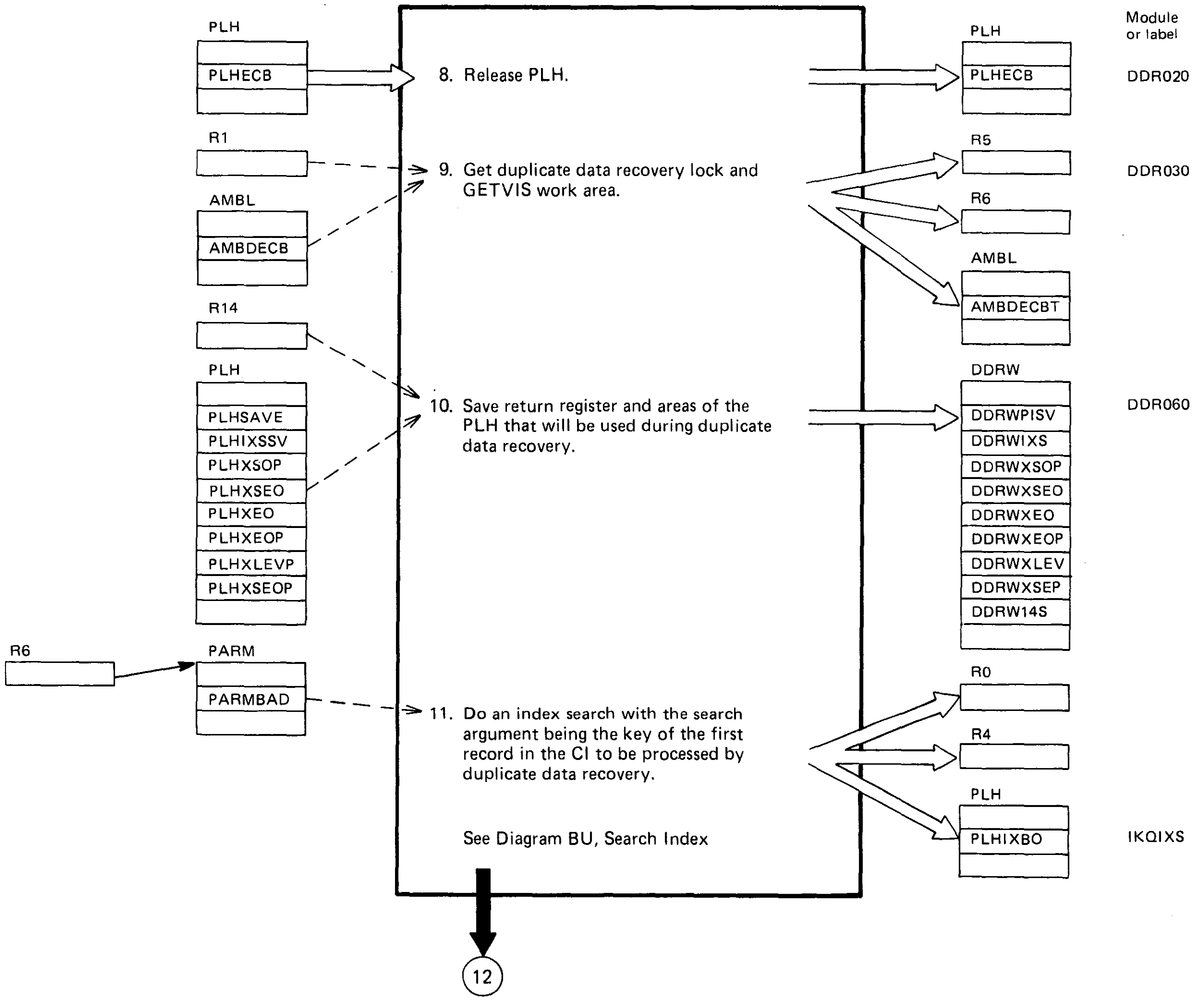
4. If ADR access is being done, it is assumed that the index is not available. No duplicate data recovery is attempted; a diagnostic is issued.

6. For ADR access when the request is a GET-with-no-update (GET NUP or GET NSP), the request is completed for the user, but a warning (R15=0) diagnostic is issued. RPLCIWNG (X'1C') is the RPL error code for this warning.

   For ADR access on GET-with-no-update, the user runs the risk of processing duplicate records if he fails to test RPLCIWNG.

   CIDFDDP is turned off at this time to allow VSAM to complete the request.

Diagram CE2. Duplicate Data Recovery


Module or label
PLH
PLHECB
8. Release PLH.
PLH
PLHECB
DDR020
R1
9. Get duplicate data recovery lock and GETVIS work area.
AMBL
AMBDECB
R5
R6
AMBL
AMBDECBT
DDR030
R14
PLH
PLHSAVE
PLHIXSSV
PLHXSOP
PLHXSEO
PLHXEO
PLHXEOP
PLHXLEVP
PLHXSEOP
10. Save return register and areas of the PLH that will be used during duplicate data recovery.
DDRW
DDRWPISV
DDRWIXS
DDRWXSOP
DDRWXSEO
DDRWXEO
DDRWXEOP
DDRWXLEV
DDRWXSEP
DDRW14S
DDR060
R6
PARM
PARMBAD
11. Do an index search with the search argument being the key of the first record in the CI to be processed by duplicate data recovery.
See Diagram BU, Search Index
R0
R4
PLH
PLHIXBO
IKQIXS
12

Notes for Diagram CE2

9. It has been determined that actual duplicate data recovery is to be performed. A lock is obtained to serialize all duplicate data recovery for this ACB. A work area is GETVISed. The only purpose of the lock is to make the storage requirement for duplicate data recovery predictable (that is, one work area per ACB).

   Register 1 is copied into register 6 to serve as a base for the Buffer Manager Parameter List.

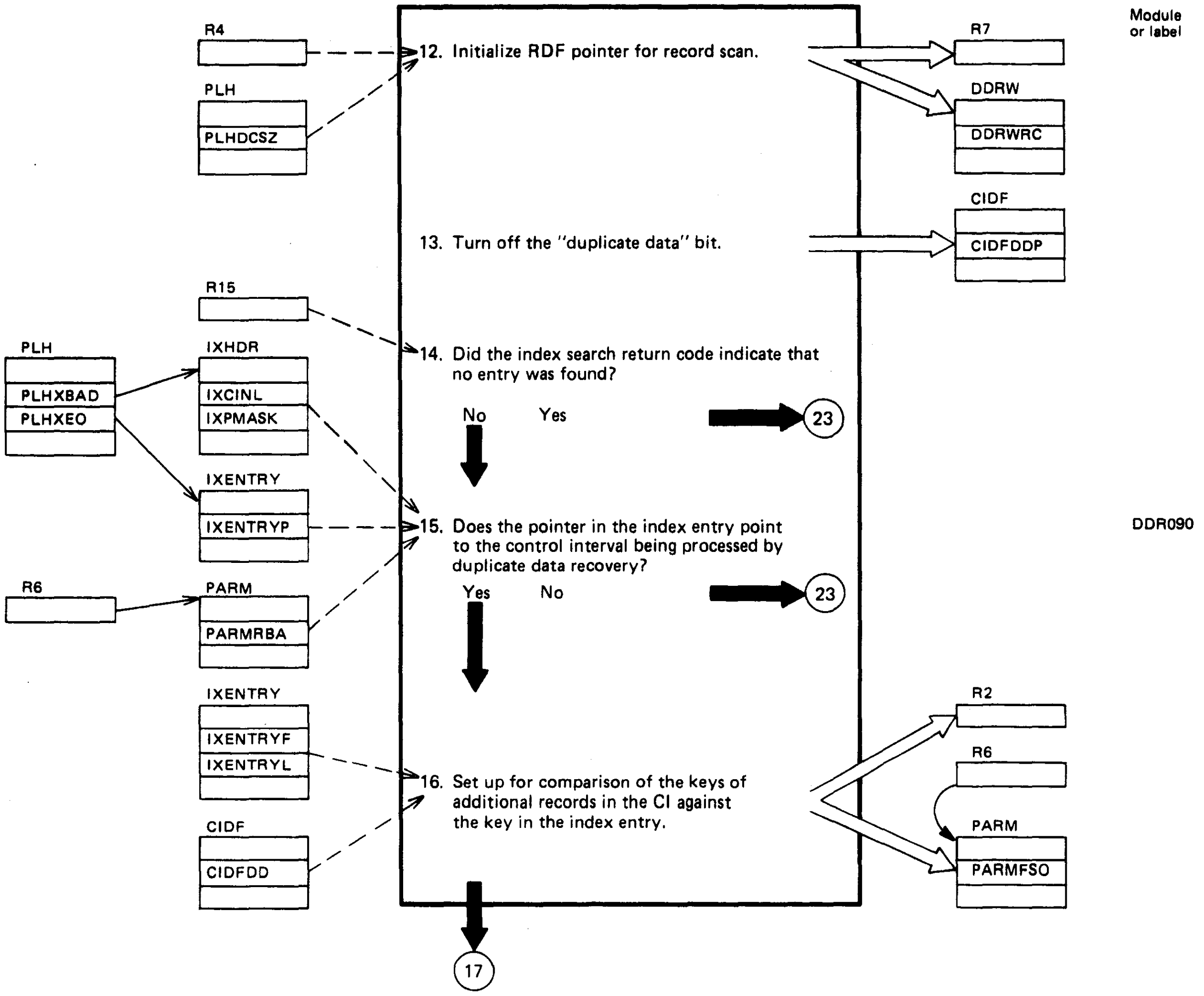
10. The information saved will be destroyed when IKQDDR calls index search (IKQIXS) and the Buffer Manager (IKQBFA).

11. A search is done against the index record in working storage to find the entry that corresponds to the key of the first data record in the CI to be processed by duplicate data recovery.

    The flag PLHIXBO is set on to tell index search not to follow the horizontal index chain pointer if it fails to find the appropriate index entry in the current index record.

    Register 4 is set up as a pointer to the next record in the CI to be processed by duplicate data recovery.

Diagram CE3. Duplicate Data Recovery


Module or label
R4
PLH
PLHDCSZ
12. Initialize RDF pointer for record scan.
R7
DDRW
DDRWRC
CIDF
13. Turn off the "duplicate data" bit.
CIDFDDP
R15
PLH
PLHXBAD
PLHXEO
IXHDR
IXCINL
IXPMASK
14. Did the index search return code indicate that no entry was found?
No
Yes
23
IXENTRY
IXENTRYP
15. Does the pointer in the index entry point to the control interval being processed by duplicate data recovery?
DDR090
R6
PARM
PARMRBA
Yes
No
23
IXENTRY
IXENTRYF
IXENTRYL
R2
R6
16. Set up for comparison of the keys of additional records in the CI against the key in the index entry.
CIDF
CIDFDD
PARM
PARMFSO
17

Notes for Diagram CE3

12. The record scan that follows will determine if the index entry located by the index search (step 11) is correct for each additional record in the CI to be processed by duplicate data recovery. The search for duplicate records ends when the first duplicate record is found. All subsequent records will have higher keys and will be duplicates also.

    The record count accumulator (DDRWRC) used during the scan is cleared at this time.

13. CIDFDDP is turned off here because addressability is now available to the CIDF.

14. If the index search did not identify the first logical record as belonging to this CA (that is, its key is greater than the highest key in the CA), then it and all records in the CI are duplicates, and the scan stops here.

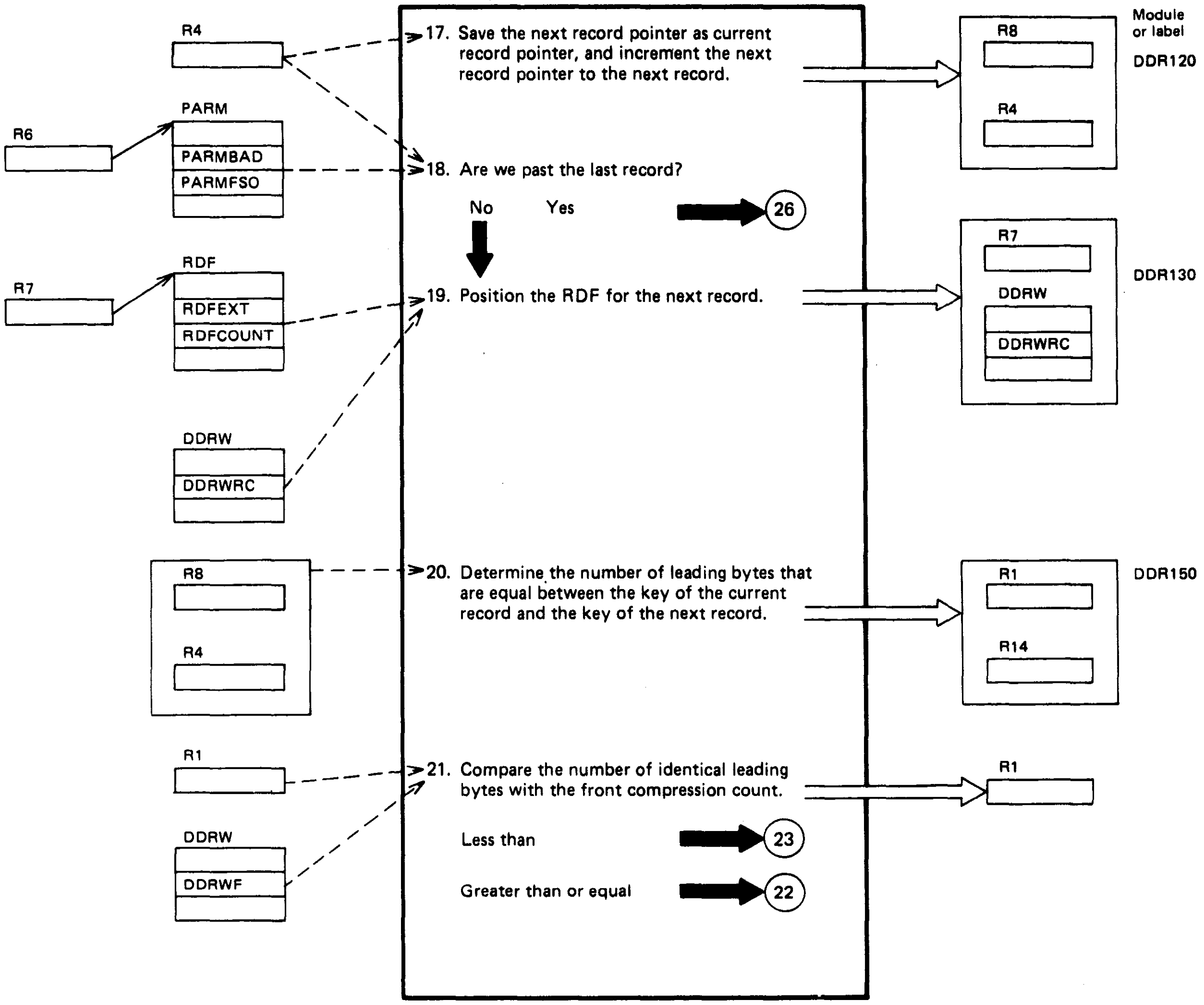
15. If the index entry points to a different data CI, then the first record and all records following it are duplicates and must be erased from the CI.

16. At this point it has been verified that the first logical record in the CI is not a duplicate record. Now the rest of the logical records in the CI must be checked. The algorithm for making the check takes advantage of the nature of front compression. (See Diagram BT for a description of front compression.)

    Because front compression consists of eliminating the leading positions in the index entry key that are duplicates of the preceding CI's key entry, all keys in the CI have the same number of identical leading characters as the front compression count. The algorithm for checking records after the first record takes advantage of this fact. Because there is no preceding CI for the first index entry in a sequence set record, it will always have a front compression count of zero. This is acceptable within this algorithm.

    Register 2 is set up as a pointer to the index entry key.

Diagram CE4. Duplicate Data Recovery



17. Save the next record pointer as current record pointer, and increment the next record pointer to the next record.

18. Are we past the last record?

19. Position the RDF for the next record.

20. Determine the number of leading bytes that are equal between the key of the current record and the key of the next record.

21. Compare the number of identical leading bytes with the front compression count.

Notes for Diagram CE4

18. If the current record is the last record, the search is ended with no record found.

19. If the current RDF is an extended RDF for multiple records of the same length, the record count accumulator (DDRWRC) is compared to the record count in the extended RDF. If they are equal, VSAM moves to the next RDF and resets DDRWRC to zero. If they are not equal, one is added to DDRWRC, and RDF positioning remains unchanged.

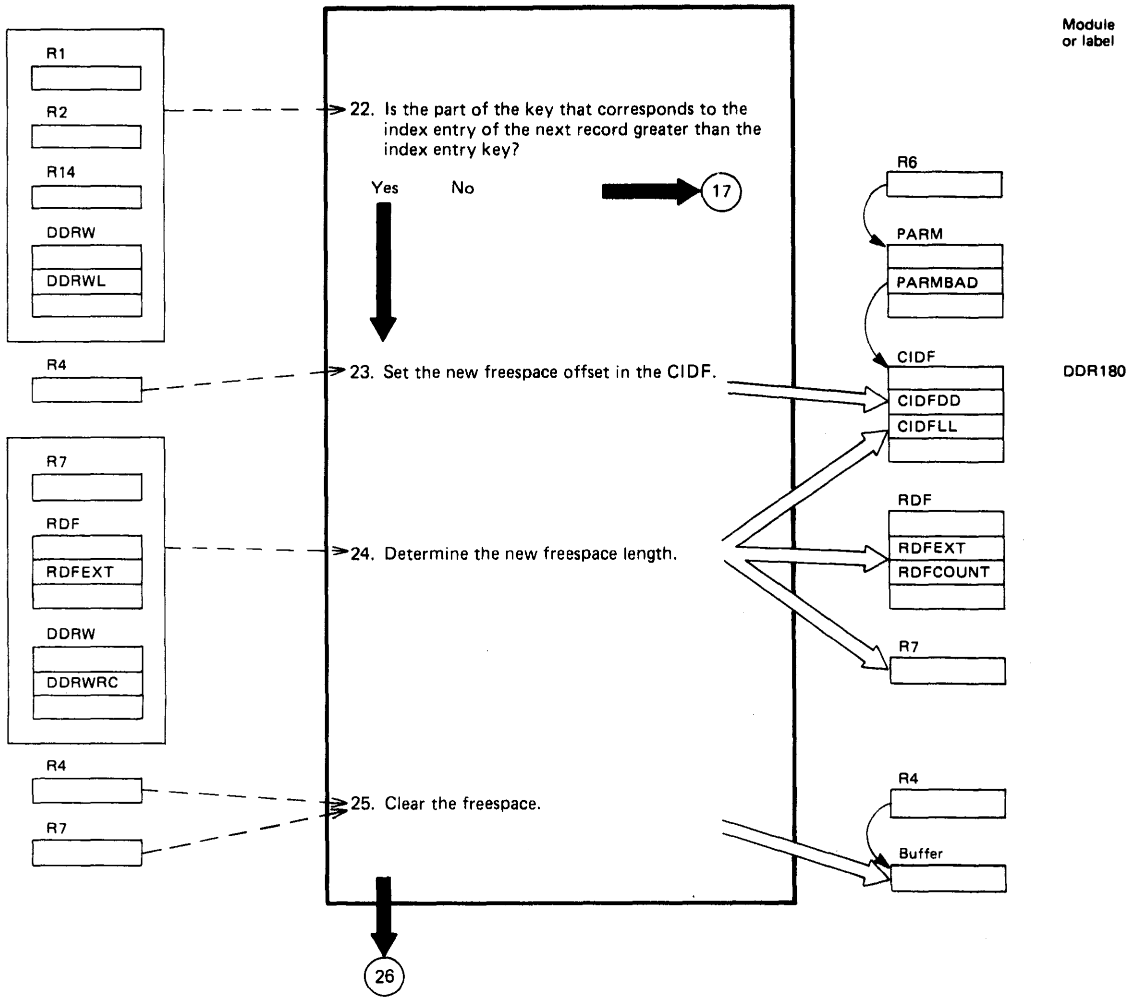
    If a duplicate record is found that is one of a string of records of identical length, DDRWRC contains the number of non-duplicate records in the string.

20. The key of the current record, which is not a duplicate, is compared with the key of the next record to get the number of leading bytes that are identical. This number must be equal to or greater than the front compression count of the index entry for this CI. If it is less, the next record is a duplicate, and the search terminates.

    If the number of identical leading bytes is greater than or equal to the front compression count, the part of the key of the next record that corresponds to the index entry key must be compared to the index entry key. (The length of the compare is the length of the key entry in the index.) If the part of the key in the record is greater than the index entry key, the record is a duplicate. Otherwise, the search continues.

    This step places the number of identical leading bytes in R1. R14 contains the address of the first byte that is not identical in the next record.

Diagram CE5. Duplicate Data Recovery



22. Is the part of the key that corresponds to the index entry of the next record greater than the index entry key?

Yes No

23. Set the new freespace offset in the CIDF.

24. Determine the new freespace length.

25. Clear the freespace.

Notes for Diagram CE5

23. At this point, the first duplicate record has been found. R4 points to this record, and R7 points to the associated RDF. In the logic that follows, this record and all following (higher key) records will be erased.

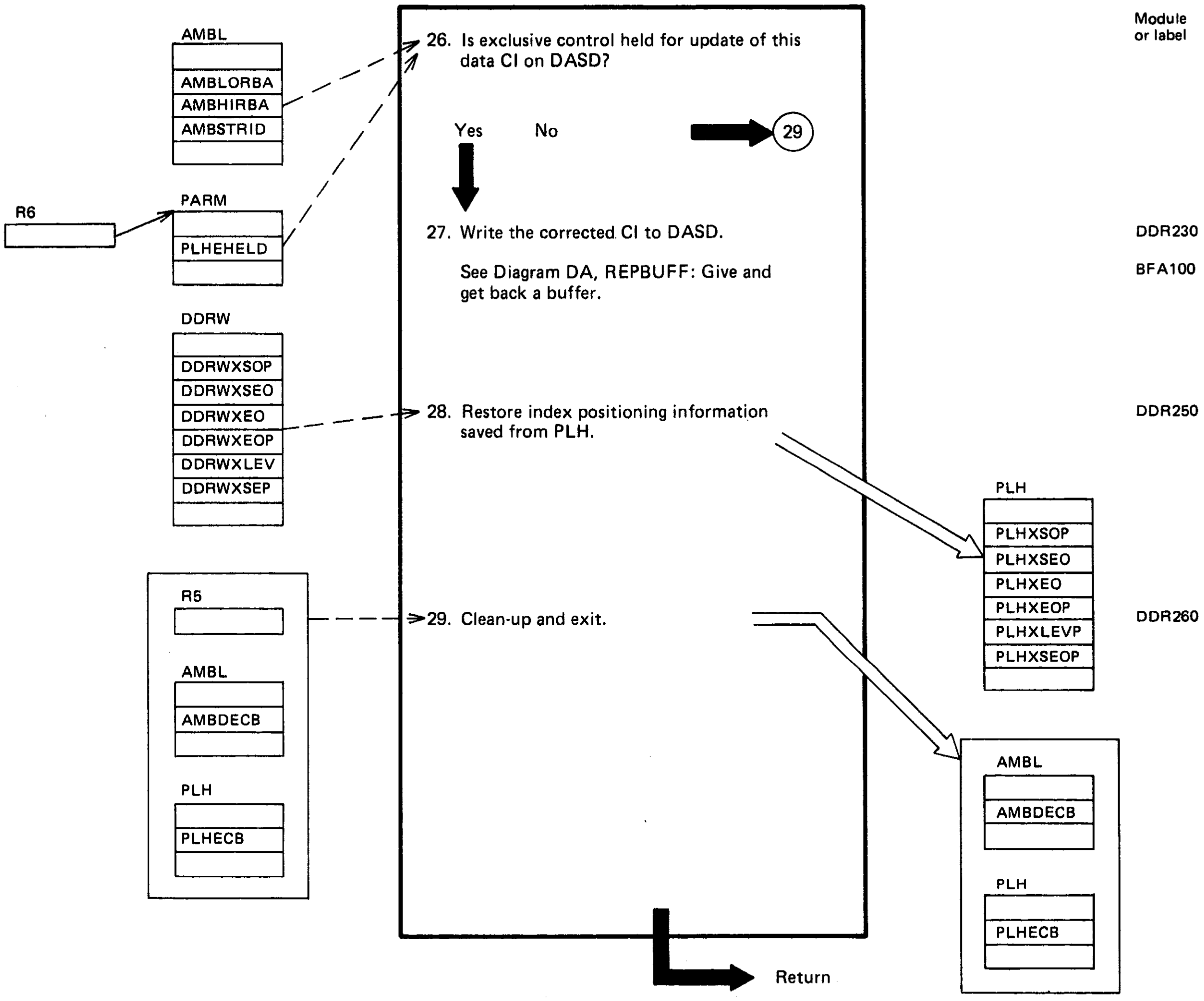
24. If the RDF of the first duplicate record is not an extended RDF, the RDF pointer is merely incremented by the length of a simple RDF, and the RDF of the first duplicate record becomes absorbed as a part of the new freespace area.

    If the RDF associated with the first duplicate record is an extended RDF, the record count accumulator (DDRWRC) is equal to the number of non-duplicate records this RDF is associated with.

    If DDRWRC is zero, all records identified by the extended RDF are duplicates, and the entire extended RDF becomes freespace. If DDRWRC is one, the extended RDF must be converted to a simple RDF, and the extension becomes freespace. If DDRWRC is more than one, the RDF remains intact, but with a new count. Additional RDFs associated with any subsequent records are absorbed as part of the freespace.

    At completion of this step, R7 contains the freespace length.

Diagram CE6. Duplicate Data Recovery



Notes for Diagram CE6

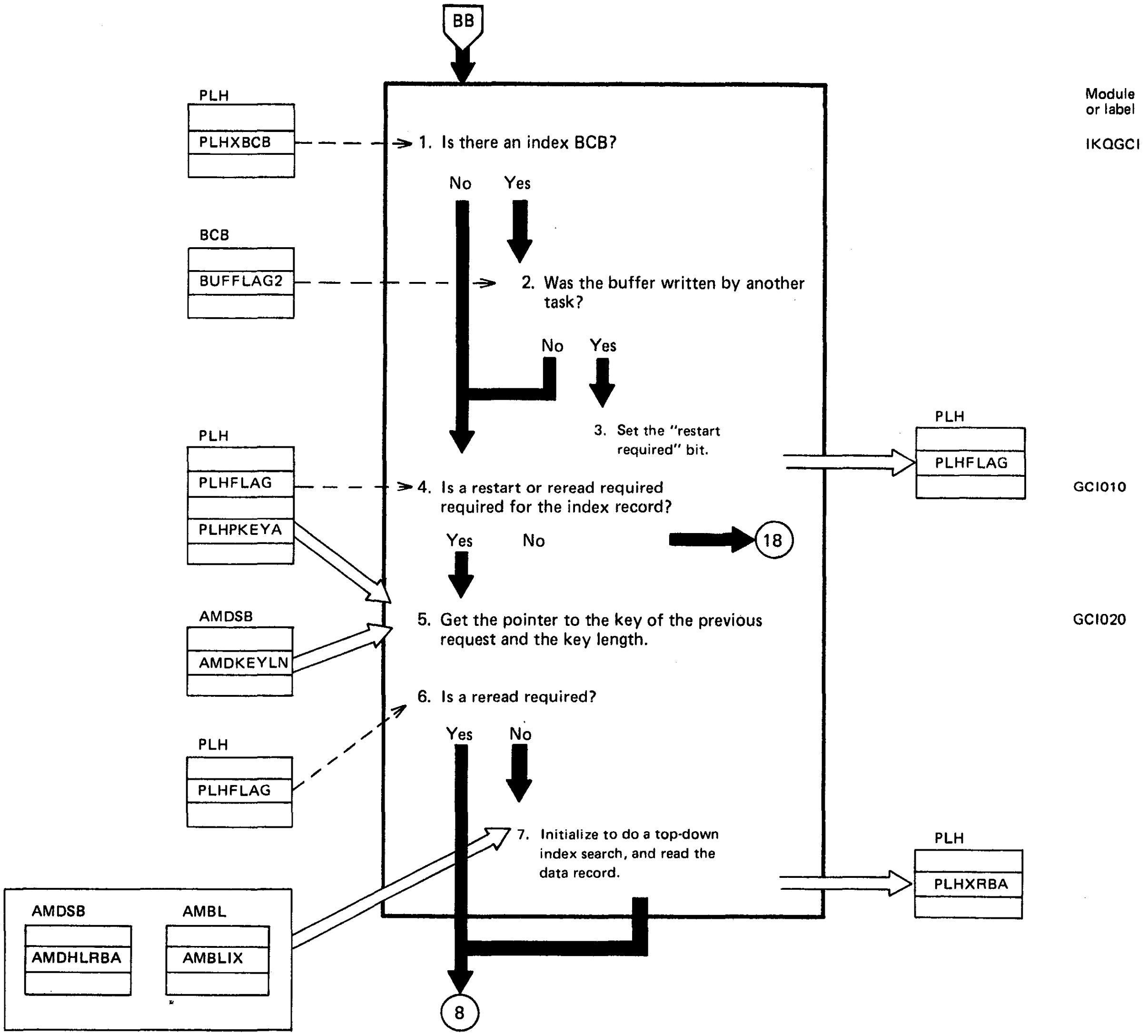
26. The corrected CI (including the case of merely turning off CIDFDDP when there are no actual duplicate records) is written to DASD only if exclusive control for update has been obtained previously (that is, if the request was a GET UPD, PUT, or ERASE).

27. Note that this is a recursive call to the Buffer Manager because IKQDDR was called from the Buffer Manager. Exclusive control is, of course, not released by this request to the Buffer Manager. The request is handled so that the Buffer Manager will complete the I/O before returning to IKQDDR.

    For share option 4, buffer invalidation is suppressed because the lock on the CA protects the buffer.

29. Clean-up includes setting the correct return code, freeing the storage occupied by the DDR work area, releasing the duplicate data recovery lock, and reacquiring the PLH lock (PLHECB).

Diagram CF1. Get Control Interval by Key



BB
Module or label
PLH
PLHXBCB
1. Is there an index BCB?
IKQGCI
No
Yes
BCB
BUFFLAG2
2. Was the buffer written by another task?
No
Yes
3. Set the "restart required" bit.
PLH
PLHFLAG
PLH
PLHFLAG
PLHPKEYA
4. Is a restart or reread required required for the index record?
GCI010
Yes
No
18
AMDSB
AMDKEYLN
5. Get the pointer to the key of the previous request and the key length.
GCI020
6. Is a reread required?
Yes
No
PLH
PLHFLAG
7. Initialize to do a top-down index search, and read the data record.
PLH
PLHXRBA
AMDSB
AMDHLRBA
AMBL
AMBLIX
8

Notes for Diagram CF1

1-3. The PLH is tested to determine if it contains a pointer to an index BCB. If there is no index BCB, or if the buffer contents are invalid, a top-down index search must be done. If there is an index BCB but the buffer contents are invalid, the "restart required" bit (PLHRST) is set to force a top-down index search.

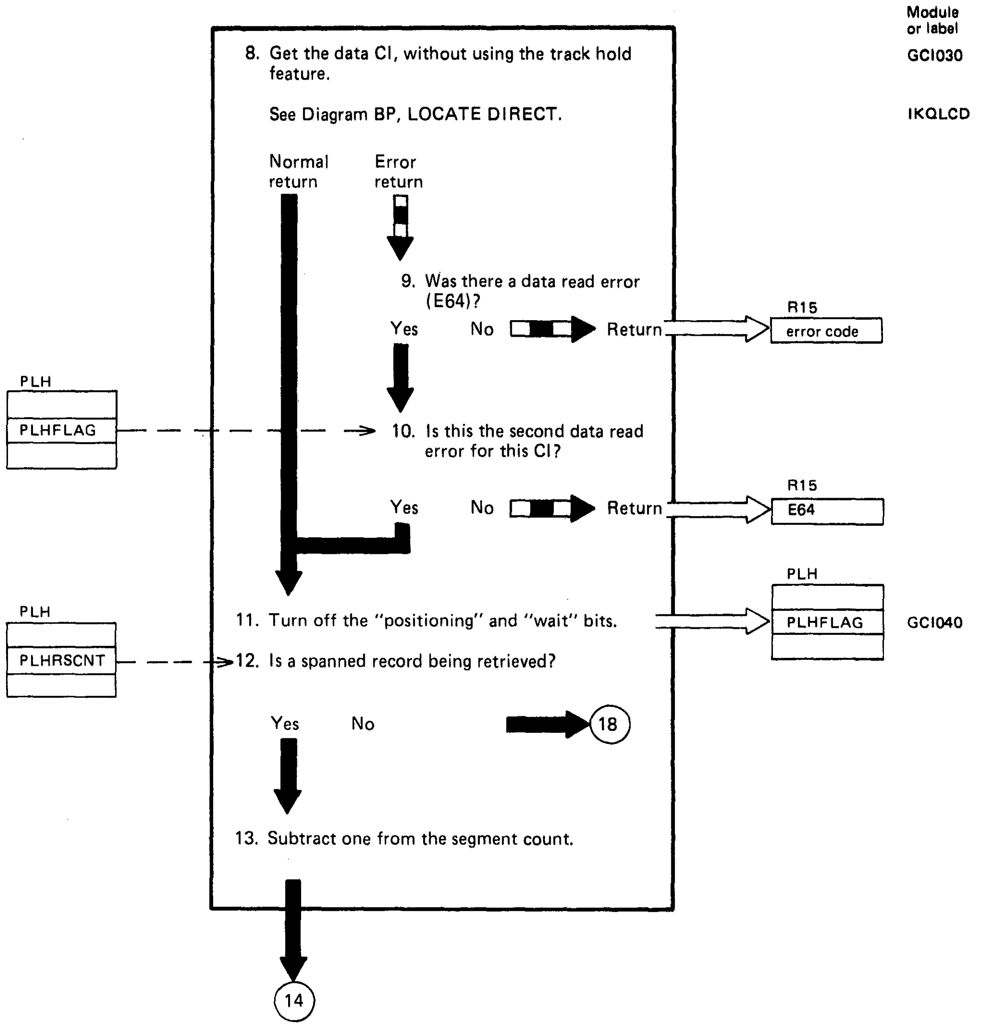
4. If either a restart (PLHRST) or a reread (PLHRREAD) is required, positioning is performed by a top-down index search to the previous CI.

   Note: PLHRST is set when an event has occurred that makes positioning information unreliable. Positioning can be re-established by using the key saved from the last record retrieved. If PLHRREAD is set, it indicates that positioning still points to the correct sequence set record, but the record must be reread and searched for the given key. PLHRREAD is set only if the BUFWRINV bit (buffer invalid) was set.

5. This step sets up repositioning to the last CI that was successfully retrieved. The key is that of the last record in the CI.

6-7. If a restart is required, a top-down index search must be done because positioning information is no longer valid.

Diagram CF2. Get Control Interval by Key


Module
or label
8. Get the data CI, without using the track hold feature.
GCI030
See Diagram BP, LOCATE DIRECT.
IKQLCD
Normal return
Error return
9. Was there a data read error (E64)?
Yes
No
Return
R15
error code
PLH
PLHFLAG
10. Is this the second data read error for this CI?
Yes
No
Return
R15
E64
PLH
PLHFLAG
PLH
11. Turn off the "positioning" and "wait" bits.
GCI040
PLH
PLHRSCNT
12. Is a spanned record being retrieved?
Yes
No
18
13. Subtract one from the segment count.
14

Notes for Diagram CF2

8-10. Locate Direct is called, and share option 4 processing is ignored. The last CI processed is located to re-establish positioning for retrieving the next CI.

If no errors occur, normal processing continues. If any error has occurred (except a data read error), control returns to the caller.

If there is a data read error, the "skip" bit (PLHSKIP) is tested. If the bit is on, IKQGCI continues to call IKQGNX (via a loop) until a successful retrieval occurs. A data record is reread only if it is in the first CI is a contiguous group of "non-readable" CIs.

11. The "positioning" (PLHPOS - X'40') and "wait" (PLHWAIT - X'10') bits are reset at this point. The combinations of these two bits have the following meanings:

.1.1 . . . . The next CI has been requested; a wait is required for I/O completion. IKQGCI must call the Buffer Manager to retrieve the CI.

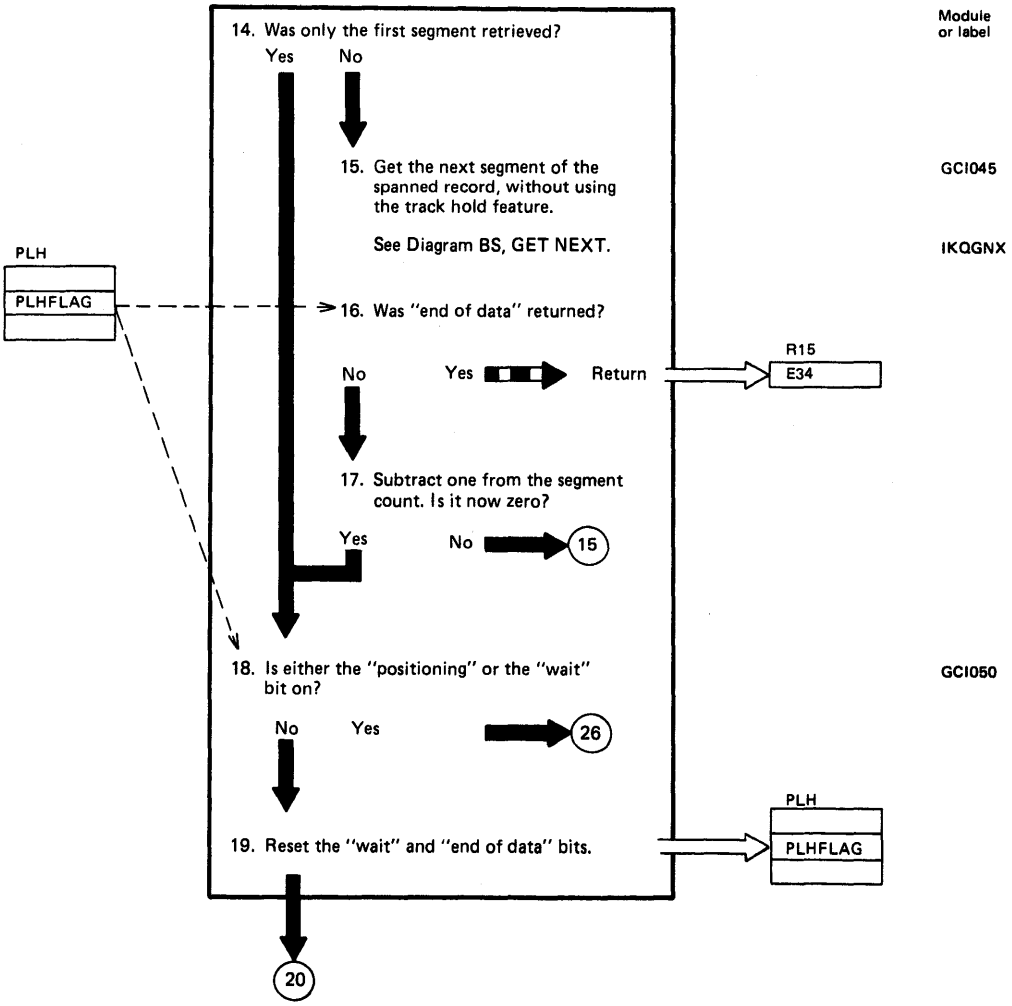
.1.0 . . . . The next CI is in the buffer and available for processing.

.0.1 . . . . Invalid combination; should not occur.

.0.0 . . . . The pointer is still positioned to the previous CI. I/O must be performed for the next CI.

12-17. If spanned records are being processed and restart occurred, Locate Direct has repositioned to the first segment of the spanned record. That segment and all following segments (including the last segment read) were bypassed before the restart occurred. A test is made to determine if *only* the first segment has been previously read (step 14).

Diagram CF3. Get Control Interval by Key

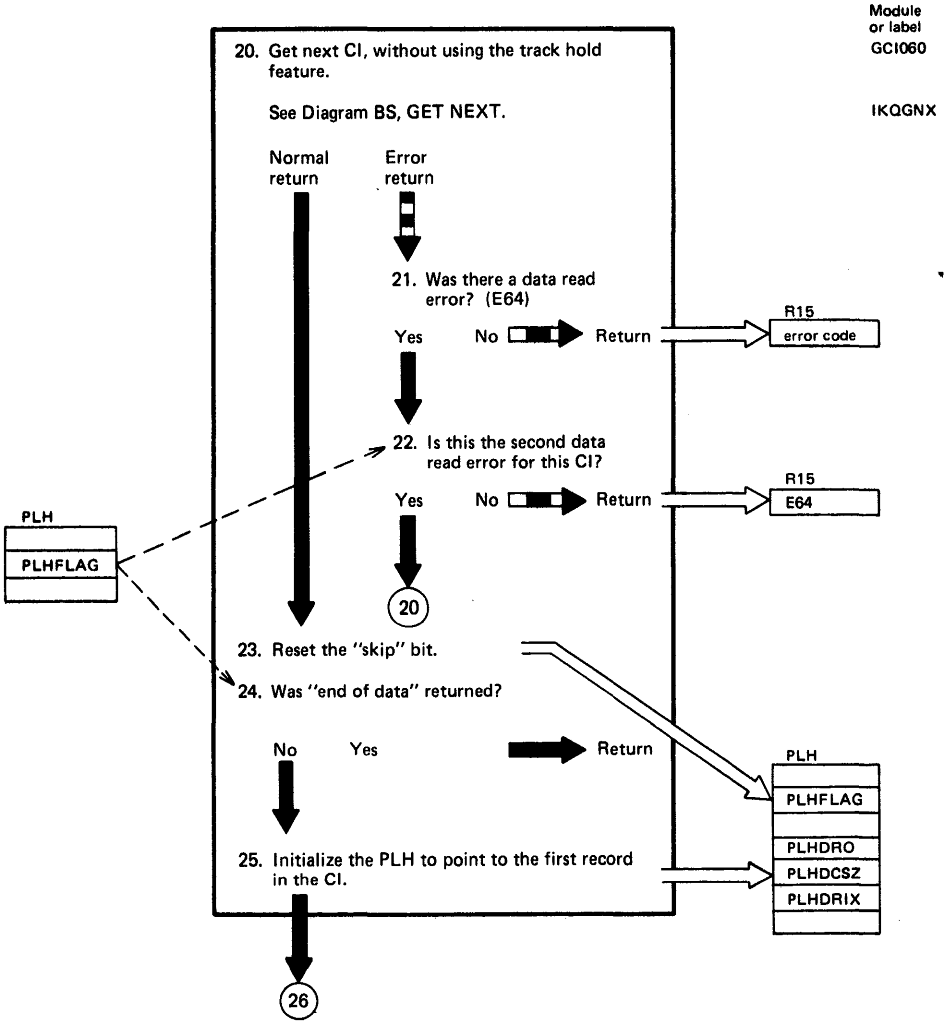

Module or label
14. Was only the first segment retrieved?
Yes
No
15. Get the next segment of the spanned record, without using the track hold feature.
GCI045
See Diagram BS, GET NEXT.
IKQGNX
PLH
PLHFLAG
16. Was "end of data" returned?
No
Yes
Return
R15
E34
17. Subtract one from the segment count. Is it now zero?
Yes
No
15
18. Is either the "positioning" or the "wait" bit on?
GCI050
No
Yes
26
PLH
19. Reset the "wait" and "end of data" bits.
PLHFLAG
20

Notes for Diagram CF3

18. A test is made to determine whether the next CI has been read and is ready for processing. If the "positioning" bit (PLHPOS) is on and the "wait" bit (PLHWAIT) is off, the next record is available.

19-22. If no read-ahead has been done for the next CI, it must be read and waited on for I/O completion. If a data read error occurred, the CI is reread. If a second data read error occurs, the CI is skipped (along with any contiguous CIs following it that also cause a data read error) until a CI is read that does not cause a data read error.

Diagram CF4. Get Control Interval by Key



20. Get next CI, without using the track hold feature.

See Diagram BS, GET NEXT.

21. Was there a data read error? (E64)

22. Is this the second data read error for this CI?

23. Reset the "skip" bit.

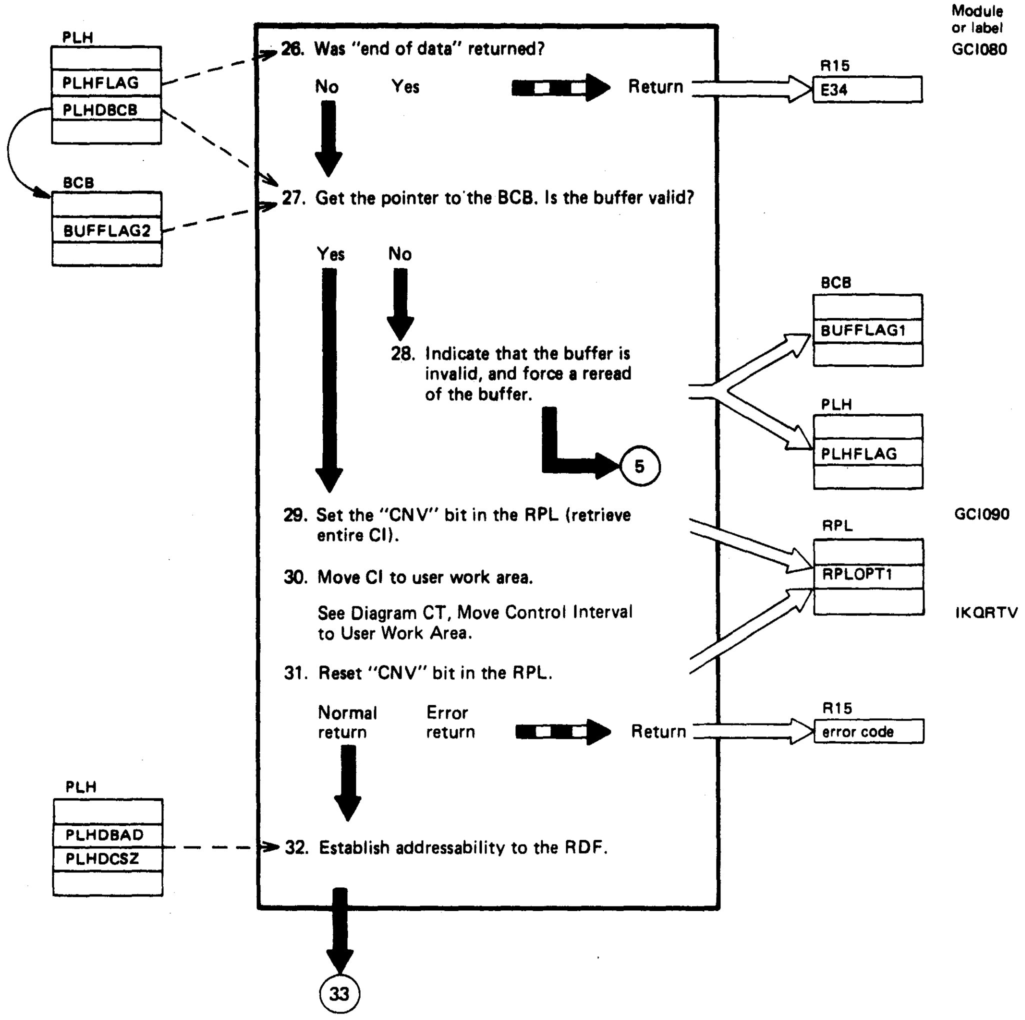
24. Was "end of data" returned?

25. Initialize the PLH to point to the first record in the CI.

Notes for Diagram CF4

23-25. The "skip" bit (PLHSKIP) is reset, and if an "end of data" (PLHEOD) condition did not occur when reading the last CI, the PLH is set to point to the first record in the CI.

Diagram CF5. Get Control Interval by Key


Module
or label
PLH
PLHFLAG
PLHDBCB
BCB
BUFFLAG2
26. Was "end of data" returned?
No
Yes
Return
R15
E34
GC1080
27. Get the pointer to the BCB. Is the buffer valid?
Yes
No
BCB
BUFFLAG1
28. Indicate that the buffer is invalid, and force a reread of the buffer.
5
PLH
PLHFLAG
29. Set the "CNV" bit in the RPL (retrieve entire CI).
RPL
RPLOPT1
GC1090
30. Move CI to user work area.
See Diagram CT, Move Control Interval to User Work Area.
IKQRTV
31. Reset "CNV" bit in the RPL.
Normal return
Error return
Return
R15
error code
PLH
PLHDBAD
PLHDCSZ
32. Establish addressability to the RDF.
33

Notes for Diagram CF5

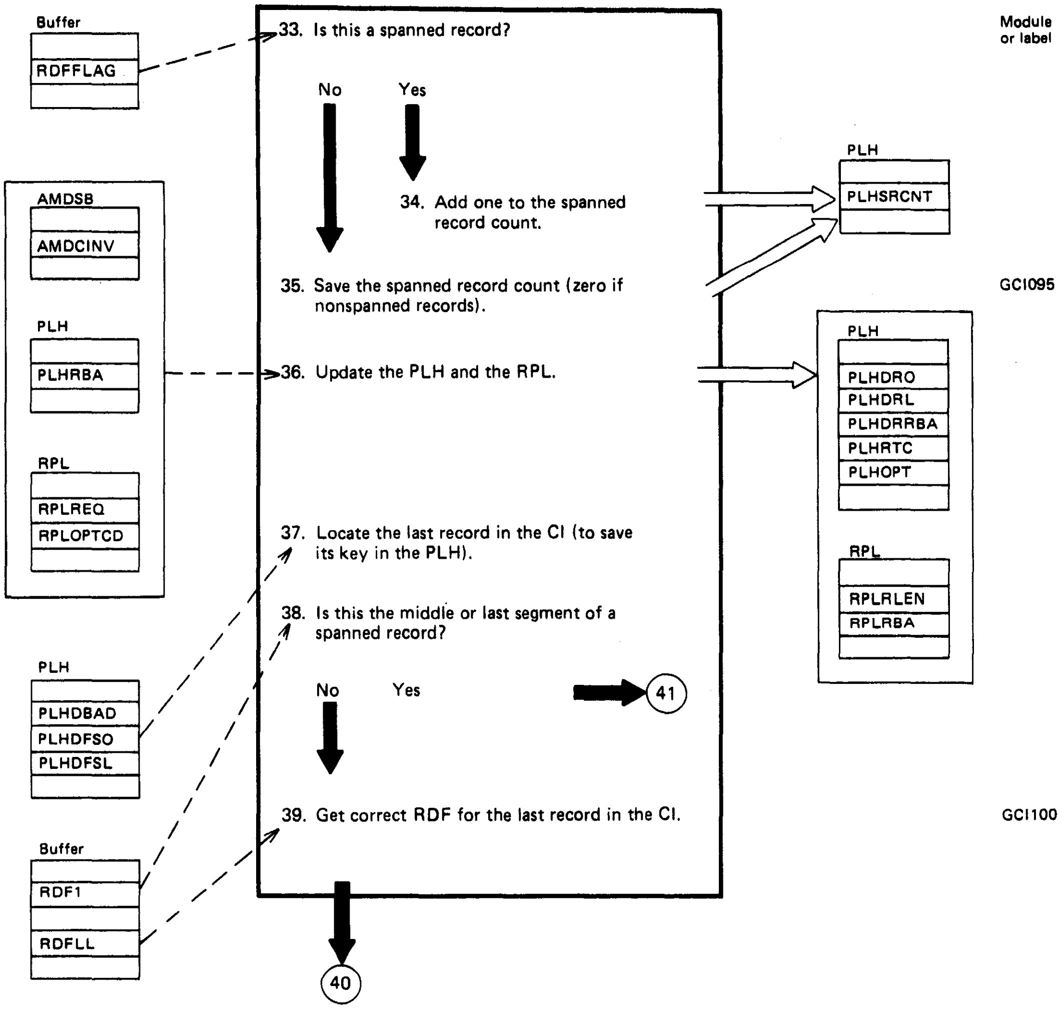
26. "End of data" must be tested again because this step could have been branched to from step 18. Either read-ahead or the call to Locate Direct (IKQLCD) might have reached the end of data condition.

27-28. If the buffer contents are not valid, the "valid" bit (BUFCVAL) is turned off, and the "reread" bit (PLHRREAD) is turned on to force a reread of the buffer. A branch to step 5 is taken to reposition to the last CI read.

29-31. If the buffer contents are valid, the CI is moved to the user work area, or the pointer to the buffer is stored in the user work area. The "CNV" bit in the RPL must be turned on to indicate CI processing (in IKQRTV) and reset after return.

32-35. Addressability is established to the first RDF pair in the CI to determine whether a spanned record is being processed.
If it is, the spanned record count (PLHSRCNT) is incremented by one to indicate the number of segments read. For nonspanned records, the count is always zero.

Diagram CF6. Get Control Interval by Key


Buffer
RDFFLAG
AMDSB
AMDCINV
PLH
PLHRBA
RPL
RPLREQ
RPLOPTCD
PLH
PLHDBAD
PLHDFSO
PLHDFSL
Buffer
RDF1
RDFLL
33. Is this a spanned record?
No
Yes
34. Add one to the spanned record count.
35. Save the spanned record count (zero if nonspanned records).
36. Update the PLH and the RPL.
37. Locate the last record in the CI (to save its key in the PLH).
38. Is this the middle or last segment of a spanned record?
No
Yes
41
39. Get correct RDF for the last record in the CI.
40
Module or label
PLH
PLHSRCNT
GC1095
PLH
PLHDRO
PLHDRL
PLHDRRBA
PLHRTC
PLHOPT
RPL
RPLRLEN
RPLRBA
GC1100

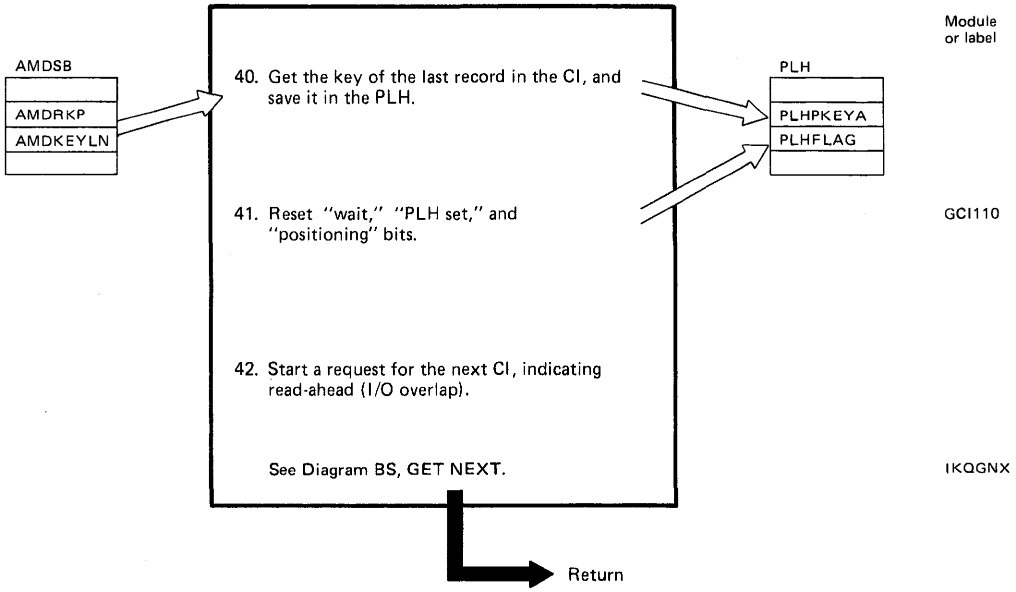
Notes for Diagram CF6

36. The PLH and RPL are updated to indicate the CI length and its RBA, and the RPL option and request type codes are saved in the PLH.

37-40. The last record (or only record, if spanned) in the CI is located, and the key is moved to the PLH in case repositioning or restart is required.

Note: For spanned records, only the first segment can contain the key; therefore, there is no key to save in the PLH when the CIs containing the middle or last segments are processed.

Diagram CF7. Get Control Interval by Key



40. Get the key of the last record in the CI, and save it in the PLH.

41. Reset "wait," "PLH set," and "positioning" bits.

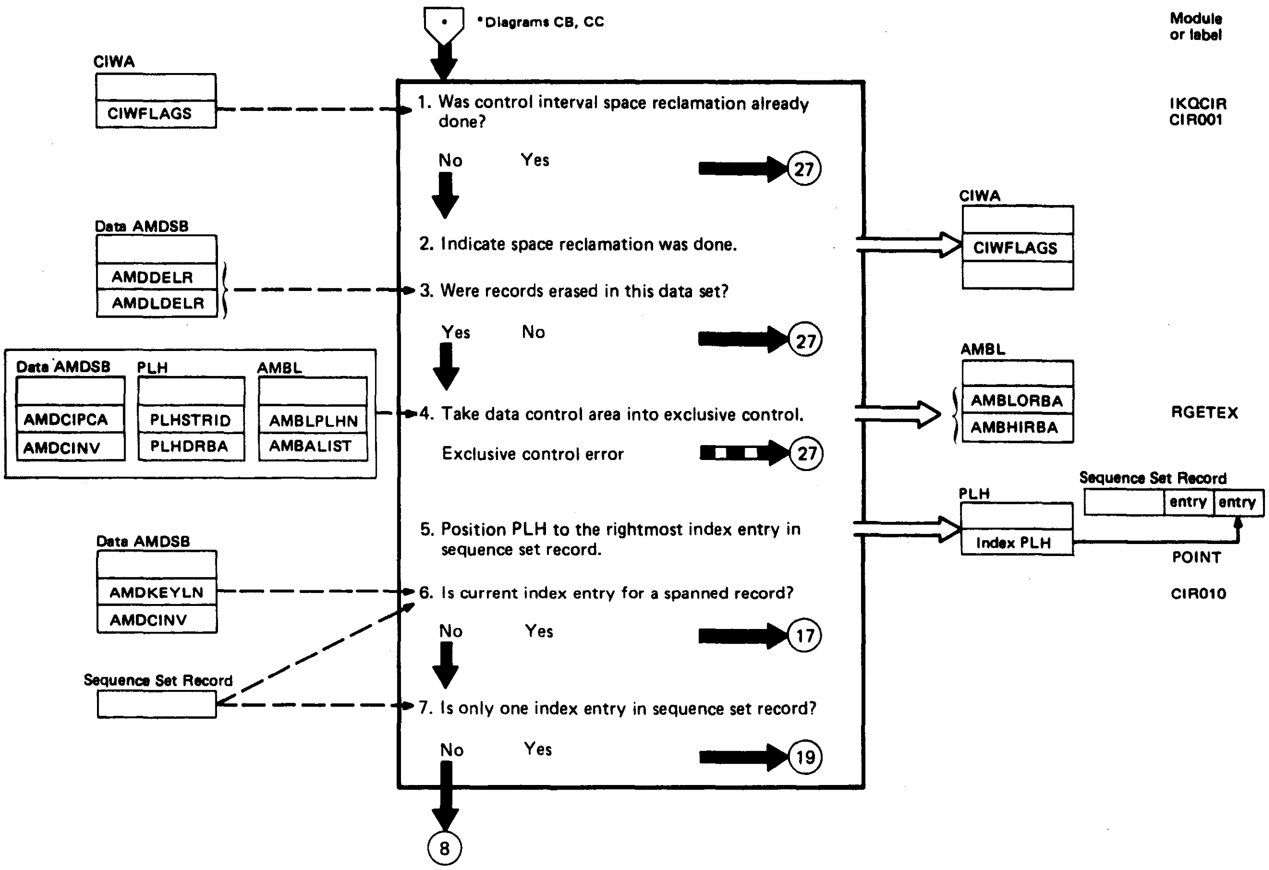
42. Start a request for the next CI, indicating read-ahead (I/O overlap).

See Diagram BS, GET NEXT.

Notes for Diagram CF7

41-42. The "wait", "PLH set", and "positioning" bits (PLHWAIT, PLHST, and PLHPOS) are turned on for the next request, and read-ahead is started for the next CI.

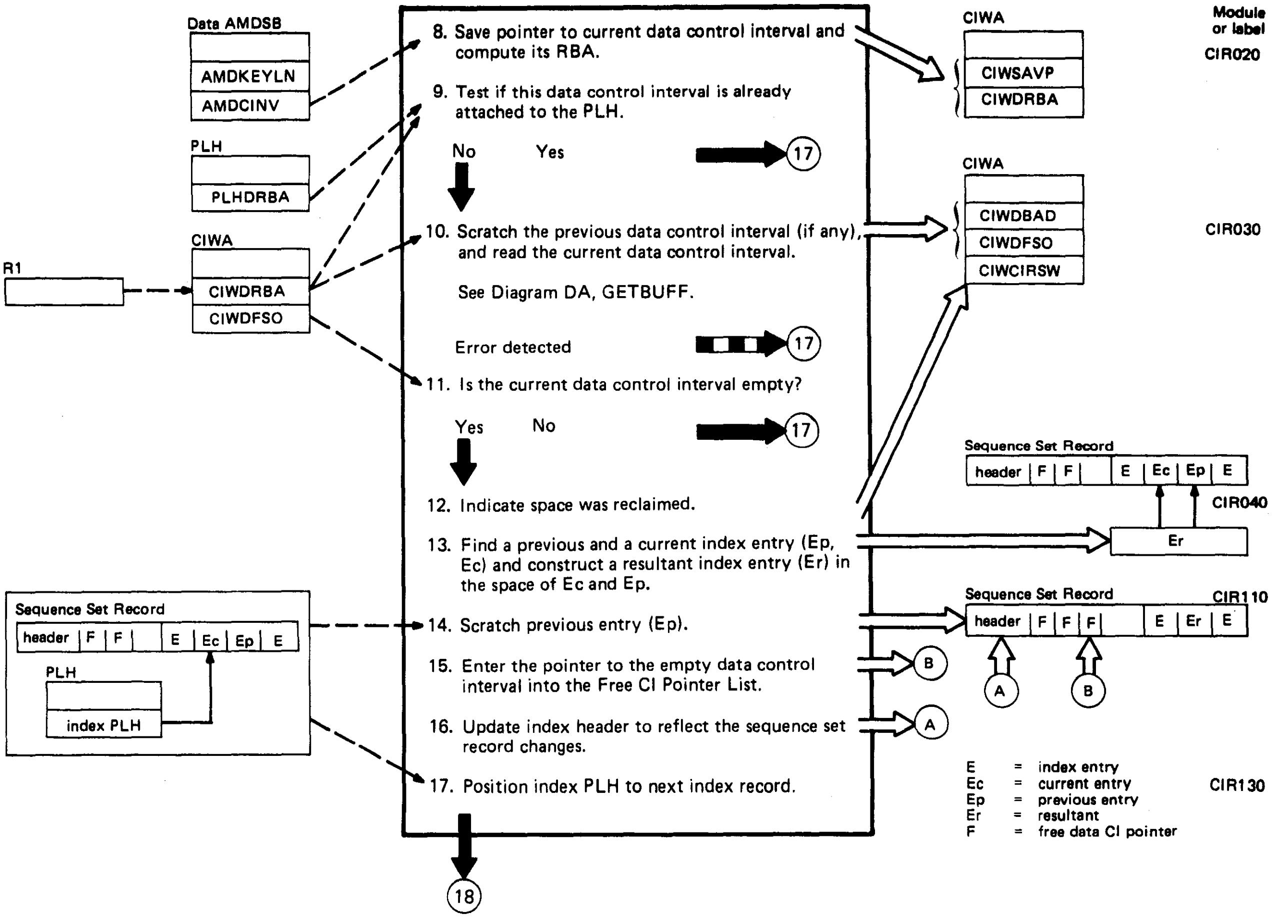
Diagram CG1. Control Interval Space Reclamation


•Diagrams CB, CC
Module or label
CIWA
CIWFLAGS
1. Was control interval space reclamation already done?
No
Yes
27
IKQCIR
CIR001
Data AMDSB
AMDDELR
AMDLDELR
2. Indicate space reclamation was done.
CIWA
CIWFLAGS
3. Were records erased in this data set?
Yes
No
27
Data AMDSB
PLH
AMBL
AMDCIPCA
AMDCINV
PLHSTRID
PLHDRBA
AMBLPLHN
AMBALIST
4. Take data control area into exclusive control.
Exclusive control error
27
AMBL
AMBLORBA
AMBHIRBA
RGETEX
5. Position PLH to the rightmost index entry in sequence set record.
PLH
Index PLH
Sequence Set Record
entry
entry
POINT
Data AMDSB
AMDKEYLN
AMDCINV
6. Is current index entry for a spanned record?
No
Yes
17
CIR010
Sequence Set Record
7. Is only one index entry in sequence set record?
No
Yes
19
8

Notes for Diagram CG1

If IKQCIS00 determines that there are no CIs available for a split (IKQCAS00 is required), IKQCIR is first called. IKQCIR ensures that records have been deleted from the data set and then reads *every* CI within the appropriate CA, looking for all CIDFs that indicate that all records in the CI have been deleted. If such a CI is *not* found, then the CA must be split. If one or more deleted CIs are found within the CA, the index sequence set is adjusted to indicate that CIs are available. Control returns to IKQCIS00 to process the now-freed data CI.

Diagram CG2. Control Interval Space Reclamation



8. Save pointer to current data control interval and compute its RBA.

9. Test if this data control interval is already attached to the PLH.

   No Yes → 17

10. Scratch the previous data control interval (if any), and read the current data control interval.

    See Diagram DA, GETBUFF.

    Error detected → 17

11. Is the current data control interval empty?

    Yes No → 17

12. Indicate space was reclaimed.

13. Find a previous and a current index entry (Ep, Ec) and construct a resultant index entry (Er) in the space of Ec and Ep.

14. Scratch previous entry (Ep).

15. Enter the pointer to the empty data control interval into the Free CI Pointer List.

16. Update index header to reflect the sequence set record changes.

17. Position index PLH to next index record.

E = index entry
Ec = current entry
Ep = previous entry
Er = resultant
F = free data CI pointer

Notes for Diagram CG2

13. The current index entry is the one pointing to the empty data CI, and the previous entry is the one to the right of this in the index record, except where the current entry is the first (rightmost) entry in the index record. In this case, the pointer from the next entry (to the left) is transferred to this entry and the next entry is used as the current entry, with the entry that points to the empty data CI now acting as the previous entry. This modification is necessary in oıuer to allow the use of uniform processing for both cases.

    The resulting index entry (Er) is formed from the current (Ec) and previous (Ep) entries in the following manner:

    If the front compression count (Fp) of the previous entry is greater than, or equal to, the front compression count (Fc) of the current entry, the resultant entry consists of the key (Kc), front compression count (Fc), and key length (Lc) of the current entry, and the pointer (Pp) of the previous entry. (See examples 1 and 2.)

    If the front compression count (Fp) of the previous entry is less than the front compression count (Fc) of the current entry, the resultant entry consists of:

    Kr: The first (Fc-Fp) characters of the previous key (Kp), followed by the current key (Kc).

    Fr: The front compression count (Fp) of the previous entry.

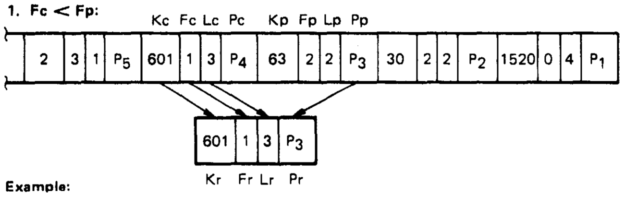
    Lr: The key length (Lc) of the current entry plus the difference between the front compression counts (Fc-Fp).

    Pr: The pointer (Pp) from the previous entry.

    This is also shown in Example 3.

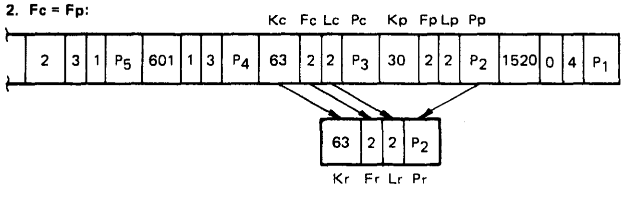
Example:

1. Fc < Fp:



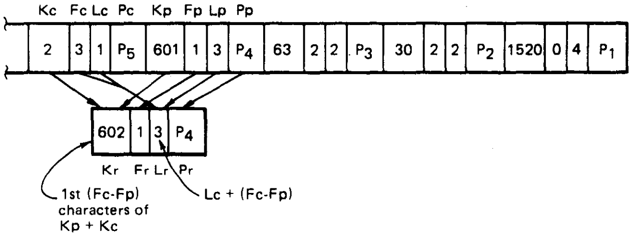
Example:

2. Fc = Fp:



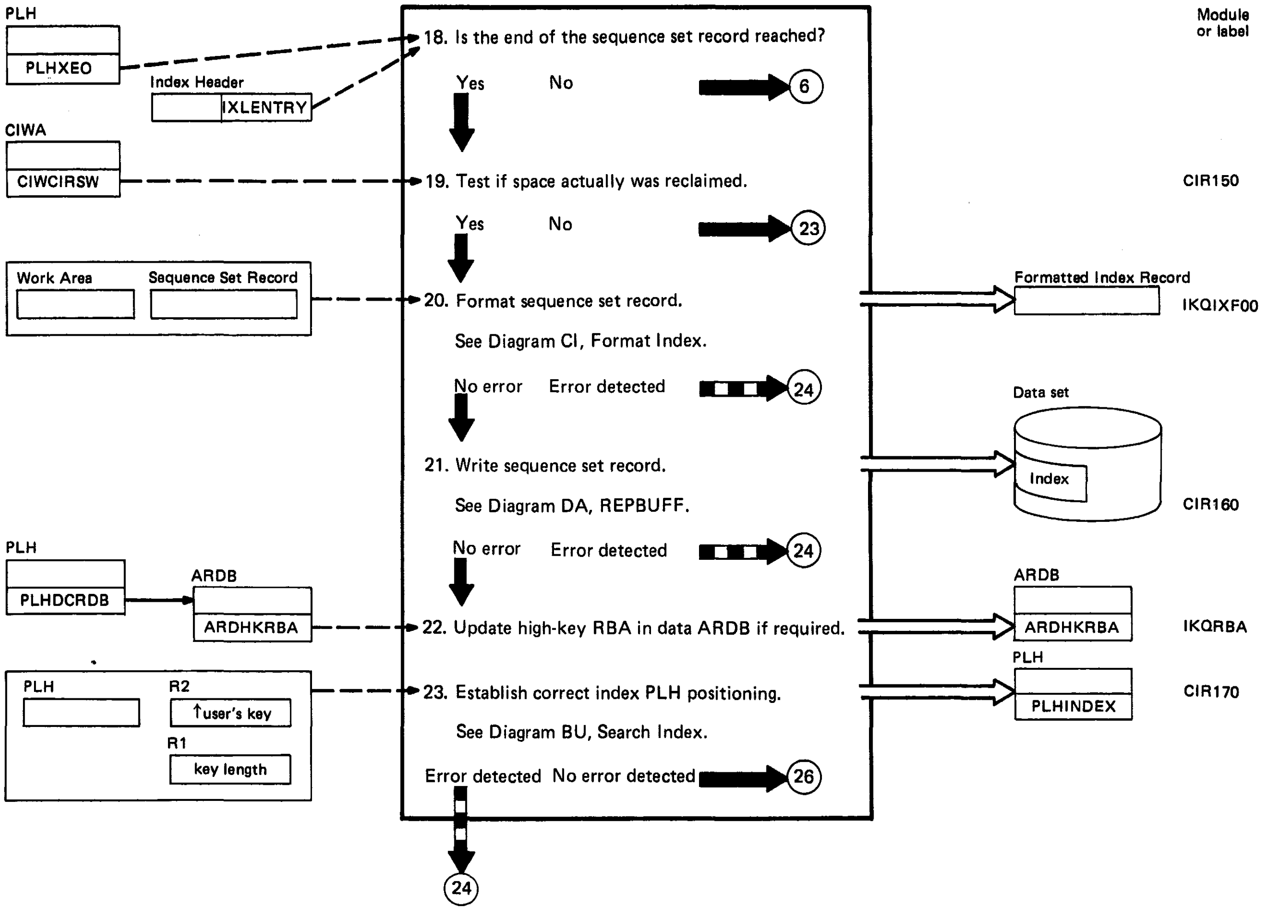
Example:

3. Fc > Fp:



Kc Fc Lc Pc Kp Fp Lp Pp
2 3 1 P5 601 1 3 P4 63 2 2 P3 30 2 2 P2 1520 0 4 P1
602 1 3 P4
Kr Fr Lr Pr
1st (Fc-Fp) characters of Kp + Kc
Lc + (Fc-Fp)

Diagram CG3. Control Interval Space Reclamation



18. Is the end of the sequence set record reached?

19. Test if space actually was reclaimed.

20. Format sequence set record.

See Diagram CI, Format Index.

21. Write sequence set record.

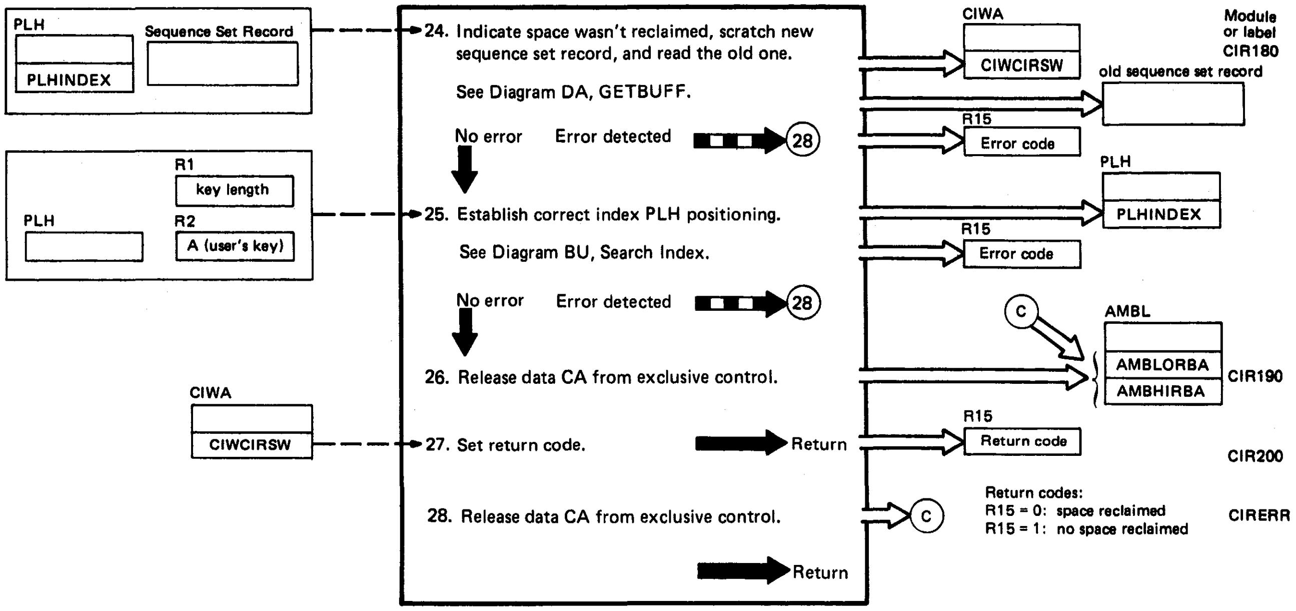
See Diagram DA, REPBUFF.

22. Update high-key RBA in data ARDB if required.

23. Establish correct index PLH positioning.

See Diagram BU, Search Index.

Diagram CG4. Control Interval Space Reclamation



24. Indicate space wasn't reclaimed, scratch new sequence set record, and read the old one.

See Diagram DA, GETBUFF.

25. Establish correct index PLH positioning.

See Diagram BU, Search Index.

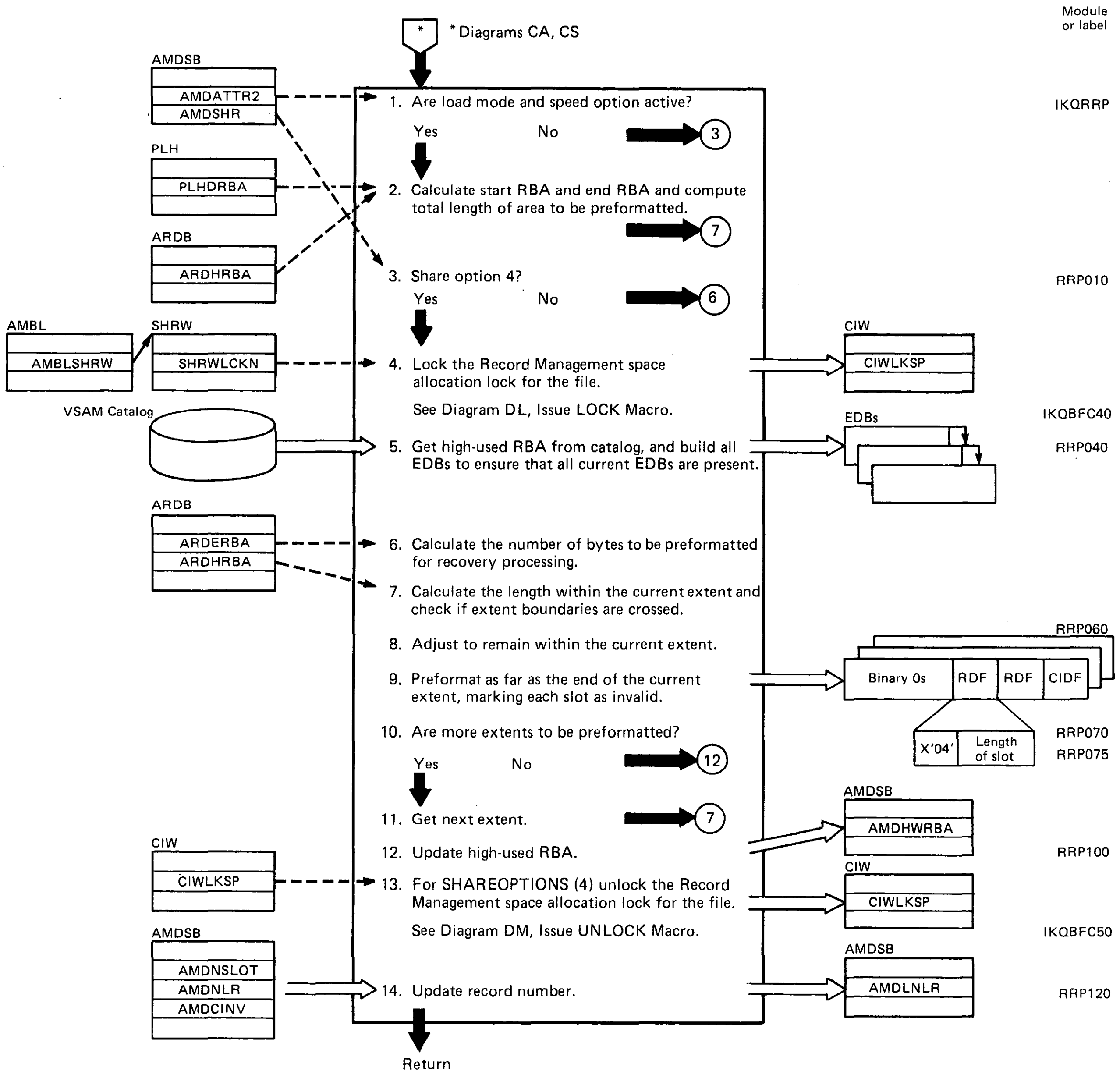
26. Release data CA from exclusive control.

27. Set return code.

28. Release data CA from exclusive control.

Return codes:
R15 = 0: space reclaimed
R15 = 1: no space reclaimed

Diagram CH1. Preformat Relative Record Data Set



1. Are load mode and speed option active?
2. Calculate start RBA and end RBA and compute total length of area to be preformatted.
3. Share option 4?
4. Lock the Record Management space allocation lock for the file.

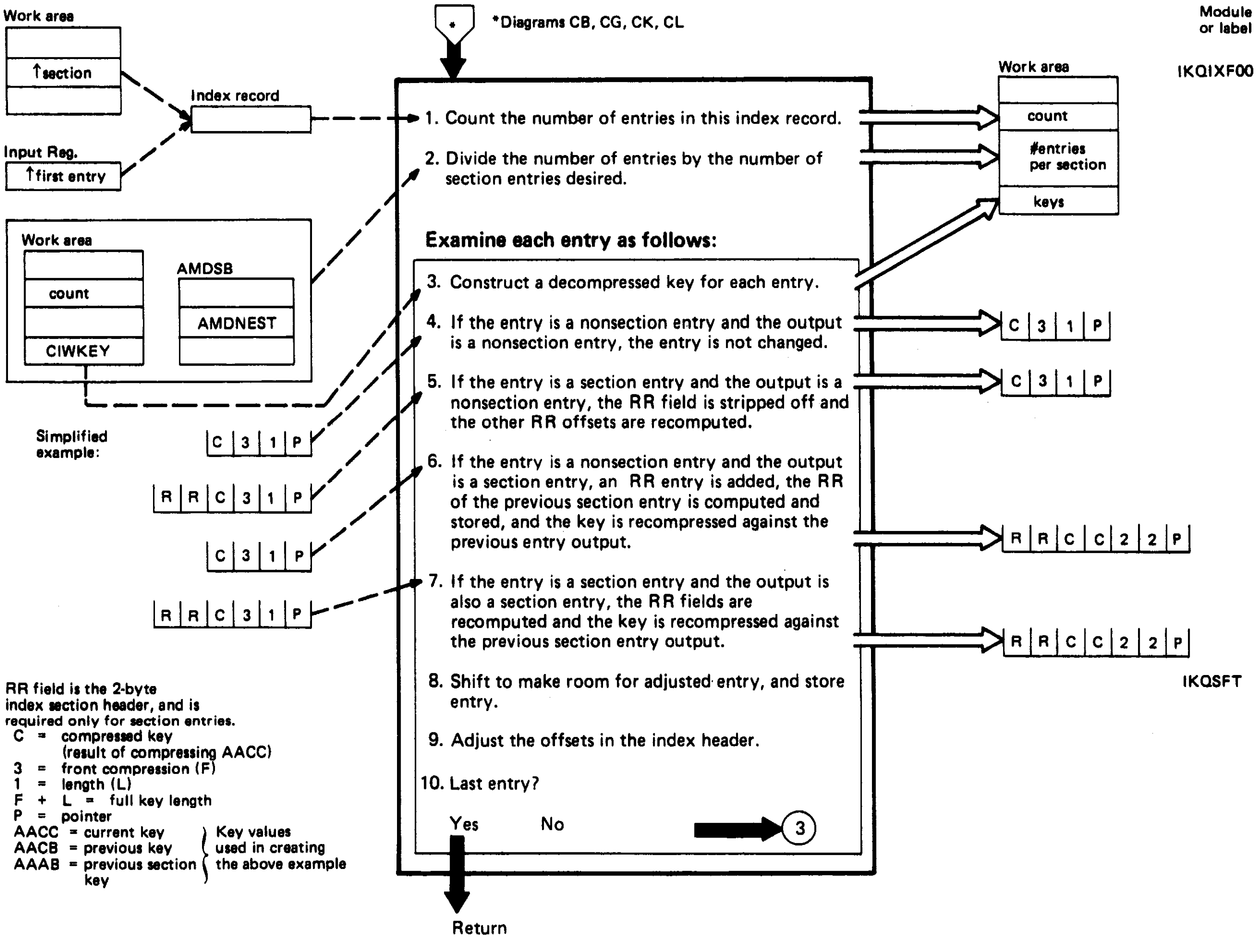
   See Diagram DL, Issue LOCK Macro.
5. Get high-used RBA from catalog, and build all EDBs to ensure that all current EDBs are present.
6. Calculate the number of bytes to be preformatted for recovery processing.
7. Calculate the length within the current extent and check if extent boundaries are crossed.
8. Adjust to remain within the current extent.
9. Preformat as far as the end of the current extent, marking each slot as invalid.
10. Are more extents to be preformatted?
11. Get next extent.
12. Update high-used RBA.
13. For SHAREOPTIONS (4) unlock the Record Management space allocation lock for the file.

    See Diagram DM, Issue UNLOCK Macro.
14. Update record number.

Notes for Diagram CH1

4. A LOCK macro is issued to perform the locking. The LOCK name is the base lock name for the file (see note for step 1 of Diagram DG), except that the last two bytes are X'0005'.

Diagram CI1. Format Index



1. Count the number of entries in this index record.
2. Divide the number of entries by the number of section entries desired.

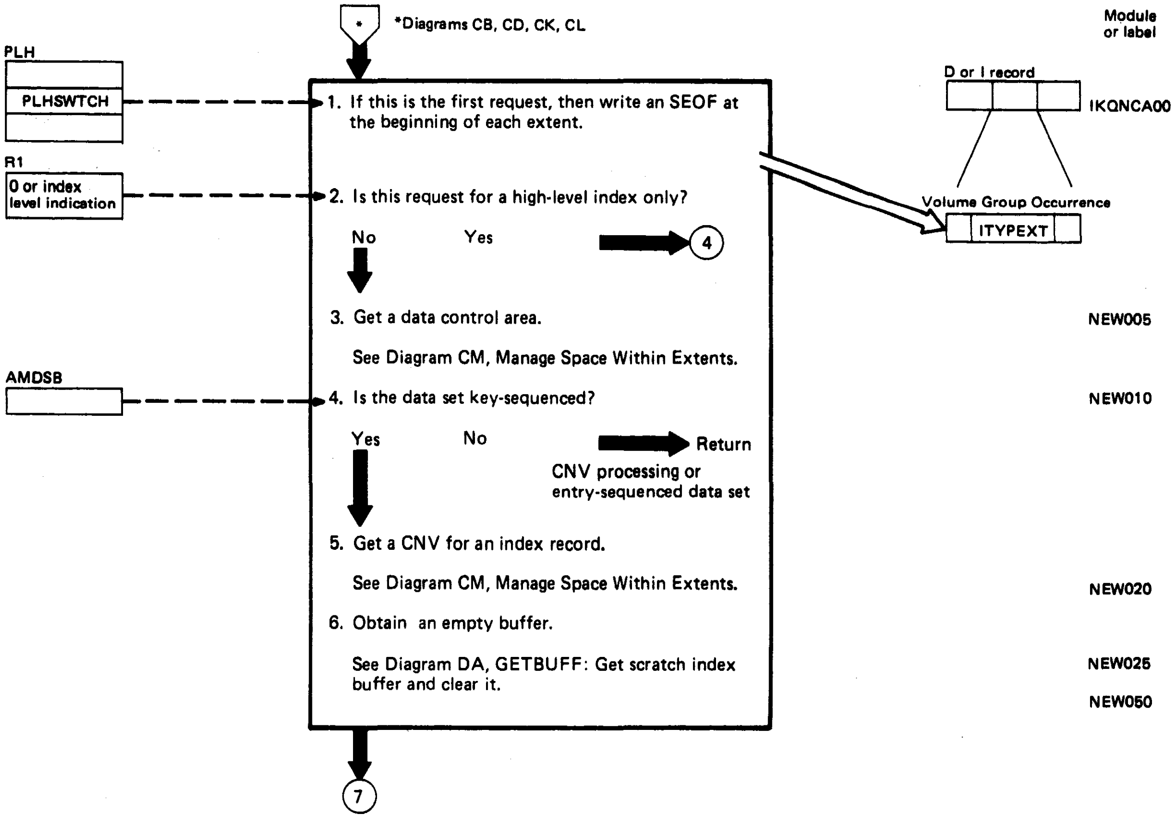
Examine each entry as follows:

3. Construct a decompressed key for each entry.
4. If the entry is a nonsection entry and the output is a nonsection entry, the entry is not changed.
5. If the entry is a section entry and the output is a nonsection entry, the RR field is stripped off and the other RR offsets are recomputed.
6. If the entry is a nonsection entry and the output is a section entry, an RR entry is added, the RR of the previous section entry is computed and stored, and the key is recompressed against the previous entry output.
7. If the entry is a section entry and the output is also a section entry, the RR fields are recomputed and the key is recompressed against the previous section entry output.
8. Shift to make room for adjusted entry, and store entry.
9. Adjust the offsets in the index header.
10. Last entry?

RR field is the 2-byte index section header, and is required only for section entries.

C = compressed key (result of compressing AACC)
3 = front compression (F)
1 = length (L)
F + L = full key length
P = pointer
AACC = current key
AACB = previous key
AAAB = previous section key
} Key values used in creating the above example

Diagram CJ1. Obtain New Control Area



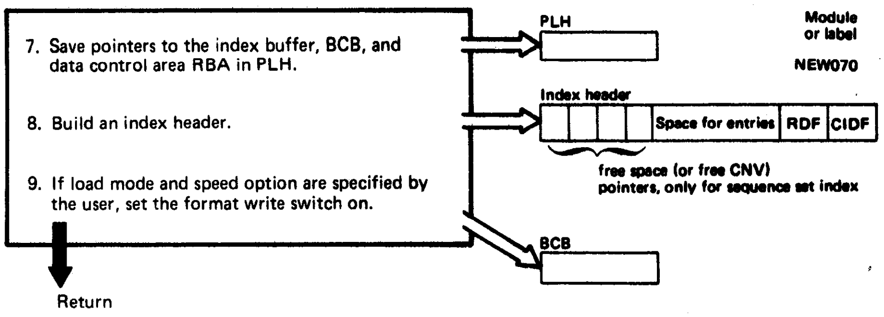
1. If this is the first request, then write an SEOF at the beginning of each extent.
2. Is this request for a high-level index only?
3. Get a data control area.

   See Diagram CM, Manage Space Within Extents.
4. Is the data set key-sequenced?
5. Get a CNV for an index record.

   See Diagram CM, Manage Space Within Extents.
6. Obtain an empty buffer.

   See Diagram DA, GETBUFF: Get scratch index buffer and clear it.

Diagram CJ2. Obtain New Control Area

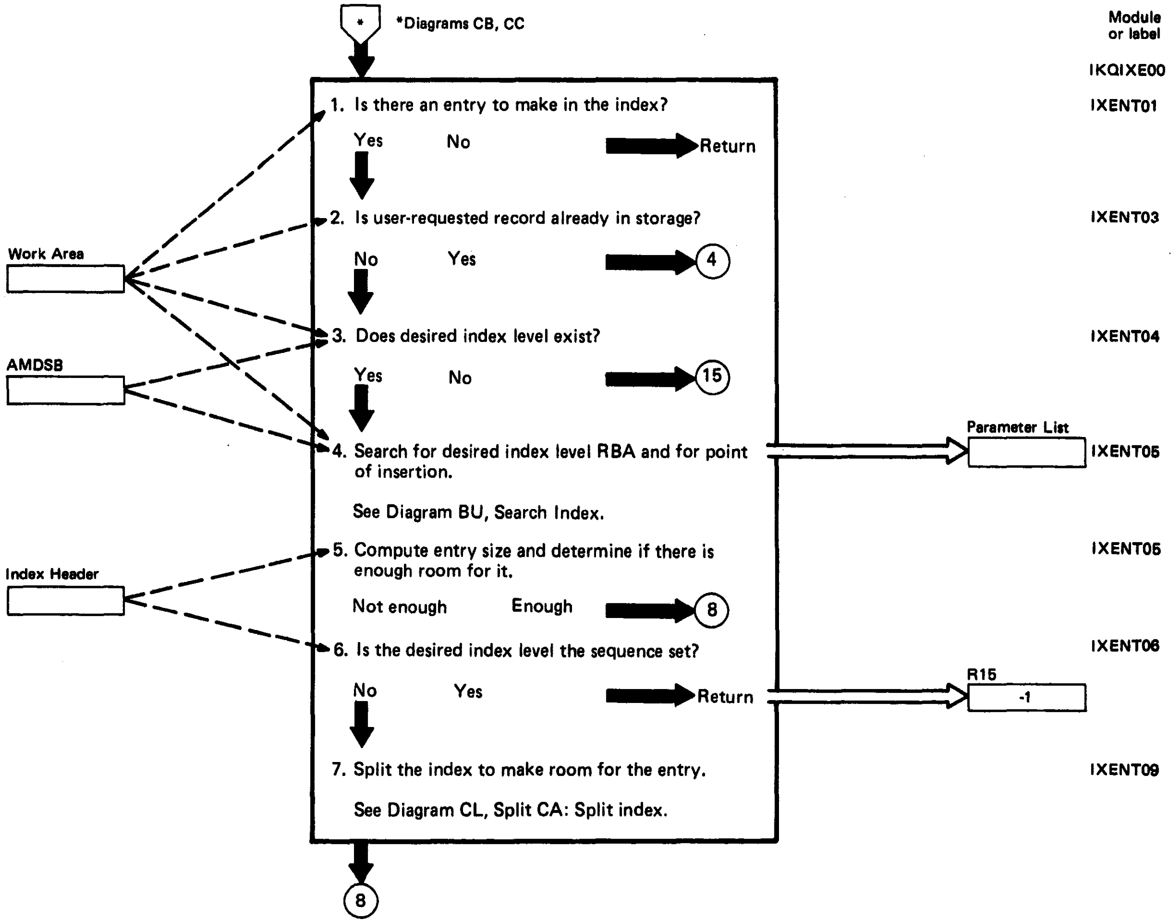


7. Save pointers to the index buffer, BCB, and data control area RBA in PLH.

8. Build an index header.

9. If load mode and speed option are specified by the user, set the format write switch on.

Diagram CK1. Create Index Entry

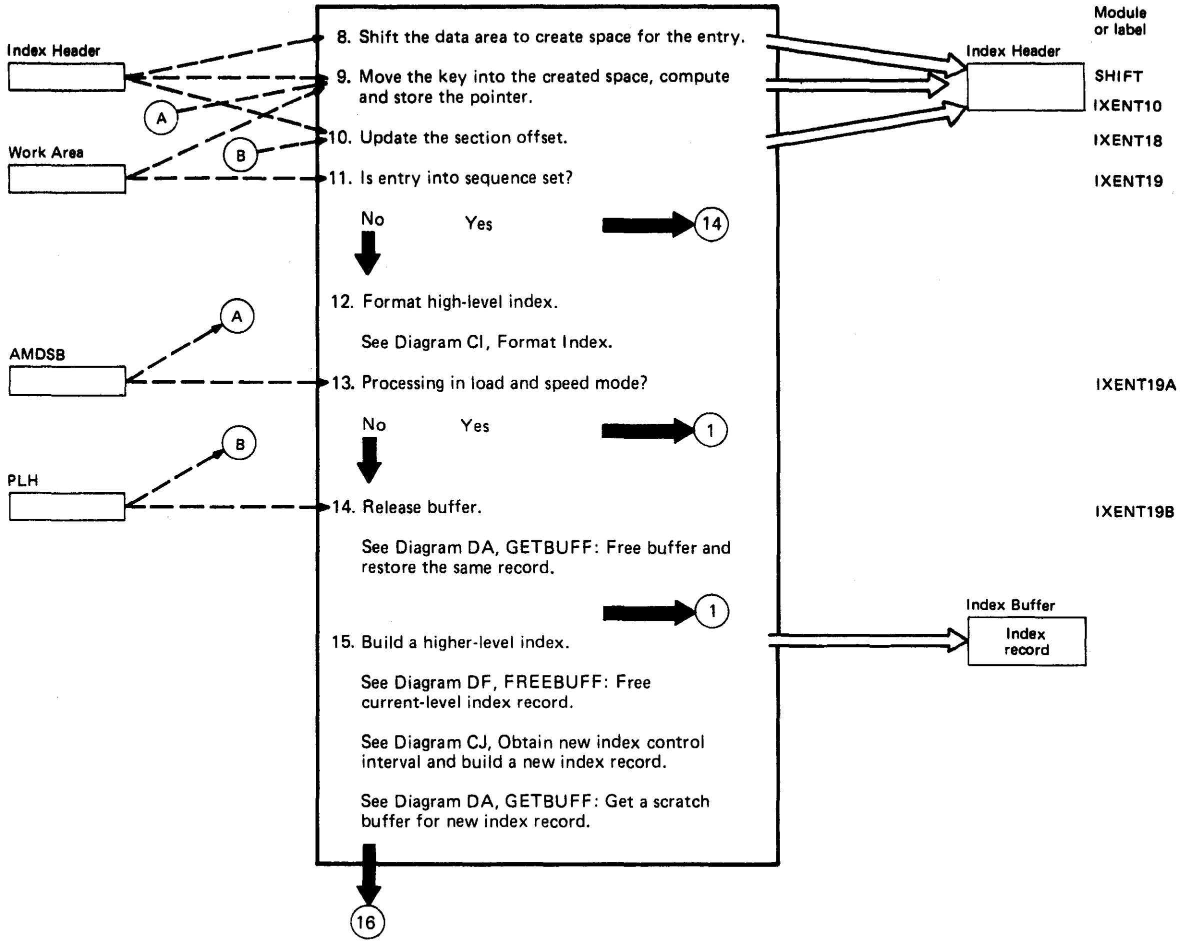


*Diagrams CB, CC
Module or label
IKQIXE00
1. Is there an entry to make in the index?
Yes
No
Return
IXENT01
2. Is user-requested record already in storage?
No
Yes
4
IXENT03
Work Area
3. Does desired index level exist?
Yes
No
15
IXENT04
AMDSB
4. Search for desired index level RBA and for point of insertion.
See Diagram BU, Search Index.
Parameter List
IXENT05
5. Compute entry size and determine if there is enough room for it.
Not enough
Enough
8
IXENT05
Index Header
6. Is the desired index level the sequence set?
No
Yes
Return
R15
-1
IXENT06
7. Split the index to make room for the entry.
See Diagram CL, Split CA: Split index.
IXENT09
8

Notes for Diagram CK1

4. The front compression count that will be used for the new key is determined during this index search. (See Diagram BU, Search Index, for a description of front compression.)

Diagram CK2. Create Index Entry



8. Shift the data area to create space for the entry.

9. Move the key into the created space, compute and store the pointer.

10. Update the section offset.

11. Is entry into sequence set?

No Yes

12. Format high-level index.

See Diagram CI, Format Index.

13. Processing in load and speed mode?

No Yes

14. Release buffer.

See Diagram DA, GETBUFF: Free buffer and restore the same record.

15. Build a higher-level index.

See Diagram DF, FREEBUFF: Free current-level index record.

See Diagram CJ, Obtain new index control interval and build a new index record.

See Diagram DA, GETBUFF: Get a scratch buffer for new index record.

Notes for Diagram CK2

8.-14. There are two types of index entries, normal and split (see examples below). The only difference between a normal entry and a split entry is in the point of insertion. An index entry is composed of a key and pointer. A normal entry is inserted after the pointer; a split entry, however, is inserted after the key and the new entry is composed of a pointer and key. The result is two entries composed of the old key and new pointer, and the new key and old pointer.

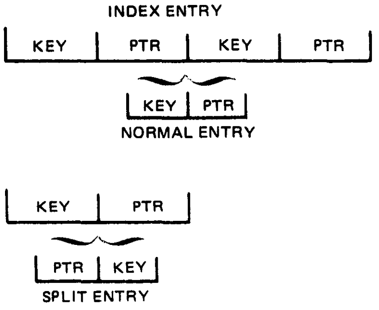
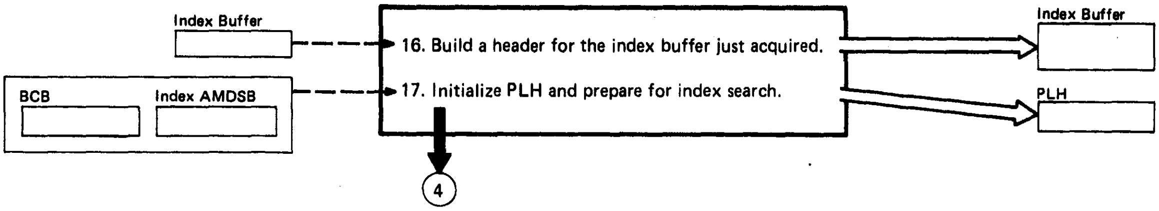


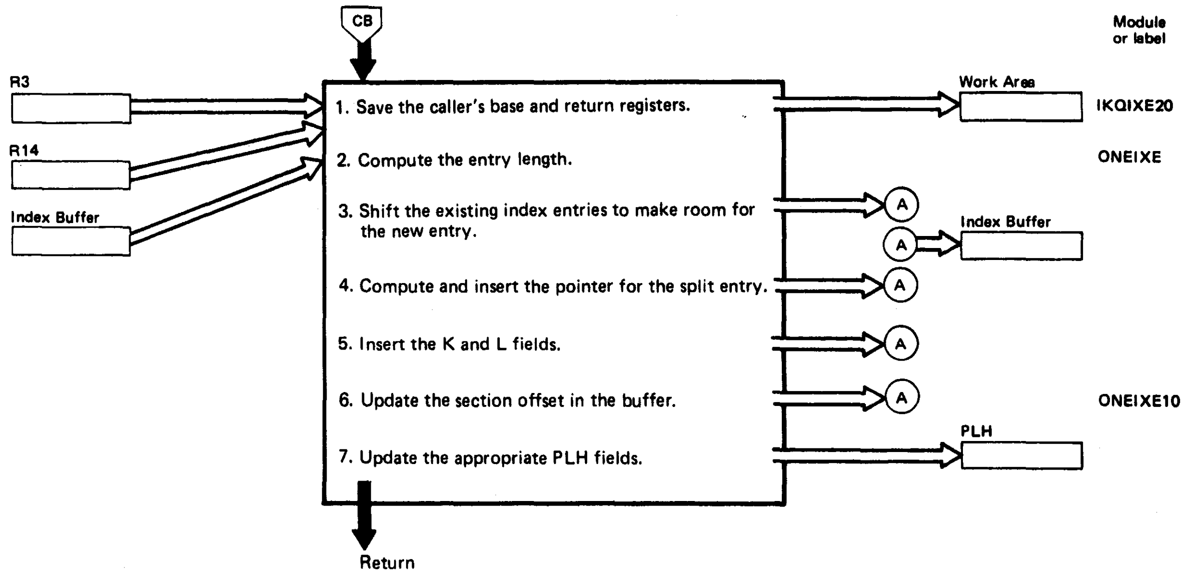
Diagram CK3. Create Index Entry



16. Build a header for the index buffer just acquired.

17. Initialize PLH and prepare for index search.

Diagram CK4. Create Index Entry



1. Save the caller's base and return registers.
2. Compute the entry length.
3. Shift the existing index entries to make room for the new entry.
4. Compute and insert the pointer for the split entry.
5. Insert the K and L fields.
6. Update the section offset in the buffer.
7. Update the appropriate PLH fields.

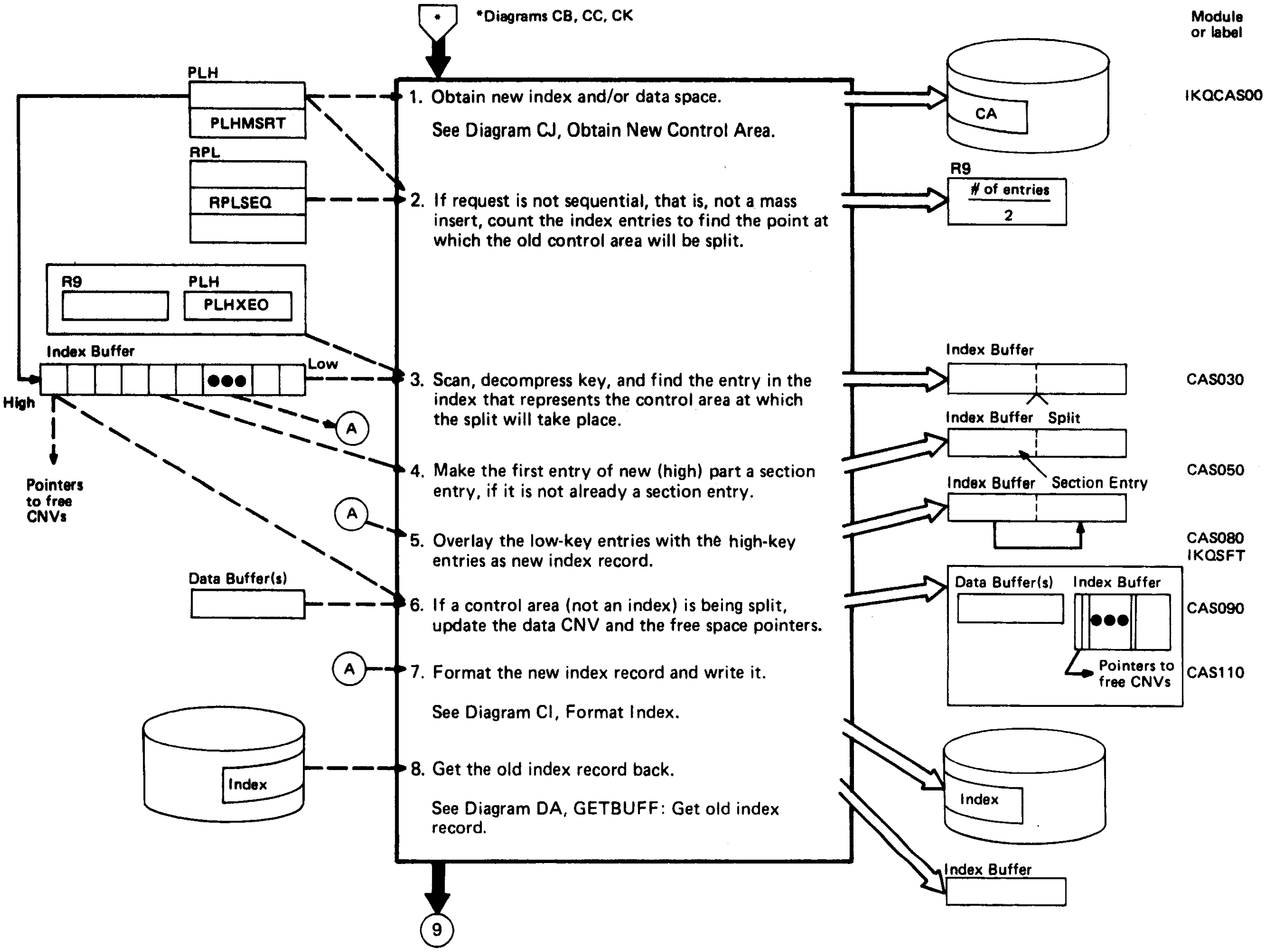
Notes for Diagram CK4

The index entry for a spanned record is a special case of the split entry mentioned in the notes for Diagrams CK1-CK3. When a spanned record is stored, the first index entry is created by the normal method (CK1-CK3). Index entries for segments after the first one are created by the routine shown in Diagram CK4.

The index entry for the first segment originally holds the key, and its F byte contains the actual key compression count. Index entries for further segments are inserted in the middle of the first entry, thus causing the key to be moved to the left and to remain in the entry for the last segment of the spanned record.

The entries for the second and subsequent segments do not contain a key, and their F byte contains a compression count equal to the keylength, thus indicating a keylength (in the entry) of zero.

Diagram CL1. Split Control Area



1. Obtain new index and/or data space.

   See Diagram CJ, Obtain New Control Area.

2. If request is not sequential, that is, not a mass insert, count the index entries to find the point at which the old control area will be split.

3. Scan, decompress key, and find the entry in the index that represents the control area at which the split will take place.

4. Make the first entry of new (high) part a section entry, if it is not already a section entry.

5. Overlay the low-key entries with the high-key entries as new index record.

6. If a control area (not an index) is being split, update the data CNV and the free space pointers.

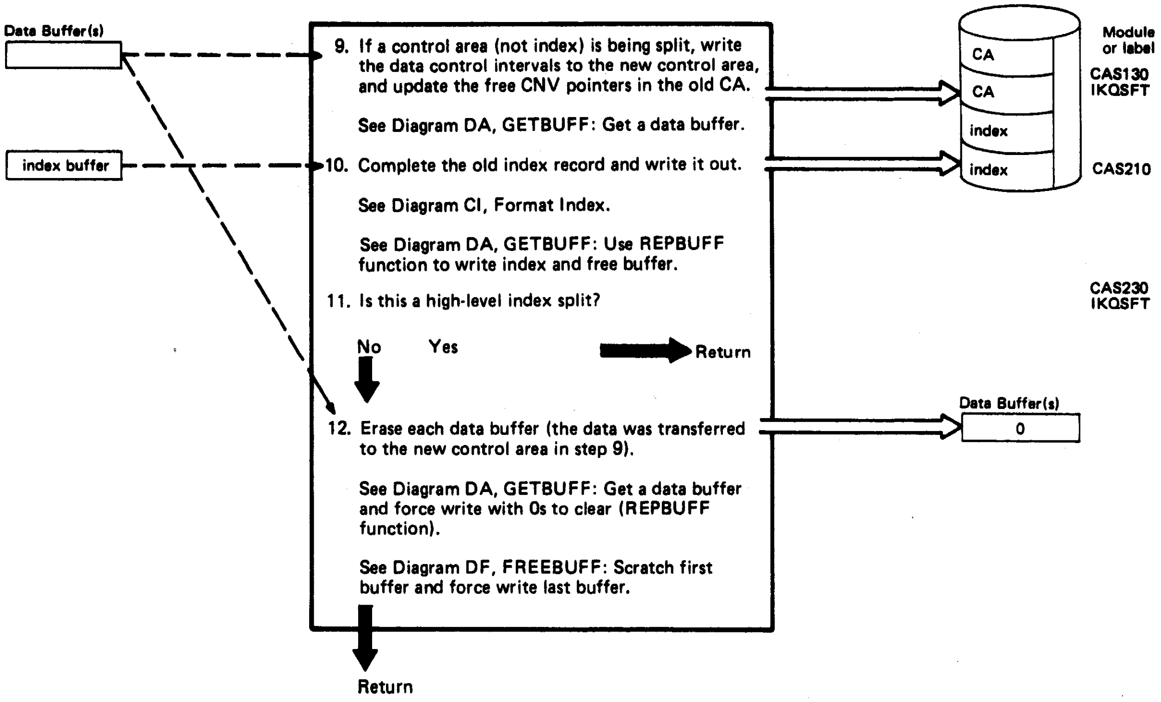
7. Format the new index record and write it.

   See Diagram CI, Format Index.

8. Get the old index record back.

   See Diagram DA, GETBUFF: Get old index record.

Diagram CL2. Split Control Area



9. If a control area (not index) is being split, write the data control intervals to the new control area, and update the free CNV pointers in the old CA.

   See Diagram DA, GETBUFF: Get a data buffer.

10. Complete the old index record and write it out.

    See Diagram CI, Format Index.

    See Diagram DA, GETBUFF: Use REPBUFF function to write index and free buffer.

11. Is this a high-level index split?

    No Yes Return

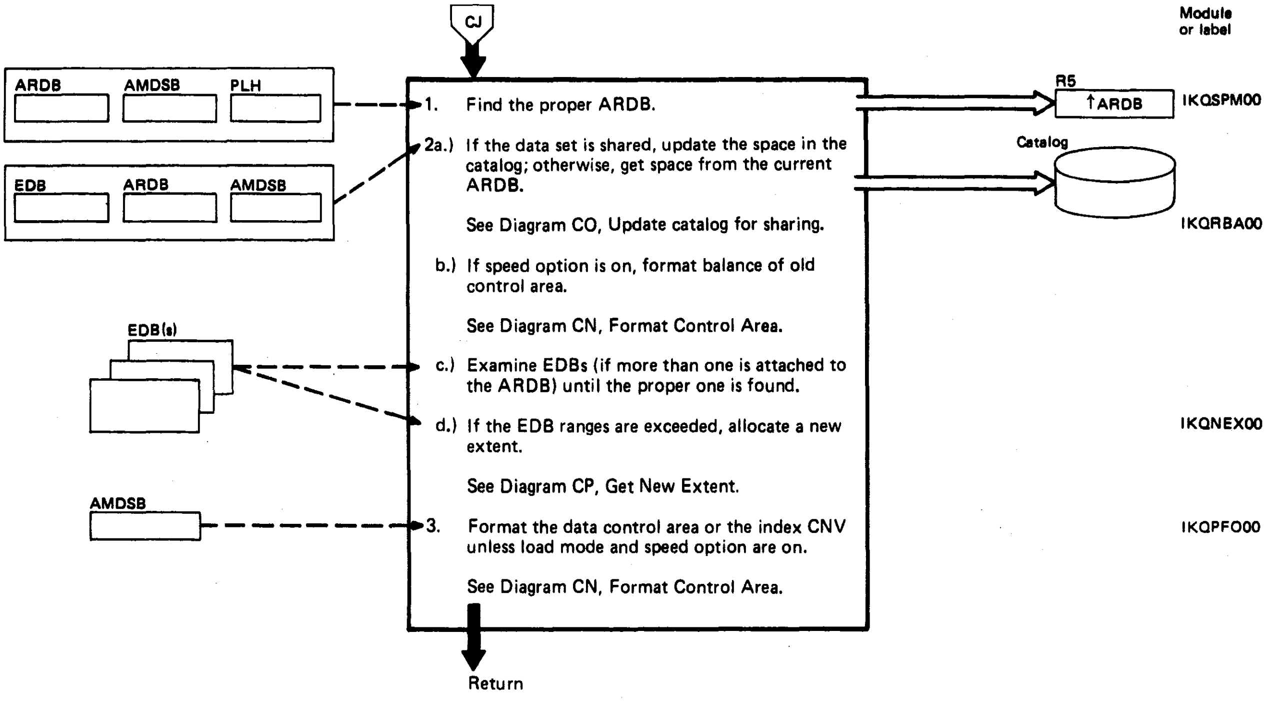
12. Erase each data buffer (the data was transferred to the new control area in step 9).

    See Diagram DA, GETBUFF: Get a data buffer and force write with 0s to clear (REPBUFF function).

    See Diagram DF, FREEBUFF: Scratch first buffer and force write last buffer.

Return

Diagram CM1. Manage Space Within Extents



1. Find the proper ARDB.

2a.) If the data set is shared, update the space in the catalog; otherwise, get space from the current ARDB.

See Diagram CO, Update catalog for sharing.

b.) If speed option is on, format balance of old control area.

See Diagram CN, Format Control Area.

c.) Examine EDBs (if more than one is attached to the ARDB) until the proper one is found.

d.) If the EDB ranges are exceeded, allocate a new extent.

See Diagram CP, Get New Extent.

3. Format the data control area or the index CNV unless load mode and speed option are on.

See Diagram CN, Format Control Area.

Notes for Diagram CM1

1. The logic of finding the proper ARDB is determined by using the following decision table where certain conditions result in certain operations taking place. For example, looking at the rightmost column, if there is an index, a key range and a change of key range, and the sequence set is embedded with data, then the ARDB is pointed to from the PLH, and the ARDPREL, a field in the data ARDB, points to the associated index ARDB.

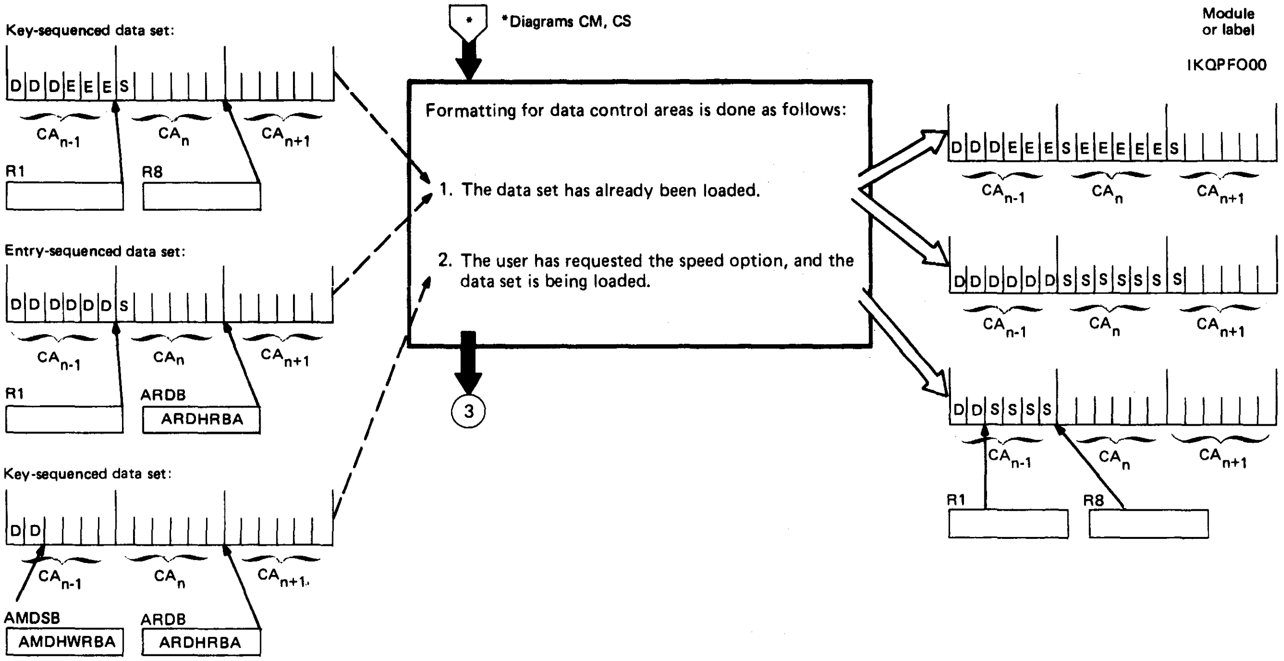
2. "Space" in this context means the serial apportionment of a single control interval (for an index) or control area (for data) for immediate use. This space is a subdivision of an extent as defined by DADSM allocation.

Data or index	D	D	D	I	I	I	I	I	I	I
Key range	N	Y	Y	N	N	N	Y	Y	Y	Y
Key range change	—	Y	N	—	—	—	—	—	N	Y
Sequence set or high level	—	—	—	—	HL	SS	—	HL	SS	SS
Sequence set with data	—	—	—	N	Y	Y	N	Y	Y	Y
The ARDB is found as follows:										
ARDB from AMDSB	X		X	X		X	X		X	
ARDB from PLH		X								X
Search ARDBs for key range			X			X			X	
Use ARDPREL*										X
Search ARDBs for high-level					X			X		

*ARDPREL is a field in the data ARDB that points to the associated index ARDB

Legend:
N = no
Y = yes
D = data
I = index
— = don't care
X = indicates how the proper ARDB is found
SS = sequence set
HL = high level

Diagram CN1. Format Data CA or Index CNV



*Diagrams CM, CS

Formatting for data control areas is done as follows:

1. The data set has already been loaded.

2. The user has requested the speed option, and the data set is being loaded.

Notes for Diagram CN1

1.-3. IKQBFA00 and IKQSFT are called to help process each of the operations described.

1. The speed option can be specified by the user, but load mode will automatically be invoked by VSAM if the data set is empty or if the end of a key range or data set has been reached.

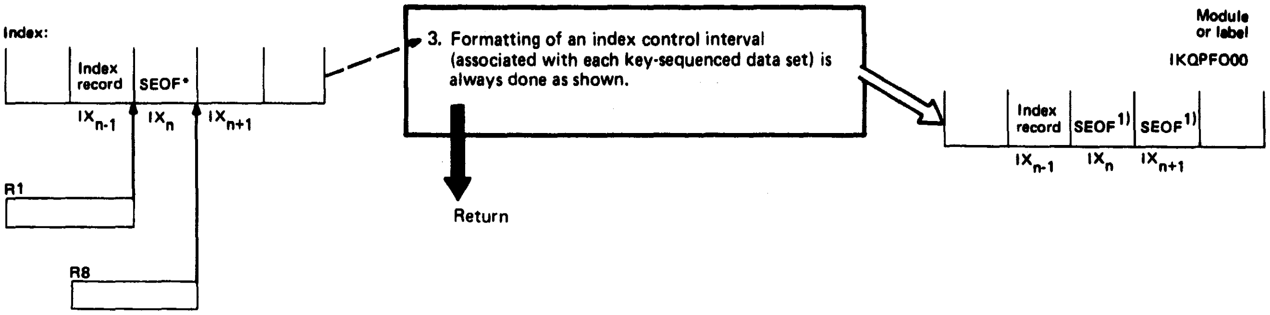
2. Once loading has been completed, that is, one or more records have been loaded into the data set and the data set has been closed, load mode may still be invoked automatically, but the speed option will be ignored.

1. & 3. IKQRBA is called to turn off the ARDB preformat bit in the catalog after any extend operation.

D	=	data control interval
E	=	empty control interval (a CNV filled with 0 plus CIDF)
S	=	software end-of-file (a CNV filled with binary 0s)
CA	=	control area
CNV	=	control interval
IX	=	index CNV (for index record)
n	=	number

Note: The three steps in this diagram are not sequential. They simply show "before" and "after" examples of the various types of preformatting.

Diagram CN2. Format Data CA or Index CNV

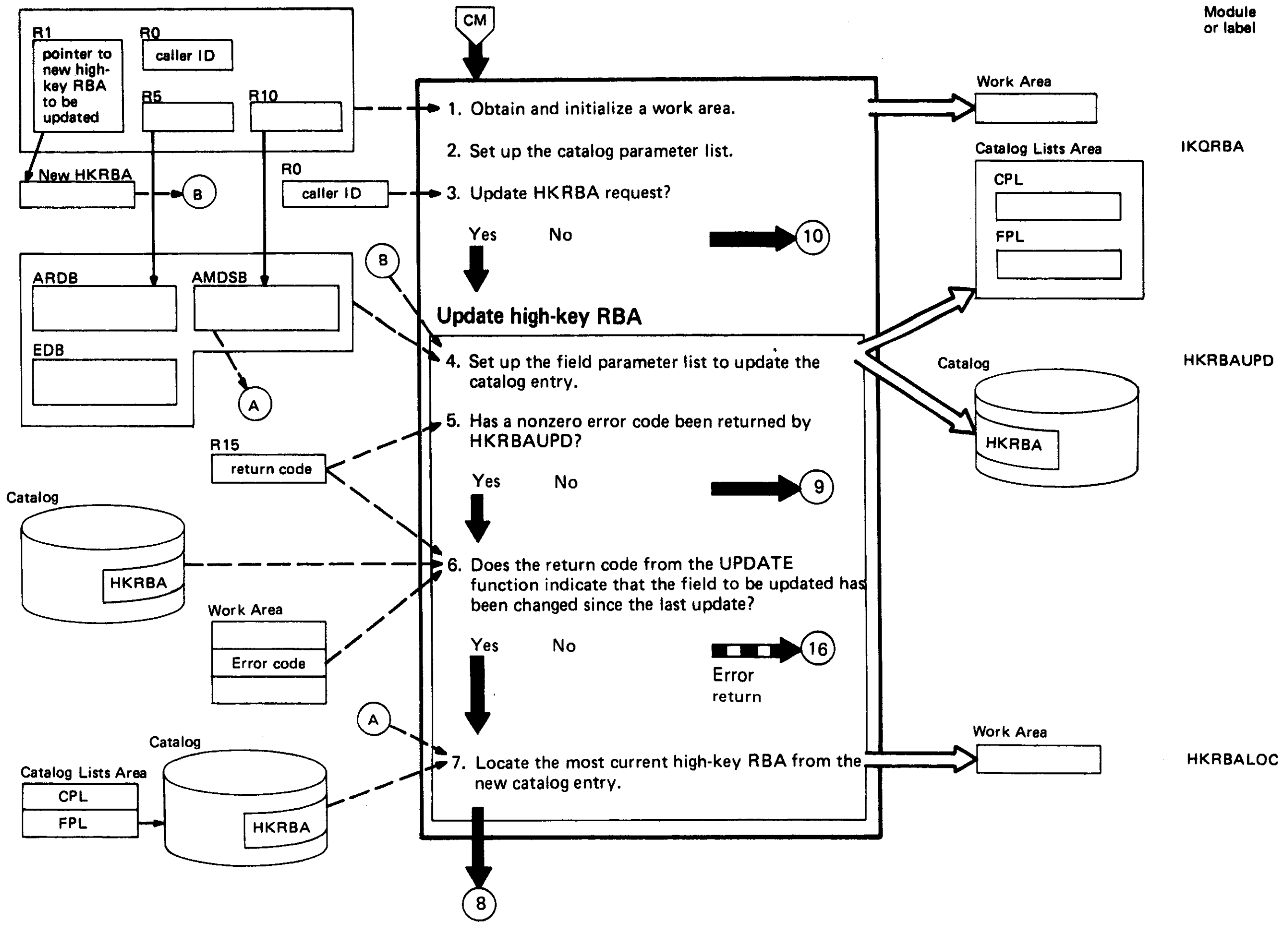


3. Formatting of an index control interval (associated with each key-sequenced data set) is always done as shown.

*Does not exist for first control interval (not written).

1) Both SEOFs (at n and n + 1) are written, even though redundant.

Diagram CO1. Update Catalog



1. Obtain and initialize a work area.
2. Set up the catalog parameter list.
3. Update HKRBA request?

Yes No

Update high-key RBA

4. Set up the field parameter list to update the catalog entry.
5. Has a nonzero error code been returned by HKRBAUPD?

Yes No

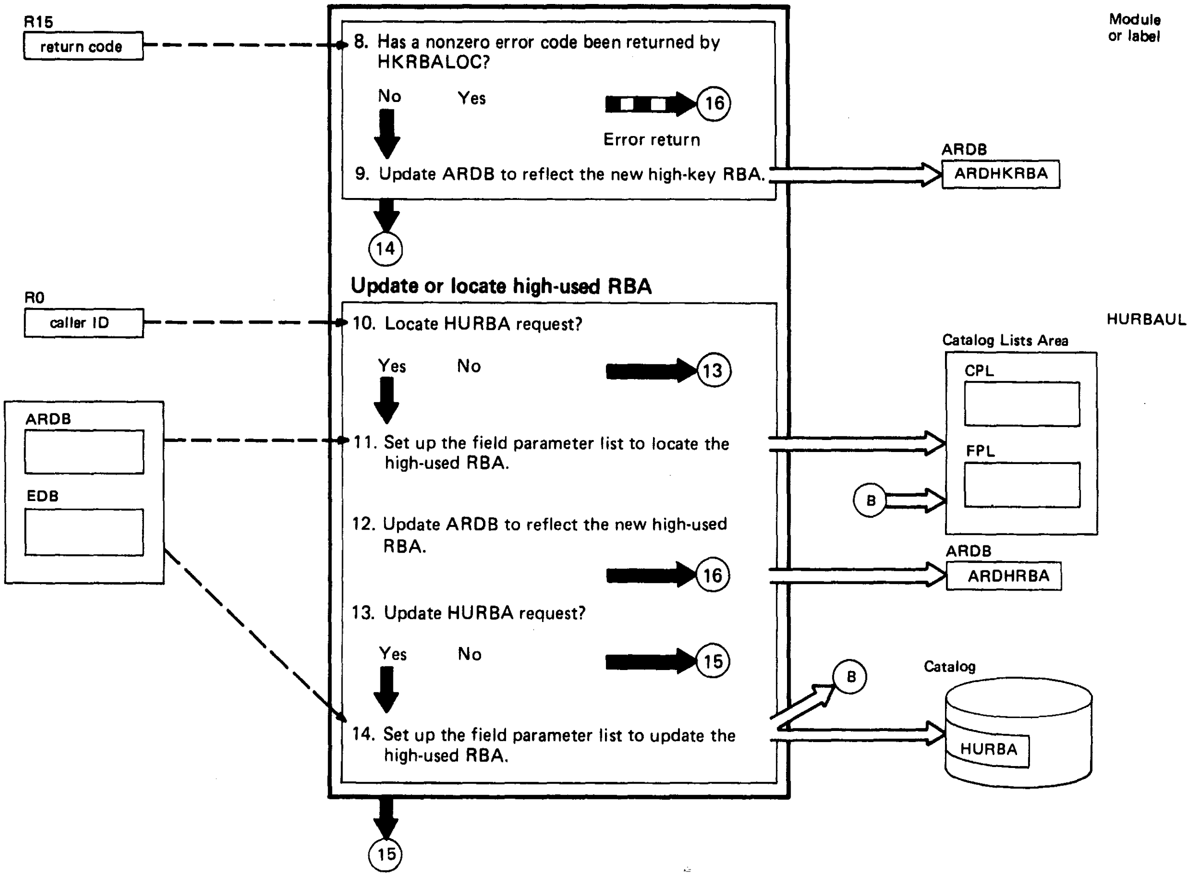
6. Does the return code from the UPDATE function indicate that the field to be updated has been changed since the last update?

Yes No

Error return

7. Locate the most current high-key RBA from the new catalog entry.

Diagram CO2. Update Catalog



8. Has a nonzero error code been returned by HKRBALOC?

9. Update ARDB to reflect the new high-key RBA.

Update or locate high-used RBA

10. Locate HURBA request?

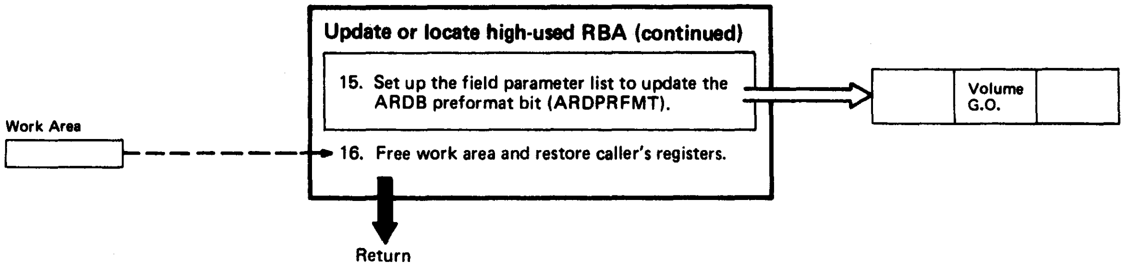
11. Set up the field parameter list to locate the high-used RBA.

12. Update ARDB to reflect the new high-used RBA.

13. Update HURBA request?

14. Set up the field parameter list to update the high-used RBA.

Diagram CO3. Update Catalog

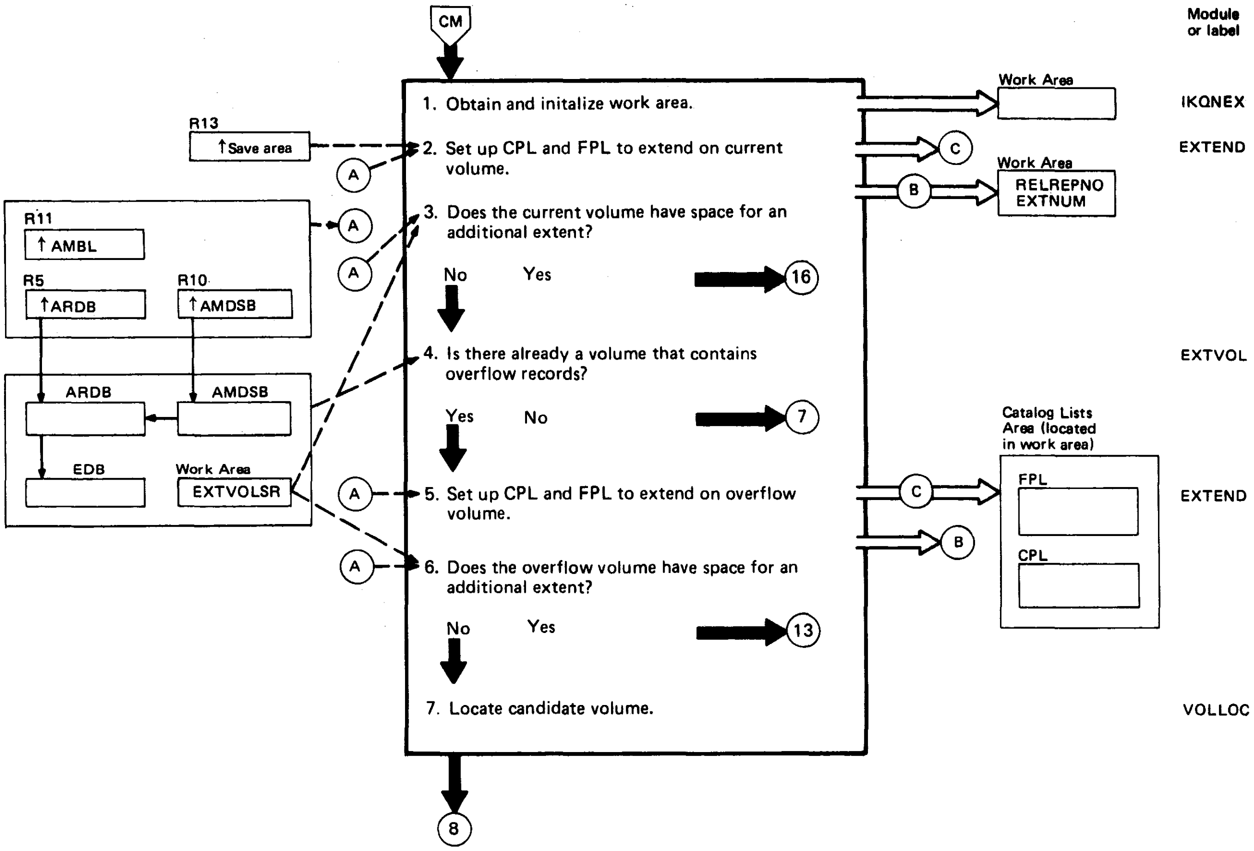


Update or locate high-used RBA (continued)

15. Set up the field parameter list to update the ARDB preformat bit (ARDPRFMT).

16. Free work area and restore caller's registers.

Diagram CP1. Get New Extent



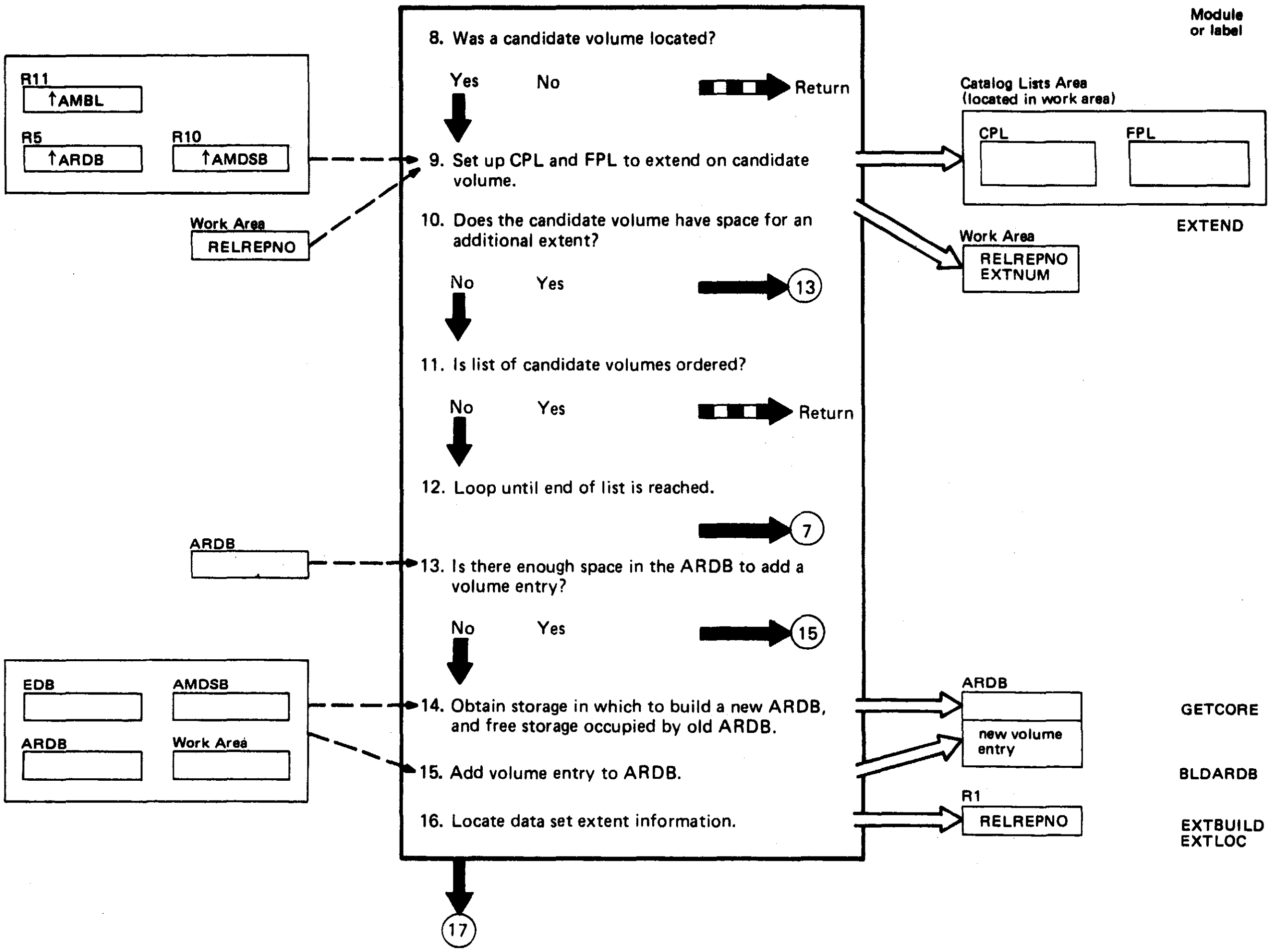
1. Obtain and initalize work area.
2. Set up CPL and FPL to extend on current volume.
3. Does the current volume have space for an additional extent?
4. Is there already a volume that contains overflow records?
5. Set up CPL and FPL to extend on overflow volume.
6. Does the overflow volume have space for an additional extent?
7. Locate candidate volume.

Notes for Diagram CP1

Note: This module falls into four logical groupings:

- Steps 1-2 where an attempt is made to extend on the current volume.
- Steps 3-5 where an attempt is made to extend on an overflow volume, if the current volume did not have enough space for an additional extent.
- Steps 6-11 where an attempt is made to extend on a candidate volume, if the overflow volume did not have enough space for an additional extent.
- Steps 12-16 where the data set extent information is located on the proper volume, a volume entry is made to the ARDB, and EDB(s) are built.

Diagram CP2. Get New Extent



8. Was a candidate volume located?

Yes No Return

9. Set up CPL and FPL to extend on candidate volume.

10. Does the candidate volume have space for an additional extent?

No Yes 13

11. Is list of candidate volumes ordered?

No Yes Return

12. Loop until end of list is reached.

7

13. Is there enough space in the ARDB to add a volume entry?

No Yes 15

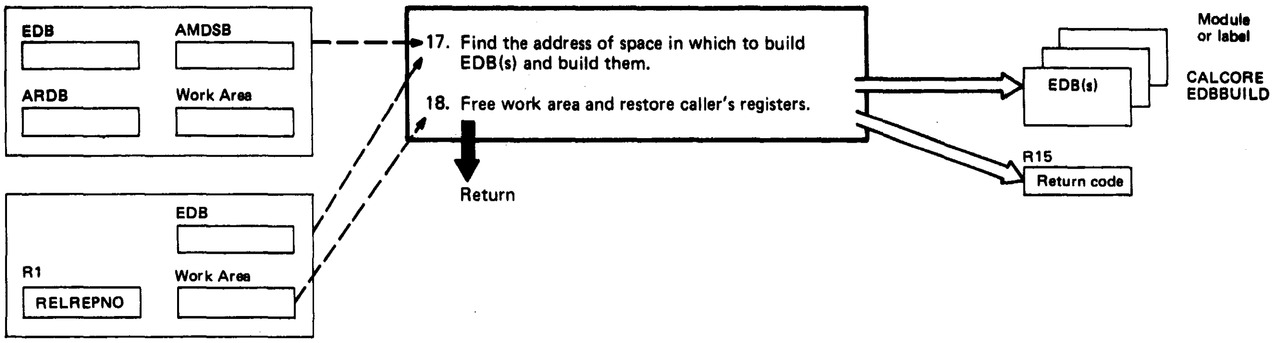
14. Obtain storage in which to build a new ARDB, and free storage occupied by old ARDB.

15. Add volume entry to ARDB.

16. Locate data set extent information.

17

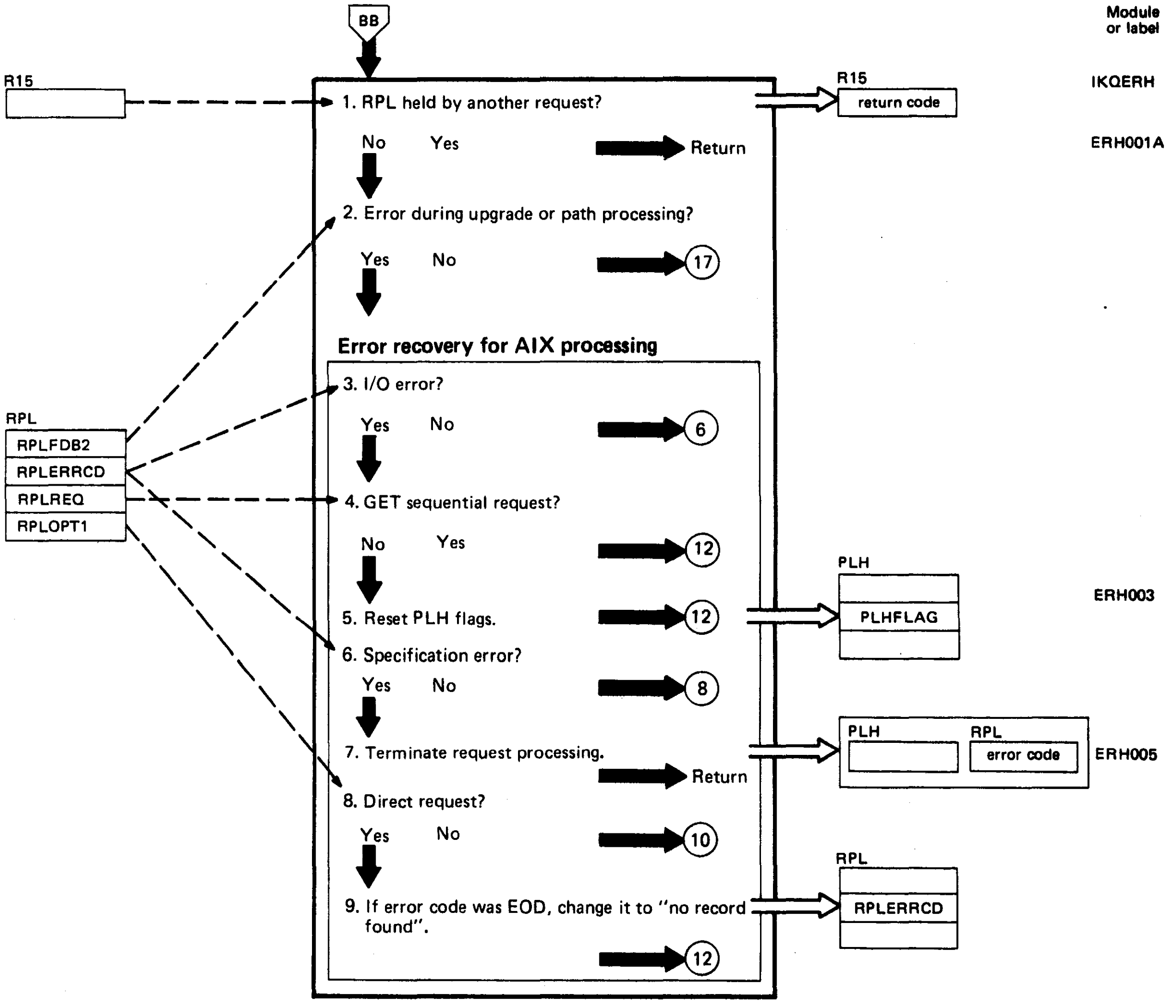
Diagram CP3. Get New Extent



17. Find the address of space in which to build EDB(s) and build them.

18. Free work area and restore caller's registers.

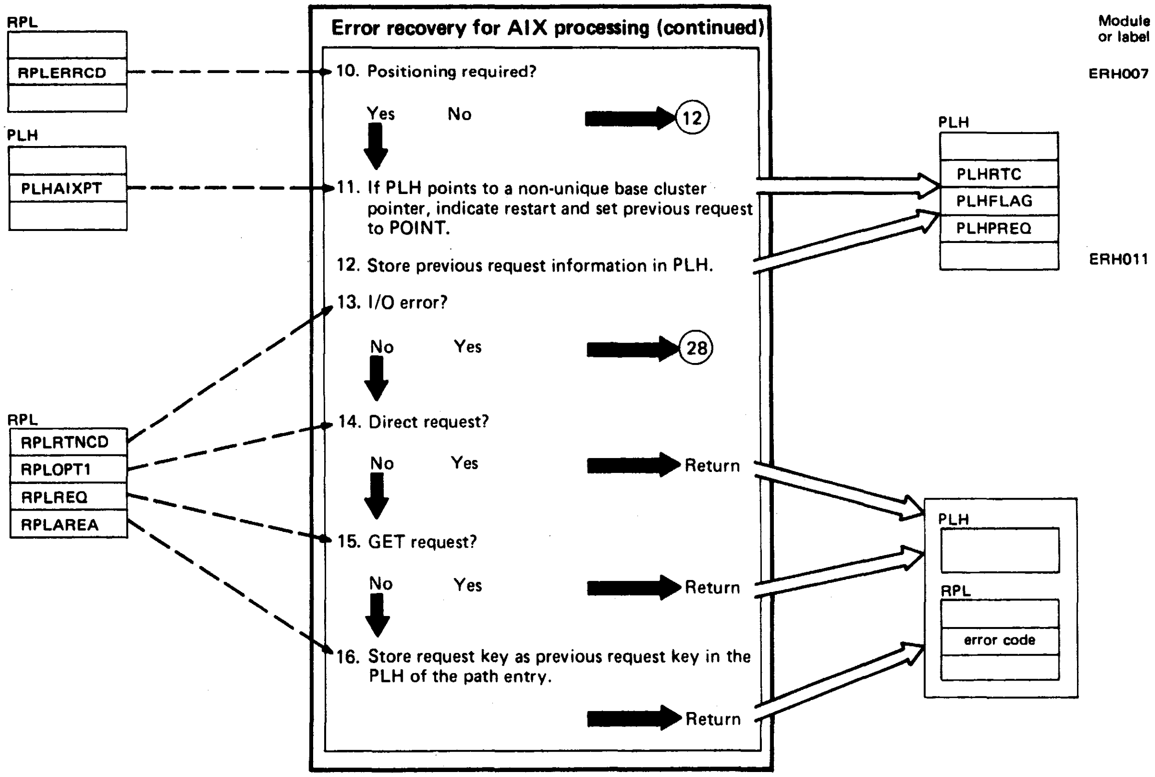
Diagram CQ1. Error Handler



Notes for Diagram CQ1

1. If the user did not point to a valid RPL or the RPL is in use (used by another task), no feedback information can be stored into the RPL.

Diagram CQ2. Error Handler



Error recovery for AIX processing (continued)

10. Positioning required?

Yes ↓ No → (12)

11. If PLH points to a non-unique base cluster pointer, indicate restart and set previous request to POINT.

12. Store previous request information in PLH.

13. I/O error?

No ↓ Yes → (28)

14. Direct request?

No ↓ Yes → Return

15. GET request?

No ↓ Yes → Return

16. Store request key as previous request key in the PLH of the path entry.

→ Return

Diagram CQ3. Error Handler

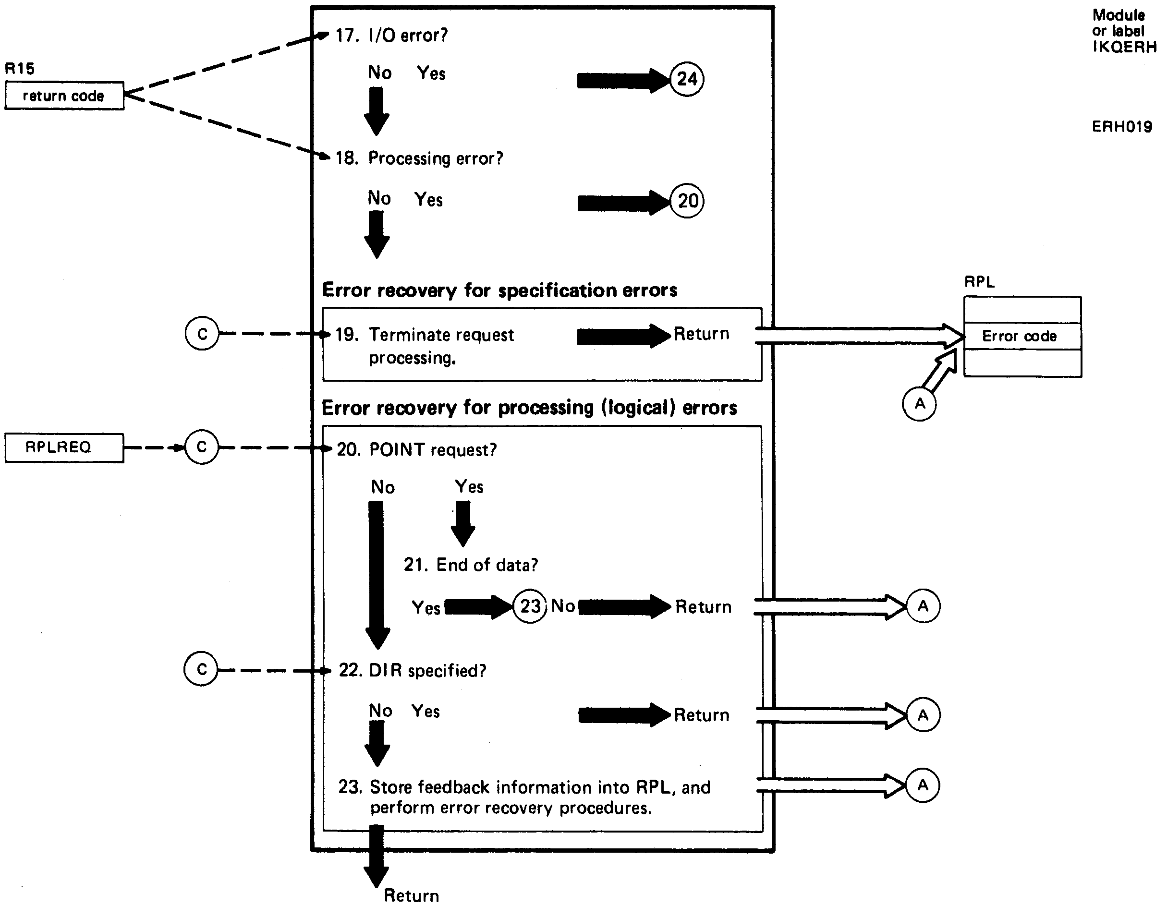
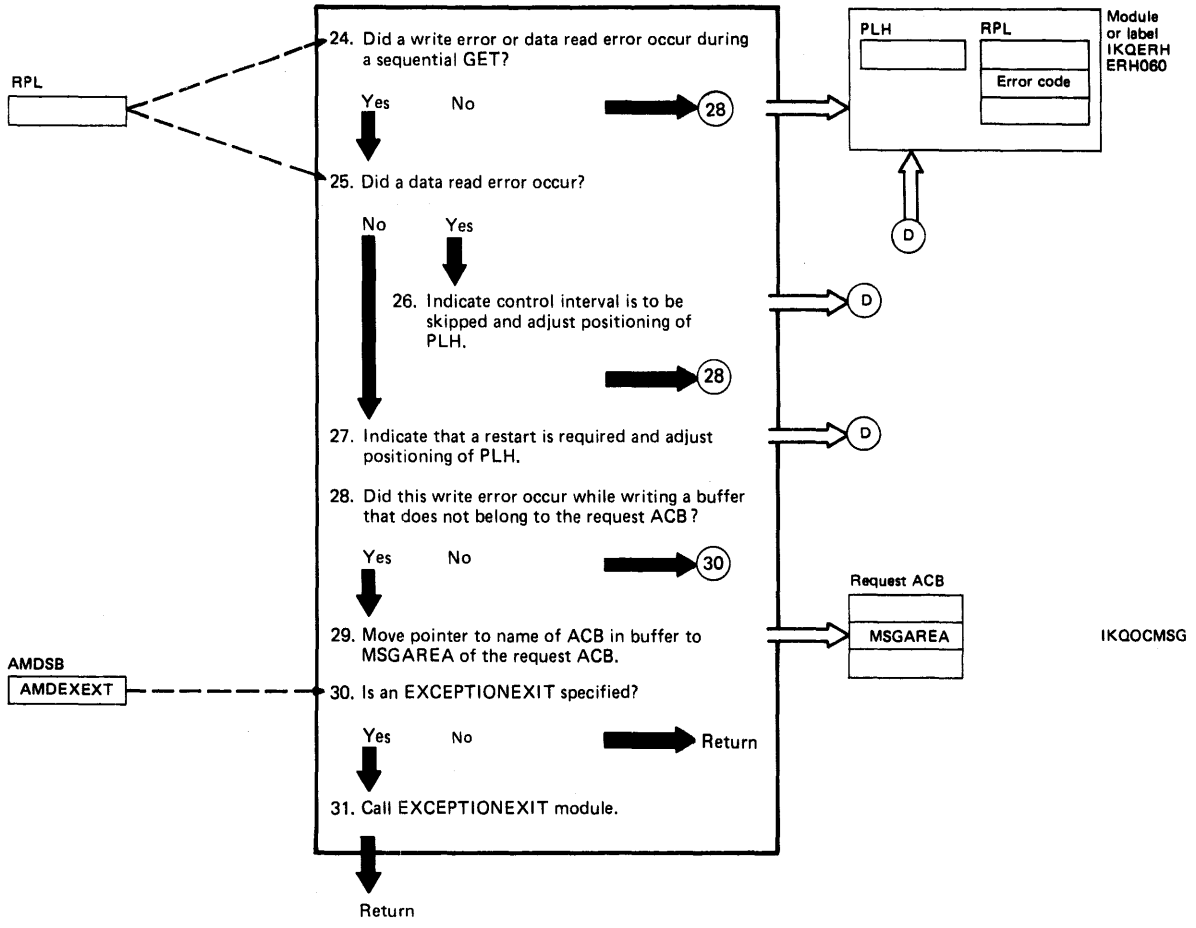


Diagram CQ4. Error Handler

Error recovery for I/O (physical) errors



24. Did a write error or data read error occur during a sequential GET?

25. Did a data read error occur?

26. Indicate control interval is to be skipped and adjust positioning of PLH.

27. Indicate that a restart is required and adjust positioning of PLH.

28. Did this write error occur while writing a buffer that does not belong to the request ACB?

29. Move pointer to name of ACB in buffer to MSGAREA of the request ACB.

30. Is an EXCEPTIONEXIT specified?

31. Call EXCEPTIONEXIT module.

Notes for Diagram CQ4

24. A write error can occur on a GET when buffers (that have not been written) are written because more are needed for the GET.

26. For a data read error occurring during a sequential GET, the user may continue processing. The next GET issued will skip the erroneous data control interval.

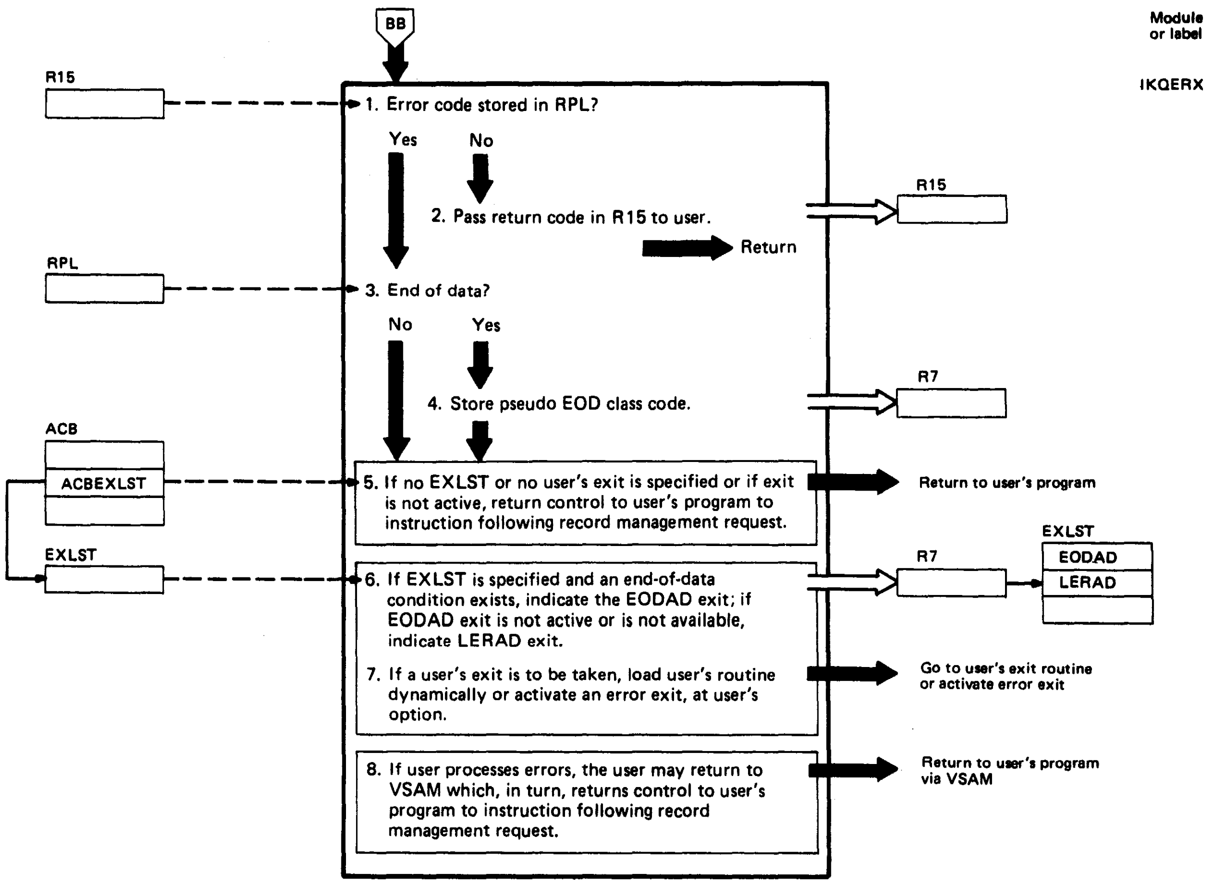
    If necessary, IKQBFA is called to release exclusive control and track hold.

27. For a write error occurring during a sequential GET, positioning will automatically be reestablished for the next request, that is, a restart is performed. To the user's program, the erroneous operation looks like a no-operation except that an error exit is taken or an error code is returned. A subsequent sequential GET will cause processing to continue.

28. If the write error occurs while writing a buffer that does not belong to the request ACB, field PLHACB points to the ACB to which the buffer belongs. Otherwise PLHACB is 0.

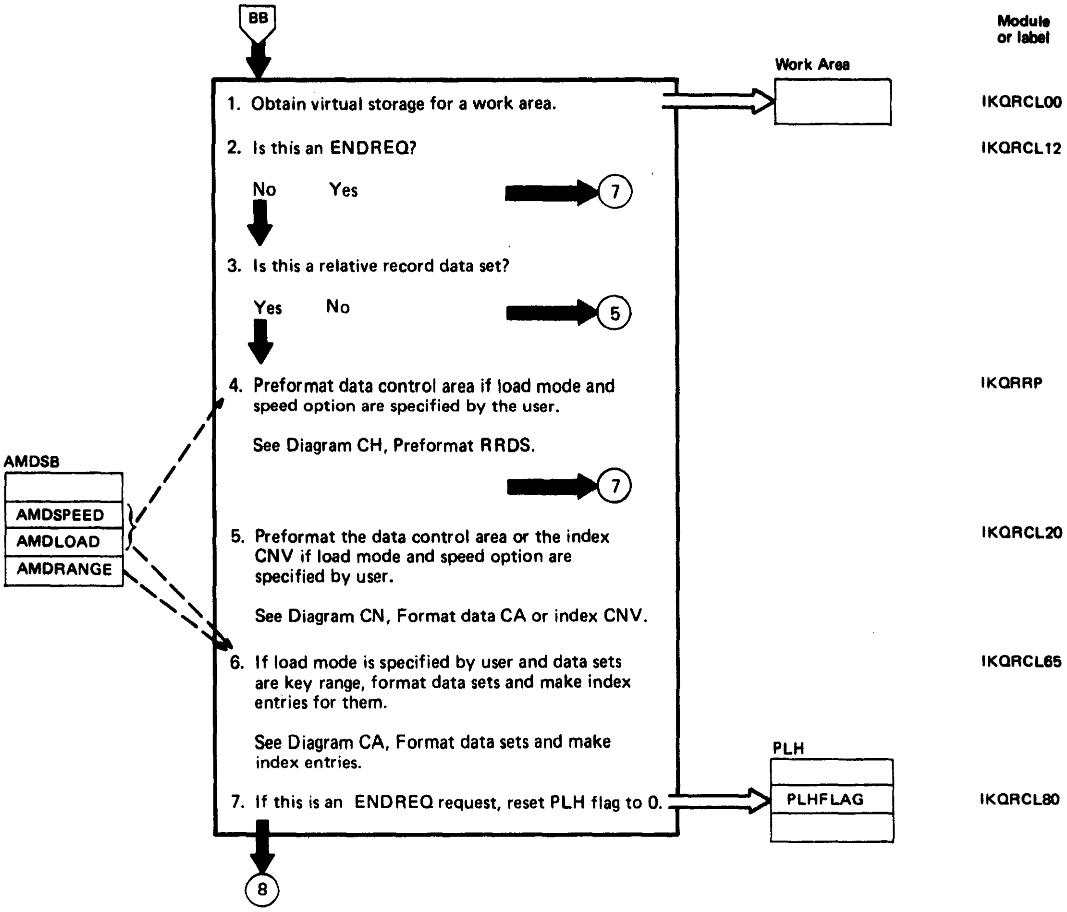
29. A pointer to the ACB name is moved into the MSGAREA of the request ACB for user information.

Diagram CR1. Error Exit



1. Error code stored in RPL?
2. Pass return code in R15 to user.
3. End of data?
4. Store pseudo EOD class code.
5. If no EXLST or no user's exit is specified or if exit is not active, return control to user's program to instruction following record management request.
6. If EXLST is specified and an end-of-data condition exists, indicate the EODAD exit; if EODAD exit is not active or is not available, indicate LERAD exit.
7. If a user's exit is to be taken, load user's routine dynamically or activate an error exit, at user's option.
8. If user processes errors, the user may return to VSAM which, in turn, returns control to user's program to instruction following record management request.

Diagram CS1. Record Management Close



1. Obtain virtual storage for a work area.

2. Is this an ENDREQ?

   No Yes → 7

3. Is this a relative record data set?

   Yes No → 5

4. Preformat data control area if load mode and speed option are specified by the user.

   See Diagram CH, Preformat RRDS.

   → 7

5. Preformat the data control area or the index CNV if load mode and speed option are specified by user.

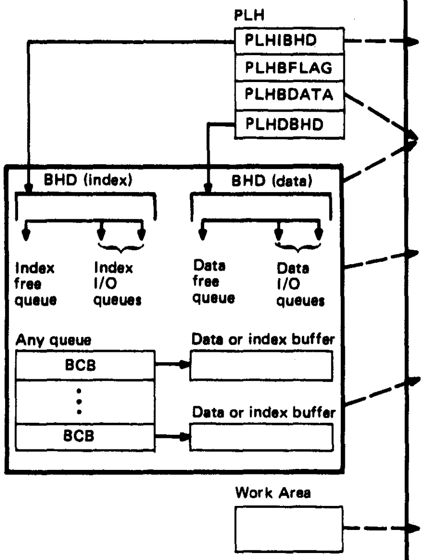
   See Diagram CN, Format data CA or index CNV.

6. If load mode is specified by user and data sets are key range, format data sets and make index entries for them.

   See Diagram CA, Format data sets and make index entries.

7. If this is an ENDREQ request, reset PLH flag to 0.

Diagram CS2. Record Management Close



Module or label

8. See Diagram DA, GETBUFF, for the following process: CBF020

(a) If a wait for completion of I/O is pending on the I/O scheduled queue, first clear the wait for the data buffers; then, for keyed processing, the wait for index buffers is cleared.

(b) If the PLH indicates any outstanding write requests not already on I/O queues, the write BCBs are placed on the data or index I/O queues.

(c) Read-only requests are purged from the data or index I/O queues, and I/O is scheduled for all remaining write requests (including both BCBs on I/O queues at the time a close was issued and those placed on the I/O queues by this routine).

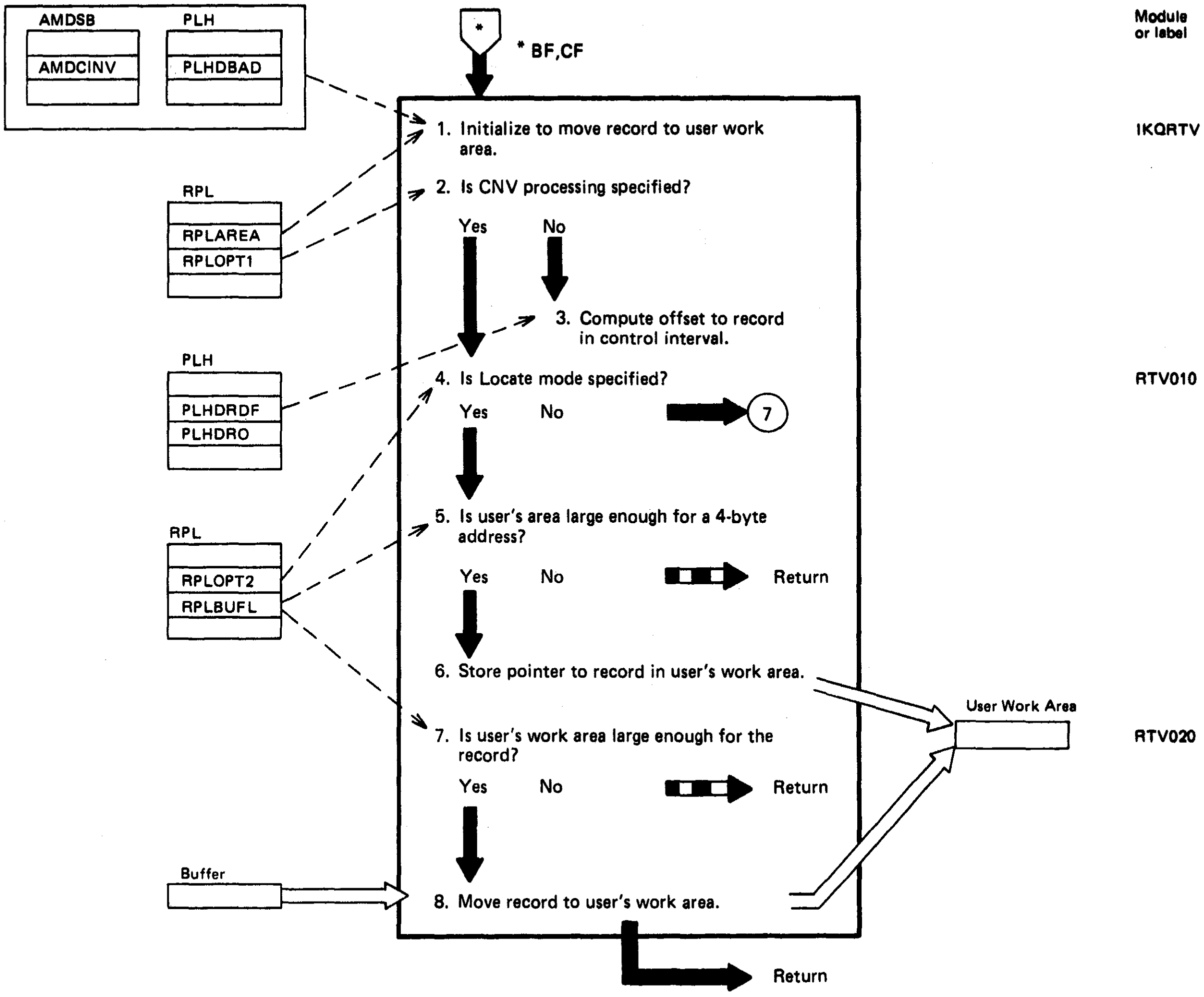
(d) When the waits for completion of write requests have been cleared, transfer BCBs from I/O schedule queue to free queue.

(e) Free any data or index control intervals that were held.

9. Free work area. IKQRCL90

Return

Diagram CT1. Move Record to User Work Area

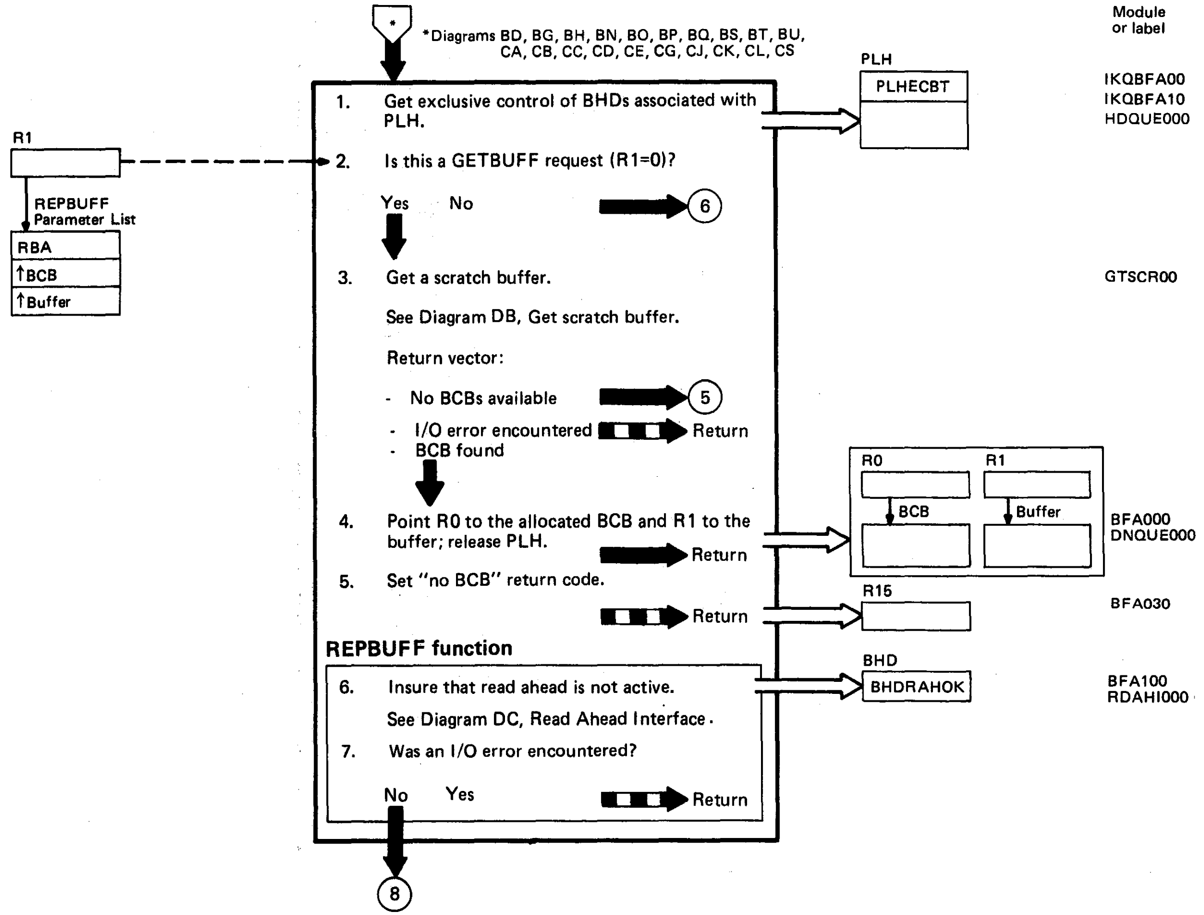


1. Initialize to move record to user work area.
2. Is CNV processing specified?
3. Compute offset to record in control interval.
4. Is Locate mode specified?
5. Is user's area large enough for a 4-byte address?
6. Store pointer to record in user's work area.
7. Is user's work area large enough for the record?
8. Move record to user's work area.

Notes for Diagram CT1

This routine moves the record to the user work area if MVE was specified in the RPL. (The CI is moved if CNV was also specified.) If LOC was specified, the pointer to the record (or CI) is stored in the user work area.

Diagram DA1. Buffer Manager: GETBUFF


*
*Diagrams BD, BG, BH, BN, BO, BP, BQ, BS, BT, BU, CA, CB, CC, CD, CE, CG, CJ, CK, CL, CS
Module or label
R1
REPBUFF Parameter List
RBA
↑BCB
↑Buffer
1. Get exclusive control of BHDs associated with PLH.
2. Is this a GETBUFF request (R1=0)?
Yes No
6
3. Get a scratch buffer.
See Diagram DB, Get scratch buffer.
Return vector:
- No BCBs available
5
- I/O error encountered
Return
- BCB found
4. Point R0 to the allocated BCB and R1 to the buffer; release PLH.
Return
5. Set "no BCB" return code.
Return
REPBUFF function
6. Insure that read ahead is not active.
See Diagram DC, Read Ahead Interface.
7. Was an I/O error encountered?
No Yes
Return
8
PLH
PLHECBT
IKQBFA00
IKQBFA10
HDQUE000
GTSCR00
R0
R1
BCB
Buffer
BFA000
DNQUE000
R15
BFA030
BHD
BHDRAHOK
BFA100
RDAHI000

Notes for Diagram DA1

The two main functions of this module are FREEBUFF (free a buffer) and GETBUFF (get a buffer). The REPBUFF function is a combination of the two, except when "no read" is specified. In this case, the function is basically a FREEBUFF.

FREEBUFF:
Free the input BCB contained in the parameter list:

a. Do any necessary I/O operations.
b. Place the BCB on the free queue.

Note: The first step may force additional I/O on the queues.

GETBUFF:
Get a buffer with the requested RBA:

a. Search the queues for the requested RBA, in order to determine if it is already in storage. If so, detach the buffer from the queue and complete the parameter list for return to the caller.
b. If the RBA is not located on the free queue, as indicated by the BCB "buffer contents valid" flag, but is found on the scheduled or nonscheduled queue, force completion of the I/O, thus forcing the BCB into the free queue, where it can be used to satisfy the request.
   Note for LSR processing:
   If the RBA is not found in one of the queues, the subpool of the VSAM shared resources pool is searched for the requested RBA.
c. If the RBA is not in storage, get a scratch buffer and read in the RBA. Then repeat step a. above.
   Note: Once the request has been satisfied, the BCB/buffer is unknown to the buffer manager, because it has been detached from the queues.

NO READ:
If the "no read" flag (PLHNORD) is set in the parameter list, the request is really a FREEBUFF request via the REPBUFF interface. This means that the "requested" RBA is not read in. Only the BCB in the parameter list is freed.

Definitions of the queues
There are four queues that can hold BCBs. These are shown below, with their pointers from the BHD, and their contents.

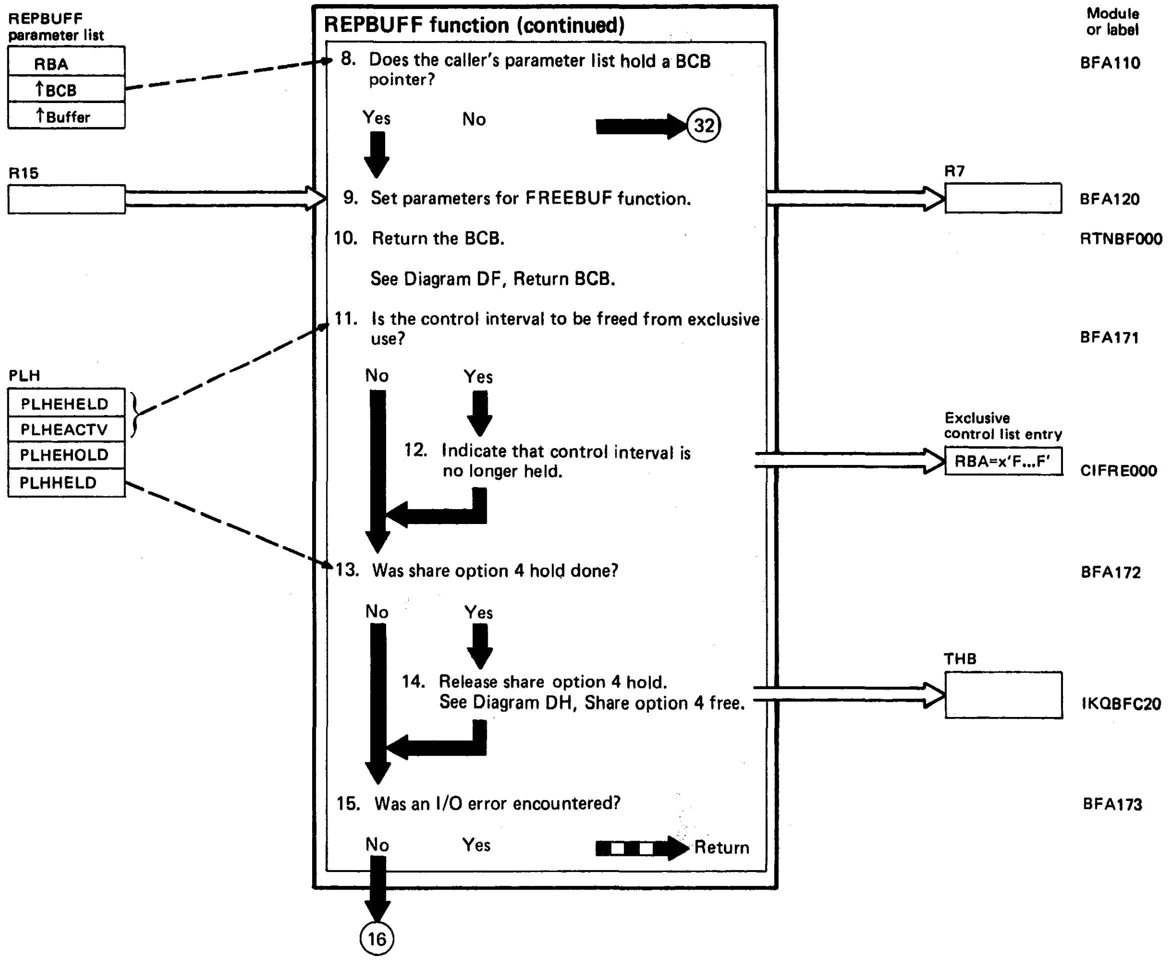
Scheduled queue:
Located by BHDSKDQ. Contains BCBs for which I/O has been started without wait.

Nonscheduled queue:
Located by BHDNSKDQ. Contains BCBs for which I/O needs to be done but has not been started yet.

I/O queue:
Located by BHD1STW. Contains BCBs on which the I/O manager is presently working (to build channel programs). This is the primary input from the buffer manager to the I/O manager.

Free queue:
Located by BHD1STF. Contains BCBs (with and without buffers with valid contents) that are available to record management.

Diagram DA2. Buffer Manager: GETBUFF



REPBUFF function (continued)

8. Does the caller's parameter list hold a BCB pointer?

Yes / No → 32

9. Set parameters for FREEBUF function.

10. Return the BCB.

See Diagram DF, Return BCB.

11. Is the control interval to be freed from exclusive use?

No / Yes

12. Indicate that control interval is no longer held.

13. Was share option 4 hold done?

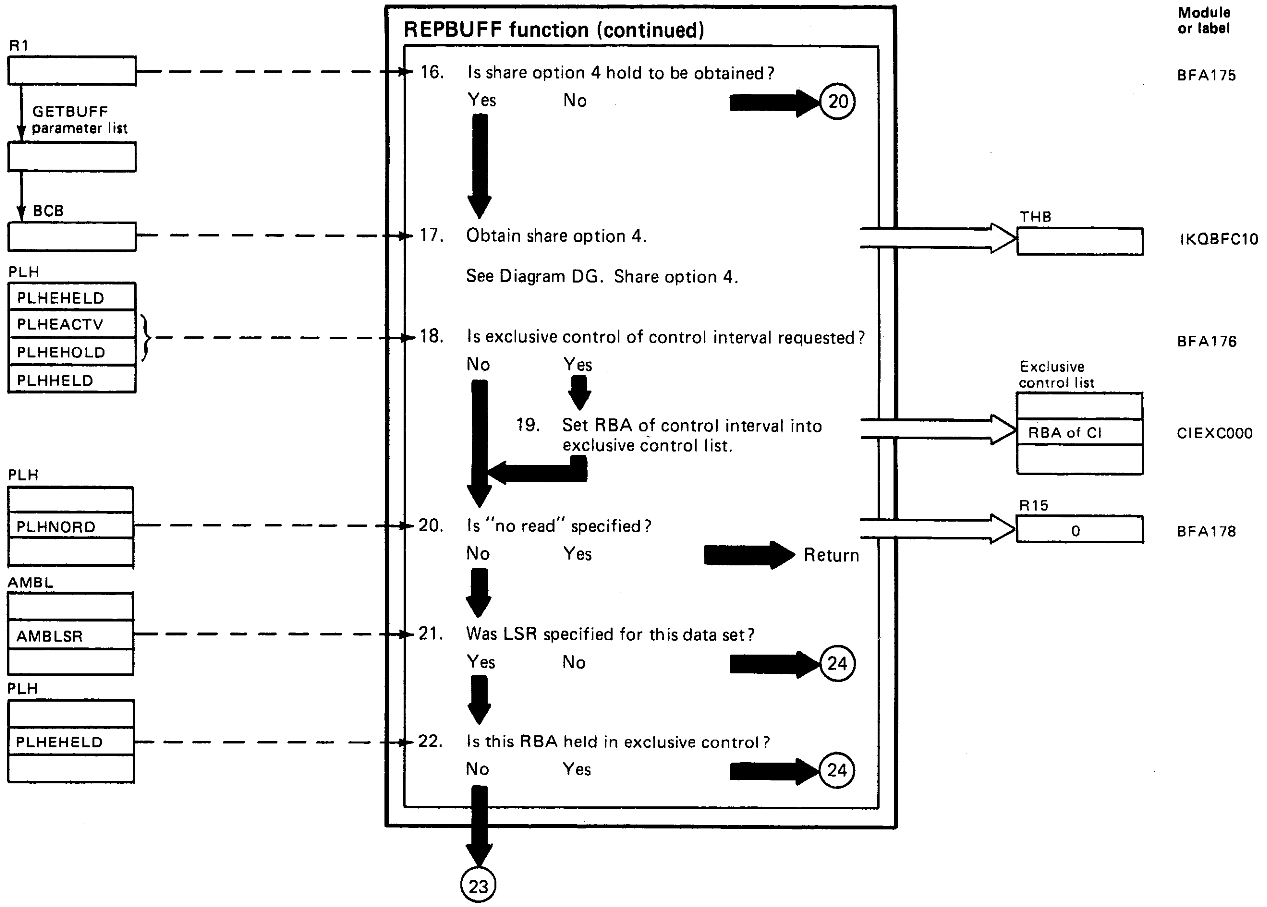
No / Yes

14. Release share option 4 hold. See Diagram DH, Share option 4 free.

15. Was an I/O error encountered?

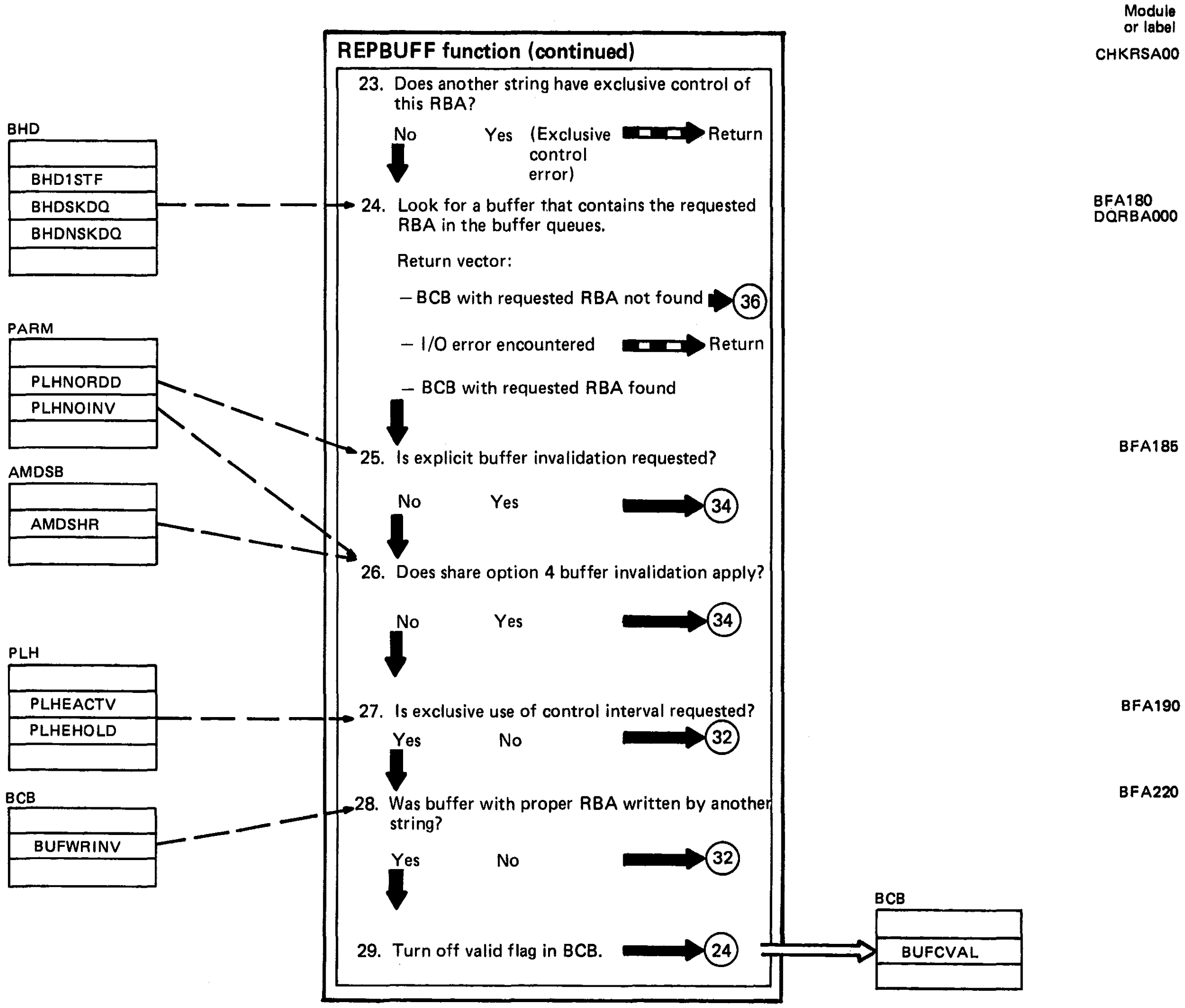
No → 16 / Yes → Return

Diagram DA3. Buffer Manager: GETBUFF



REPBUFF function (continued)
Module or label
R1
GETBUFF parameter list
BCB
PLH
PLHEHELD
PLHEACTV
PLHEHOLD
PLHHELD
PLH
PLHNORD
AMBL
AMBLSR
PLH
PLHEHELD
16. Is share option 4 hold to be obtained?
Yes
No
20
BFA175
17. Obtain share option 4.
See Diagram DG. Share option 4.
THB
IKQBFC10
18. Is exclusive control of control interval requested?
No
Yes
BFA176
19. Set RBA of control interval into exclusive control list.
Exclusive control list
RBA of CI
CIEXC000
20. Is "no read" specified?
No
Yes
Return
R15
0
BFA178
21. Was LSR specified for this data set?
Yes
No
24
22. Is this RBA held in exclusive control?
No
Yes
24
23

Diagram DA4. Buffer Manager: GETBUFF

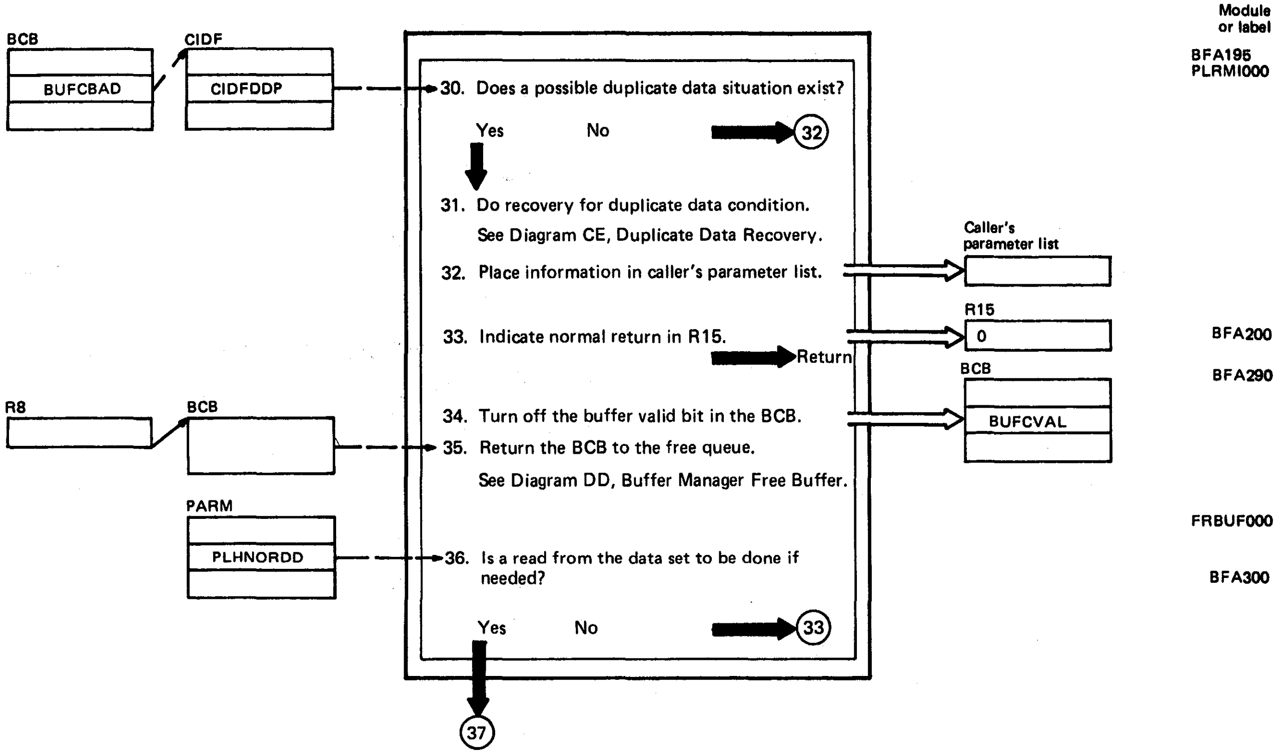

Module or label
REPBUFF function (continued)
23. Does another string have exclusive control of this RBA?
No
Yes (Exclusive control error)
Return
CHKRSA00
BHD
BHD1STF
BHDSKDQ
BHDNSKDQ
24. Look for a buffer that contains the requested RBA in the buffer queues.
Return vector:
– BCB with requested RBA not found
36
– I/O error encountered
Return
– BCB with requested RBA found
BFA180
DQRBA000
PARM
PLHNORDD
PLHNOINV
AMDSB
AMDSHR
25. Is explicit buffer invalidation requested?
No
Yes
34
BFA185
26. Does share option 4 buffer invalidation apply?
No
Yes
34
PLH
PLHEACTV
PLHEHOLD
27. Is exclusive use of control interval requested?
Yes
No
32
BFA190
BCB
BUFWRINV
28. Was buffer with proper RBA written by another string?
Yes
No
32
BFA220
29. Turn off valid flag in BCB.
24
BCB
BUFCVAL

Notes for Diagram DA4

25. A special interpretation is given when "no read from data set" (PLHNORDD) is specified but invalidation is allowed (PLHNOINV is off in the Buffer Manager Parameter List). This special interpretation is to explicitly invalidate any existing buffer with the RBA specified by PARMRBA (regardless of share option specification).

26. Share option 4 invalidation applies to any share option 4 data set (AMDSHR is on) for which invalidation has not been suppressed (PLHNOINV is *off*).

Diagram DA5. Buffer Manager: GETBUFF



30. Does a possible duplicate data situation exist?

31. Do recovery for duplicate data condition. See Diagram CE, Duplicate Data Recovery.

32. Place information in caller's parameter list.

33. Indicate normal return in R15.

34. Turn off the buffer valid bit in the BCB.

35. Return the BCB to the free queue. See Diagram DD, Buffer Manager Free Buffer.

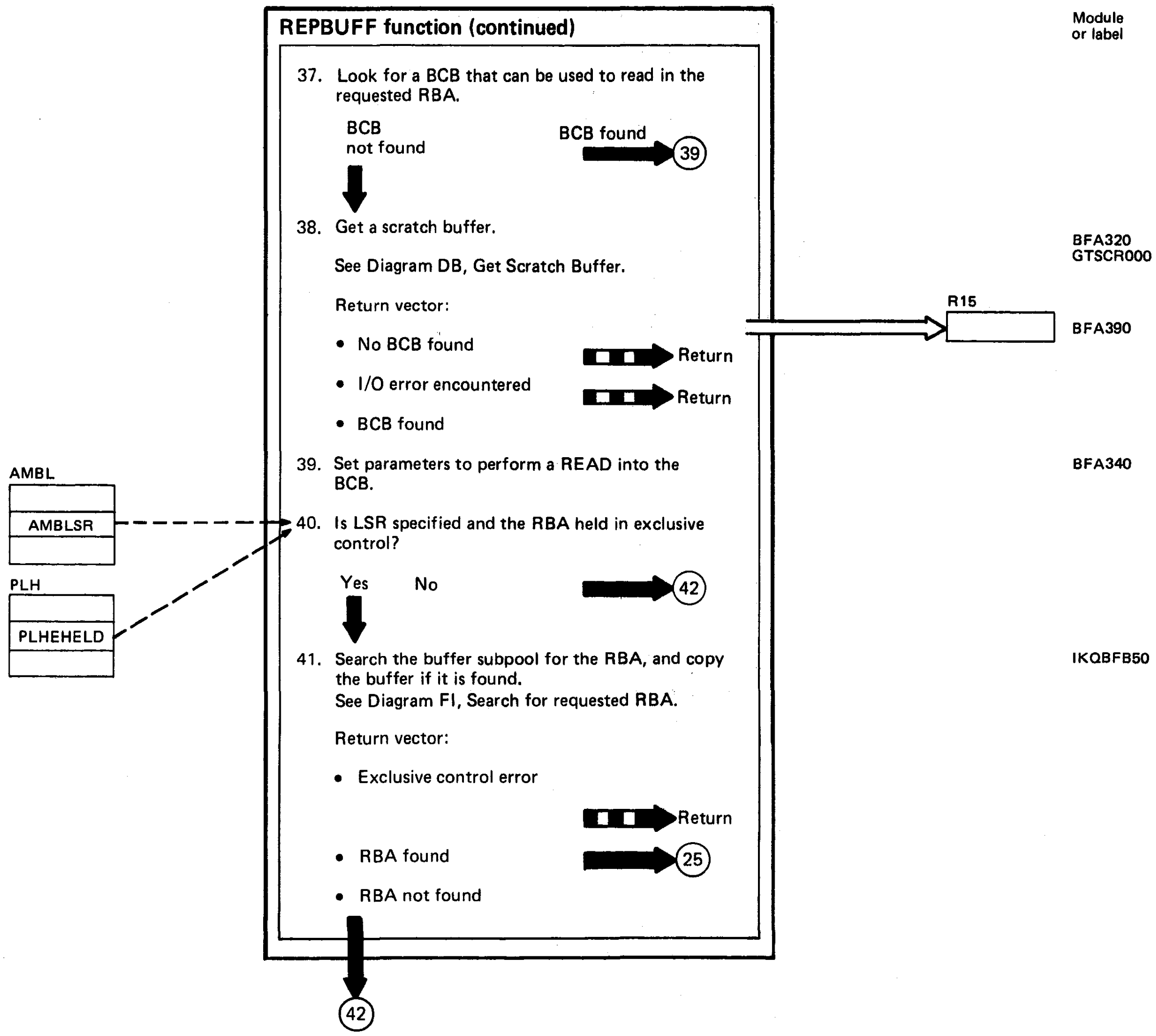
36. Is a read from the data set to be done if needed?

Notes for Diagram DA5

31. Invocation of duplicate data recovery is suppressed if the PLHNODDR bit is on in the buffer manager parameter list. PLHNODDR is used during a CI split when a CI is being written while bit CIDFDDP is on (indicating a possible duplicate data condition exists).

34.-35. The buffer is invalidated and returned to the free queue as a scratch buffer.

Diagram DA6. Buffer Manager: GETBUFF



REPBUFF function (continued)

Module or label

37. Look for a BCB that can be used to read in the requested RBA.

38. Get a scratch buffer. — BFA320 GTSCR000

    See Diagram DB, Get Scratch Buffer.

    Return vector: — BFA390

    - No BCB found
    - I/O error encountered
    - BCB found

39. Set parameters to perform a READ into the BCB. — BFA340

40. Is LSR specified and the RBA held in exclusive control?

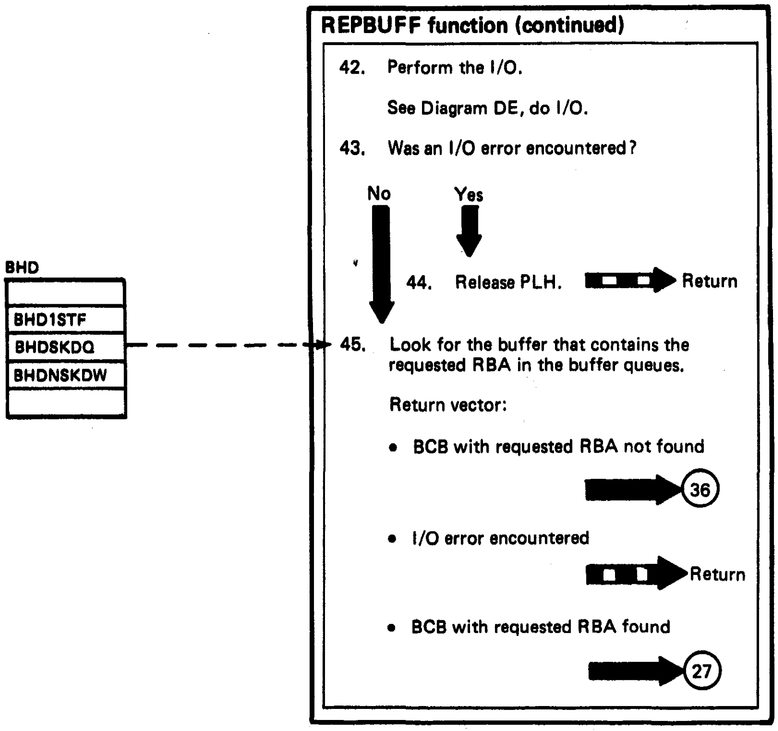
41. Search the buffer subpool for the RBA, and copy the buffer if it is found. — IKQBFB50

    See Diagram FI, Search for requested RBA.

    Return vector:

    - Exclusive control error
    - RBA found
    - RBA not found

Diagram DA7. Buffer Manager: GETBUFF



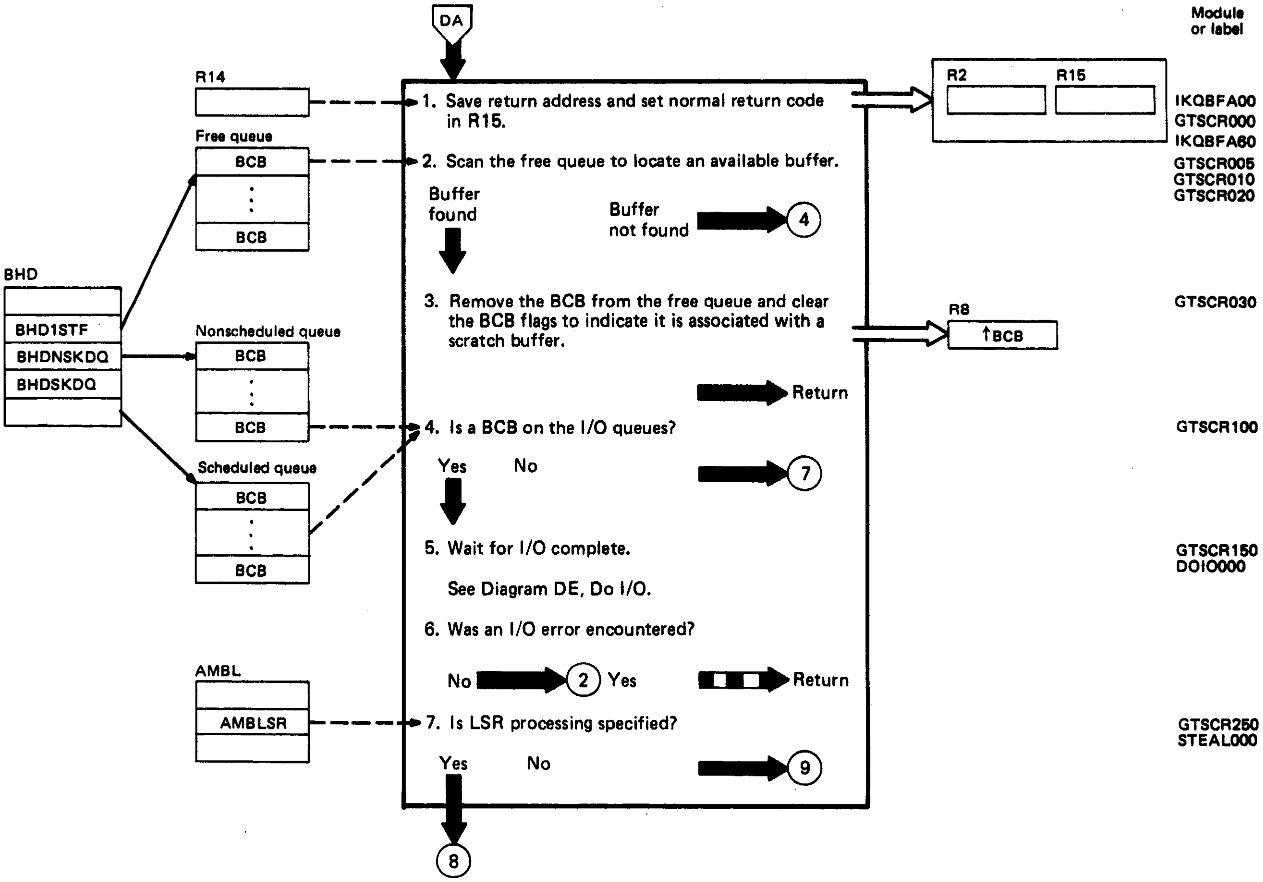
REPBUFF function (continued)

	Module or label
42. Perform the I/O.	BFA370
See Diagram DE, do I/O.	DOIO000
43. Was an I/O error encountered?	BFA210
44. Release PLH.	DNQUE000
45. Look for the buffer that contains the requested RBA in the buffer queues.	BFA380 DQRBA000

Return vector:

- BCB with requested RBA not found
- I/O error encountered
- BCB with requested RBA found

Diagram DB1. Buffer Manager: Get Scratch Buffer



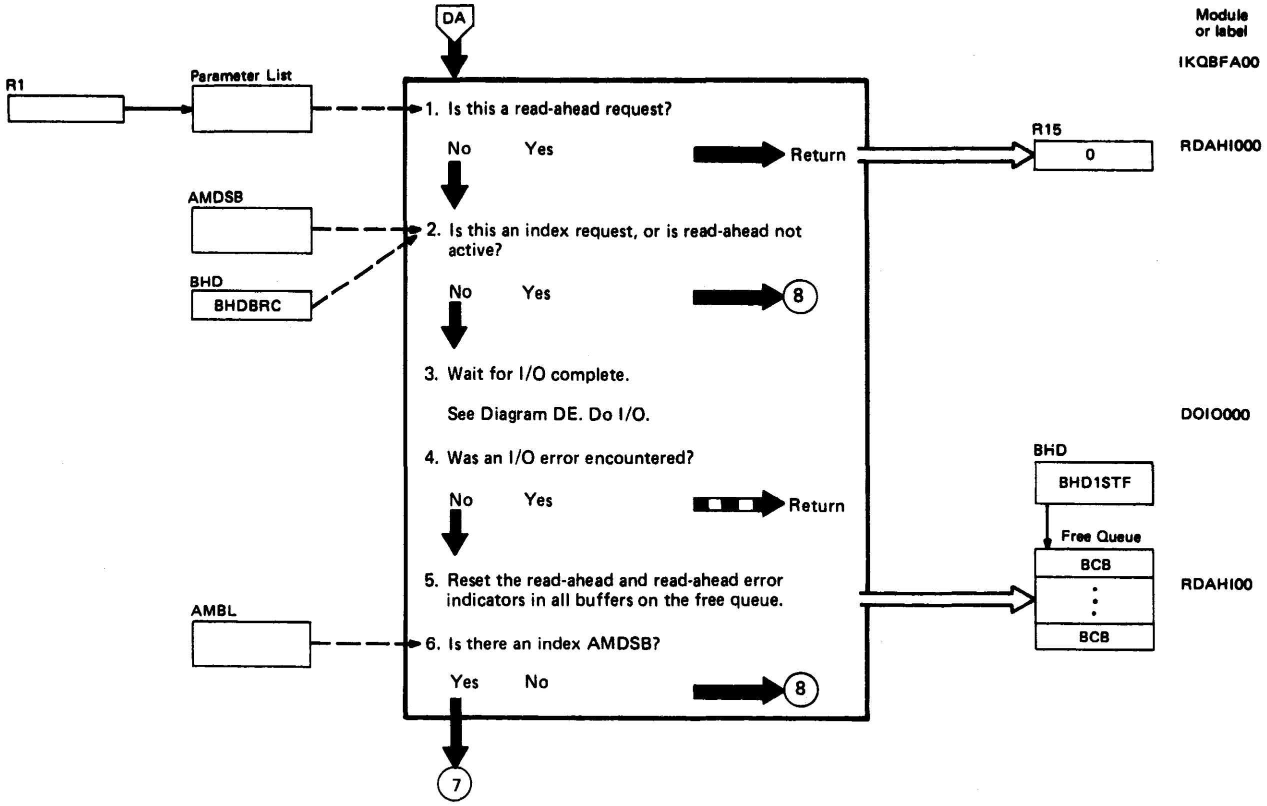
1. Save return address and set normal return code in R15.
2. Scan the free queue to locate an available buffer.
3. Remove the BCB from the free queue and clear the BCB flags to indicate it is associated with a scratch buffer.
4. Is a BCB on the I/O queues?
5. Wait for I/O complete.

   See Diagram DE, Do I/O.
6. Was an I/O error encountered?
7. Is LSR processing specified?

Diagram DB2. Buffer Manager: Get Scratch Buffer

	Module or label
8. Search buffer subpool for a scratch buffer. See Diagram FG, Get a scratch buffer. → ②	IKQBFB30
9. Must get a buffer from another string. Return vector: - Buffer found → ③ - Buffer not found → Return	IKQBFA20 STEAL020

Diagram DC1. Buffer Manager: Read-ahead Interface

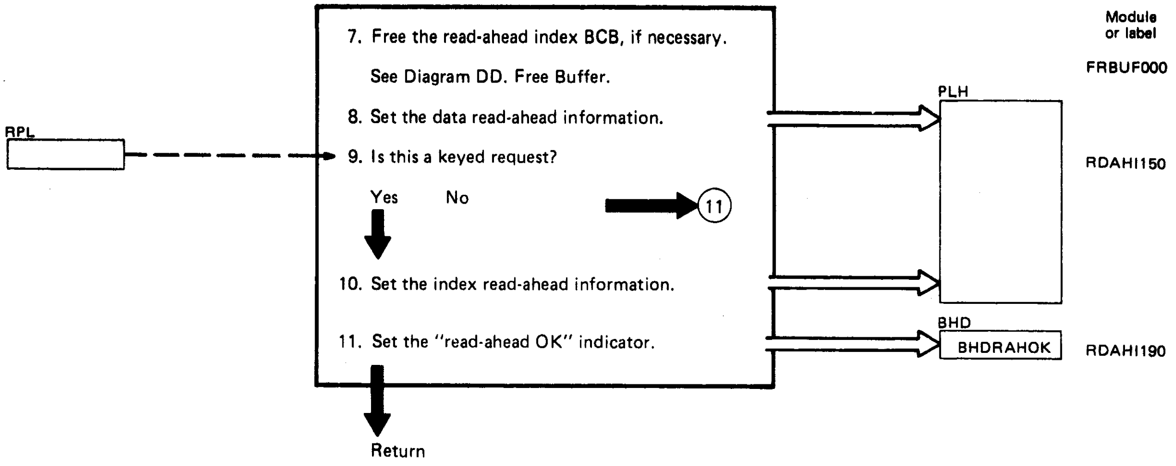

DA
Module or label
IKQBFA00
R1
Parameter List
1. Is this a read-ahead request?
No Yes
Return
R15
0
RDAHI000
AMDSB
2. Is this an index request, or is read-ahead not active?
BHD
BHDBRC
No Yes
8
3. Wait for I/O complete.
See Diagram DE. Do I/O.
DOIO000
BHD
BHD1STF
4. Was an I/O error encountered?
No Yes
Return
Free Queue
BCB
BCB
5. Reset the read-ahead and read-ahead error indicators in all buffers on the free queue.
RDAHI00
AMBL
6. Is there an index AMDSB?
Yes No
8
7

Notes for Diagram DC1

This routine handles any non-sequential request that occurs while read-ahead is active. If a non-sequential request, such as GET DIRECT, is received after read-ahead has been started, this interrupts the read-ahead operations, and the read-ahead routine in deactivated, and the read-ahead control information is reset. Read-ahead can be activated on a subsequent GET SEQUENTIAL when a control interval is read (GETNXT processing).

5. If a read-ahead error occurs, the buffer is flagged invalid.

Diagram DC2. Buffer Manager: Read-ahead Interface



7. Free the read-ahead index BCB, if necessary.

   See Diagram DD. Free Buffer.

8. Set the data read-ahead information.

9. Is this a keyed request?

   Yes No

10. Set the index read-ahead information.

11. Set the "read-ahead OK" indicator.

Return

Diagram DD1. Buffer Manager: Free Buffer

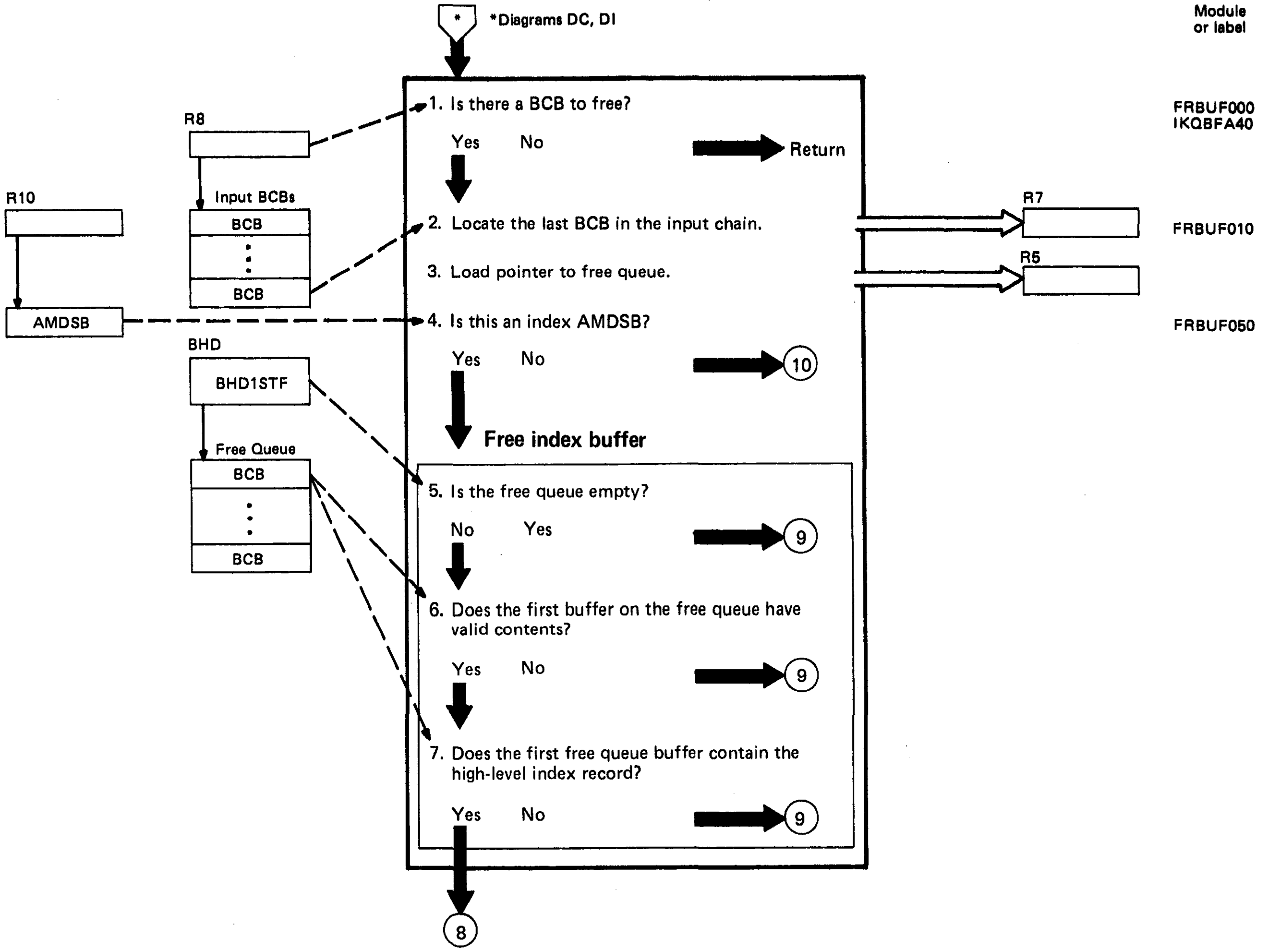
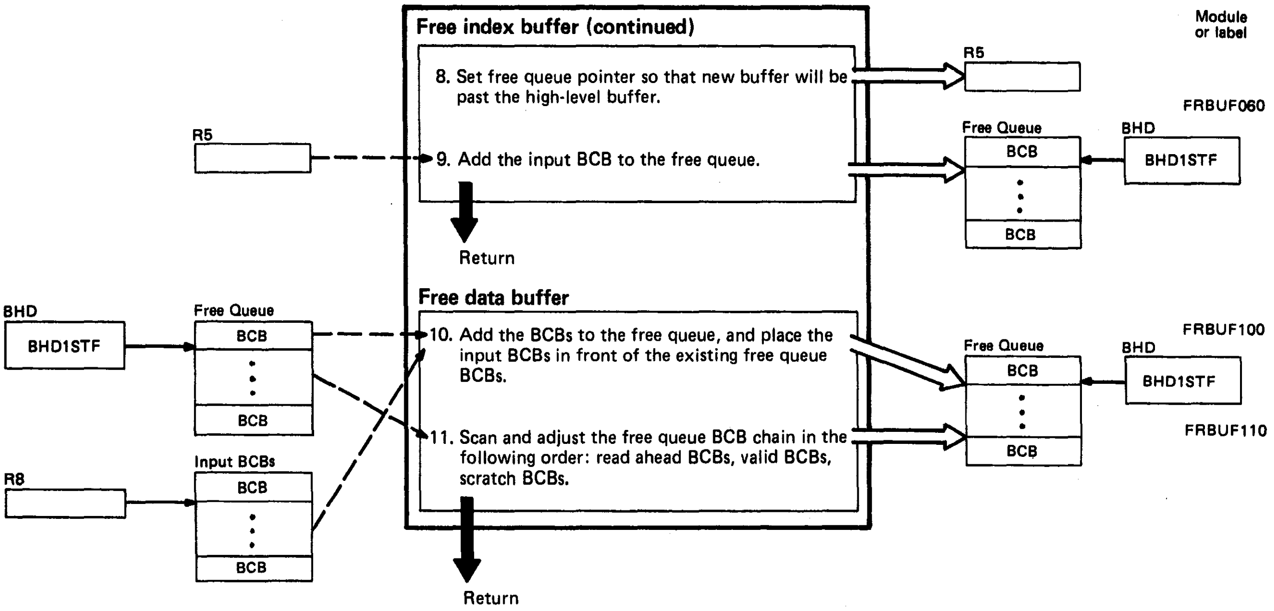

*Diagrams DC, DI
Module or label
R8
R10
Input BCBs
BCB
BCB
AMDSB
BHD
BHD1STF
Free Queue
BCB
BCB
1. Is there a BCB to free?
Yes No
Return
2. Locate the last BCB in the input chain.
3. Load pointer to free queue.
4. Is this an index AMDSB?
Yes No
10
Free index buffer
5. Is the free queue empty?
No Yes
9
6. Does the first buffer on the free queue have valid contents?
Yes No
9
7. Does the first free queue buffer contain the high-level index record?
Yes No
9
8
R7
R5
FRBUF000
IKQBFA40
FRBUF010
FRBUF050

Diagram DD2. Buffer Manager: Free Buffer



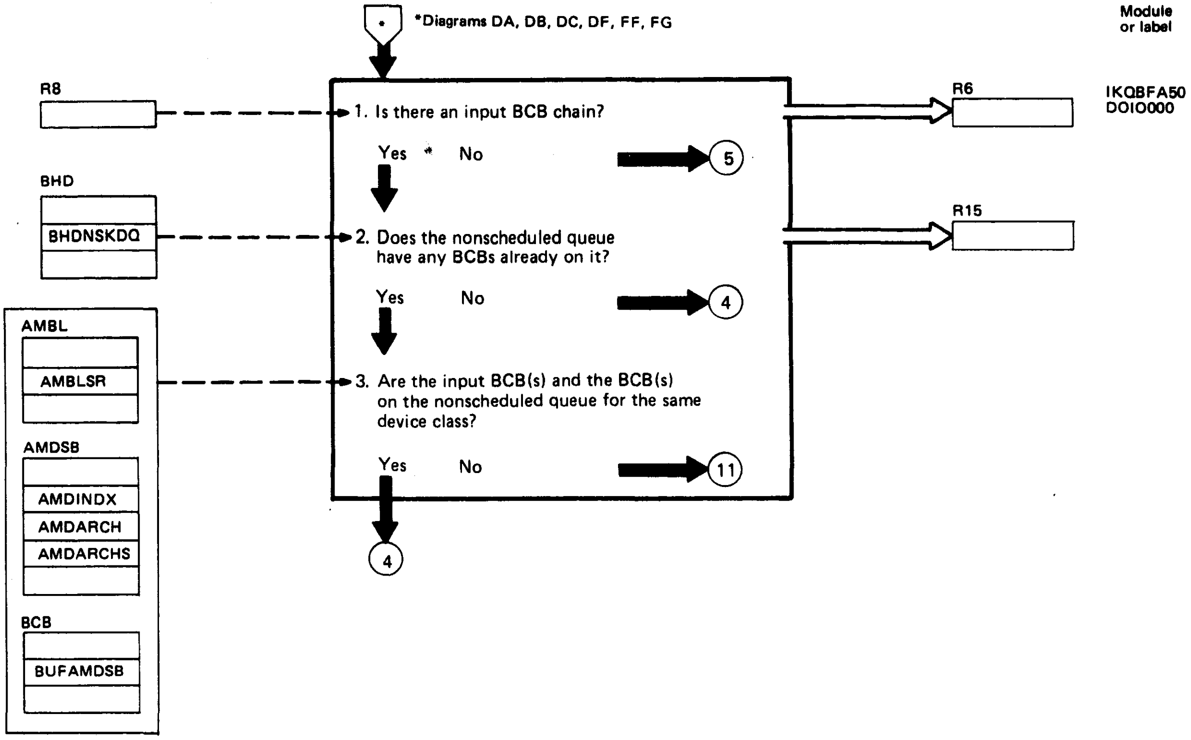
Module or label
Free index buffer (continued)
R5
8. Set free queue pointer so that new buffer will be past the high-level buffer.
R5
FRBUF060
R5
9. Add the input BCB to the free queue.
Free Queue
BCB
BCB
BHD
BHD1STF
Return
Free data buffer
BHD
BHD1STF
Free Queue
BCB
BCB
10. Add the BCBs to the free queue, and place the input BCBs in front of the existing free queue BCBs.
FRBUF100
Free Queue
BCB
BCB
BHD
BHD1STF
11. Scan and adjust the free queue BCB chain in the following order: read ahead BCBs, valid BCBs, scratch BCBs.
FRBUF110
Input BCBs
R8
BCB
BCB
Return

Notes for Diagram DD2

8. Make sure a buffer containing the high-level index control interval is lowest priority to be scratched for reuse.

11. Make sure the free queue BCB chain is in the following order of increasing priority to be scratched for reuse:
    - Read-ahead BCBs (valid contents)
    - Other BCBs with valid contents
    - Scratch BCBs (no valid contents)

Diagram DE1. Buffer Manager: Do I/O



*Diagrams DA, DB, DC, DF, FF, FG
Module or label
R8
BHD
BHDNSKDQ
AMBL
AMBLSR
AMDSB
AMDINDX
AMDARCH
AMDARCHS
BCB
BUFAMDSB
1. Is there an input BCB chain?
Yes
No
5
2. Does the nonscheduled queue have any BCBs already on it?
Yes
No
4
3. Are the input BCB(s) and the BCB(s) on the nonscheduled queue for the same device class?
Yes
No
11
4
R6
R15
IKQBFA50
DOIO000

Notes for Diagram DE1

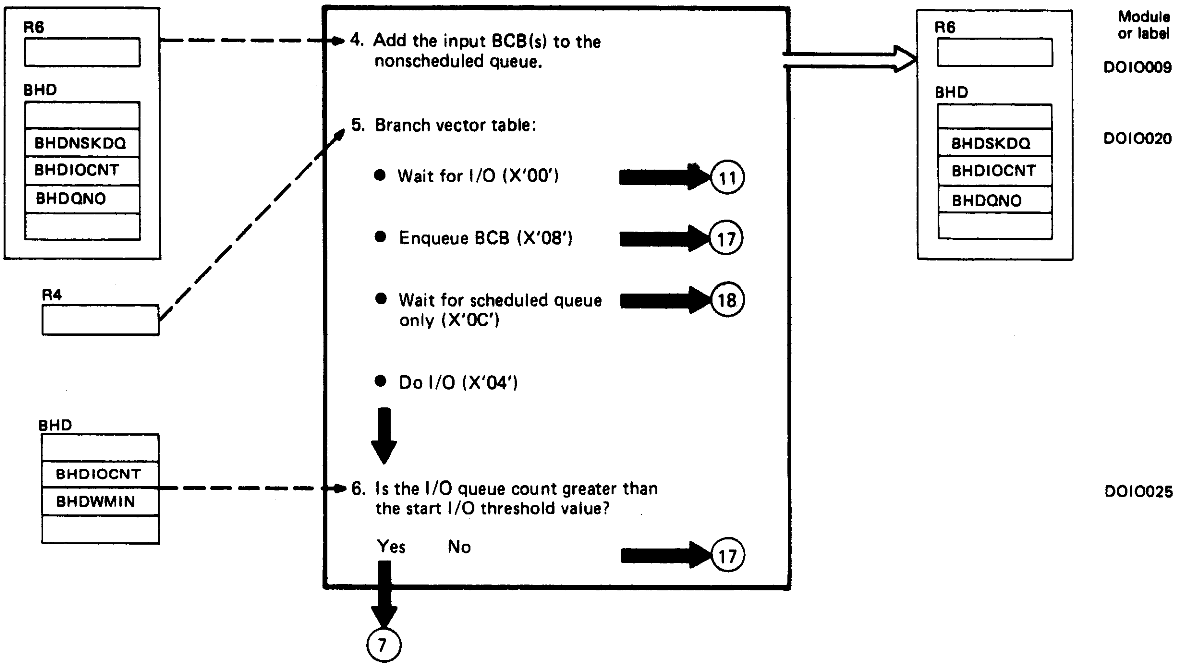
3. This test ensures that, in normal processing, only fixed block BCBs or only CKD BCBs will be given to the I/O Manager at one time.

   A device class conflict could occur in two cases:

   - If the high level index is on one device class and the data with an imbedded sequence set is on the other, there is a potential device class conflict in processing the index.
   - With LSR, the user could do a WRTBFR at a time when there are already BCBs for the other device class on the nonscheduled queue. (WRTBFR only invokes the Do I/O routine to write BCBs belonging to a single AMDSB at a time.)

   If there is a device class conflict, the I/O Manager is called to complete all I/O currently on the scheduled queue and the nonscheduled queue, before the input BCB(s) are enqueued. During the call(s) to the I/O Manager, a pointer to the input BCB(s) is saved in register 6.

Diagram DE2. Buffer Manager: Do I/O



Notes for Diagram DE2

4. Add the BCB(s) pointed to by register 6 to the nonscheduled queue. The count of I/O operations to be done is also incremented. Register 6 is cleared at completion of this step.

5. On entry to the Do I/O routine, register 4 contains one of the following codes:

   0 – Wait for all I/O to complete, including I/O newly enqueued with this invocation.

   4 – If the threshold (see note 6) has been reached, wait for previously started I/O to complete, and start I/O not previously started.

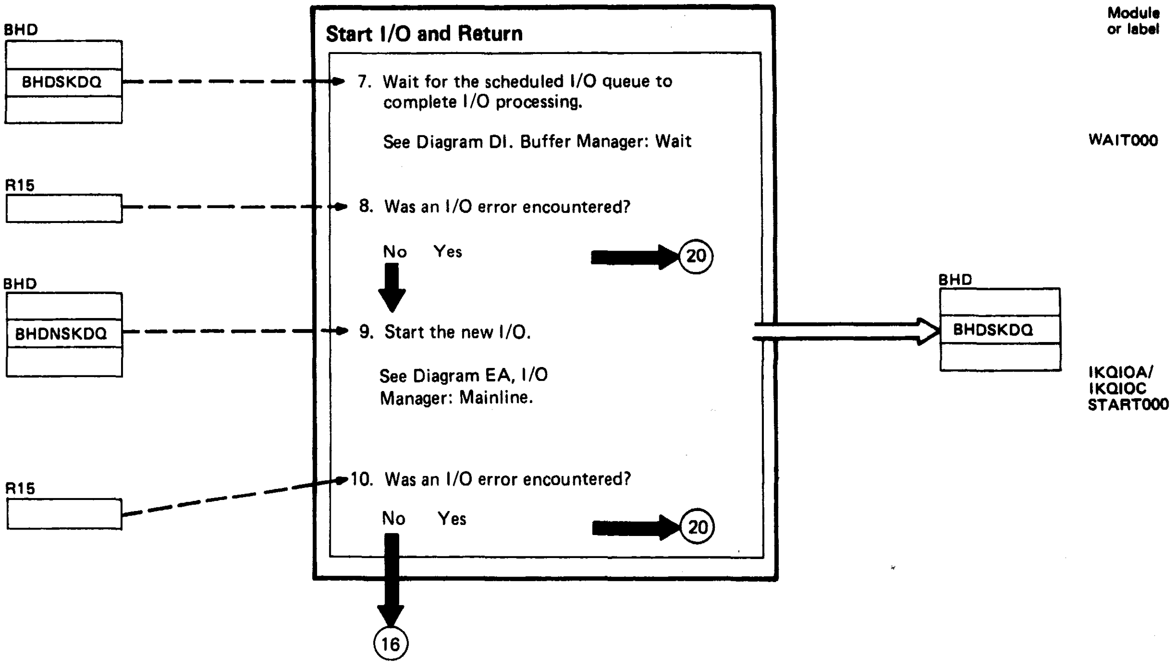
   8 – Enqueue the BCB.

   12 – Wait for previously started I/O to complete.

6. The threshold test consists of comparing the count of I/O operations to do (number of BCBs on the nonscheduled queue) to a fixed threshold value. For LSR the value is zero. For non-shared resources, the value is either:

   - The total number of BCBs owned by this PLH, if that number is less than 4.
   - One-half the total number of BCBs owned by this PLH (rounded high if an odd number), if that number is greater than or equal to 4.

Diagram DE3. Buffer Manager: Do I/O



Start I/O and Return

7. Wait for the scheduled I/O queue to complete I/O processing.

   See Diagram DI. Buffer Manager: Wait

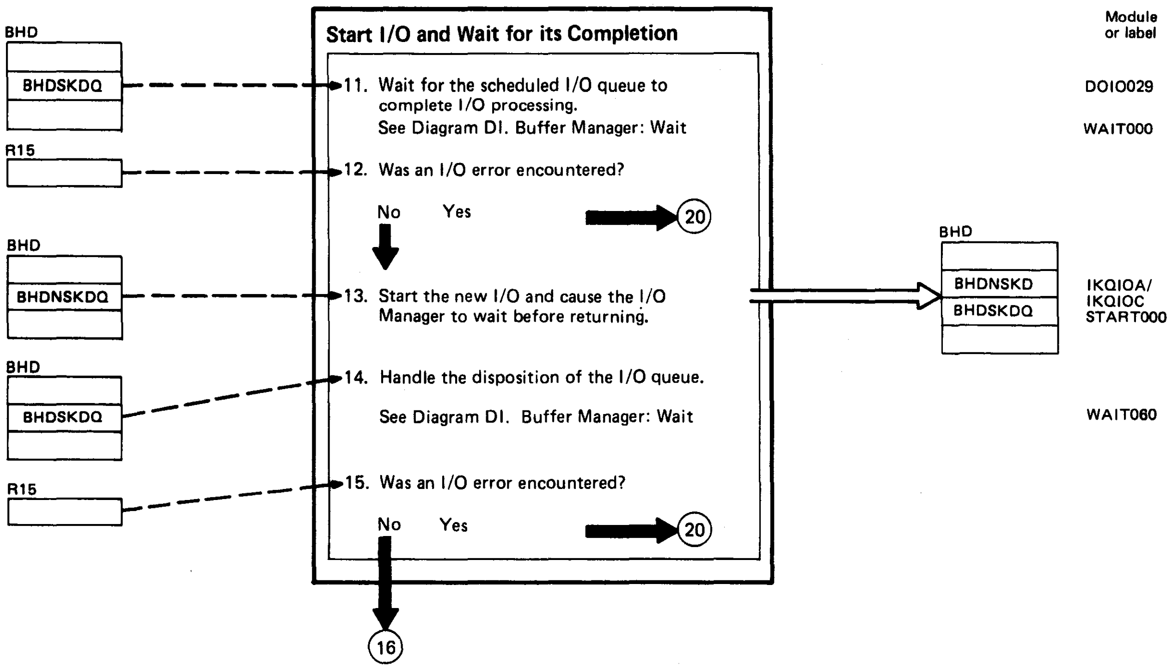
8. Was an I/O error encountered?

9. Start the new I/O.

   See Diagram EA, I/O Manager: Mainline.

10. Was an I/O error encountered?

Diagram DE4. Buffer Manager: Do I/O



Start I/O and Wait for its Completion

11. Wait for the scheduled I/O queue to complete I/O processing.
    See Diagram DI. Buffer Manager: Wait

12. Was an I/O error encountered?

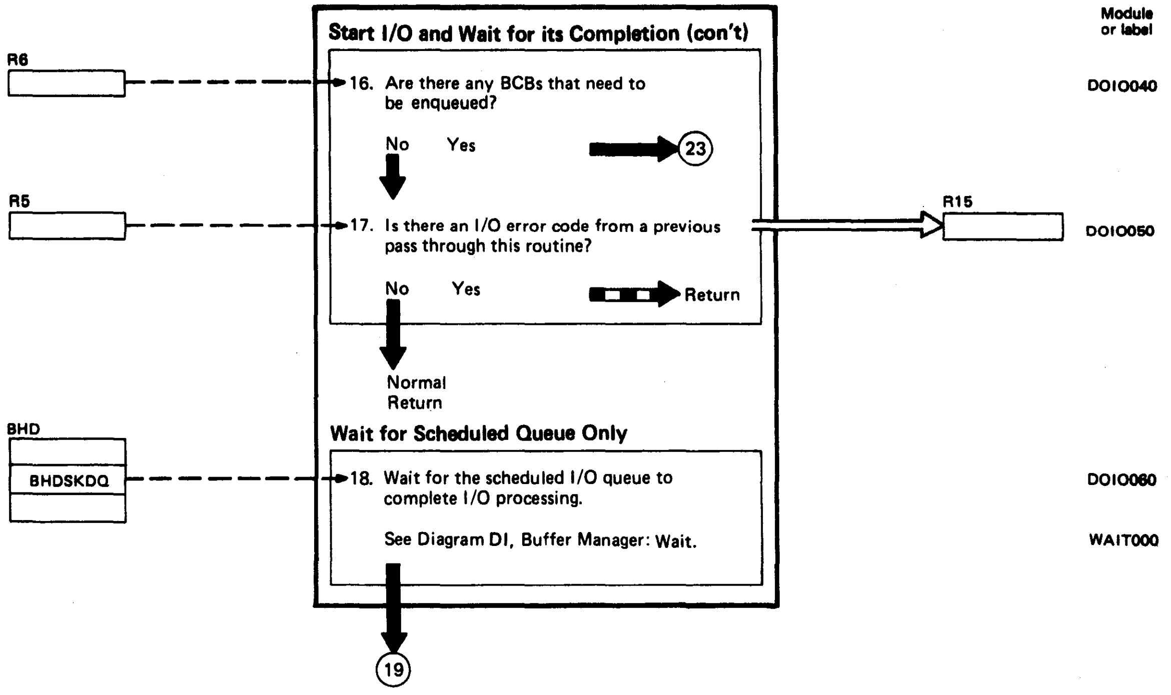
13. Start the new I/O and cause the I/O Manager to wait before returning.

14. Handle the disposition of the I/O queue.

    See Diagram DI. Buffer Manager: Wait

15. Was an I/O error encountered?

Diagram DE5. Buffer Manager: Do I/O


Module or label
Start I/O and Wait for its Completion (con't)
R6
16. Are there any BCBs that need to be enqueued?
No
Yes
23
DOIO040
R5
17. Is there an I/O error code from a previous pass through this routine?
No
Yes
Return
R15
DOIO050
Normal Return
Wait for Scheduled Queue Only
BHD
BHDSKDQ
18. Wait for the scheduled I/O queue to complete I/O processing.
See Diagram DI, Buffer Manager: Wait.
DOIO060
WAIT000
19

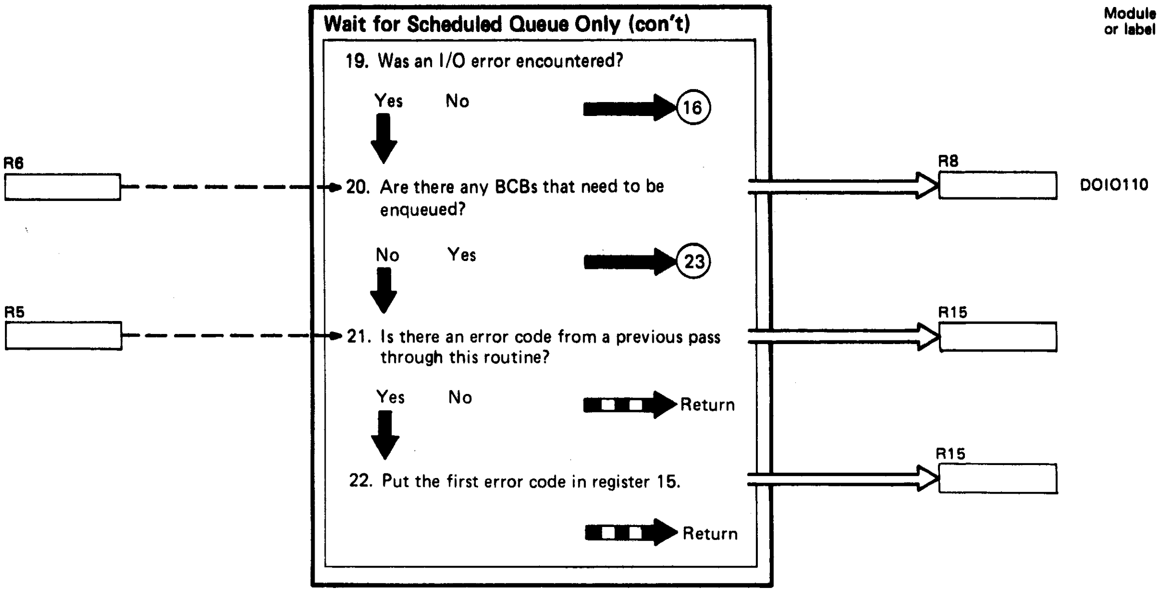
Notes for Diagram DE5

16. Register 6 is tested to see if there are any BCBs that were not enqueued due to a device class conflict, or if the Buffer Manager wait routine (see Diagram DI) caused register 6 to point to any BCBs that were "ignored" by the I/O Manager due to an I/O error on another BCB.

17. If this is a second pass because BCBs were held back (in register 6) from being enqueued, then an error may have occurred on the first pass. If such an error did occur, its code was saved in register 5 during the second pass.

    This code is now restored to register 15, and the error return to the caller is taken.

Diagram DE6. Buffer Manager: Do I/O

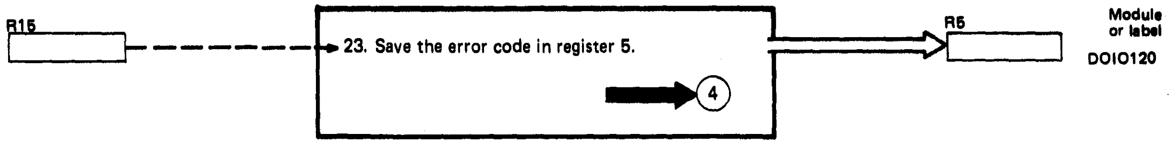

Wait for Scheduled Queue Only (con't)
Module or label
19. Was an I/O error encountered?
Yes
No
16
R6
20. Are there any BCBs that need to be enqueued?
R8
DOIO110
No
Yes
23
R5
21. Is there an error code from a previous pass through this routine?
R15
Yes
No
Return
R15
22. Put the first error code in register 15.
Return

Notes for Diagram DE6

20. See note for step 16.

21.-22. See note for step 17. In case of multiple errors (errors on both the first pass and the second pass), the first error code is returned to the caller.

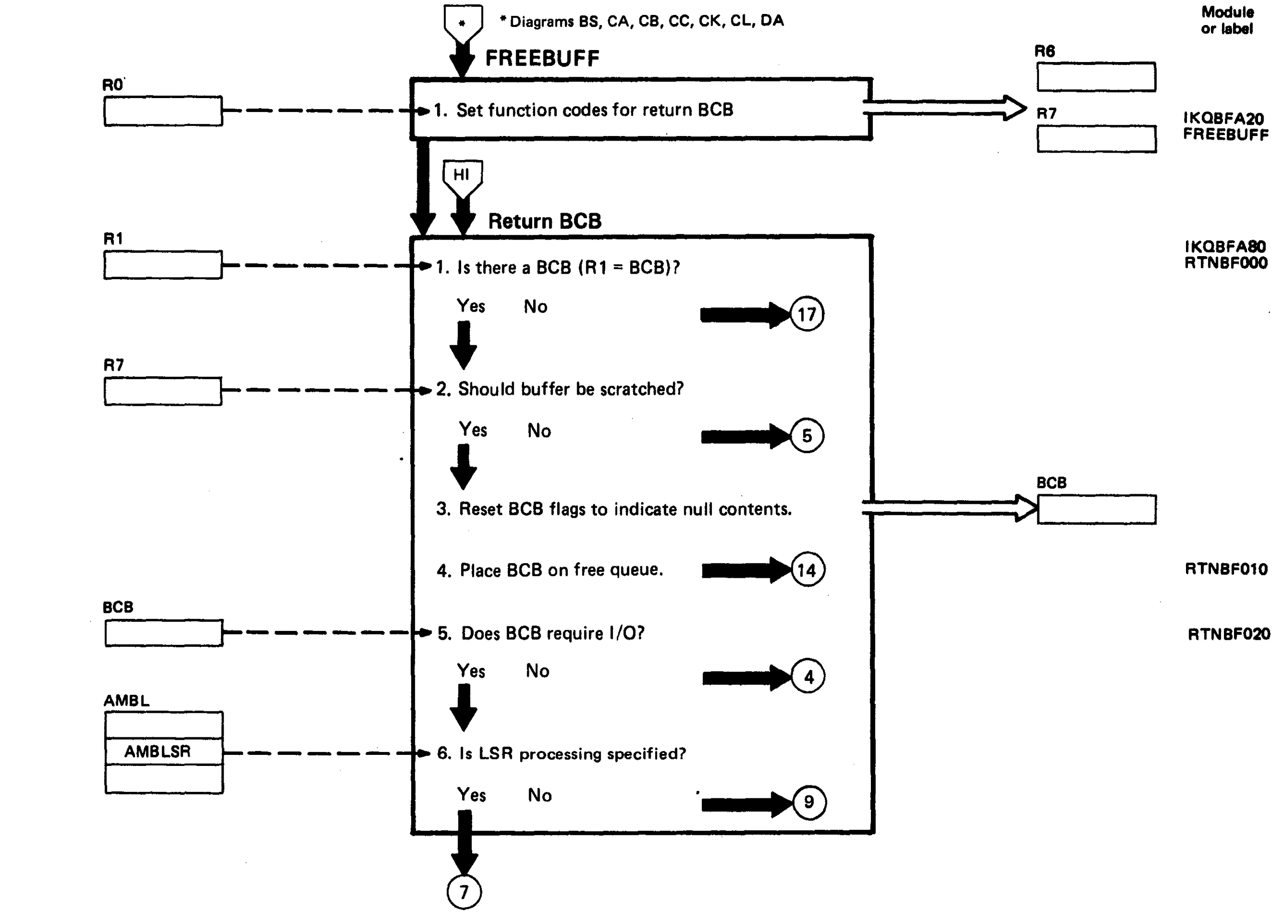
Diagram DE7. Buffer Manager: Do I/O



Notes for Diagram DE7

23. See note for step 17.

Diagram DF1. Buffer Manager: FREEBUFF and Return BCB



* Diagrams BS, CA, CB, CC, CK, CL, DA
FREEBUFF
Module or label
R0
R6
R7
1. Set function codes for return BCB
IKQBFA20 FREEBUFF
HI
Return BCB
R1
IKQBFA80 RTNBF000
1. Is there a BCB (R1 = BCB)?
Yes No
17
R7
2. Should buffer be scratched?
Yes No
5
3. Reset BCB flags to indicate null contents.
BCB
4. Place BCB on free queue.
14
RTNBF010
BCB
5. Does BCB require I/O?
RTNBF020
Yes No
4
AMBL
AMBLSR
6. Is LSR processing specified?
Yes No
9
7

Notes for Diagram DF1

1. A BCB address is passed in R1. If this address is zero, FREEBUF must wait on all I/O. (For all references to the Do I/O routine and Do I/O function codes, see Diagram DE.) If this routine is entered at IKQBFA20, the function code is passed in R0 and moved to R7, and R6 is set to zero before proceding.

   Regardless of entry point, R7 contains one of the following function codes:

   negative - BCB is to be scratched.

   zero - if I/O flags are set on in the BCB, go to the Do I/O routine to start I/O but not wait.

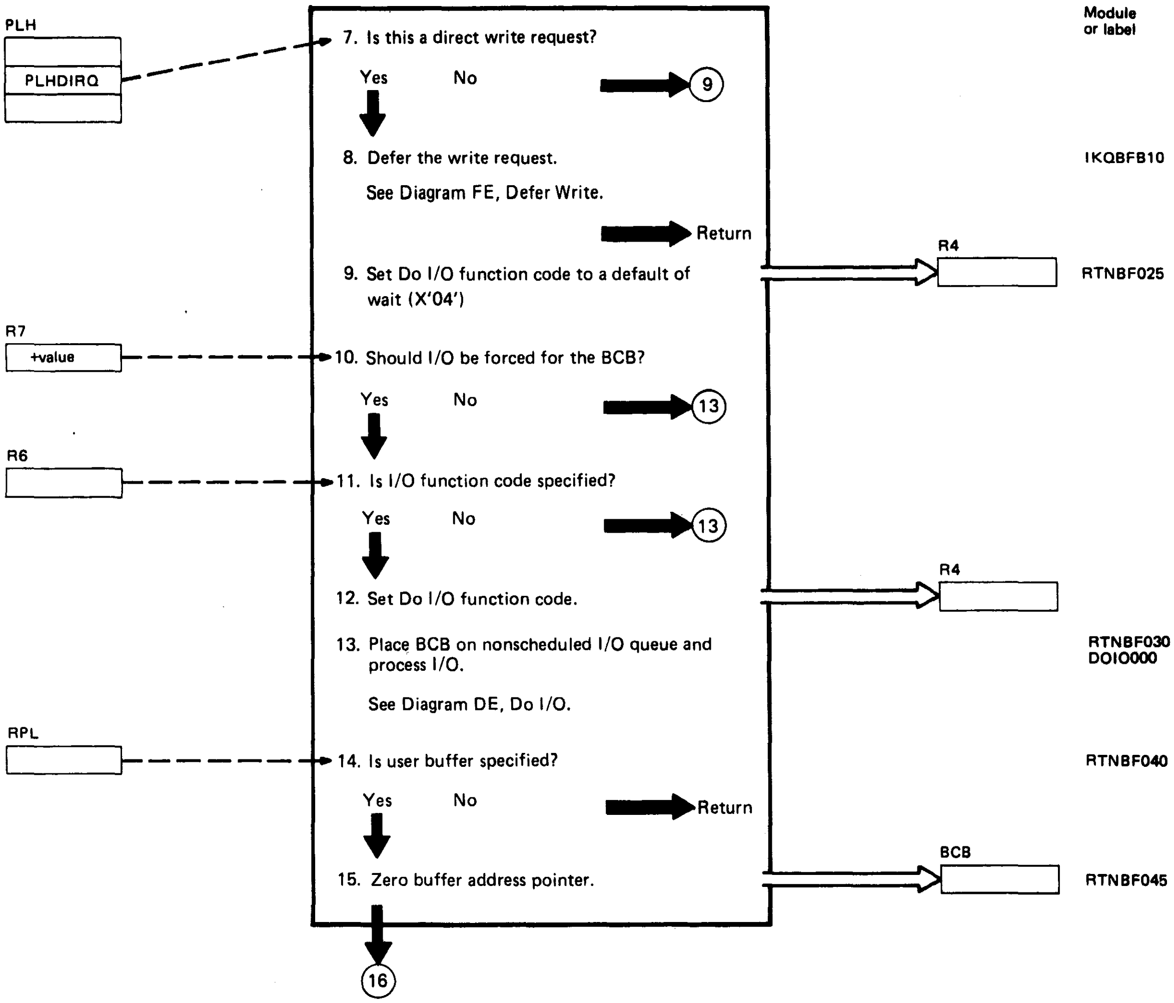
   positive - if I/O flags are set on in the BCB, go to the Do I/O routine to start I/O and wait.

   R6 may also contain a code:

   zero - Use R7 function codes as described above.

   non-zero - Override R7 function code if it was zero or positive. The override consists of decrementing by 1 the code in R6, and passing the result to the Do I/O routine as its function code.

Diagram DF2. Buffer Manager: FREEBUFF and Return BCB



7. Is this a direct write request?

8. Defer the write request.

   See Diagram FE, Defer Write.

9. Set Do I/O function code to a default of wait (X'04')

10. Should I/O be forced for the BCB?

11. Is I/O function code specified?

12. Set Do I/O function code.

13. Place BCB on nonscheduled I/O queue and process I/O.

    See Diagram DE, Do I/O.

14. Is user buffer specified?

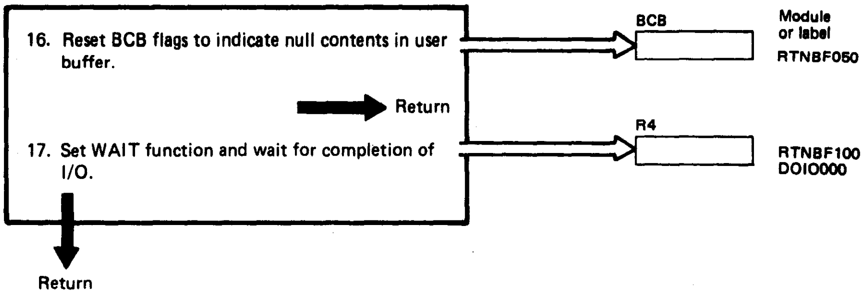
15. Zero buffer address pointer.

Notes for Diagram DF2

10. I/O is forced if the BCB is for a catalog, or if a share option 4 data set is being processed.

12. The I/O function code is set to X'04' (start all I/O) if R7 contains zero; the code is set to X'00' (wait for all I/O to complete) if R7 is positive.

Diagram DF3. Buffer Manager: FREEBUFF and Return BCB



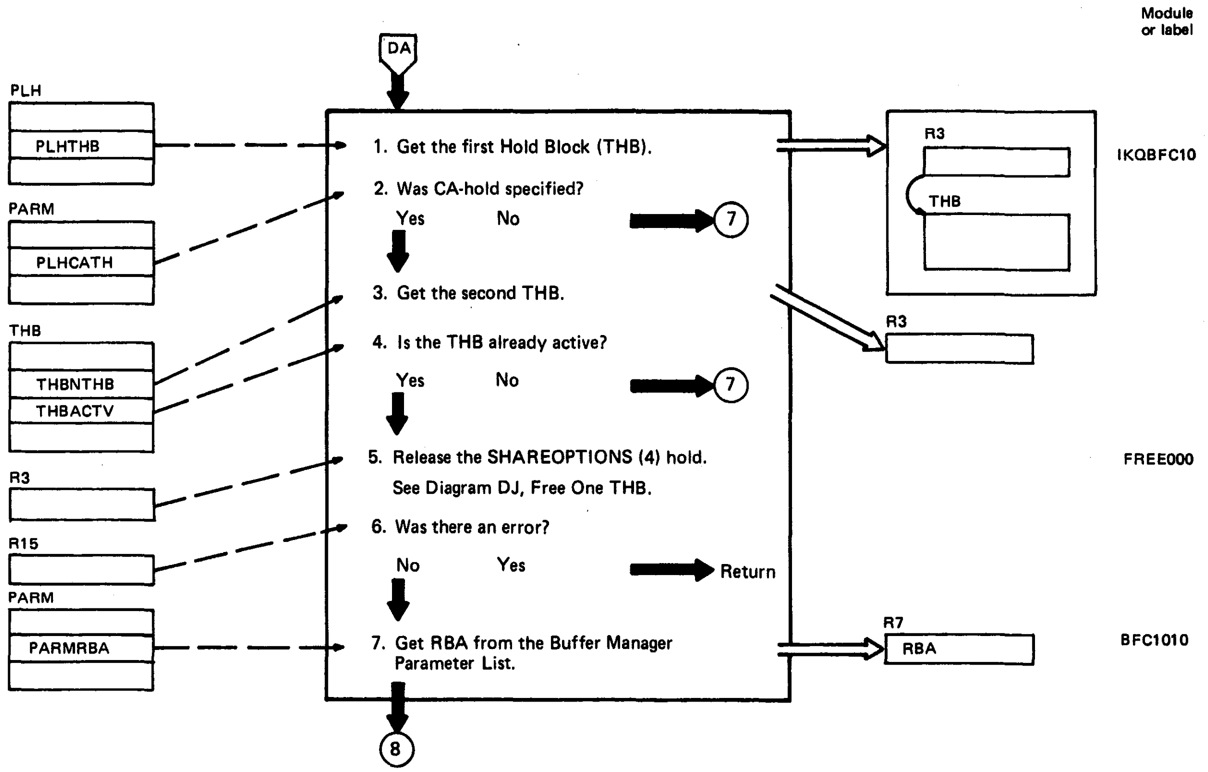
16. Reset BCB flags to indicate null contents in user buffer.

Return

17. Set WAIT function and wait for completion of I/O.

Return

Diagram DG1. Buffer Manager: Share Option 4 Hold


Module or label
DA
PLH
PLHTHB
PARM
PLHCATH
THB
THBNTHB
THBACTV
R3
R15
PARM
PARMRBA
1. Get the first Hold Block (THB).
2. Was CA-hold specified?
Yes No
7
3. Get the second THB.
4. Is the THB already active?
Yes No
7
5. Release the SHAREOPTIONS (4) hold.
See Diagram DJ, Free One THB.
6. Was there an error?
No Yes
Return
7. Get RBA from the Buffer Manager Parameter List.
8
R3
THB
R3
R7
RBA
IKQBFC10
FREE000
BFC1010

Notes for Diagram DG1

1. Share option 4 hold uses the LOCK supervisor service to lock an individual control area across all partitions of the system. A lock name is constructed that uniquely identifies the file and the control area within the file. The format of the lock name (12 bytes) is V.volid.CInumber.CAnumber where:

   V (1 byte) is the literal "V" that identifies VSAM lock names.

   volid (6 bytes) as the volid of the catalog that owns the file.

   CI number (3 bytes) is the control interval number of the catalog record that describes the data component of the file.

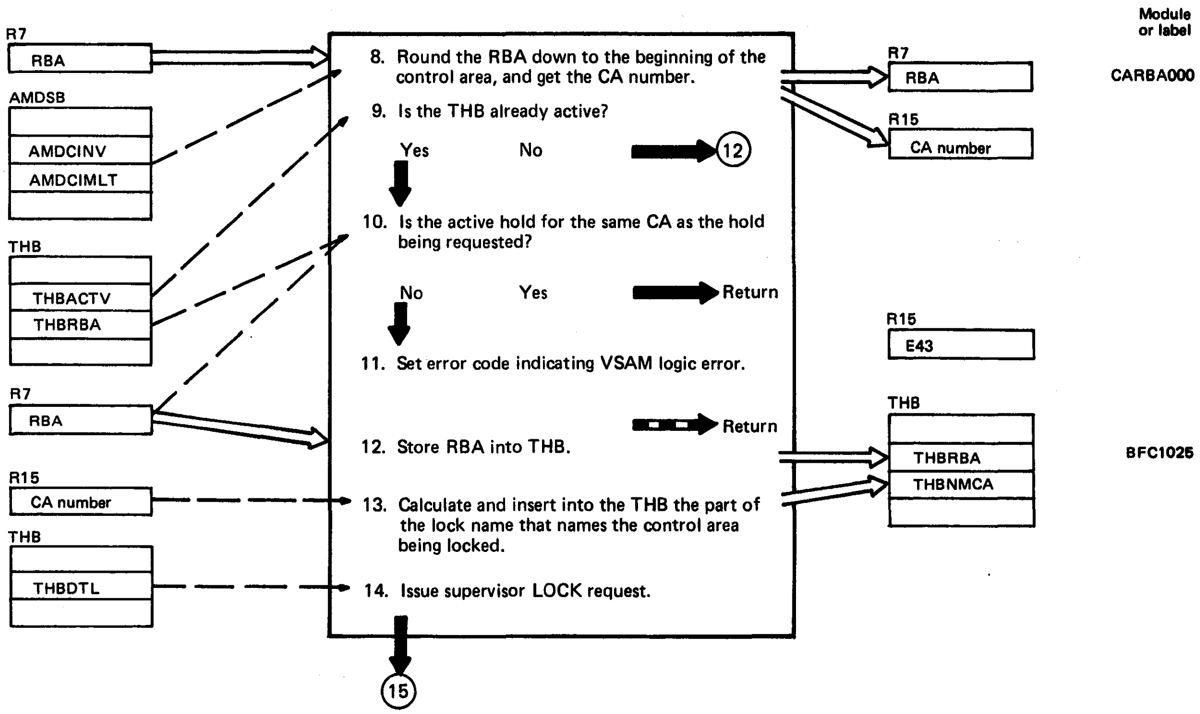
   CA number (2 bytes) is either the number of the data control area within the file that is being locked, or for keyed access to a KSDS, the control interval number of the sequence set record that points to the control area. This CA number has the constant 1024 added to it, so that 1024 lock names are reserved for uses other share option 4 locking by Record Management.

   Internally, VSAM uses one share option 4 hold block (THB) to maintain the status of each outstanding request. For each string (PLH), only one request at a time is normally allowed. There is one additional THB (referred to as the CA-split THB) that allows one additional request for each string under either of two special circumstances:

   - During a CA split – to lock the new CA.
   - When a GET KGE (key-greater-than-or-equal) fails to find the exact key requested and the search for the next higher key leads to a different CA. In this case, the hold on the original CA is retained in the CA-split THB, while the next higher CA is locked using the normal THB. This special case is for applications that do a GET KGE to find out whether a record exists, and then expect that status of existence or non-existence to not change until another operation is performed (such as a PUT to insert the record).

4. The CA-split THB is only active if it was used for the special case described in step 1, but the GET KGE was followed by a PUT UPD that caused a CA split.

Diagram DG2. Buffer Manager: Share Option 4 Hold



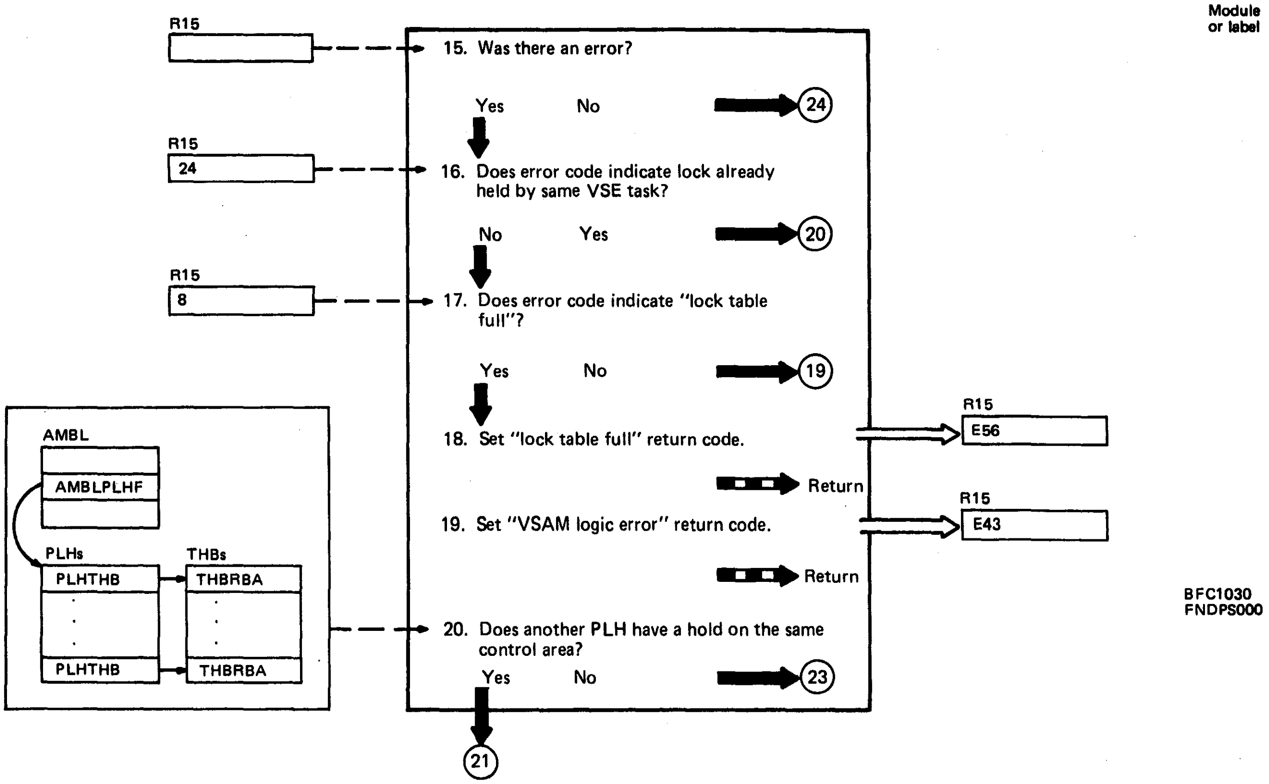
8. Round the RBA down to the beginning of the control area, and get the CA number.
9. Is the THB already active?
10. Is the active hold for the same CA as the hold being requested?
11. Set error code indicating VSAM logic error.
12. Store RBA into THB.
13. Calculate and insert into the THB the part of the lock name that names the control area being locked.
14. Issue supervisor LOCK request.

Notes for Diagram DG2

13. The part of the lock name that identifies the control area is the last two bytes of the lock name, as described in the note for step 1.

14. The PLH is released during the LOCK request. This avoids a possible deadlock with the "buffer steal" mechanism. Before the LOCK macro is issued, the bit THBNOTPR is turned off (as a potential problem determination aid). After return from the LOCK macro, the bit is turned back on.

Diagram DG3. Buffer Manager: Share Option 4 Hold



15. Was there an error?

16. Does error code indicate lock already held by same VSE task?

17. Does error code indicate "lock table full"?

18. Set "lock table full" return code.

19. Set "VSAM logic error" return code.

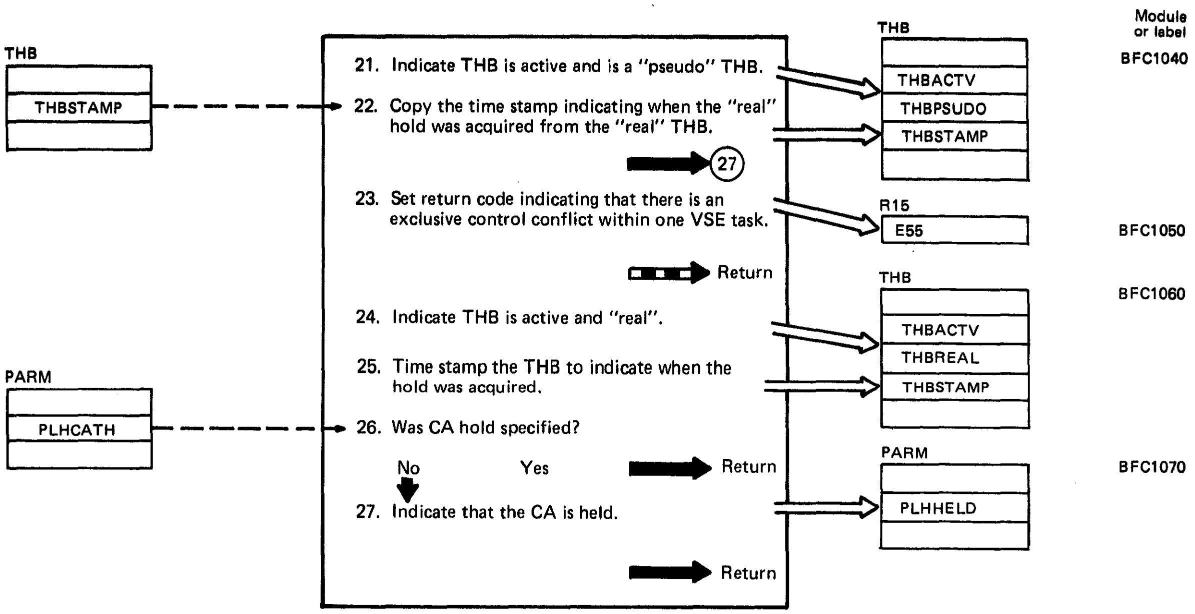
20. Does another PLH have a hold on the same control area?

Notes for Diagram DG3

20. It is necessary to determine if the other lock for the same control area is one that is already known. If the lock is in a THB of another string, the request is completed normally, but the THB is flagged as a "pseudo hold." The other string will check for such "pseudo holds" when it releases its hold.

    When one task requests a second hold (under a different PLH) for the same control area, the hold given to the second request is referred to as a "pseudo hold."

Diagram DG4. Buffer Manager: Share Option 4 Hold



21. Indicate THB is active and is a "pseudo" THB.

22. Copy the time stamp indicating when the "real" hold was acquired from the "real" THB.

23. Set return code indicating that there is an exclusive control conflict within one VSE task.

24. Indicate THB is active and "real".

25. Time stamp the THB to indicate when the hold was acquired.

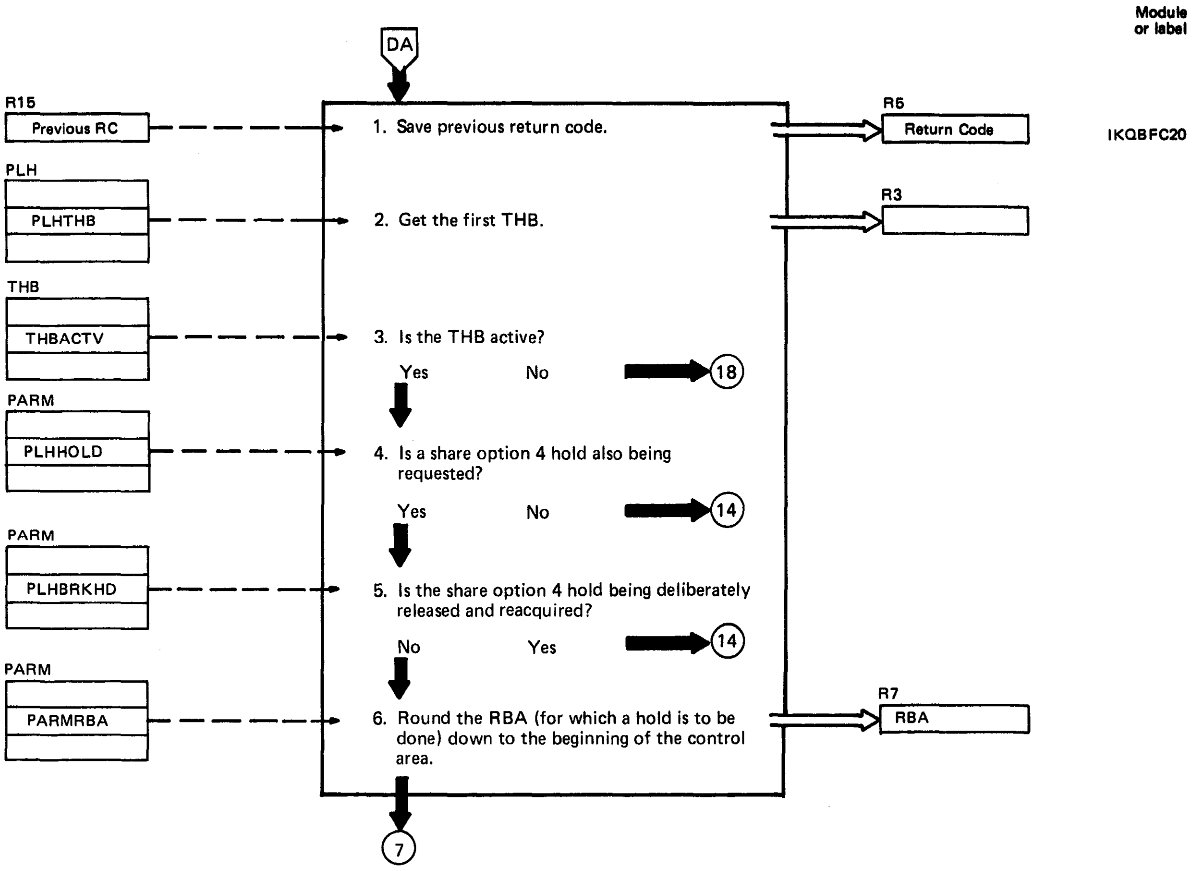
26. Was CA hold specified?

27. Indicate that the CA is held.

Notes for Diagram DG4

25. The time-of-day clock is stored into an 8-byte field in the THB.

Diagram DH1. Buffer Manager: Share Option 4 Free

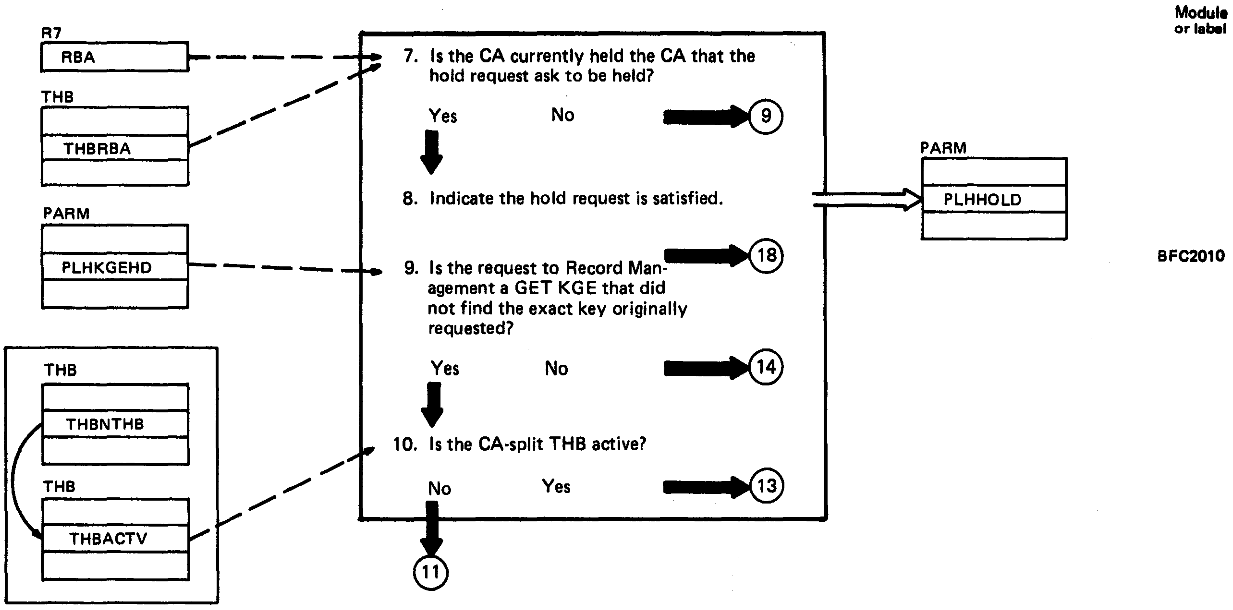


Module or label
DA
R15
Previous RC
PLH
PLHTHB
THB
THBACTV
PARM
PLHHOLD
PARM
PLHBRKHD
PARM
PARMRBA
1. Save previous return code.
2. Get the first THB.
3. Is the THB active?
Yes
No
18
4. Is a share option 4 hold also being requested?
Yes
No
14
5. Is the share option 4 hold being deliberately released and reacquired?
No
Yes
14
6. Round the RBA (for which a hold is to be done) down to the beginning of the control area.
7
R6
Return Code
R3
R7
RBA
IKQBFC20

Notes for Diagram DH1

1. The previous contents of register 15 are saved so they will not be inadvertently destroyed. This is necessary because IKQBFC20 may be called to release outstanding share option 4 holds for IKQBFA error condition processing.
2. The first THB is for normal processing; the second is the CA-split hold THB.
4. The primary reason for testing if a hold is being done at the same time as this free is so that no exclusive control hole will be caused if the same CA is specified for both free and hold. This type of free and hold occurs when IKQLCD has been called by IKQMDY after IKQMDY got a minus one return code from IKQCIS.
5. If the share option 4 hold has been held for an excessively long time, then the exclusive control is temporarily interrupted. In this case, the PLHBRKHD flag will have been set by IKQLCN, IKQGNX, or IKQGPT.

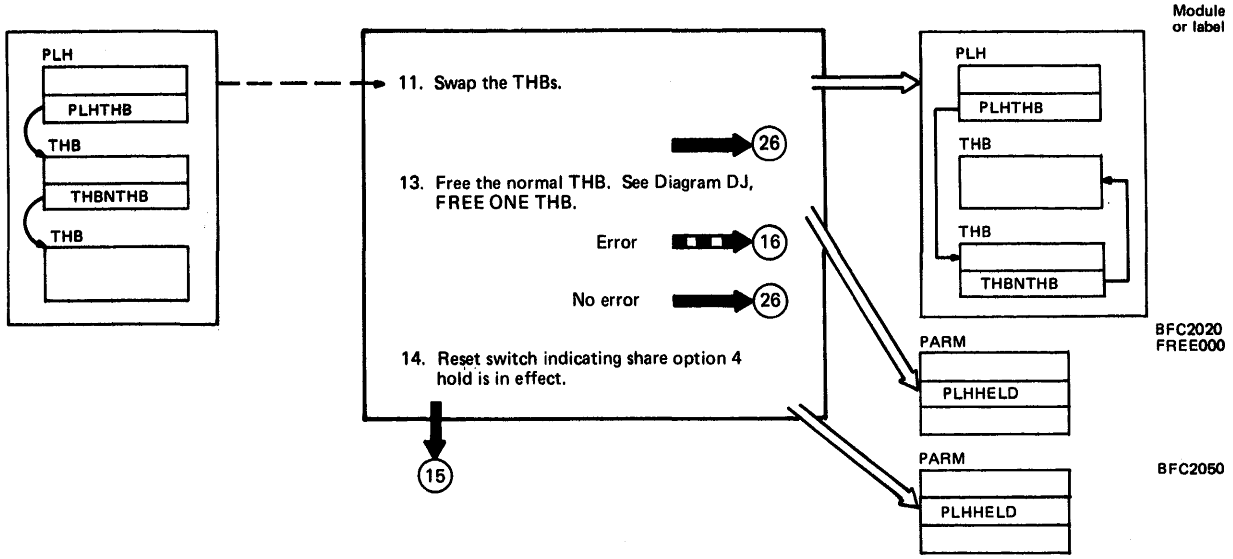
Diagram DH2. Buffer Manager: Share Option 4 Free


Module or label
R7
RBA
THB
THBRBA
PARM
PLHKGEHD
THB
THBNTHB
THB
THBACTV
7. Is the CA currently held the CA that the hold request ask to be held?
Yes
No
9
8. Indicate the hold request is satisfied.
18
9. Is the request to Record Management a GET KGE that did not find the exact key originally requested?
Yes
No
14
10. Is the CA-split THB active?
No
Yes
13
11
PARM
PLHHOLD
BFC2010

Notes for Diagram DH2

9. This is a test for the special case described in the note for step 1 of Diagram DG1.
10. The only way in which the CA-split THB could be active is if the GET KGE search for the next higher key had proceeded to the next CA and that CA has been found completely empty; the search now proceeds to a third CA.

Diagram DH3. Buffer Manager: Share Option 4 Free



11. Swap the THBs.

13. Free the normal THB. See Diagram DJ, FREE ONE THB.

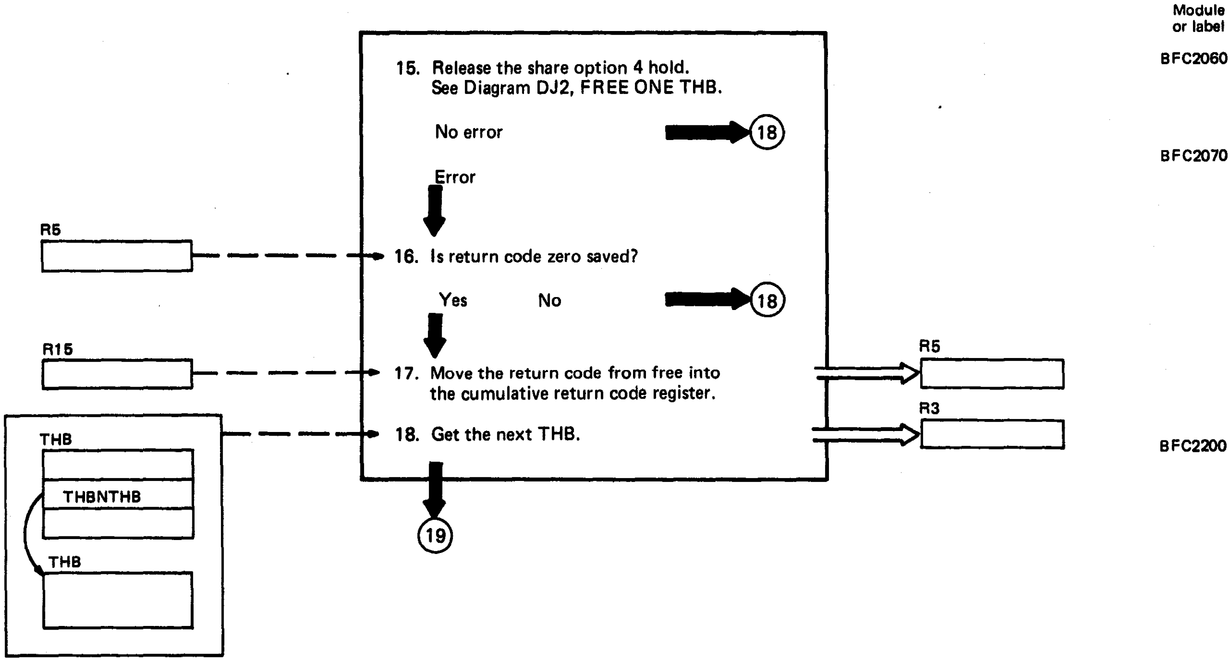
14. Reset switch indicating share option 4 hold is in effect.

Notes for Diagram DH3

11. This step retains the hold on the CA originally searched on the GET KGE in "CA-split THB", and frees the "normal THB", so it can be used for a hold on the next CA.

13. The 'next' CA is freed, so that a hold can be done on the third CA.

Diagram DH4. Buffer Manager: Share Option 4 Free



15. Release the share option 4 hold. See Diagram DJ2, FREE ONE THB.

16. Is return code zero saved?

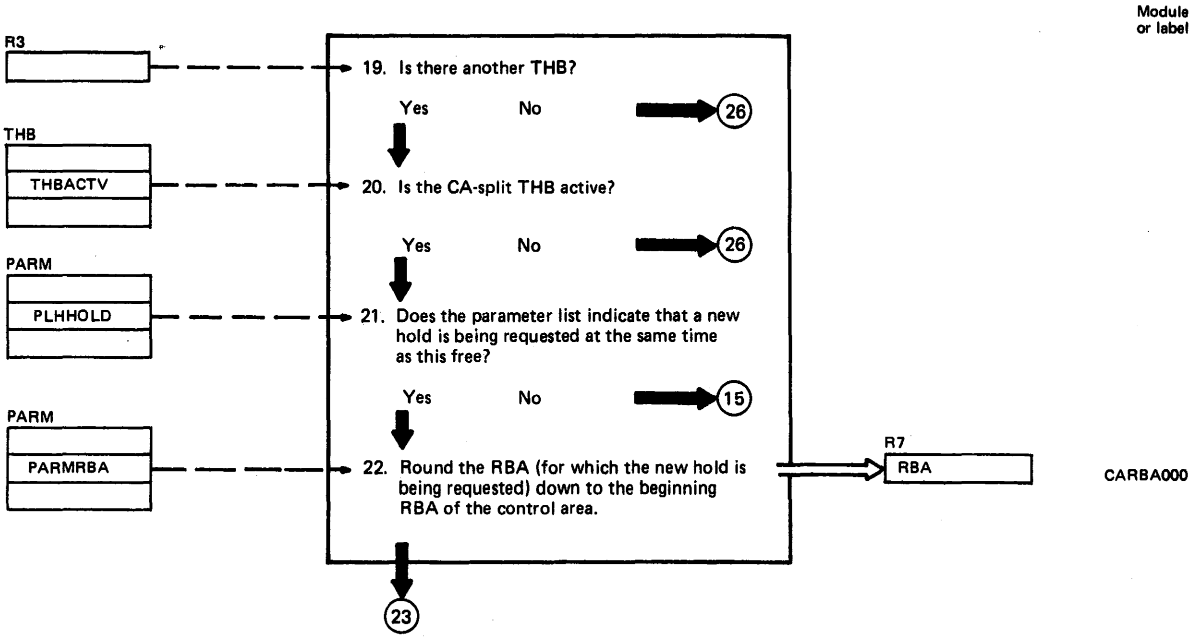
17. Move the return code from free into the cumulative return code register.

18. Get the next THB.

Notes for Diagram DH4

16.-17. The error code returned to the requester is always the first nonzero return code.

Diagram DH5. Buffer Manager: Share Option 4 Free

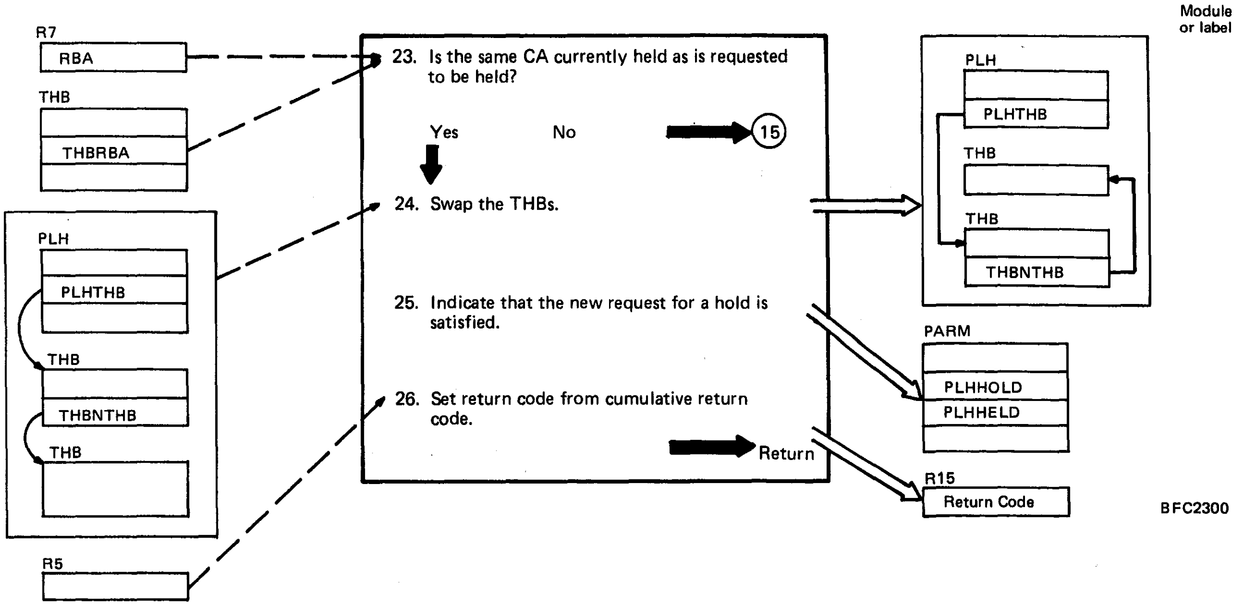

Module or label
R3
THB
THBACTV
PARM
PLHHOLD
PARM
PARMRBA
19. Is there another THB?
Yes
No
26
20. Is the CA-split THB active?
Yes
No
26
21. Does the parameter list indicate that a new hold is being requested at the same time as this free?
Yes
No
15
22. Round the RBA (for which the new hold is being requested) down to the beginning RBA of the control area.
R7
RBA
CARBA000
23

Notes for Diagram DH5

19. If there is another THB, it is because we have just processed the normal THB, and have not yet processed the CA-split THB.

21. See note for step 4.

Diagram DH6. Buffer Manager: Share Option 4 Free



23. Is the same CA currently held as is requested to be held?

Yes No ⑮

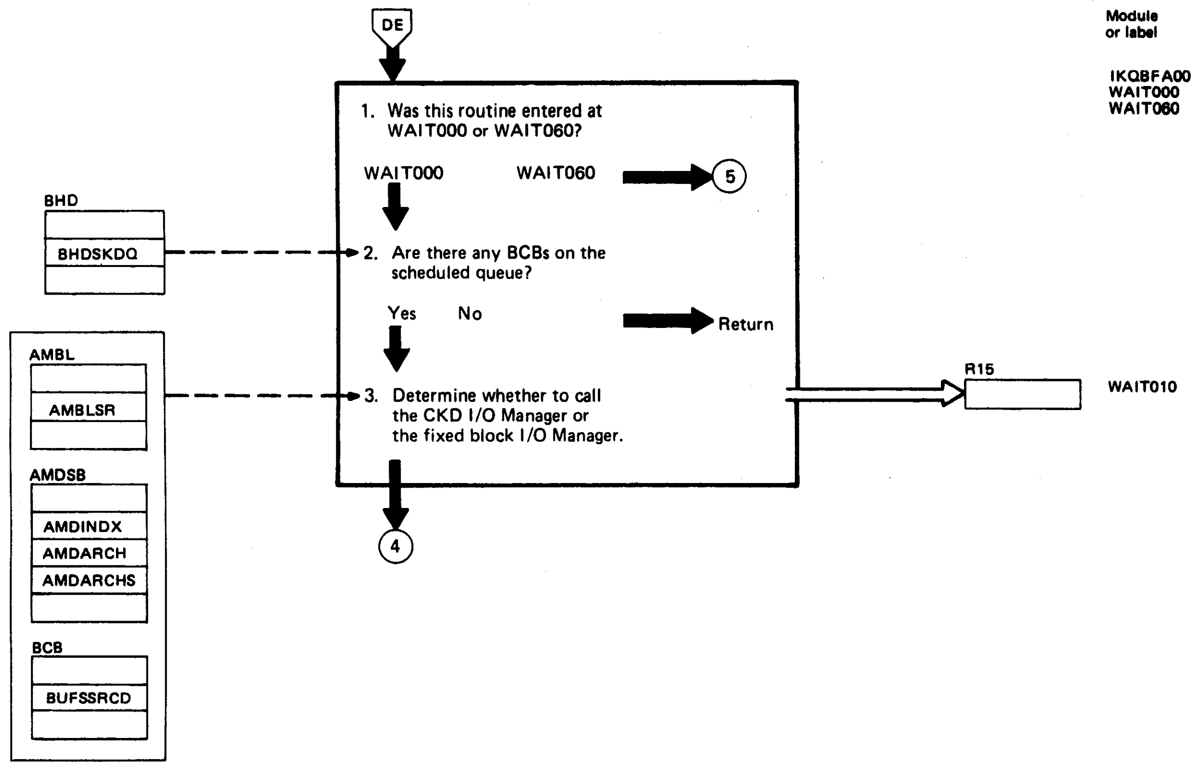
24. Swap the THBs.

25. Indicate that the new request for a hold is satisfied.

26. Set return code from cumulative return code.

Return

Diagram DI1. Buffer Manager: Wait



1. Was this routine entered at WAIT000 or WAIT060?

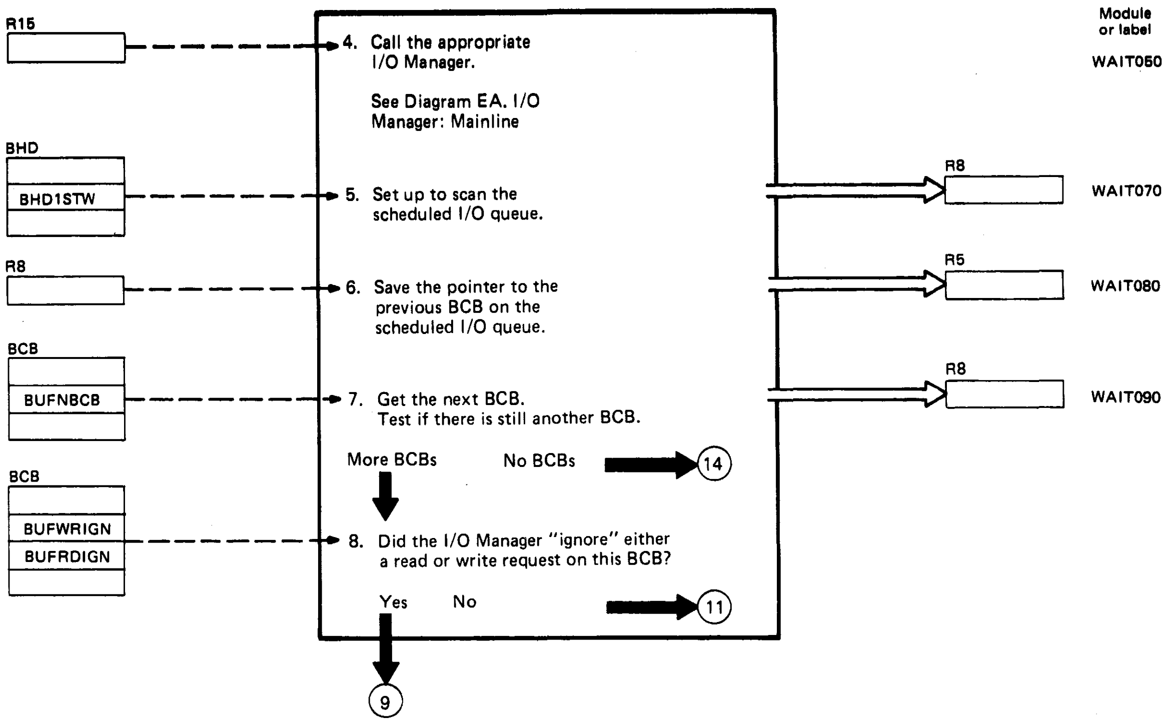
2. Are there any BCBs on the scheduled queue?

3. Determine whether to call the CKD I/O Manager or the fixed block I/O Manager.

Notes for Diagram DI1

1. This decision is not part of the code. It appears here for convenience in documentation only.
3. All requests on the scheduled queue are assumed to be for the same I/O Manager. The determination of which I/O Manager to use is based on bit AMDARCH or AMDARCHS in the AMDSB for the first BCB on the scheduled queue, according to the following conditions:
   - If the AMDSB is for a data component, AMDARCH determines which I/O Manager to use.
   - If the AMDSB is for an index and the I/O is for a high-level index CI (BUFSSRCD=0), AMDARCH determines which I/O Manager to use.
   - If the ADMSB is for an index component and the I/O is for a sequence set CI (BUFSSRCD=1), AMDARCHS determines which I/O Manager to use.

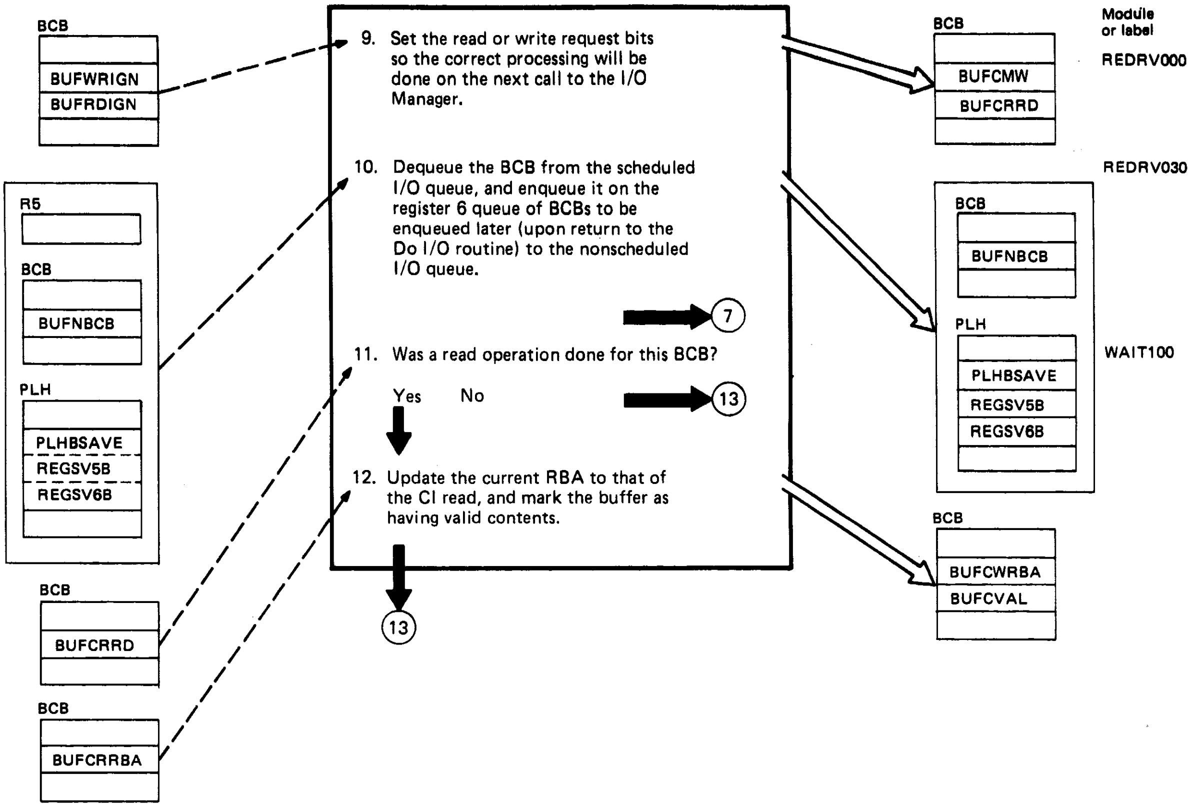
Diagram DI2. Buffer Manager: Wait


Module or label
R15
4. Call the appropriate I/O Manager.
See Diagram EA. I/O Manager: Mainline
WAIT060
BHD
BHD1STW
5. Set up to scan the scheduled I/O queue.
R8
WAIT070
R8
6. Save the pointer to the previous BCB on the scheduled I/O queue.
R5
WAIT080
BCB
BUFNBCB
7. Get the next BCB. Test if there is still another BCB.
R8
WAIT090
More BCBs
No BCBs
14
BCB
BUFWRIGN
BUFRDIGN
8. Did the I/O Manager "ignore" either a read or write request on this BCB?
Yes
No
11
9

Notes for Diagram DI2

5. Register 8 is initialized so that BHD1STW can be addressed as if it were the BUFNBCB field.
8. A read or write request may have been flagged by the I/O Manager as "ignored" due to an I/O error on another buffer. The Buffer Manager will retry these requests.

Diagram DI3. Buffer Manager: Wait



9. Set the read or write request bits so the correct processing will be done on the next call to the I/O Manager.

10. Dequeue the BCB from the scheduled I/O queue, and enqueue it on the register 6 queue of BCBs to be enqueued later (upon return to the Do I/O routine) to the nonscheduled I/O queue.

11. Was a read operation done for this BCB?

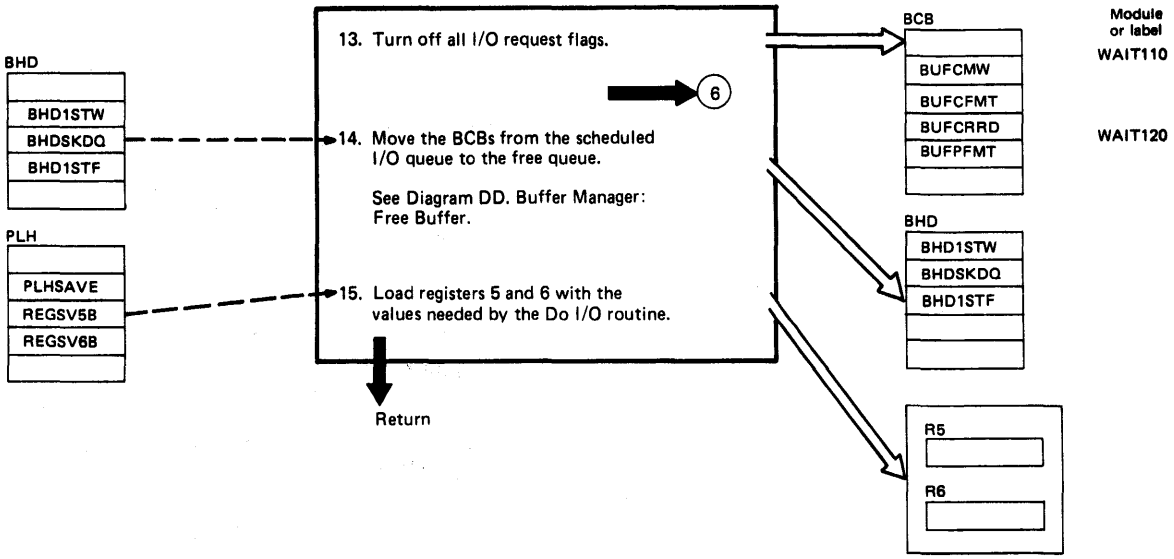
12. Update the current RBA to that of the CI read, and mark the buffer as having valid contents.

Notes for Diagram DI3

9. The read or write request bits will have been turned off if a read or write ignore was indicated.

10. At this time, the R6 value to be used is contained in the I/O Manager save area (PLHBSAVE + 52). The saved function code (PLHBSAVE + 48) for the Do I/O routine is set in this step to force starting and waiting on all I/O.

Diagram DI4. Buffer Manager: Wait



13. Turn off all I/O request flags.

14. Move the BCBs from the scheduled I/O queue to the free queue.

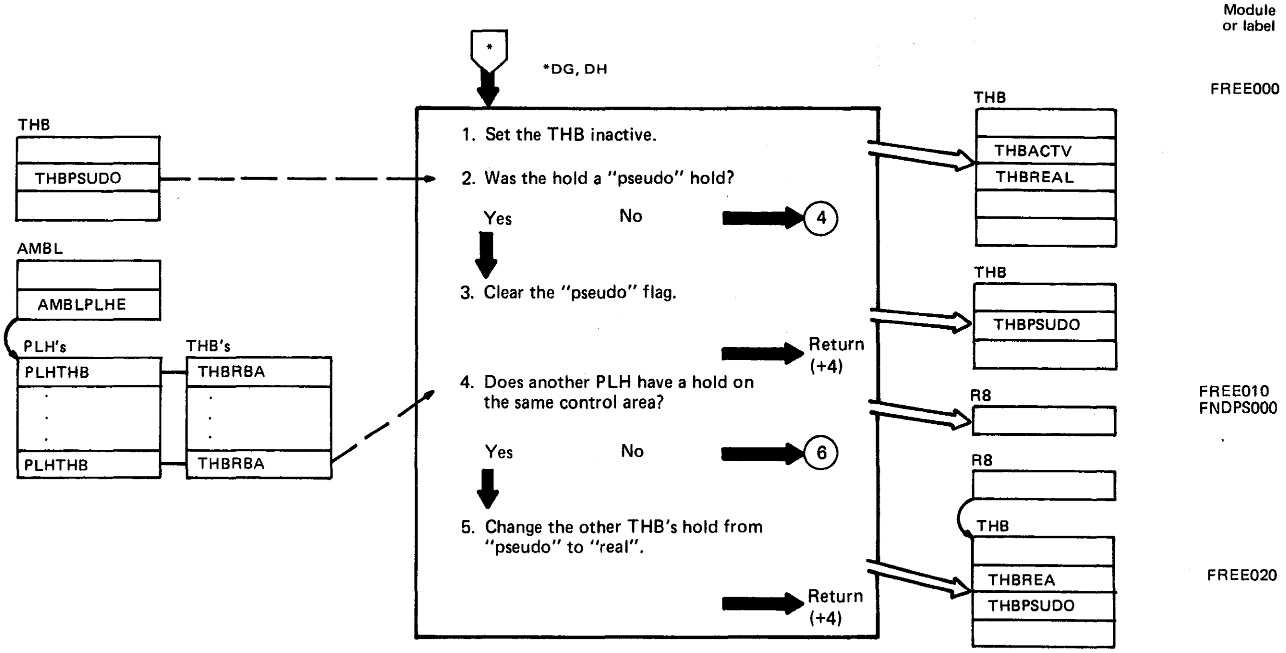
    See Diagram DD. Buffer Manager: Free Buffer.

15. Load registers 5 and 6 with the values needed by the Do I/O routine.

Notes for Diagram DI4

15. The R5 and R6 values needed by the Do I/O routine are in the I/O Manager save area in the PLH (PLHBSAVE + 48).

Diagram DJ1. Buffer Manager: Free One THB



1. Set the THB inactive.
2. Was the hold a "pseudo" hold?
3. Clear the "pseudo" flag.
4. Does another PLH have a hold on the same control area?
5. Change the other THB's hold from "pseudo" to "real".

Notes for Diagram DJ1

1. Register 3 contains a pointer to the THB to be freed upon entry to this routine.

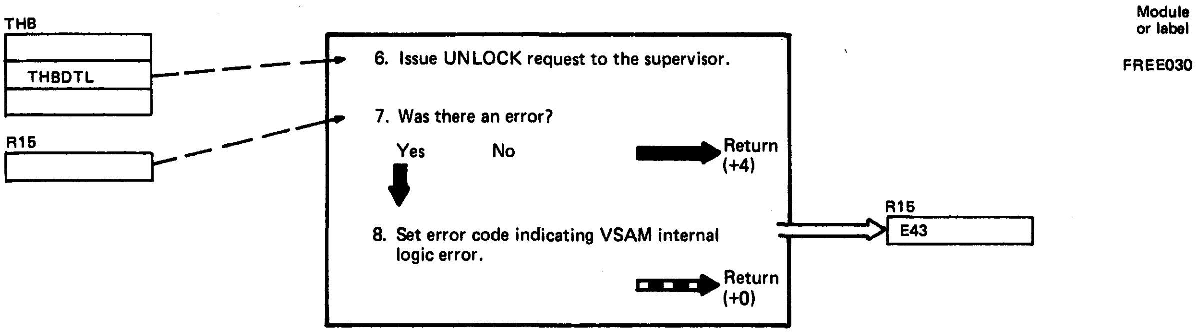
   There are two returns from FREE000. The returns are by means of a branch vector following the call. Return to the return address plus displacement:

   +0 – indicates error. An error code has been set into register 15.
   +4 – indicates no error.

2. See the note for step 21 of chart DG for a description of "pseudo" hold.

4. A search is made of THBs pointed to by other PLHs.

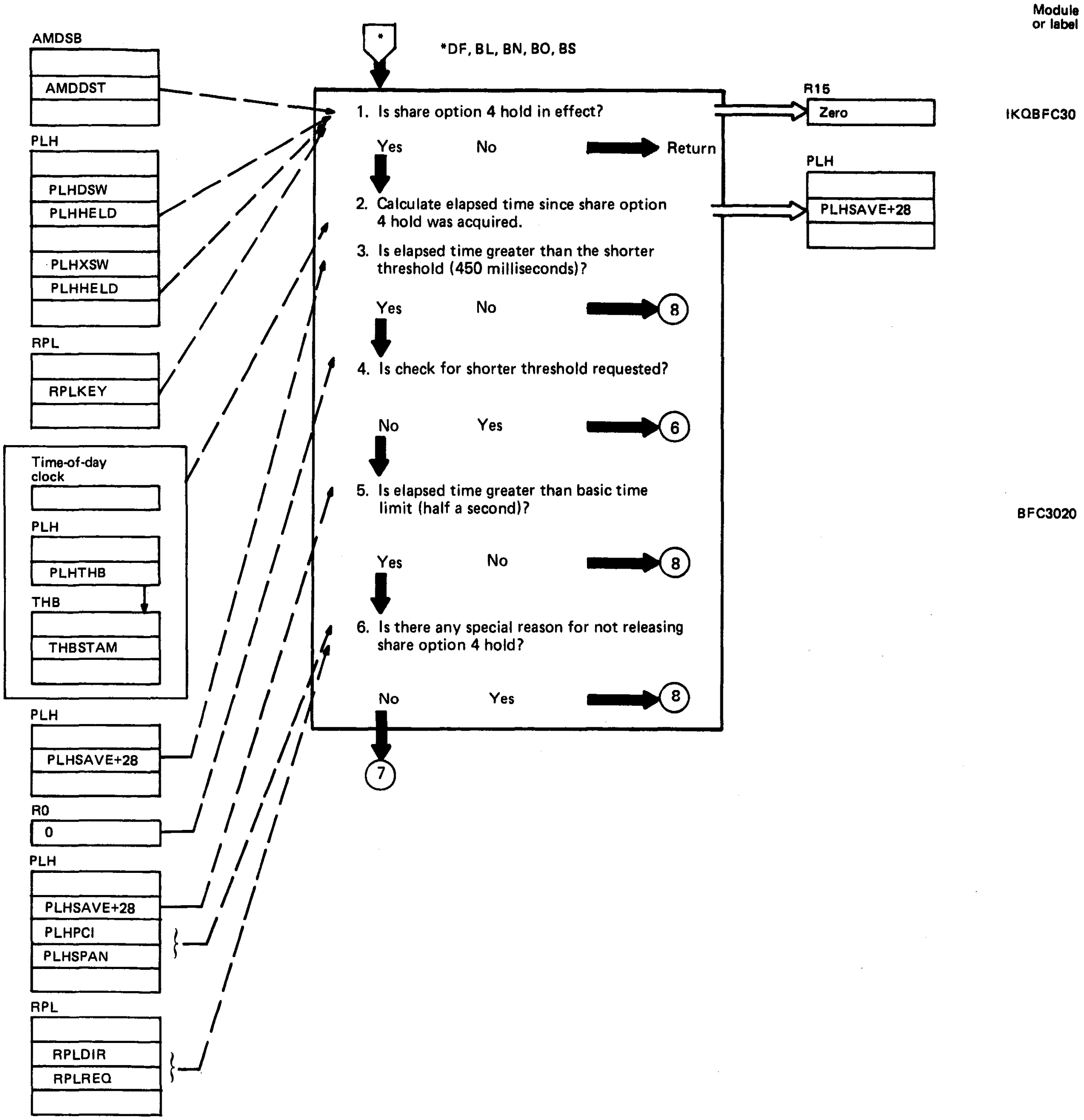
Diagram DJ2. Buffer Manager: Free One THB


Module or label
THB
THBDTL
R15
6. Issue UNLOCK request to the supervisor.
FREE030
7. Was there an error?
Yes
No
Return (+4)
R15
E43
8. Set error code indicating VSAM internal logic error.
Return (+0)

Notes for Diagram DJ2

8. The errors that are possible on UNLOCK should not occur. If any do occur, it is a logic error.

Diagram DK1. Buffer Manager: Share Option 4 Hold Time-out



1. Is share option 4 hold in effect?
2. Calculate elapsed time since share option 4 hold was acquired.
3. Is elapsed time greater than the shorter threshold (450 milliseconds)?
4. Is check for shorter threshold requested?
5. Is elapsed time greater than basic time limit (half a second)?
6. Is there any special reason for not releasing share option 4 hold?

Notes for Diagram DK1

1. This routine checks to see if the elapsed time (since the share option 4 hold was acquired) has exceeded one or two values:
   - A basic fixed limit (half a second)
   - A slightly shorter threshold (450 milliseconds).

   One of these values is selected, based on an input parameter in register 0 (R0):
   - R0 = nonzero – check against the basic time limit
   - R0 = zero – check against the shorter threshold.

   If the shorter threshold has elapsed, then the basic limit is about to expire. The check for the shorter threshold is used when a transition is being made to a new control interval, because it is cheaper to release and reaquire the hold then, rather than after the control interval has once been read.

   There are two return codes in register 15 from IKQBFC30:

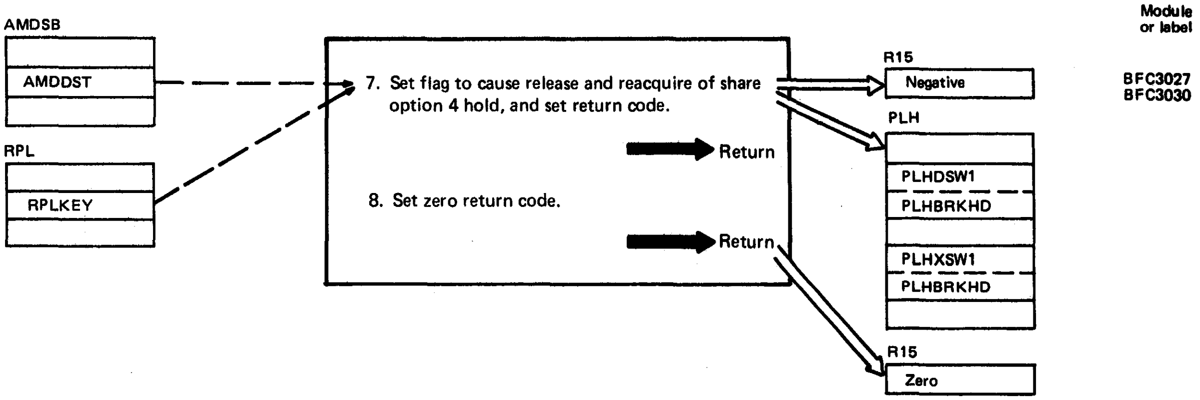
   negative – indicates the share option 4 hold should be released and reacquired

   zero – indicates the share option 4 hold should not be released.

6. The share option 4 hold should not be released under the following conditions:
   - During direct (DIR) processing.
   - During retrieval of a spanned record.
   - On a request other than a GET-UPDATE or PUT-ADD.
   - In backward processing, when the bit PLHPCI is on.

   The return code indicating that the hold should be released is not given if status has previously been set indicating that the share option 4 hold should be broken (PLHBRKHD on in PLHDSW1 or PLHXSW1).

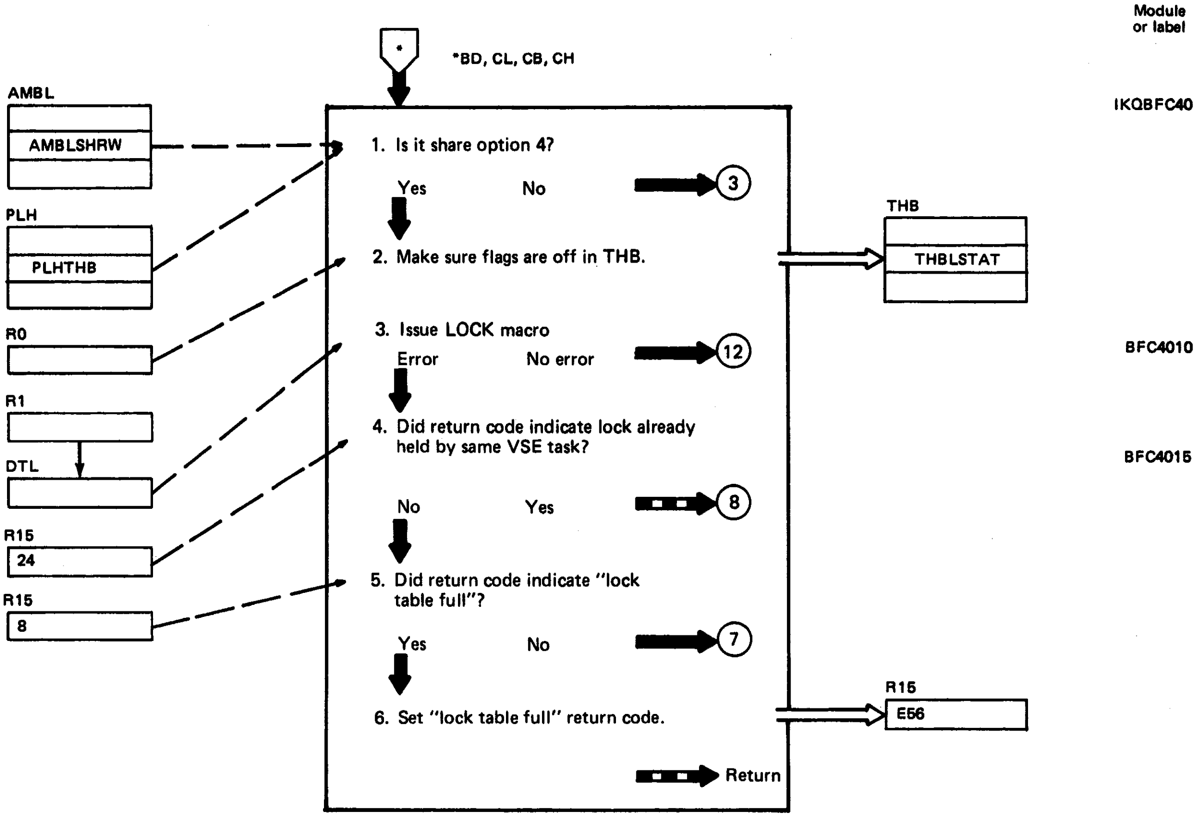
Diagram DK2. Buffer Manager: Share Option 4 Hold Time-out



Notes for Diagram DK2

7. The flag PLHBRKHD is set in byte PLHSXW1 for keyed access to a KSDS. In all other cases PLHBRKHD is set in byte PLHDSW1.

Diagram DL1. Buffer Manager: Issue LOCK Macro

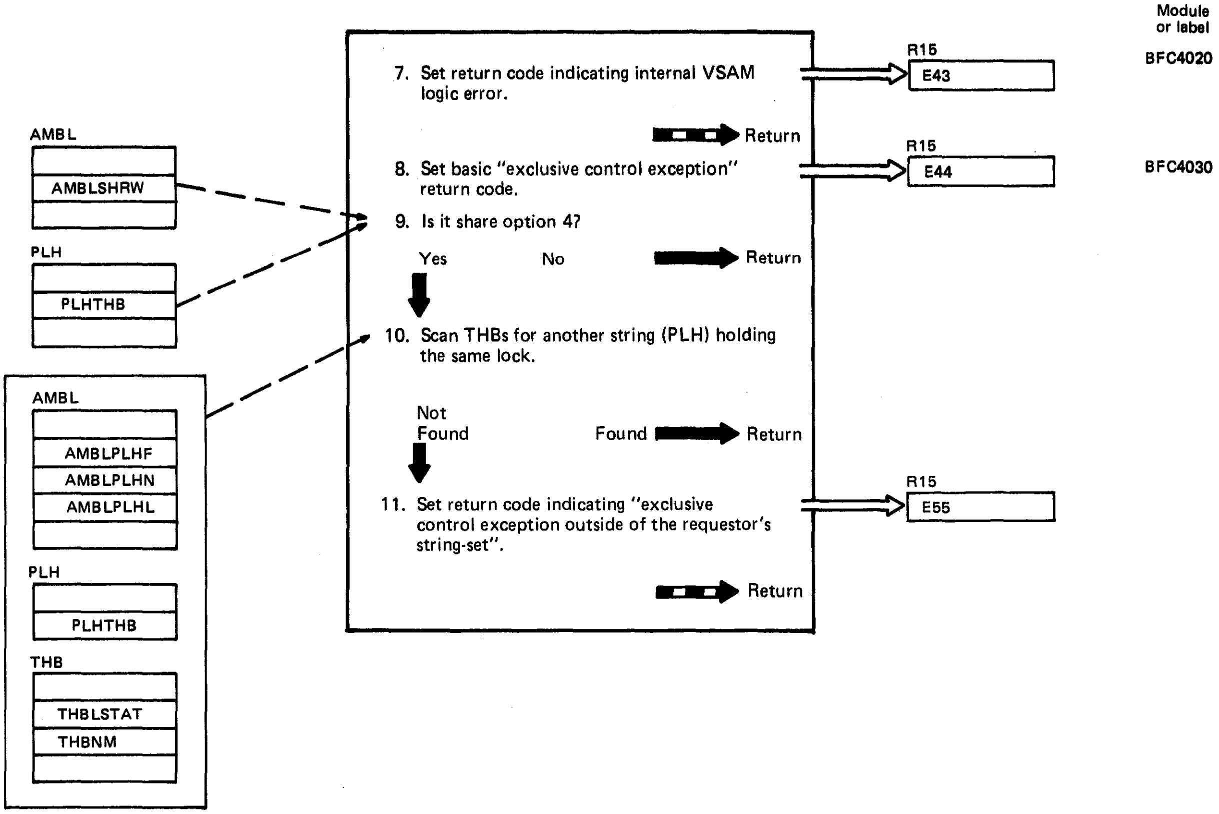


Notes for Diagram DL1.

1. IKQBFC40 is called to issue a LOCK macro and determine the Record Management internal code. It is used for locks other than the basic share option 4 control area locks.

   If IKQBFC40 is called for a share option 4 file, status is kept in the THB showing which locks have been set.
2. A problem-determination-aid bit THBNOTxx is turned off, indicating that the lock corresponding to the suffix xx is in the process of being locked.
3. After return from the LOCK macro the problem-determination-aid bit THBNOTxx is turned back on.
4. The lock would be already held by a VSAM request different from the one currently being processed. This type of locking conflict will only occur here if the second VSAM request is issued in an exit from the first (e.g. an EXCPAD exit).

Diagram DL2. Buffer Manager: Issue LOCK Macro



7. Set return code indicating internal VSAM logic error.

8. Set basic "exclusive control exception" return code.

9. Is it share option 4?

10. Scan THBs for another string (PLH) holding the same lock.

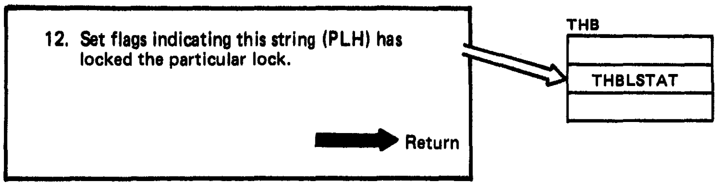
11. Set return code indicating "exclusive control exception outside of the requestor's string-set".

Notes for Diagram DL2

8.-11. For share option 4, a different return code (E44) is given if there is an "exclusive control conflict" between strings (PLHs) of the same string-set (set of PLHs), than if the "exclusive control conflict" is outside the same string-set (return code E55).

Diagram DL3. Buffer Manager: Issue LOCK Macro

Module or label



12. Set flags indicating this string (PLH) has locked the particular lock.

BFC4060

Diagram DM1. Buffer Manager: Issue UNLOCK Macro

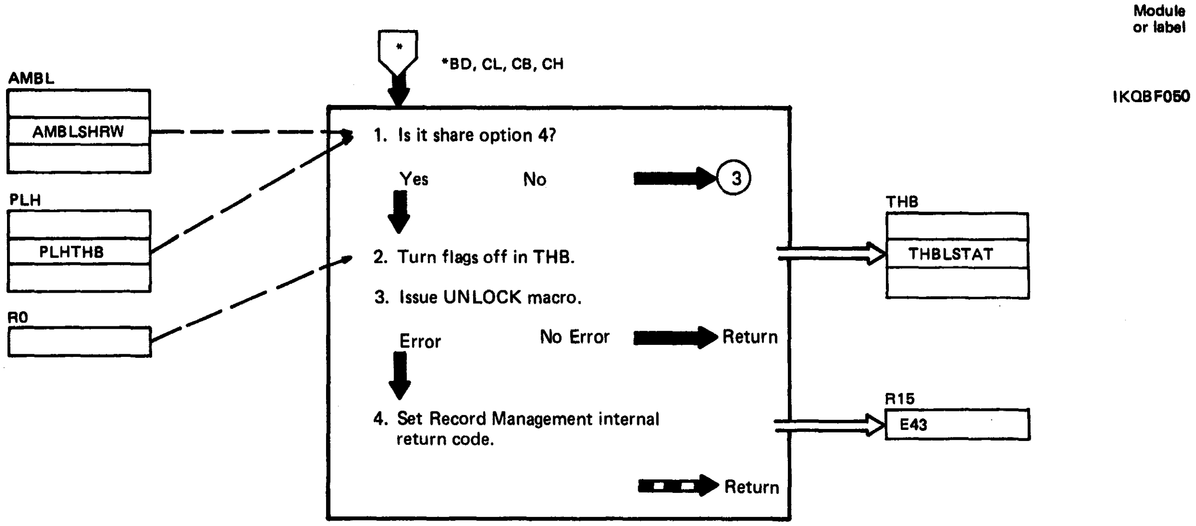
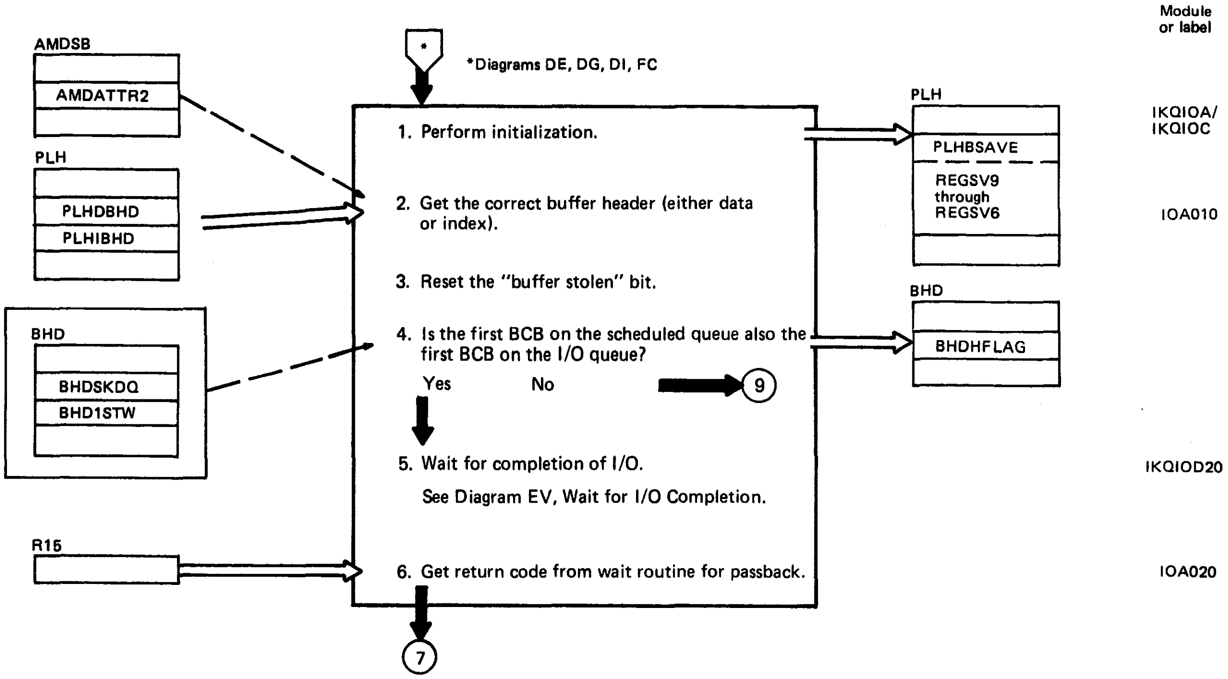


Diagram EA1. I/O Manager: Mainline



Module or label
AMDSB
AMDATTR2
PLH
PLHDBHD
PLHIBHD
BHD
BHDSKDQ
BHD1STW
R15
*
*Diagrams DE, DG, DI, FC
1. Perform initialization.
2. Get the correct buffer header (either data or index).
3. Reset the "buffer stolen" bit.
4. Is the first BCB on the scheduled queue also the first BCB on the I/O queue?
Yes
No
9
5. Wait for completion of I/O.
See Diagram EV, Wait for I/O Completion.
6. Get return code from wait routine for passback.
7
PLH
PLHBSAVE
REGSV9 through REGSV6
BHD
BHDHFLAG
IKQIOA/ IKQIOC
IOA010
IKQIOD20
IOA020

Notes for Diagram EA1

IKQIOA contains I/O Manager code for count-key-data devices. IKQIOC contains I/O Manager code for fixed block devices. Except for these device differences, the code and labels in these two modules are essentially the same.

2. When I/O Manager functions are to be performed, the PLH workarea (PLHWAREA) is set to zero because it will be used for switches, counters, etc. The address of the data or index BHD is stored in the PLH workarea; it is used for repositioning to the correct BHD throughout the program.

   NOTE: In the input and output columns of the HIPO diagrams. PLH workarea fields are shown as part of the PLH. They are listed as sub-fields (indicated by dashed lines) in the PLHWAREA field. PLHWAREA is mapped by IKQIOWKA; it is documented in the Data Areas section of this manual under the heading "I/O Workarea (IOWKA)".

3. Whenever buffers are stolen, bit BHDSTL is turned on. This bit is tested in later steps, and I/O Manager processing is terminated if no buffers are available for I/O.

4. There are four buffer queues: the scheduled queue, the nonscheduled queue, the I/O queue, and the free queue. (These queues are described in the notes for Diagram DA, Buffer Manager: GETBUFF.)

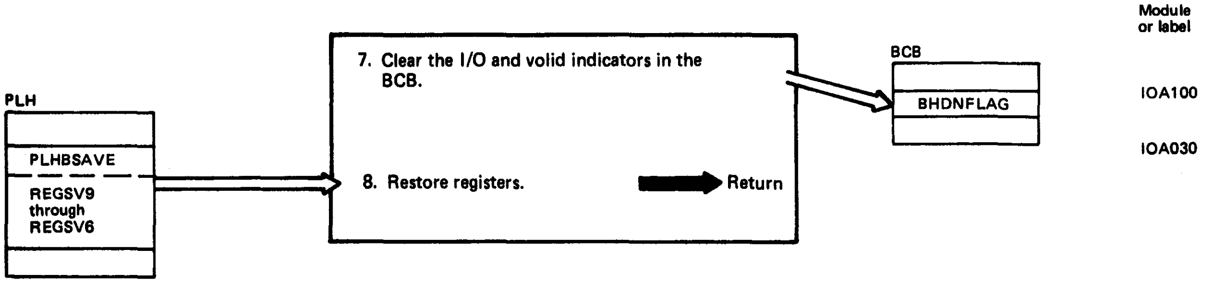
   The Buffer Manager (IKQBFA) places a pointer to a buffer queue in BHD1STW. The buffer queue requires either a WAIT only (scheduled queue) or an EXCP (nonscheduled queue), and it is now treated as an I/O queue. IKQBFA sets the BHD1STW address to that of the scheduled queue whenever the I/O operation is to be completed, or if a "buffer steal" is to be performed.

   The purge buffer routine (IKQPBF) places a pointer in BHD1STW for the buffer queue to be purged and sets the nonscheduled queue bit (BHDNSKD) in BHDHFLAG. This action causes both an EXCP and WAIT to occur.

5. The WAIT routine takes the EXCPAD exit (if specified), waits for I/O completion, and detects any I/O errors.

6.-8. These steps perform termination of the I/O Manager.

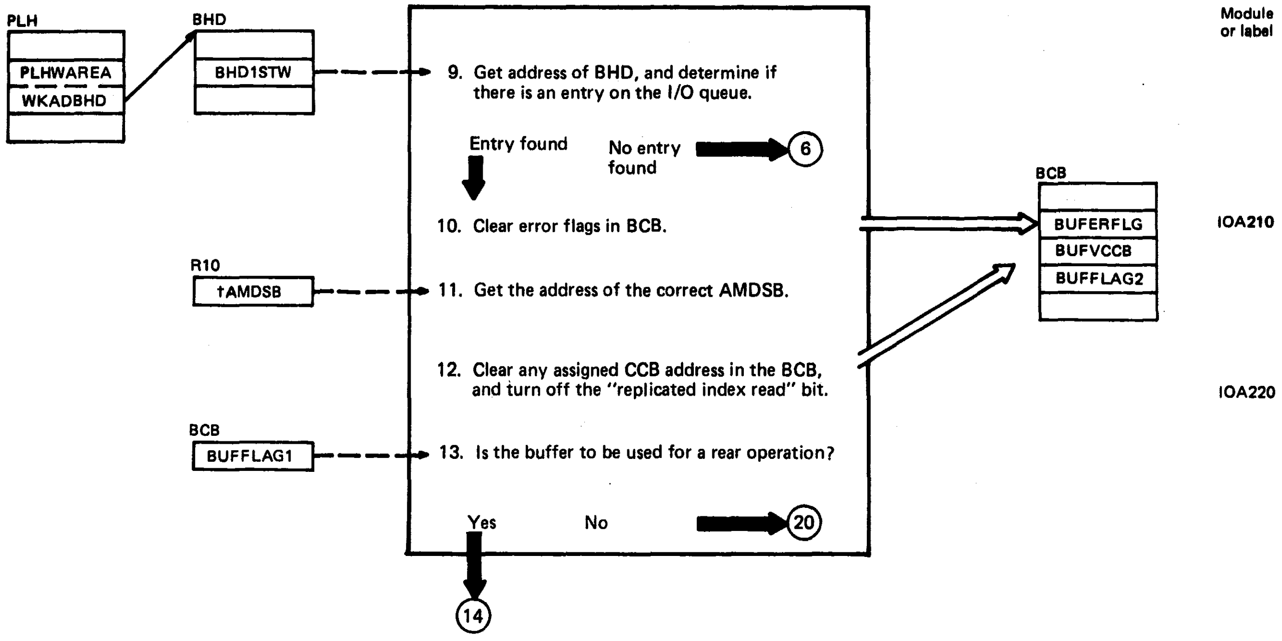
Diagram EA2. I/O Manager: Mainline



7. Clear the I/O and volid indicators in the BCB.

8. Restore registers.

Diagram EA3. I/O Manager: Mainline



9. Get address of BHD, and determine if there is an entry on the I/O queue.

10. Clear error flags in BCB.

11. Get the address of the correct AMDSB.

12. Clear any assigned CCB address in the BCB, and turn off the "replicated index read" bit.

13. Is the buffer to be used for a rear operation?

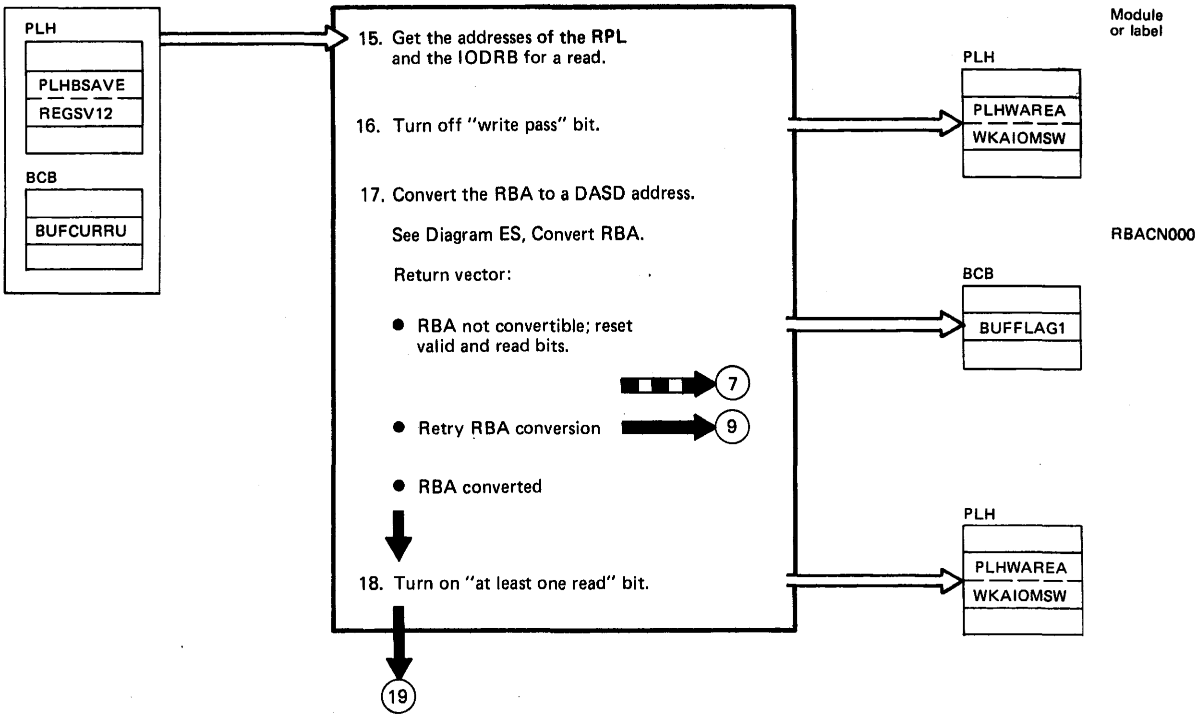
Notes for Diagram EA3

9.-12. These steps perform initialization for processing buffers on the I/O queue (requiring an EXCP).

The buffer queue must be tested for entries in the case where a "buffer steal" may have occurred. (A "buffer steal" takes all the buffers on the queue.) If there are no buffers on the queue, the I/O Manager terminates.

13.-19. These steps are performed for any buffer that requires a read I/O operation.

Diagram EA4. I/O Manager: Mainline


PLH
PLHBSAVE
REGSV12
BCB
BUFCURRU
15. Get the addresses of the RPL and the IODRB for a read.
16. Turn off "write pass" bit.
17. Convert the RBA to a DASD address.
See Diagram ES, Convert RBA.
Return vector:
● RBA not convertible; reset valid and read bits.
7
● Retry RBA conversion
9
● RBA converted
18. Turn on "at least one read" bit.
19
Module or label
PLH
PLHWAREA
WKAIOMSW
RBACN000
BCB
BUFFLAG1
PLH
PLHWAREA
WKAIOMSW

Notes for Diagram EA4

15.-17. These steps convert all RBAs to physical DASD addresses.

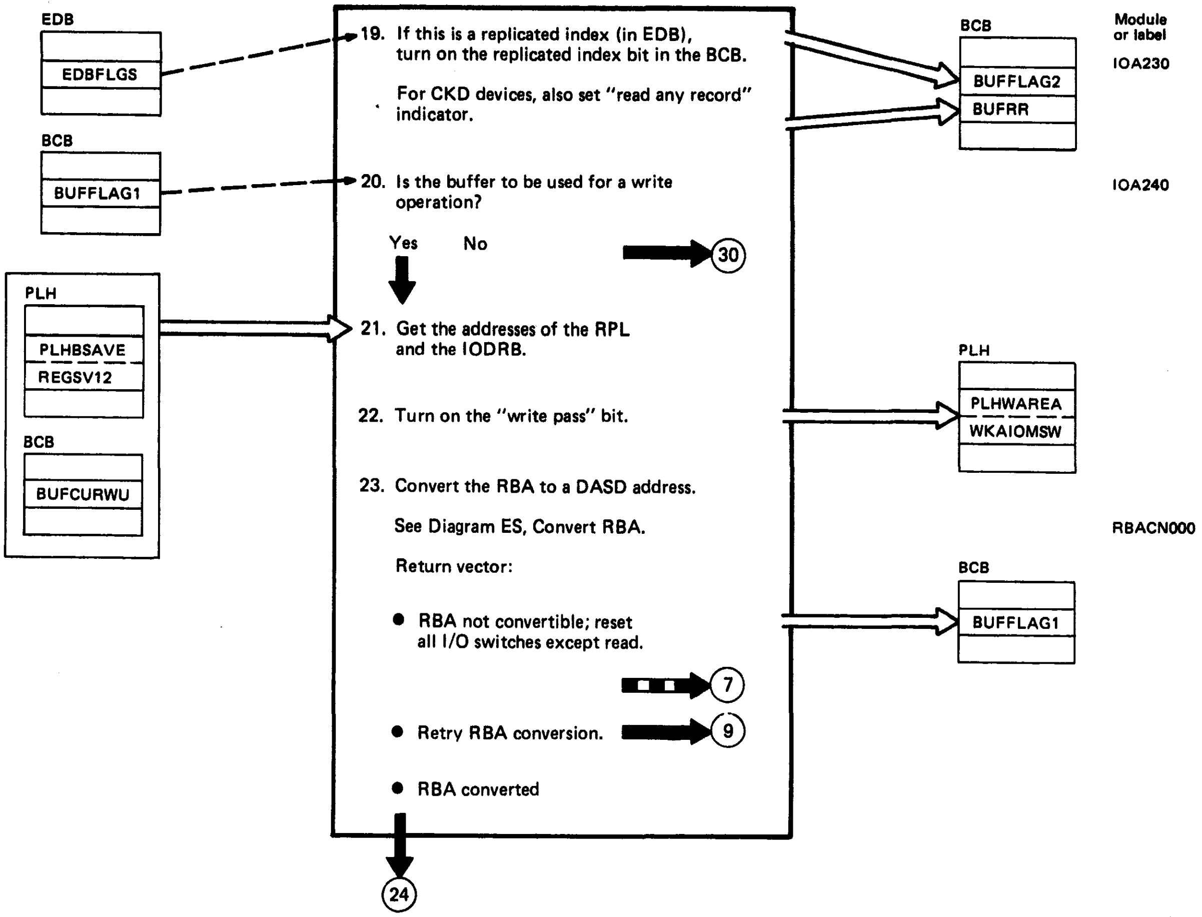
If the conversion failed, the I/O Manager returns control after clearing the read and valid bits in the buffer belonging to the invalid RBA. (Either the RBA did not exist in the current range of RBAs in the EDBs, or the volume mount failed.)

If a volume mount was successful, then the buffer queue has been changed by the purge buffer routine (IKQPBF). Normally, IKQPBF would have called the I/O Manager to write all buffers associated with the volume to be demounted (first pass) and to read all buffers associated with that volume (second pass). In this case, however, IKQPBF turns off the read (BUFCRRD) and/or write (BUFCMW and BUFCFMT) bits for these buffers so they will not be processed by the I/O Manager on its second pass.

18. If at least one buffer is found that requires a read I/O, the IOM1RD bit is turned on in the PLH for optimization.

If no buffers require a read I/O, the steps for sorting of buffers according to DASD addresses and testing of buffers for read I/O operations are bypassed later.

Diagram EA5. I/O Manager: Mainline



19. If this is a replicated index (in EDB), turn on the replicated index bit in the BCB.

    For CKD devices, also set "read any record" indicator.

20. Is the buffer to be used for a write operation?

21. Get the addresses of the RPL and the IODRB.

22. Turn on the "write pass" bit.

23. Convert the RBA to a DASD address.

    See Diagram ES, Convert RBA.

    Return vector:

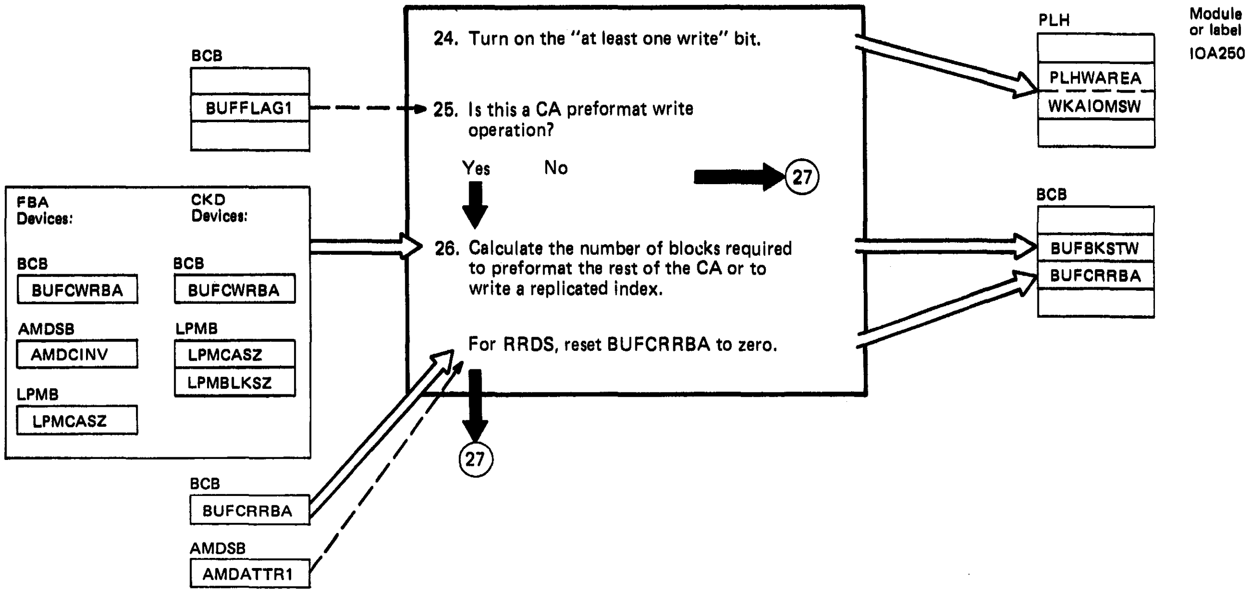
    - RBA not convertible; reset all I/O switches except read.
    - Retry RBA conversion.
    - RBA converted

Notes for Diagram EA5

20.-30. These steps are performed for any buffer that requires a write I/O operation.

23. This step performs RBA conversion in the same manner as step 17.

Diagram EA6. I/O Manager: Mainline



24. Turn on the "at least one write" bit.

25. Is this a CA preformat write operation?

26. Calculate the number of blocks required to preformat the rest of the CA or to write a replicated index.

For RRDS, reset BUFCRRBA to zero.

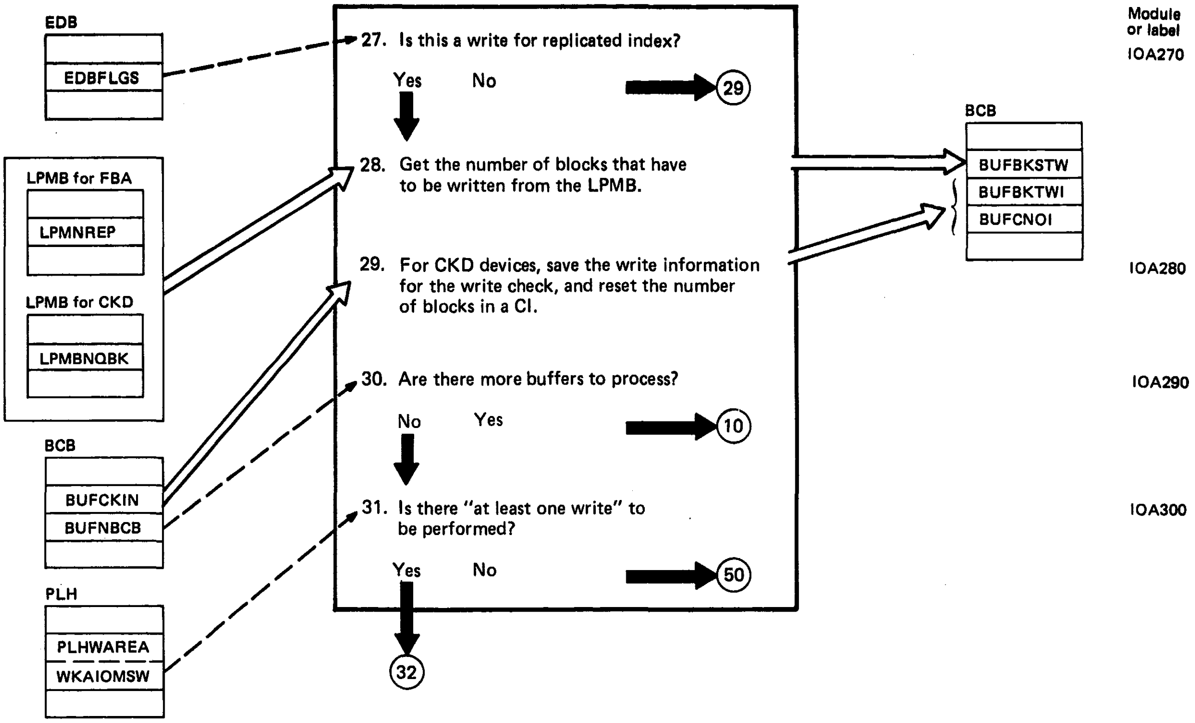
Notes for Diagram EA6

24. The buffer write optimization bit is like the read optimization bit described in step 18.

25.-28. These steps calculate the number of physical records required to format the remainder of a CA when preformatting is performed, or the blocks required to write a replicated index or sequence set.

For fixed block devices, the calculated value is the number of CIs remaining in the CA. For CKD devices, it is the number of physical records.

Diagram EA7. I/O Manager: Mainline



27. Is this a write for replicated index?

28. Get the number of blocks that have to be written from the LPMB.

29. For CKD devices, save the write information for the write check, and reset the number of blocks in a CI.

30. Are there more buffers to process?

31. Is there "at least one write" to be performed?

Notes for Diagram EA7

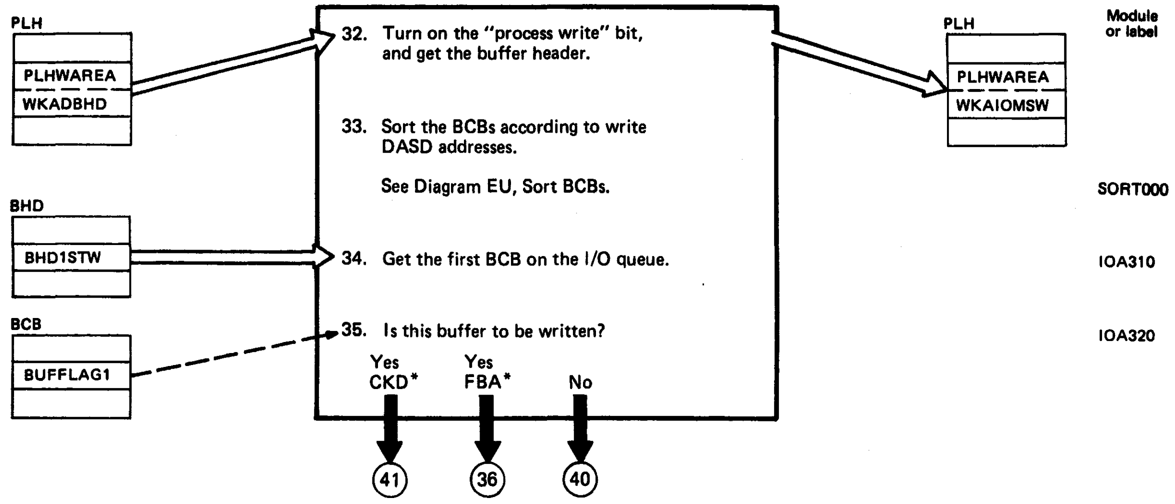
29. For CKD devices, the number of blocks and the DASD address are saved for later use when creating the write-check CCWs for the channel program.

31. If the "at least one write" bit is on, steps 32.-40. are performed.

These steps perform the write processing for the buffers. Buffers are sorted in ascending sequence according to DASD addresses to minimize rotation time. Channel programs are built to perform the write operations.

For CKD devices, processing includes write-check operations. After a CKD write pass, the same loop is executed for write-check (except for the sort operation). Write-check is indicated by bit AMDWCK in AMDATTR1 in the AMDSB.

Diagram EA8. I/O Manager: Mainline



32. Turn on the "process write" bit, and get the buffer header.

33. Sort the BCBs according to write DASD addresses.

    See Diagram EU, Sort BCBs.

34. Get the first BCB on the I/O queue.

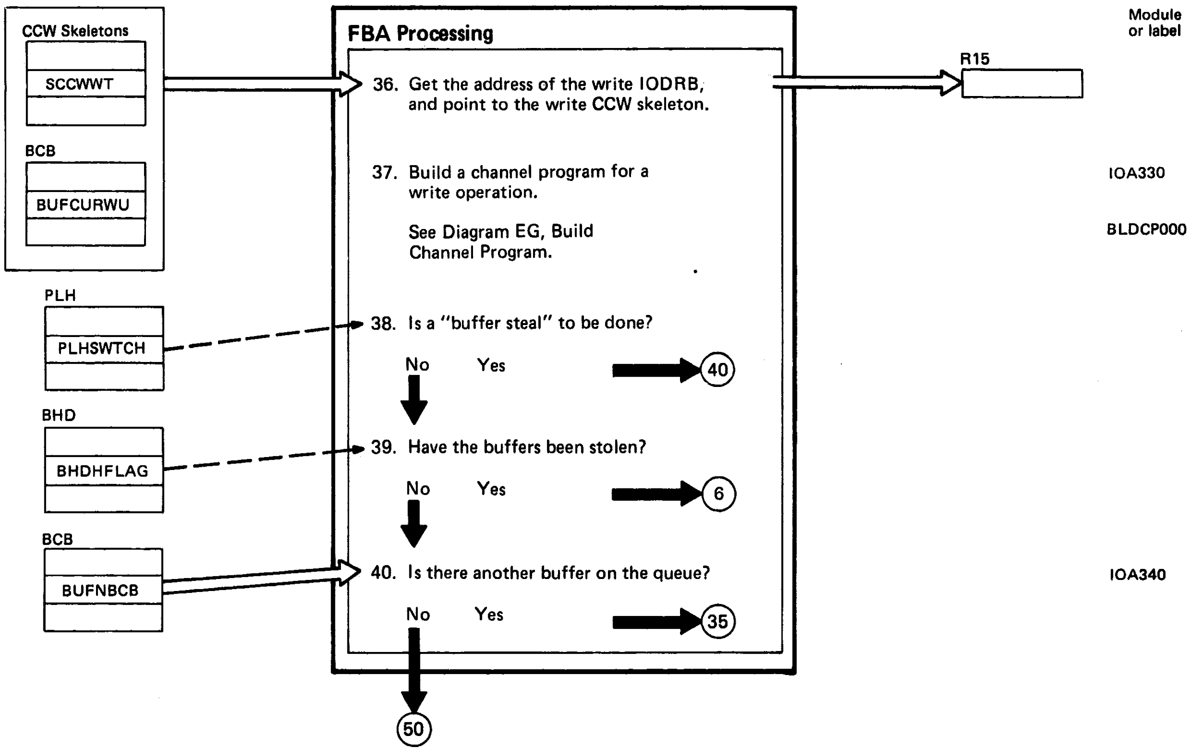
35. Is this buffer to be written?

*IKQIOA contains all CKD I/O Manager code, and IKQIOC contains all FBA I/O Manager code. Consequently, CKD vs. FBA decisions do not appear in the actual code. The "decision" is shown in the HIPO for convenience in documentation only.

Notes for Diagram EA8

32. The bit IOMPROCW in the PLH workarea indicates a write pass is being performed on the buffers when sorting and building channel programs.
33. Buffers are sorted in ascending DASD address sequence. Sorting is performed by rechaining the buffers. When the CCWs are built, they are executed in the proper sequence because each CCW in the chain points to its buffer in the correct sequence.
34. At this point there is at least one buffer on the I/O queue.
35. If the buffer is to be written, then steps 36.-40. are performed; otherwise the next buffer in the queue is obtained and tested in the same manner.

Diagram EA9. I/O Manager: Mainline


CCW Skeletons
SCCWWT
BCB
BUFCURWU
PLH
PLHSWTCH
BHD
BHDHFLAG
BCB
BUFNBCB
FBA Processing
36. Get the address of the write IODRB, and point to the write CCW skeleton.
37. Build a channel program for a write operation.
See Diagram EG, Build Channel Program.
38. Is a "buffer steal" to be done?
No
Yes
40
39. Have the buffers been stolen?
No
Yes
6
40. Is there another buffer on the queue?
No
Yes
35
50
R15
Module or label
IOA330
BLDCP000
IOA340

Notes for Diagram EA9

36.-37. These steps build a CCB, a CCW string, the I/O data blocks for the DASD arguments, and make the Fix List entries for all the buffers to be written to a CA. Each CA has its own set of CCB, CCW, and I/O data blocks; they are all built for all the writes performed during this pass. The module contains a skeleton of the CCW type to be built; the address of this skeleton is passed to the BDCCW routine via register 15.

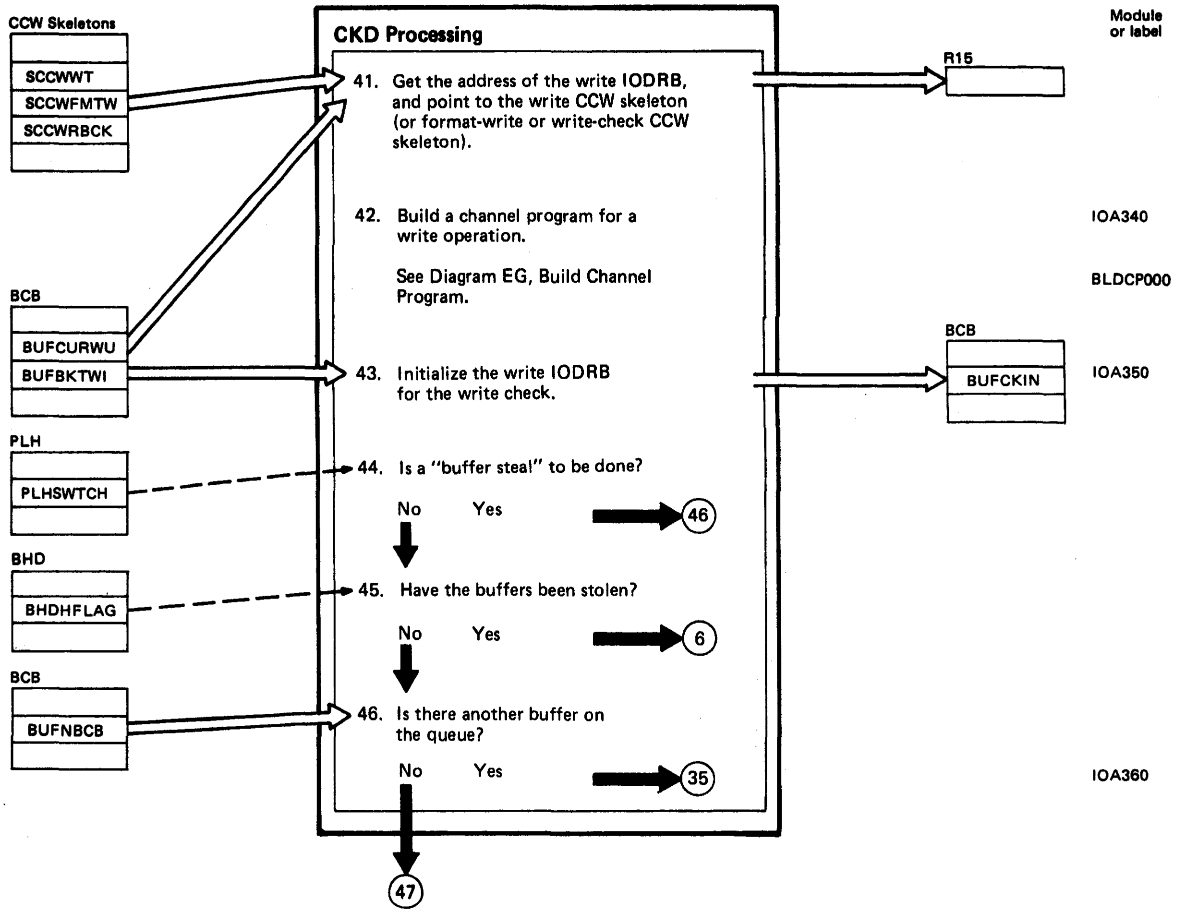
38. The "buffer steal" bit (PLHSTBCB) is turned on by the Buffer Manager whenever the I/O Manager is requested to complete the I/O of another task.

If the answer to this step is "yes", then buffers are being stolen from another task while the other task is also doing I/O Manager processing.

If the answer to this step is "no", this request to the I/O Manager is for normal processing, not a "buffer steal".

39. This step determines if buffers were stolen during the exit to the user's EXCPAD routine.

Diagram EA10. I/O Manager: Mainline



CKD Processing

41. Get the address of the write IODRB, and point to the write CCW skeleton (or format-write or write-check CCW skeleton).

42. Build a channel program for a write operation.

See Diagram EG, Build Channel Program.

43. Initialize the write IODRB for the write check.

44. Is a "buffer steal" to be done?

No Yes

45. Have the buffers been stolen?

No Yes

46. Is there another buffer on the queue?

No Yes

Notes for Diagram EA10

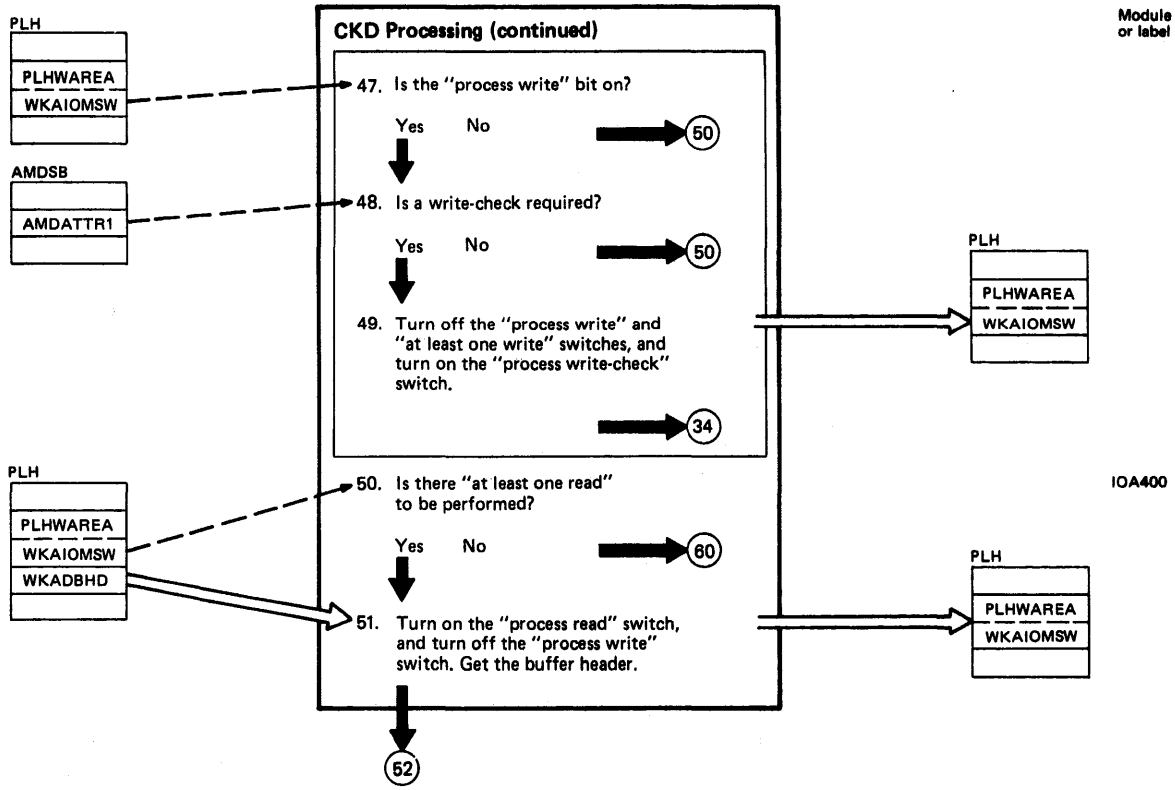
41.-42. See notes for steps 36.-37.

43. The write IODRB is initialized with the write-check data.

44. See notes for step 38.

45. See notes for step 39.

Diagram EA11. I/O Manager: Mainline



CKD Processing (continued)

47. Is the "process write" bit on?

48. Is a write-check required?

49. Turn off the "process write" and "at least one write" switches, and turn on the "process write-check" switch.

50. Is there "at least one read" to be performed?

51. Turn on the "process read" switch, and turn off the "process write" switch. Get the buffer header.

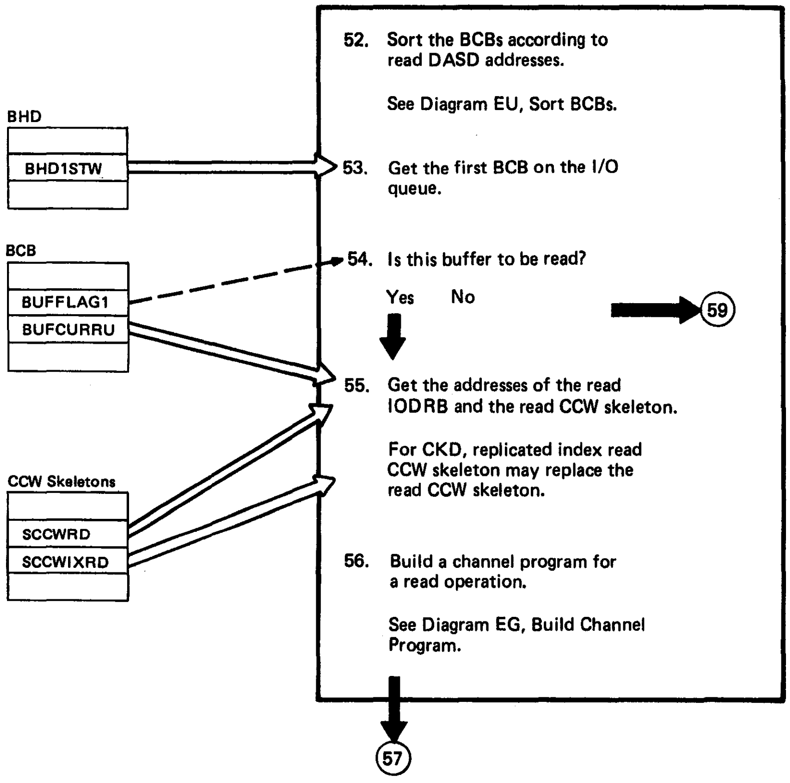
Notes for Diagram EA11

47.-49. Set up to perform a write-check pass if required.

50. These steps perform read processing for the buffers. Buffers are sorted in ascending sequence according to DASD addresses, and channel programs are built to perform read operations.

These steps are similar to the write operation described in steps 31.-40. Read CCWs are added to the end of the write CCWs when the buffers to be read are for the same CA as any writes in the preceding write pass.

Diagram EA12. I/O Manager: Mainline



Module or label

52. Sort the BCBs according to read DASD addresses.

See Diagram EU, Sort BCBs. SORT000

53. Get the first BCB on the I/O queue.

54. Is this buffer to be read? IOA410

Yes No

55. Get the addresses of the read IODRB and the read CCW skeleton.

For CKD, replicated index read CCW skeleton may replace the read CCW skeleton.

56. Build a channel program for a read operation. IOA420

See Diagram EG, Build Channel Program. BLDCP000

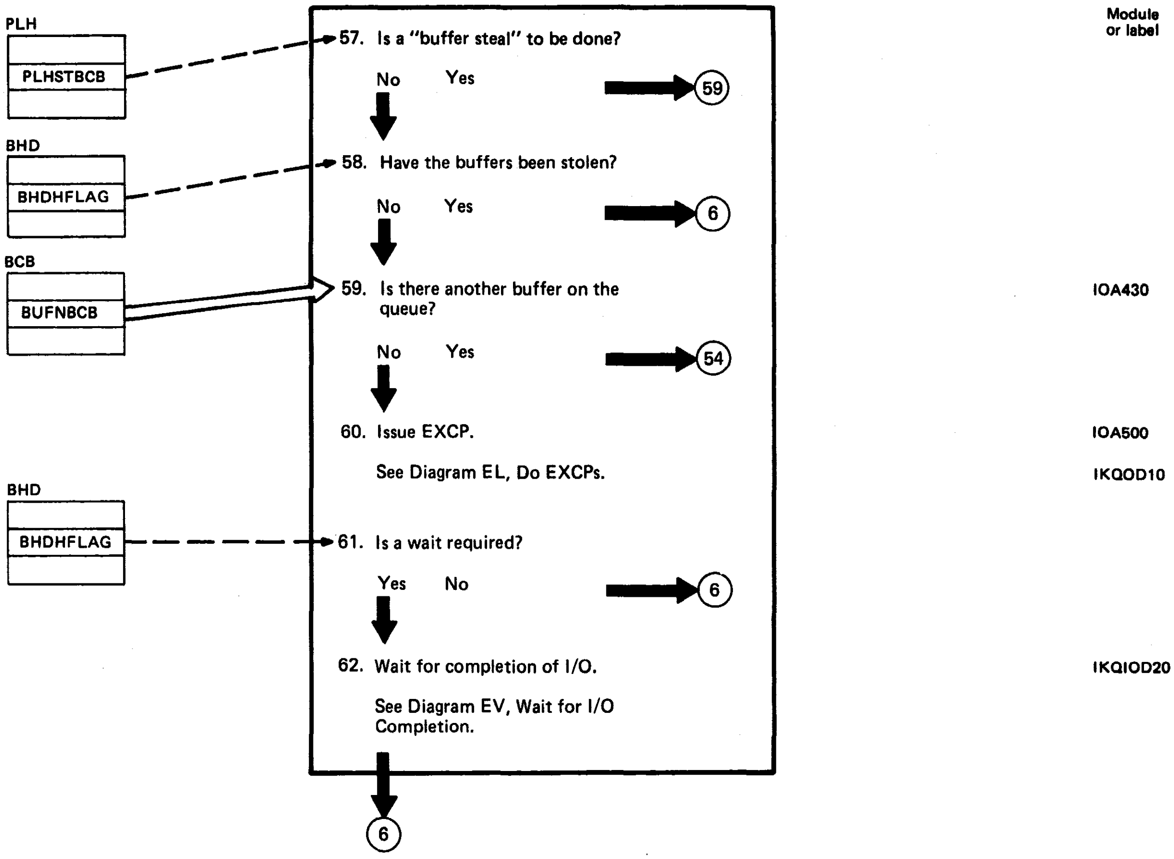
Notes for Diagram EA12

53. See notes for step 34.

54. See note for step 35.

55.-56. See notes for steps 36.-37.

Diagram EA13. I/O Manager: Mainline


PLH
PLHSTBCB
BHD
BHDHFLAG
BCB
BUFNBCB
BHD
BHDHFLAG
57. Is a "buffer steal" to be done?
No
Yes
59
58. Have the buffers been stolen?
No
Yes
6
59. Is there another buffer on the queue?
No
Yes
54
60. Issue EXCP.
See Diagram EL, Do EXCPs.
61. Is a wait required?
Yes
No
6
62. Wait for completion of I/O.
See Diagram EV, Wait for I/O Completion.
6
Module or label
IOA430
IOA500
IKQOD10
IKQIOD20

Notes for Diagram EA13

57. See notes for step 38.

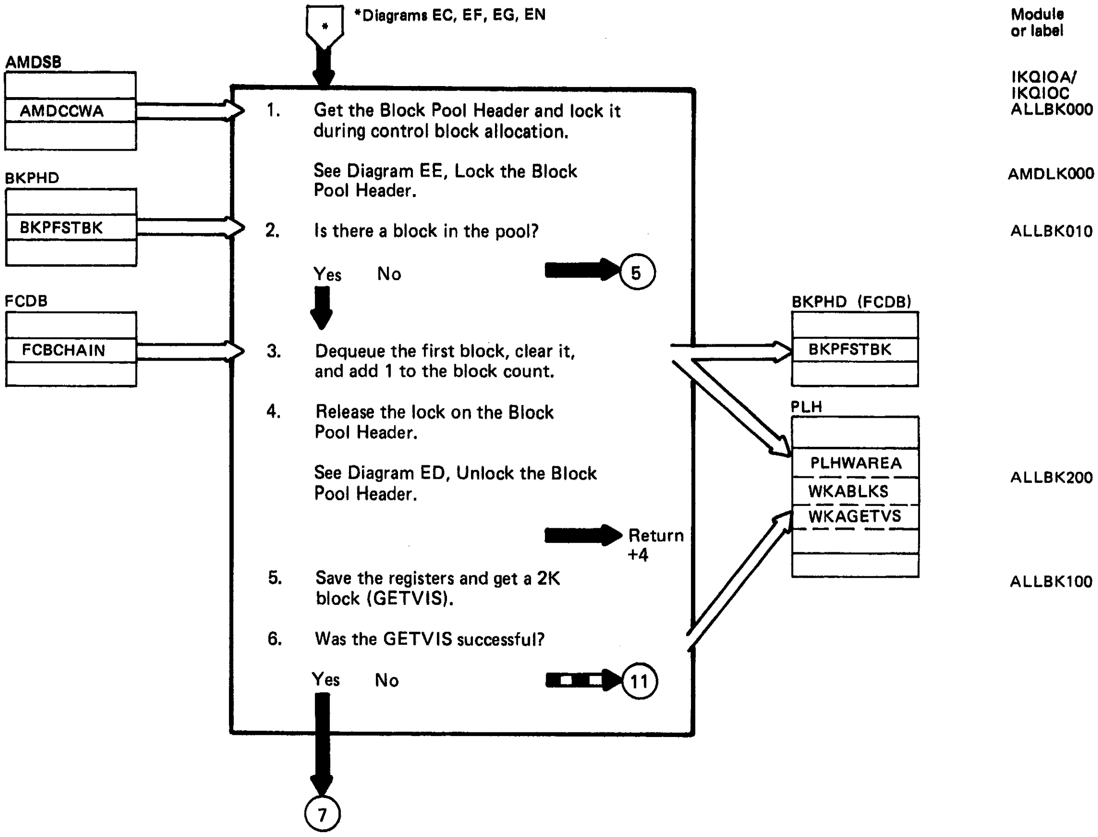
58. See notes for step 39.

60. This step causes the Fix List to be performatted in the format required by the system. An EXCP is issued for each IORB that has been properly initialized in the CCB chain. (Normally there will be only one CCB in the chain.)

For fixed block devices, the EXCP routine also performs track hold.

61.-62. If the I/O manager was called by the purge buffer routine (IKQPBF), then bit BHDNSKD in BHDHFLAG is set on, and the I/O manager waits for completion of the I/O operation just started.

Diagram EB1. I/O Manager: Allocate a Control Block



1. Get the Block Pool Header and lock it during control block allocation.

   See Diagram EE, Lock the Block Pool Header.

2. Is there a block in the pool?

3. Dequeue the first block, clear it, and add 1 to the block count.

4. Release the lock on the Block Pool Header.

   See Diagram ED, Unlock the Block Pool Header.

5. Save the registers and get a 2K block (GETVIS).

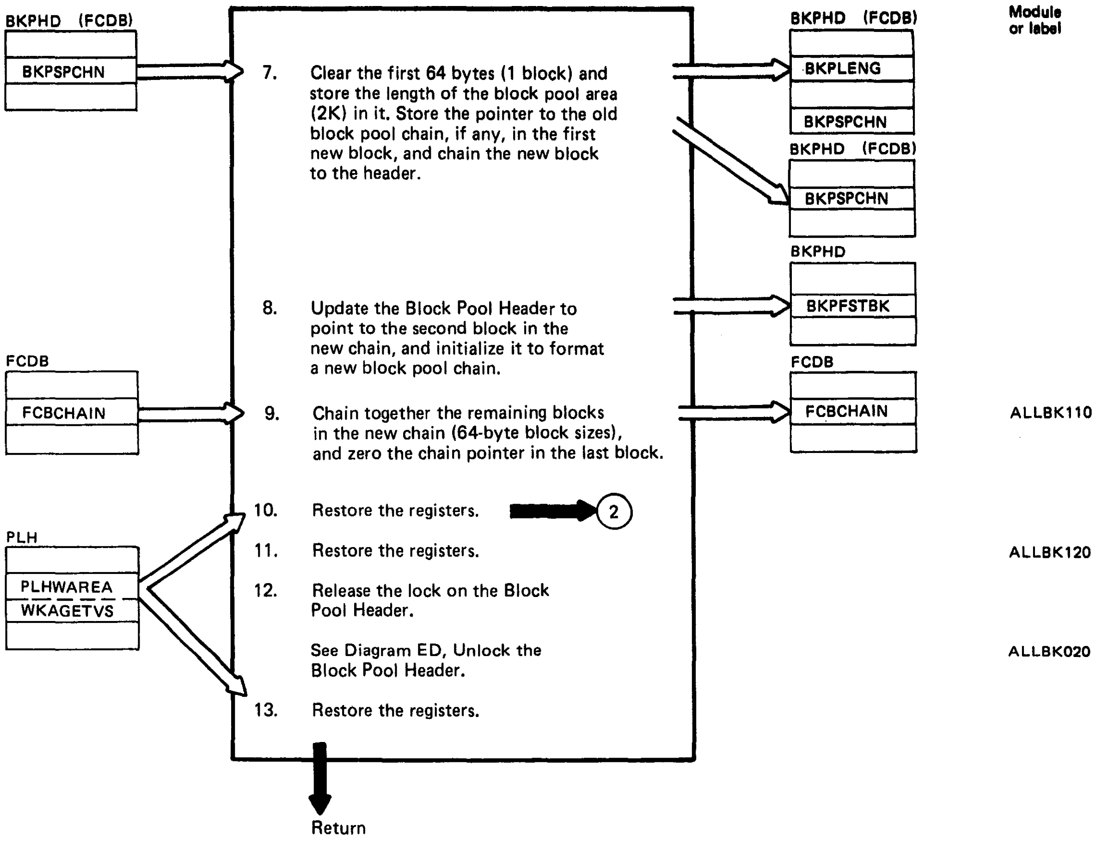
6. Was the GETVIS successful?

Notes for Diagram EB1

1.-4. The address of the Block Pool Header (BKPHD) is obtained from the AMDSB, and the Block Pool Header ECB is locked while allocation takes place. The first available block in the chain is removed from the chain and cleared to binary zeros. The PLH counter for the number of allocated blocks used for a given CCB is incremented by one. The Block Pool Header is now unlocked, and control returns to the caller.

5. If no block is available, a 2K GETVIS is issued with PAGE=YES and POOL=YES specified. If no space can be obtained, the Block Pool Header ECB is unlocked, and an error return is taken.

Diagram EB2. I/O Manager: Allocate a Control Block



7. Clear the first 64 bytes (1 block) and store the length of the block pool area (2K) in it. Store the pointer to the old block pool chain, if any, in the first new block, and chain the new block to the header.

8. Update the Block Pool Header to point to the second block in the new chain, and initialize it to format a new block pool chain.

9. Chain together the remaining blocks in the new chain (64-byte block sizes), and zero the chain pointer in the last block.

10. Restore the registers.

11. Restore the registers.

12. Release the lock on the Block Pool Header.

    See Diagram ED, Unlock the Block Pool Header.

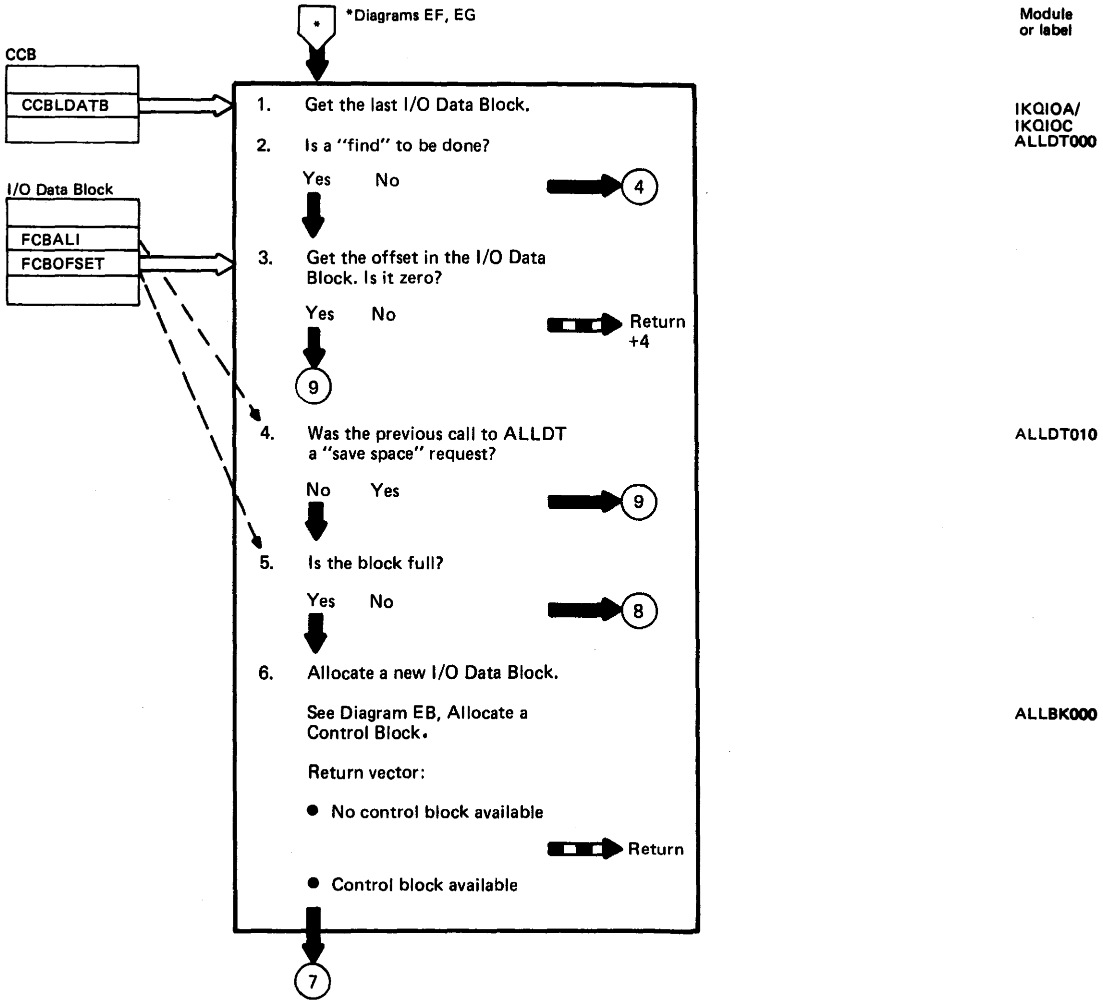
13. Restore the registers.

Notes for Diagram EB2

7.-10. After a successful GETVIS, the first 64 bytes of the gotten area are cleared to binary zeros and used as a header for a new set of blocks. This header is chained to the header of the last set of blocks allocated, and the Block Pool Header (BKPSPCHN) is chained to the new header.

The remaining space is suballocated into 64-byte blocks. These remaining blocks are chained together, and the first block of the chain is chained to the new header block and the Block Pool Header (BKPFSTBK). The allocation procedure is then retried.

Diagram EC1. I/O Manager: Allocate and Find an I/O Data Field



*Diagrams EF, EG
Module or label
CCB
CCBLDATB
I/O Data Block
FCBALI
FCBOFSET
1. Get the last I/O Data Block.
2. Is a "find" to be done?
Yes
No
4
3. Get the offset in the I/O Data Block. Is it zero?
Yes
No
Return +4
9
4. Was the previous call to ALLDT a "save space" request?
No
Yes
9
5. Is the block full?
Yes
No
8
6. Allocate a new I/O Data Block.
See Diagram EB, Allocate a Control Block.
Return vector:
● No control block available
Return
● Control block available
7
IKQIOA/ IKQIOC ALLDT000
ALLDT010
ALLBK000

Notes for Diagram EC1

This routine performs one of three functions, depending on the value passed in register 1:

Register 1 = 0: Find last-used data field in the block.

Register 1 = 4: Get next available data field in the block to use for storing the I/O DASD arguments.

Register 1 = 8: Get next available data field in the block to use as a temporary save area for the CCB currently being processed.

1.-3. The address of the last I/O Data Block in the chain (currently-used block) is obtained. Then the type of function call is determined. For a "find" function, if the offset (FCBOFSET) of the control block is zero, the routine returns to the calling routine with register 9 pointing to the start of the control block.

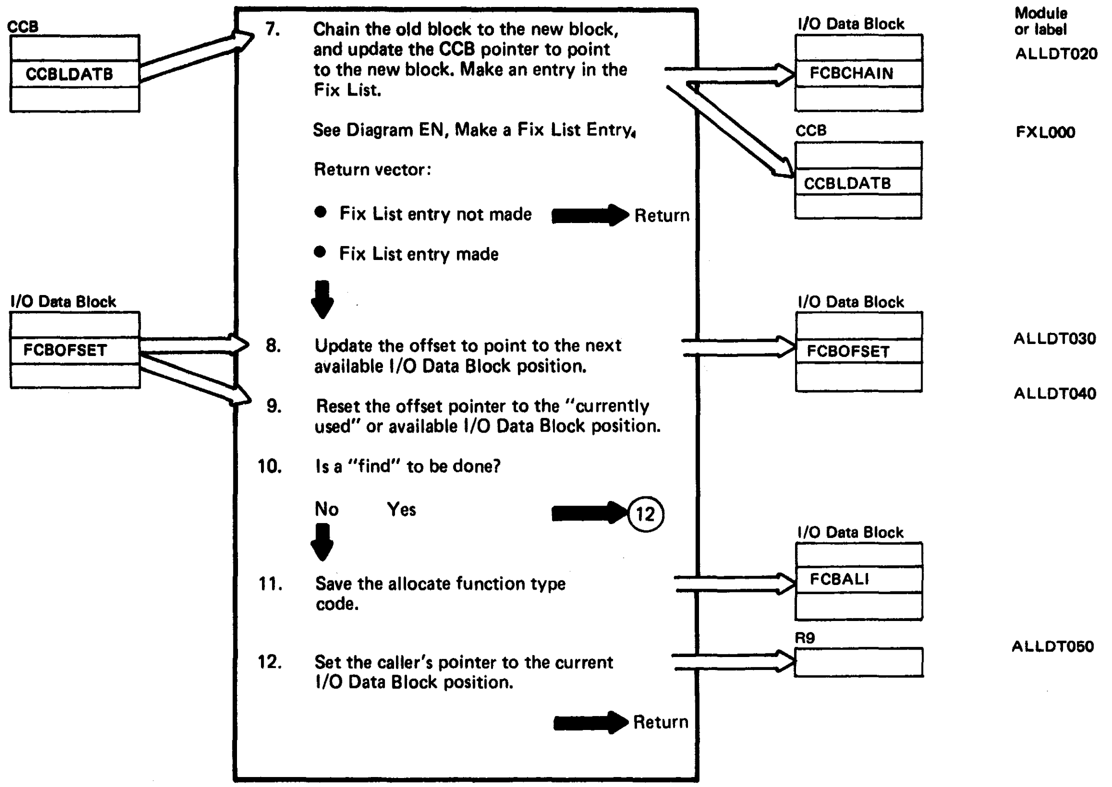
If the offset is not zero, then steps 9, 10, and 12 are performed to generate the pointer to the last-used I/O data field in the block.

4.-7. Each time arguments are stored or temporarily saved in the control block, the allocate type code (in register 1) is stored in the control block (FCBALI) for the next store or save operation. (For a temporary save, the save area space is reused for a permanent store.)

If the last entry of the control block was not just previously used as a temporary save area, a new field is allocated (if available). If all the fields are used, the "Allocate a Control Block" routine (ALLBK) is called to get another 64-byte block. The new block (if obtained) is chained to the old block. If a new block is not obtained, an error return is taken.

If the last entry of the control block was previously used as a temporary save area, steps 9-12 are performed to generate a pointer to this temporary save area (last-used area in the block) so that it can be reused.

Diagram EC2. I/O Manager: Allocate and Find an I/O Data Field



7. Chain the old block to the new block, and update the CCB pointer to point to the new block. Make an entry in the Fix List.

   See Diagram EN, Make a Fix List Entry.

   Return vector:

   - ● Fix List entry not made → Return
   - ● Fix List entry made

8. Update the offset to point to the next available I/O Data Block position.

9. Reset the offset pointer to the "currently used" or available I/O Data Block position.

10. Is a "find" to be done?

    No   Yes → (12)

11. Save the allocate function type code.

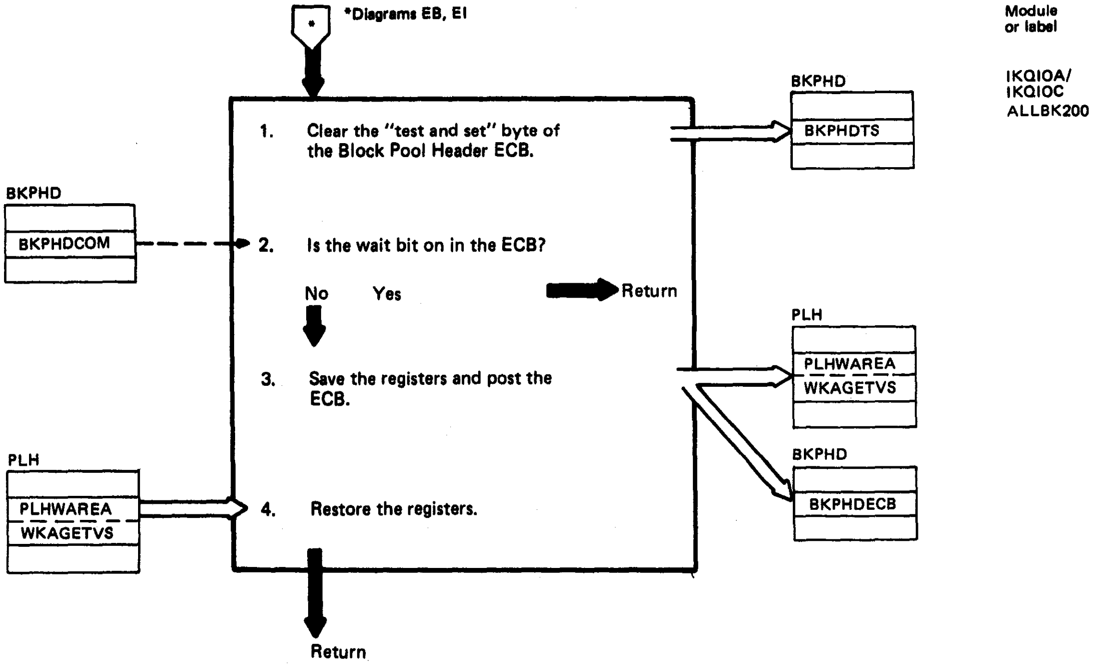
12. Set the caller's pointer to the current I/O Data Block position.

    → Return

Notes for Diagram EC2

8.-12. For the I/O data field store operation, the FCDB offset is updated to point to the next available field (after the one to be used for the current request). The pointer is then backed up to the current field to be used, and the allocate type code is saved.

Diagram ED1. I/O Manager: Unlock the Block Pool Header

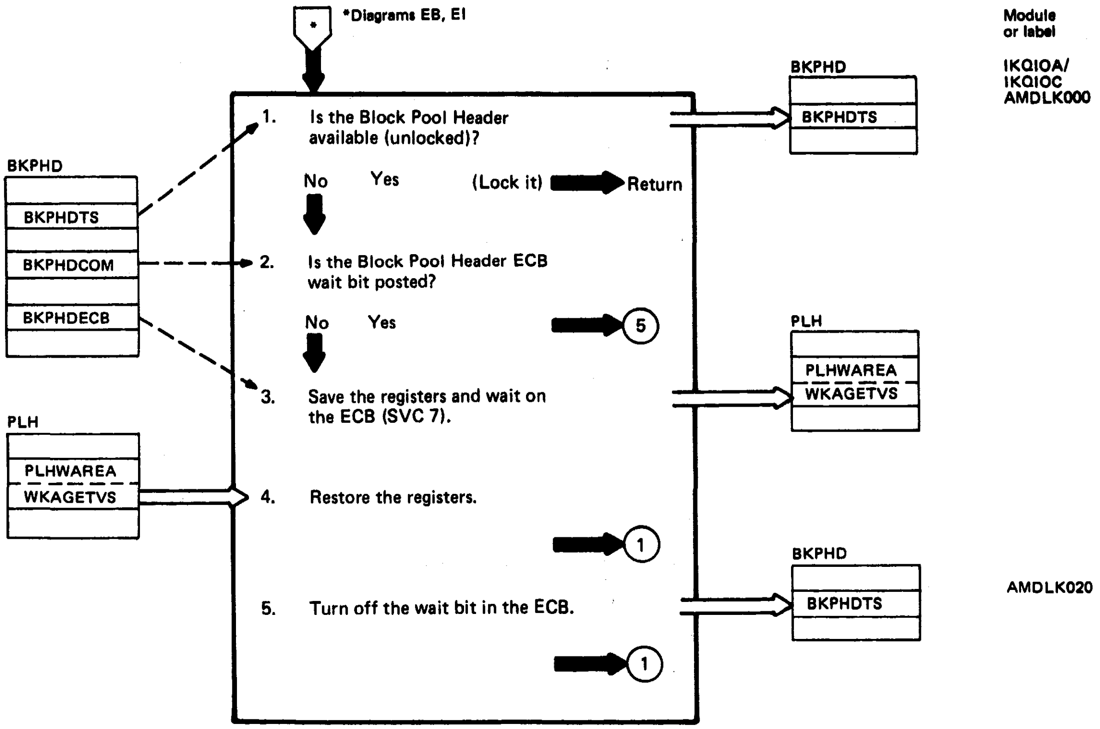


1. Clear the "test and set" byte of the Block Pool Header ECB.
2. Is the wait bit on in the ECB?
3. Save the registers and post the ECB.
4. Restore the registers.

Notes for Diagram ED1

This routine clears the "test and set" byte of the Block Pool Header ECB. If it finds the wait bit off, it posts the ECB to bring any waiting tasks out of the WAIT state.

Diagram EE1. I/O Manager: Lock the Block Pool Header



1. Is the Block Pool Header available (unlocked)?
2. Is the Block Pool Header ECB wait bit posted?
3. Save the registers and wait on the ECB (SVC 7).
4. Restore the registers.
5. Turn off the wait bit in the ECB.

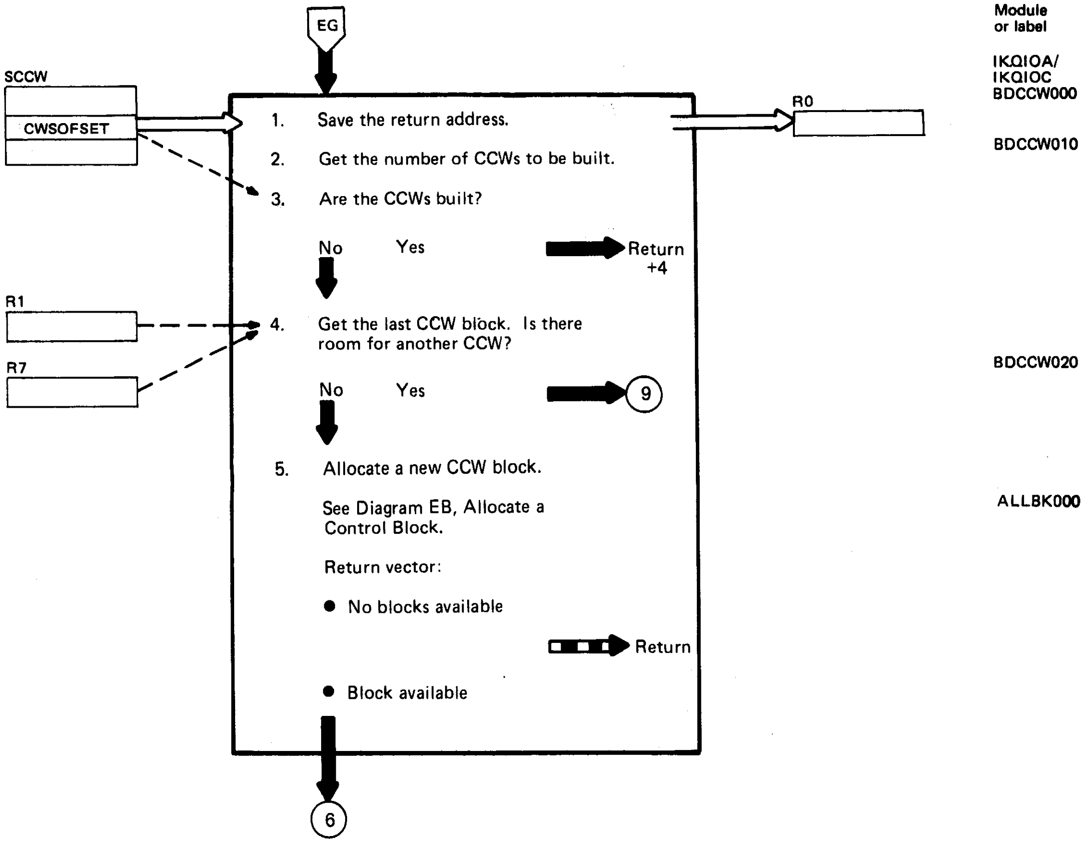
Notes for Diagram EE1

This routine performs a "test and set" on the Block Pool Header ECB. If the ECB is available, it will now be locked, and control returns to the calling routine.

If the ECB is not available, the wait bit is tested (in case posting of the ECB occurred after the "test and set"). If the wait bit is on, it is cleared, and the "test and set" is retried.

If the ECB is not posted, the lock routine goes into a wait state until the Block Pool Header is available.

Diagram EF1. I/O Manager: Build CCW



1. Save the return address.
2. Get the number of CCWs to be built.
3. Are the CCWs built?
4. Get the last CCW block. Is there room for another CCW?
5. Allocate a new CCW block.

See Diagram EB, Allocate a Control Block.

Return vector:

- No blocks available
- Block available

Notes for Diagram EF1

2.-3. Each CCW skeleton starts with an offset (that is, a multiple of 8) which indicates the number of CCWs to be built during each pass through the Build CCW routine. Each skeleton is 11 bytes long. When the last byte at the end of the skeleton(s) is a zero value, control is returned to the caller.

NOTE:

- Search ID Equal – TIC CCWs require two CCW positions in the same CCW block.
- A Read Count CCW and a Read Data CCW are treated as a CCW pair, but they do not need to be in the same CCW block. The Read Count CCW is used only when a replicated index search is to be done.
- The Seek and Set Sector CCWs are always in the same block. They will always be the first two CCWs in the channel program.

4. The last CCW block in the CCW chain is tested to see if there is sufficient room for the number of CCWs to be built during the current pass. The offset of the starting CCW skeleton indicates the amount of CCW space required. The maximum offset for the CCW block is 56 bytes (FCBCMAX).

The last entry in the CCW block is reserved for a TIC to the next CCW string in the next block, if more than 7 CCWs (6 when a CCW pair must be built as the last two entries and reside in the same CCW block) are to be built in one channel program.

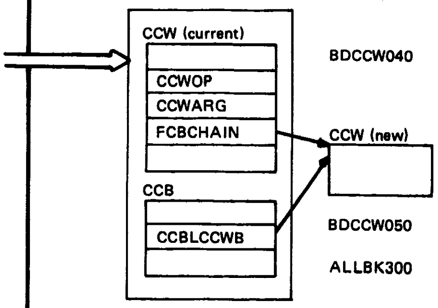
5.-8. When the current CCW block becomes full, a new CCW block is allocated, and the appropriate entry is made in the Fix List.

If allocation fails for the Fix List entry, the CCW block is deallocated because it is not yet in the CCW chain.

If either allocation fails, a return is taken to Build Channel Program routine (BLDCP), and I/O is performed for the current chain of CCBs in order to free control blocks for reuse.

Diagram EF2. I/O Manager: Build CCW

	Module or label
6. Make an entry in the Fix List for the new CCW block.	BDCCW030
See Diagram EN, Make a Fix List entry.	FXL000
Return vector: ● Fix List entry not made → 8 ● Fix List entry made ↓	
7. Put a TIC to the new CCW block after the last CCW in the current CCW block, chain the current CCW block to the new CCW block, and save the pointer to the new CCW block (in the CCB). → 4	BDCCW040
8. Deallocate the CCW block.	BDCCW050
See Diagram EI, Deallocate a control block. → Return	ALLBK300



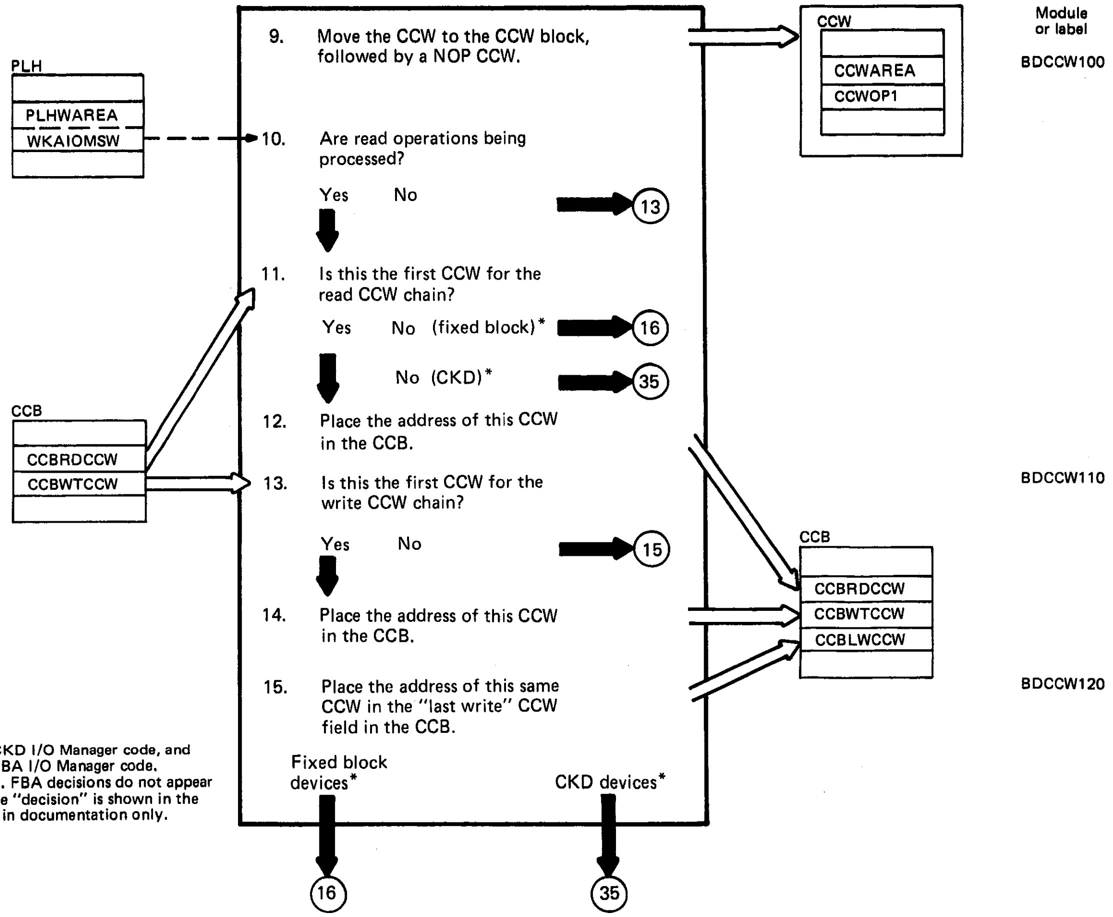
Notes for Diagram EF2

7. When a new CCW block is obtained, it is chained to the current block, and a TIC CCW is placed after the last CCW in the current block.

   For a single CCW skeleton currently being processed, the TIC overlays the chain pointer to the new CCW block.

   If a multiple CCW skeleton (Search ID Equal – TIC) is being processed (and the two CCWs must reside in the same CCW block), the TIC follows the last CCW in the block, but it does not overlay the chain pointer. There will be less than 7 CCWs in the block (excluding the TIC CCW). The $IDE CCW can never begin in the seventh entry because the channel will skip the eighth entry when the search is satisfied.

Diagram EF3. I/O Manager: Build CCW



Module or label
PLH
PLHWAREA
WKAIOMSW
CCB
CCBRDCCW
CCBWTCCW
9. Move the CCW to the CCW block, followed by a NOP CCW.
10. Are read operations being processed?
Yes No
13
11. Is this the first CCW for the read CCW chain?
Yes No (fixed block)*
16
No (CKD)*
35
12. Place the address of this CCW in the CCB.
13. Is this the first CCW for the write CCW chain?
Yes No
15
14. Place the address of this CCW in the CCB.
15. Place the address of this same CCW in the "last write" CCW field in the CCB.
Fixed block devices*
CKD devices*
16
35
CCW
CCWAREA
CCWOP1
BDCCW100
BDCCW110
CCB
CCBRDCCW
CCBWTCCW
CCBLWCCW
BDCCW120

*IKQIOA contains all CKD I/O Manager code, and IKQIOC contains all FBA I/O Manager code. Consequently, CKD vs. FBA decisions do not appear in the actual code. The "decision" is shown in the HIPO for convenience in documentation only.

Notes for Diagram EF3

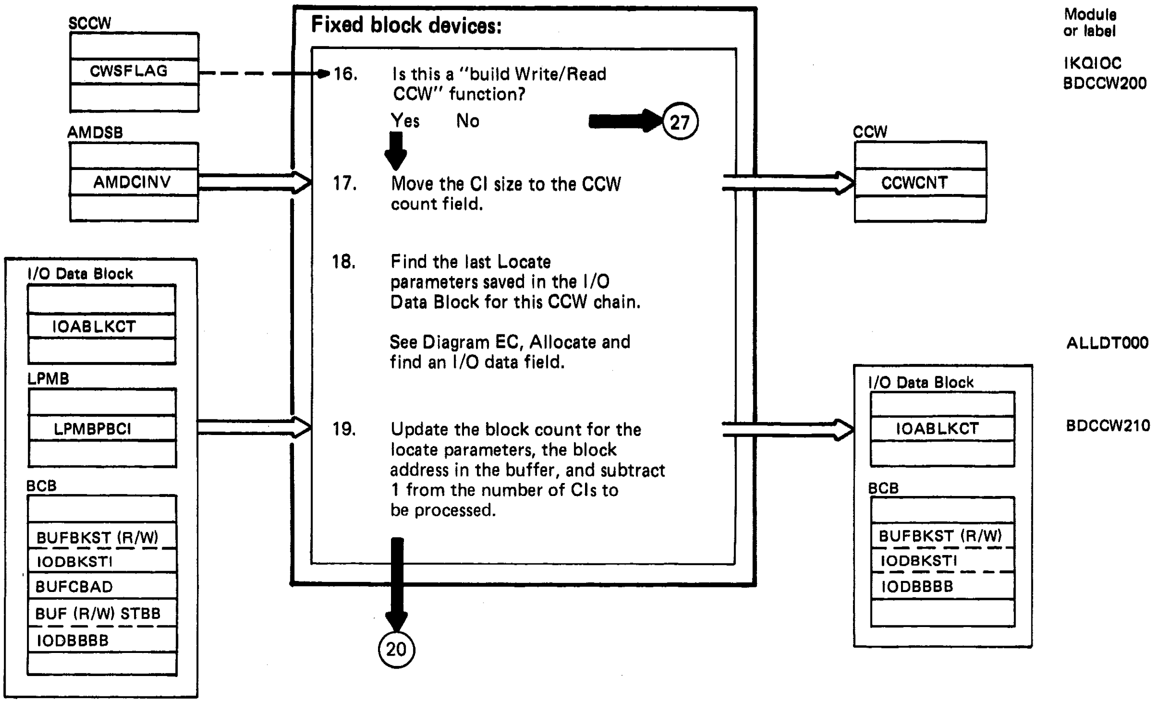
9. The CCW skeleton is moved to the CCW area; a NOP op code follows in the next CCW position.

For fixed block devices, a CCW chain consisting of only Write CCWs must be followed by a NOP if the Locate parameters specify that a write check is to be performed.

10.-15. The first CCW for a Read or Write CCW string is stored in the CCB. The last CCW in the Write CCW string is also stored in the CCB. These CCWs are used by the I/O Error Handler (IKQIOB) when handling errors for a "piggy-back" CCW chain. They are used to determine where the CCW chain is to be split for retry of the CCW chain.

The "piggy-back" indicator (CCBUERR) suppresses error retry by the system. The I/O Error Handler turns this switch off when it performs retry. The last Write CCW has the command chaining switch turned off when the write portion of the CCW string is retried. When the Read CCW string is retried, a TIC CCW to it is inserted after the Seek CCW (count-key-data devices) or Define Extent CCW (fixed block devices) to point to the start of the Read CCW string.

Diagram EF4. I/O Manager: Build CCW



16. Is this a "build Write/Read CCW" function?

17. Move the CI size to the CCW count field.

18. Find the last Locate parameters saved in the I/O Data Block for this CCW chain.

See Diagram EC, Allocate and find an I/O data field.

19. Update the block count for the locate parameters, the block address in the buffer, and subtract 1 from the number of CIs to be processed.

Notes for Diagram EF4

16.-19. If the CWSBFADC bit is on in the CWSFLAG field of the skeleton CCW, it indicates a Write/Read CCW skeleton. The CCW count field is initialized to the CI length, and the Locate arguments are obtained from the I/O Data Block for this CCW (most recent entry in block).

The block count field in the Locate arguments is updated by the total number of physical blocks required for a CI. The block address in the buffer is also updated by this same number. This is done to compare it to the block address of any succeeding blocks to determine if a Locate is needed for the next Read or Write CCW.

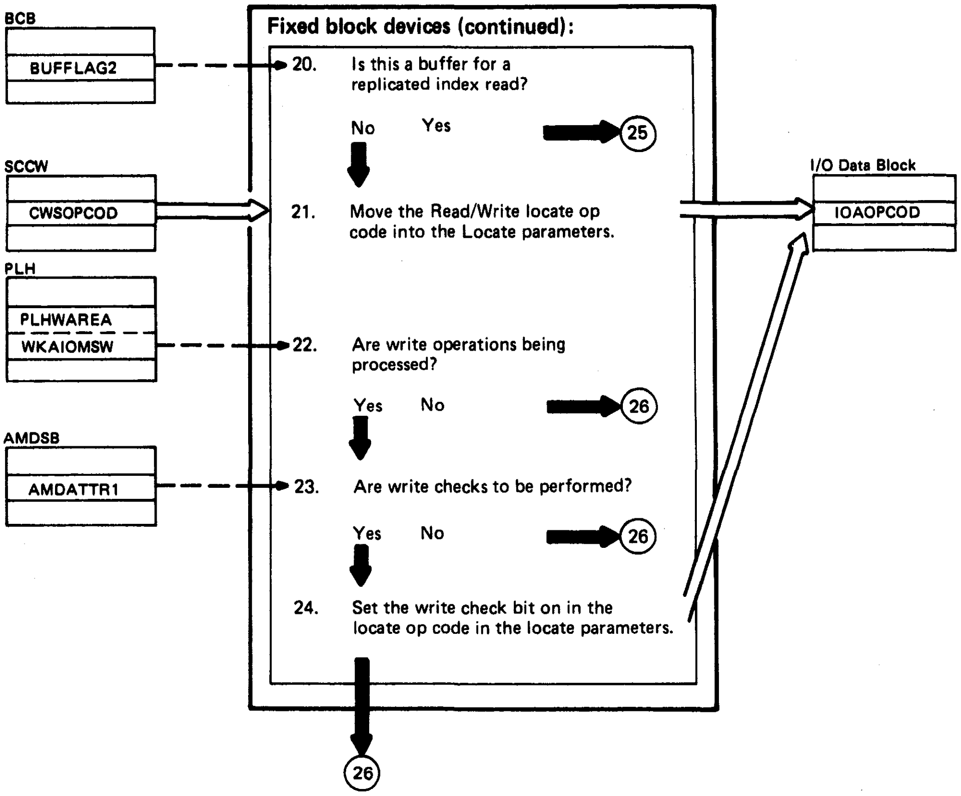
The number of CIs to be processed for the current buffer is decremented by one; this counter controls looping through BLDCP.

For Define Extent and Locate CCWs, the CWSFLAG byte has the CWSARGAD bit on. This means the I/O area to be used in the CCW is in the I/O Data Block (FCDB).

For Read and Write CCWs, the CWSFLAG byte has the CWSBFADC bit on. This means the I/O area to be used is the buffer currently being processed.

Chained Read/Write CCWs must have the data chaining flag on when they are immediately followed by other Read/Write CCWs.

Diagram EF5. I/O Manager: Build CCW



Fixed block devices (continued):

20. Is this a buffer for a replicated index read?

21. Move the Read/Write locate op code into the Locate parameters.

22. Are write operations being processed?

23. Are write checks to be performed?

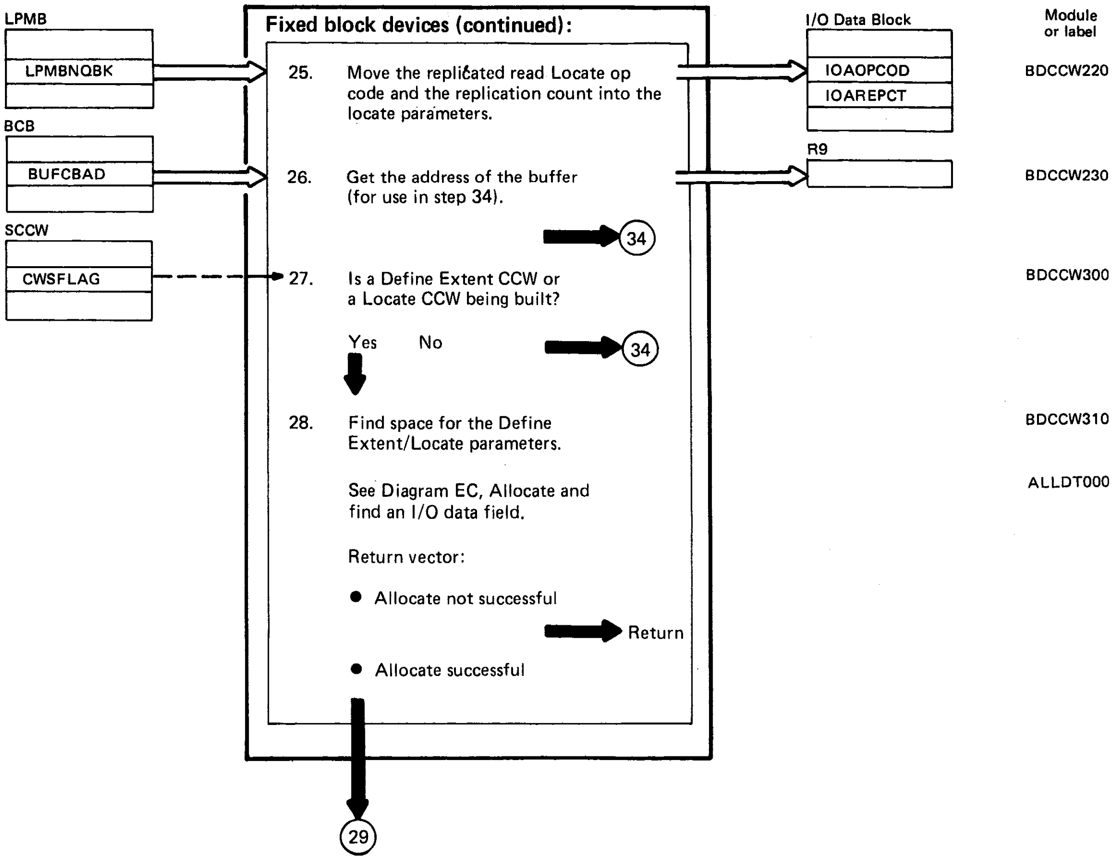
24. Set the write check bit on in the locate op code in the locate parameters.

Notes for Diagram EF5

20.-21. If this is a Read CCW for a data record or non-replicated index (or sequence set) record, the normal Read CCW is used. The Locate argument is updated to indicate a normal read operation.

22.-24. If write operations are being processed and a write-check is required, then the write-check bit (IOAWCK) is turned on in the Locate argument op code.

Diagram EF6. I/O Manager: Build CCW



Fixed block devices (continued):

25. Move the repliĉated read Locate op code and the replication count into the locate parameters.

26. Get the address of the buffer (for use in step 34).

27. Is a Define Extent CCW or a Locate CCW being built?

28. Find space for the Define Extent/Locate parameters.

See Diagram EC, Allocate and find an I/O data field.

Return vector:

- Allocate not successful
- Allocate successful

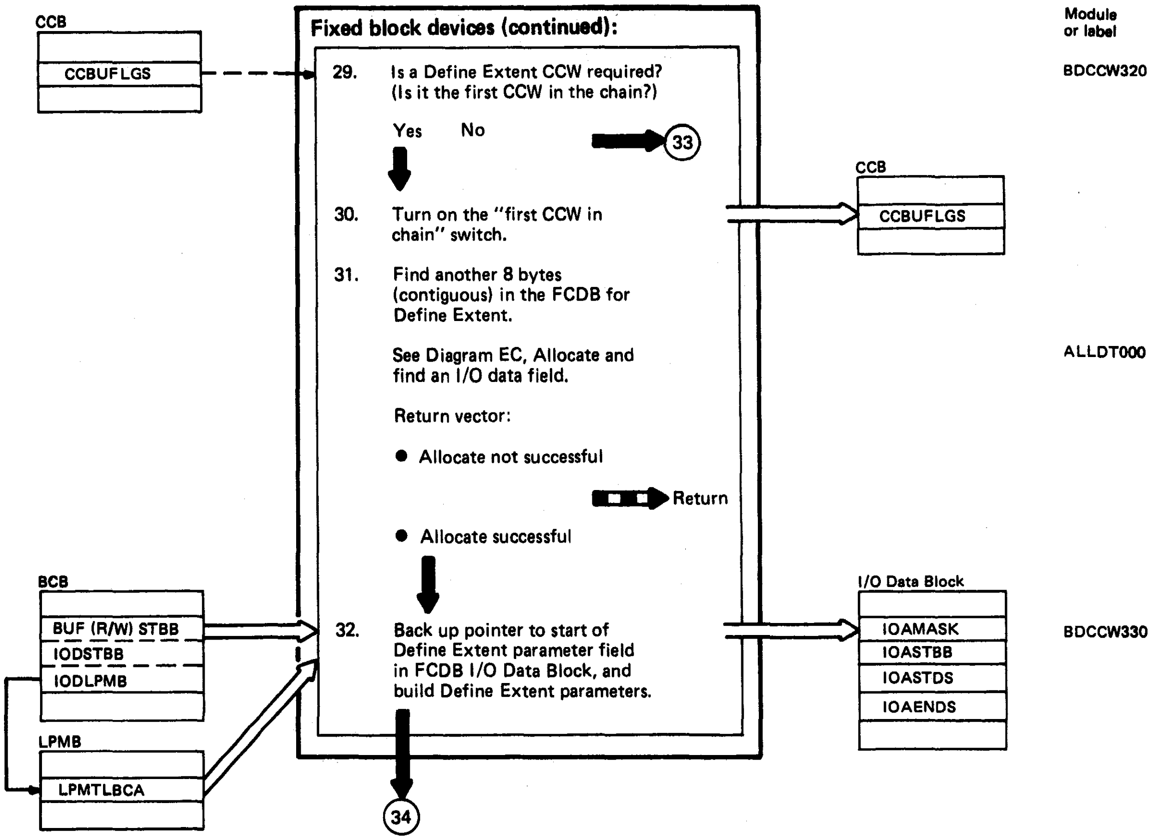
Notes for Diagram EF6

25. When a replicated index (or sequence set) read is to be done, the Locate arguments are updated to indicate a replicated read, and the replication count field is set to the total number of physical blocks actually used for the imbedded index or sequence set. (Any blocks wasted after the sequence set are not included in this value.)

27.-31. Whenever a Define Extent CCW or a Locate CCW is built (CWSARGAD), an 8-byte field in the I/O Data Block is allocated for its DASD arguments, and the CCBRPS bit is turned on to indicate that building of the channel program has started. (CCBRPS was initially set to 0 to indicate that a new channel program is to be started.)

This 8-byte field is sufficient for a Locate CCW, but a Define Extent CCW requires 16 bytes, so an additional 8 bytes must be allocated. Because there is only one Define Extent CCW in a channel program, and it is always the first CCW, its arguments are always the first 16 bytes of the I/O Data Block (assuming no allocation error occurs). Register 9 contains the address of the 8-byte field just allocated, so it must be backed up to point to the beginning of the 16-byte field.

Diagram EF7. I/O Manager: Build CCW



Fixed block devices (continued):

29. Is a Define Extent CCW required? (Is it the first CCW in the chain?)

Yes No → 33

30. Turn on the "first CCW in chain" switch.

31. Find another 8 bytes (contiguous) in the FCDB for Define Extent.

See Diagram EC, Allocate and find an I/O data field.

Return vector:

- Allocate not successful → Return
- Allocate successful

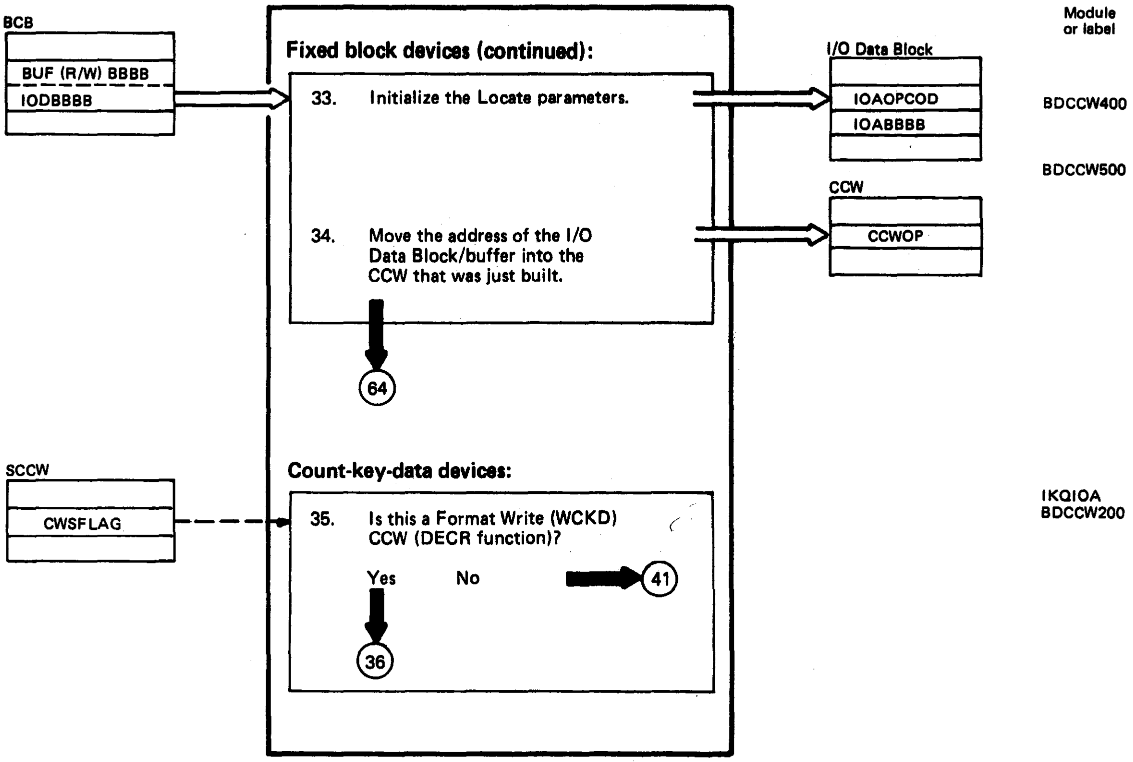
32. Back up pointer to start of Define Extent parameter field in FCDB I/O Data Block, and build Define Extent parameters.

→ 34

Notes for Diagram EF7

32. This step builds the Define Extent arguments. It sets the I/O mask field to 0 (default value), initializes the physical starting block of the CA on the device, and sets the block range to the total number of physical blocks in the CA (not including any wasted blocks at the end of the CA). The start of the block range is set to 0.

Diagram EF8. I/O Manager: Build CCW


BCB
BUF (R/W) BBBB
IODBBBB
Fixed block devices (continued):
33. Initialize the Locate parameters.
34. Move the address of the I/O Data Block/buffer into the CCW that was just built.
64
Module or label
I/O Data Block
IOAOPCOD
IOABBBB
BDCCW400
BDCCW500
CCW
CCWOP
SCCW
CWSFLAG
Count-key-data devices:
35. Is this a Format Write (WCKD) CCW (DECR function)?
Yes
No
41
36
IKQIOA
BDCCW200

Notes for Diagram EF8

33. This step initializes the fields for the Locate parameters. The CCW type op code, the block count, and the replication count fields are initialized to zeros. They will be set during Write/Read CCW processing. The physical block number of the first block to be processed by this Locate is then placed in the argument field.

34. The I/O argument address obtained in step 26 (for the buffer) or steps 28-32 from the I/O Data Block allocation (in R9) is placed in the CCW argument field, and the offset in the CCW block is updated to point to the next available CCW position.

35.-64. CWSFLAG settings in the CCW skeletons for CKD devices are listed below. See the CCW skeleton DSECT description in Section 5: Data Areas for explanation of the bit settings.

CCW Type	Flag Names	Hex
Seek	CWSIVLR,CWSARGAD,CWSNOOPT	51
Search ID Equal	CWSARGAD	10
Set Sector	CWSRPS (in CWSFLAGC)	80
TIC	CWSASTER	08
Write Count	CWSARGAD,CWSDECR,CWSNOOPT	13
Write Data 1*	CWSBFADC,CWSINCR	24
Write Data 2*	CWSBFADC,CWSINCR,CWSIVLR	64
Read Count	CWSPLHAD	80
Read Data	CWSBFADC,CWSINCR	24
Read Back Check**	CWSBFADC,CWSINCR	24

*Write Data 1 is the second part of the Format Write (CKD) CCW set and follows the Write Count CCW. The Write Count CCW is data chained to the Write Data CCW.

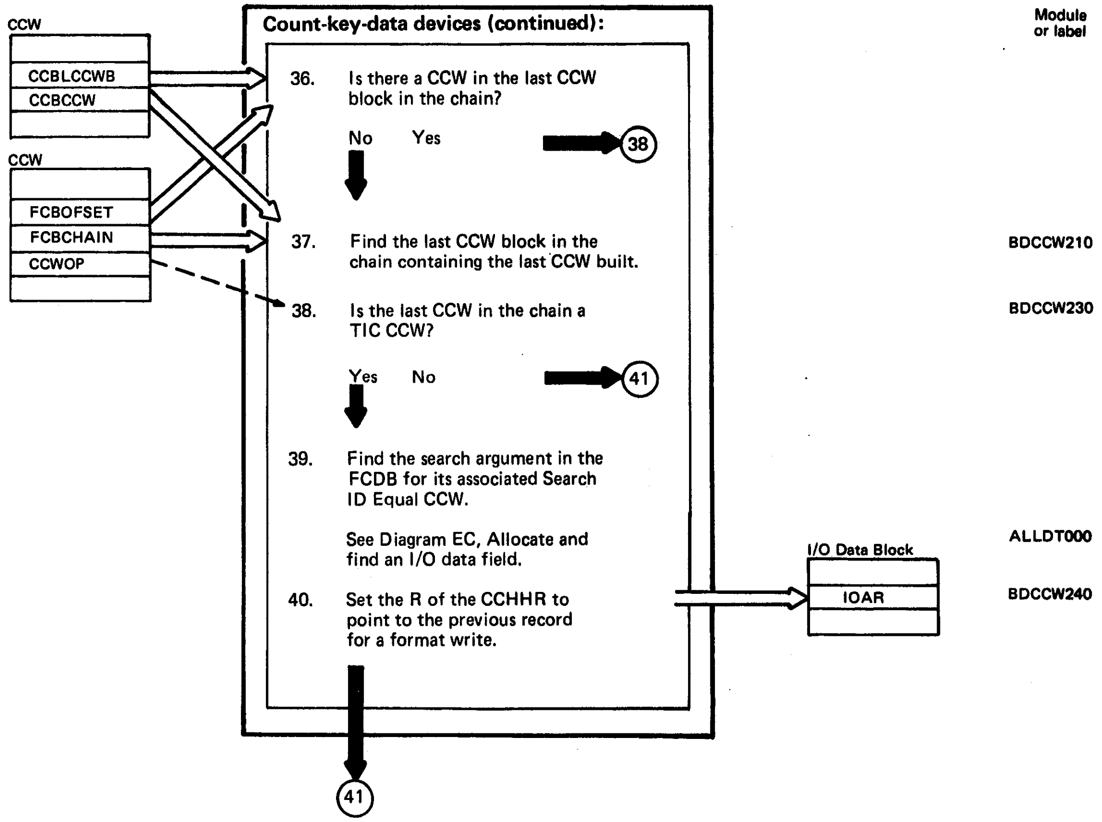
Write Data 2 is the CCW skeleton used for updating an existing DASD record.

**The Read Back Check CCW is a Read Data CCW with the SKIP bit on.

NOTE:

- The Search ID Equal – TIC CCW skeletons are processed as a pair of CCWs and must be in the same CCW block.
- The Read Count and Read Data CCW skeletons are processed as a pair for replicated reads, but they need not be in the same CCW block. The Read Count CCW only reads the CCHH field.
- The Write Count and Write Data CCW skeletons are processed as a pair for format write operations, but they need not be in the same CCW block.
- The Set Sector CCW skeleton is preceded by a Seek CCW. They are processed as a pair and are always in the same CCW block because they are the first two CCWs of any new channel program for RPS devices.

Diagram EF9. I/O Manager: Build CCW



36. Is there a CCW in the last CCW block in the chain?

37. Find the last CCW block in the chain containing the last CCW built.

38. Is the last CCW in the chain a TIC CCW?

39. Find the search argument in the FCDB for its associated Search ID Equal CCW.

See Diagram EC, Allocate and find an I/O data field.

40. Set the R of the CCHHR to point to the previous record for a format write.

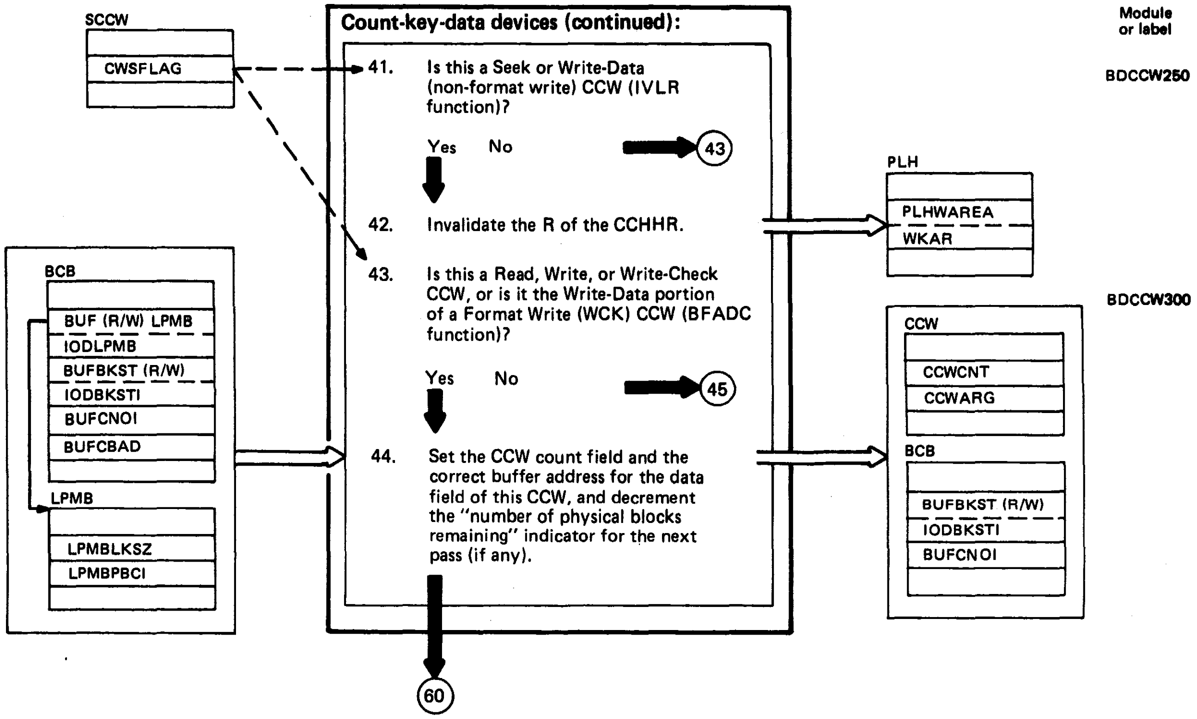
Notes for Diagram EF9

35.-40. When the Write Count of a Format Write CCW sequence is being processed, the previous CCW is checked for a TIC op code. If the Write Count is the first CCW in a new CCW block, the CCW block chain is searched until the block preceding the current block is found, and the last CCW in the preceding block is checked for a TIC op code.

If the previous CCW is a TIC CCW, the Search ID Equal argument in the I/O Data Block is set to search on the ID field of the record preceding the current record that is to be written.

If the previous CCW is not a TIC CCW, then it has already been set up to write the record preceding the record to be written, and orientation has been established.

Diagram EF10. I/O Manager: Build CCW



41. Is this a Seek or Write-Data (non-format write) CCW (IVLR function)?

42. Invalidate the R of the CCHHR.

43. Is this a Read, Write, or Write-Check CCW, or is it the Write-Data portion of a Format Write (WCK) CCW (BFADC function)?

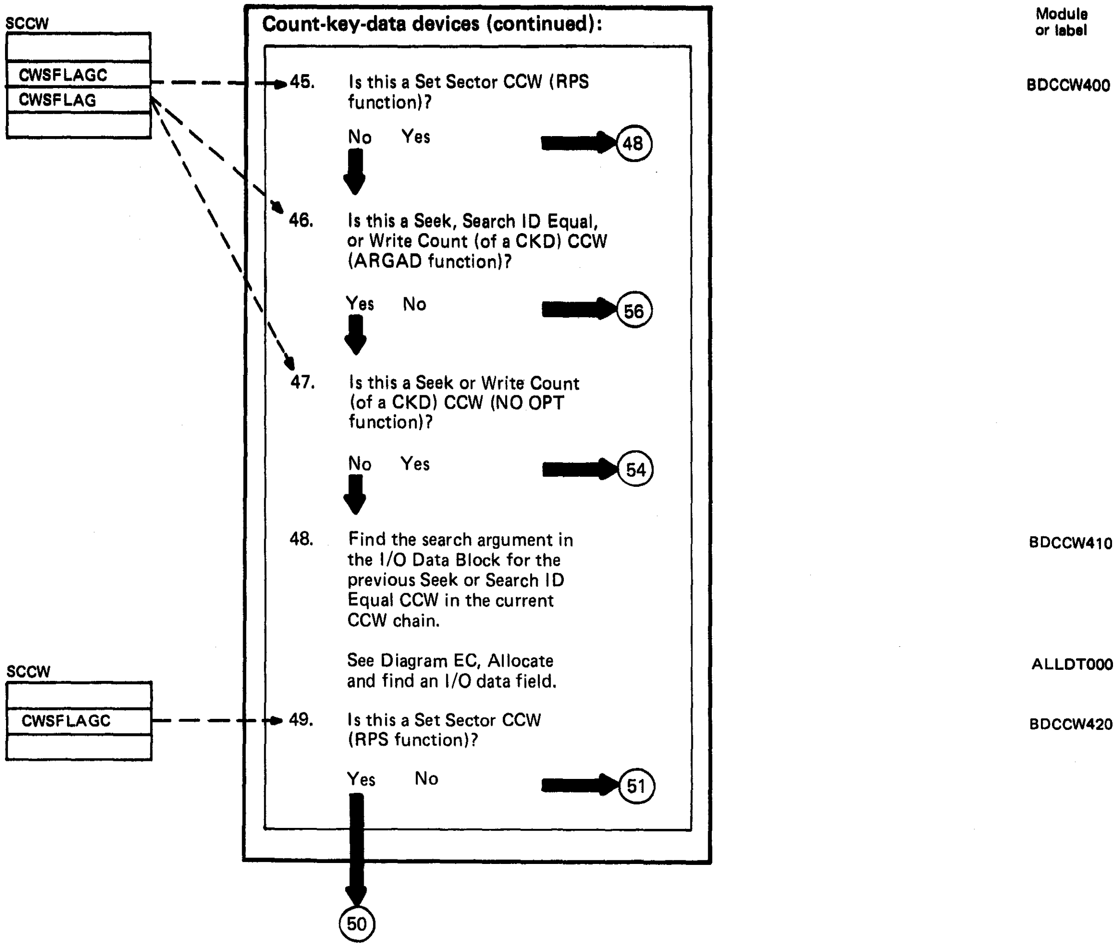
44. Set the CCW count field and the correct buffer address for the data field of this CCW, and decrement the "number of physical blocks remaining" indicator for the next pass (if any).

Notes for Diagram EF10

41.-42. If the current CCW is a Seek CCW or a Write Data CCW, the R byte (WKAR) in the PLHWAREA is invalidated for the next Write/Read CCW. This forces a Search ID Equal – TIC CCW sequence to be built. (Write Data CCWs must be oriented by a Search CCW.)

43.-44. If the CCW being processed is a Write/Read CCW (or a Write Data from the Write CKD), then the CCW count field is initialized with the physical blocksize, and the position in the buffer to be used as I/O argument address (BUFCNOI) is determined and saved in register 9. The "number of physical blocks to write" (BUFBKSTW) indicator is decremented by one. (This value is normally one except when writing replicated index records or when doing preformat writes.)

Diagram EF11. I/O Manager: Build CCW



Count-key-data devices (continued):

45. Is this a Set Sector CCW (RPS function)? BDCCW400

No Yes → 48

46. Is this a Seek, Search ID Equal, or Write Count (of a CKD) CCW (ARGAD function)?

Yes No → 56

47. Is this a Seek or Write Count (of a CKD) CCW (NO OPT function)?

No Yes → 54

48. Find the search argument in the I/O Data Block for the previous Seek or Search ID Equal CCW in the current CCW chain. BDCCW410

See Diagram EC, Allocate and find an I/O data field. ALLDT000

49. Is this a Set Sector CCW (RPS function)? BDCCW420

Yes No → 51

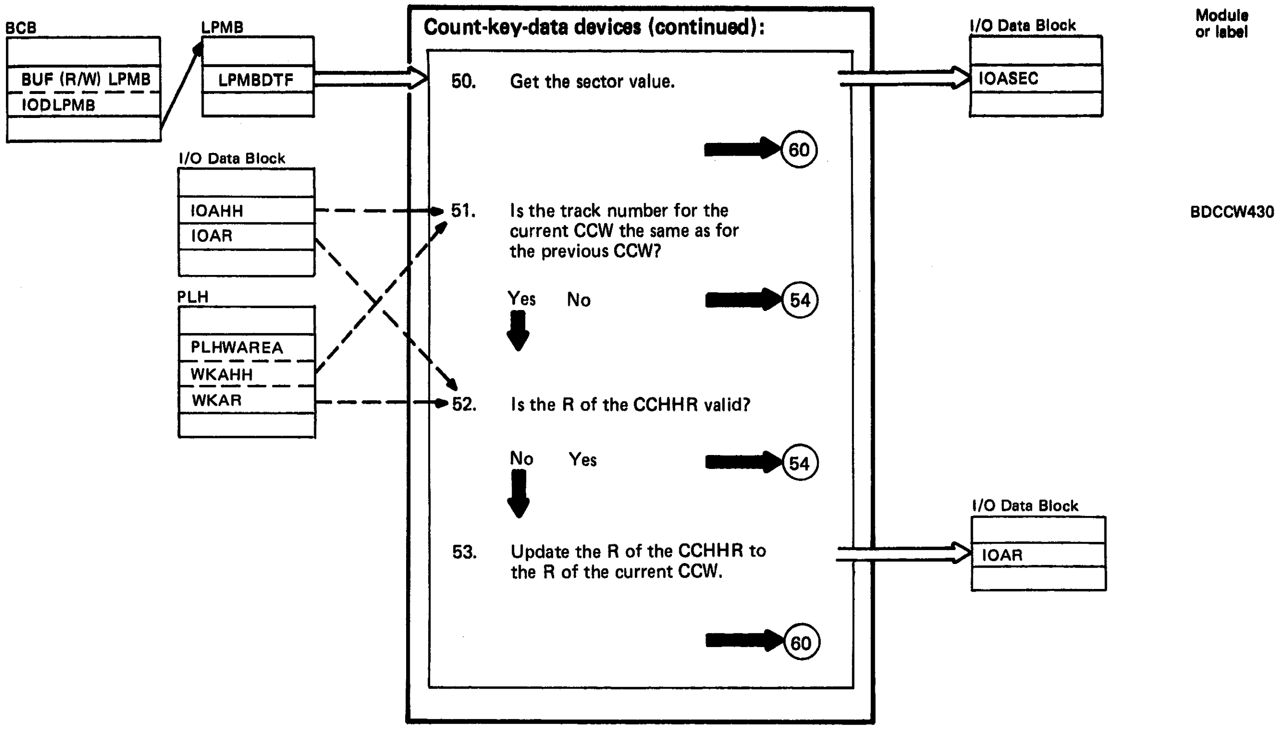
Yes → 50

Notes for Diagram EF11

45.-53. If a Set Sector or a Search ID Equal CCW is being processed, the last I/O Data Block arguments are located. For a Set Sector CCW, the device type code is placed in the I/O Data Block (IOASEC) for the SECTVAL (SVC 75) function.

For a Search ID Equal CCW following a Seek (Set Sector) CCW, the R field in the I/O Data Block is updated, when the Search ID Equal CCW is for the same track as the Seek CCW.

Diagram EF12. I/O Manager: Build CCW



Count-key-data devices (continued):

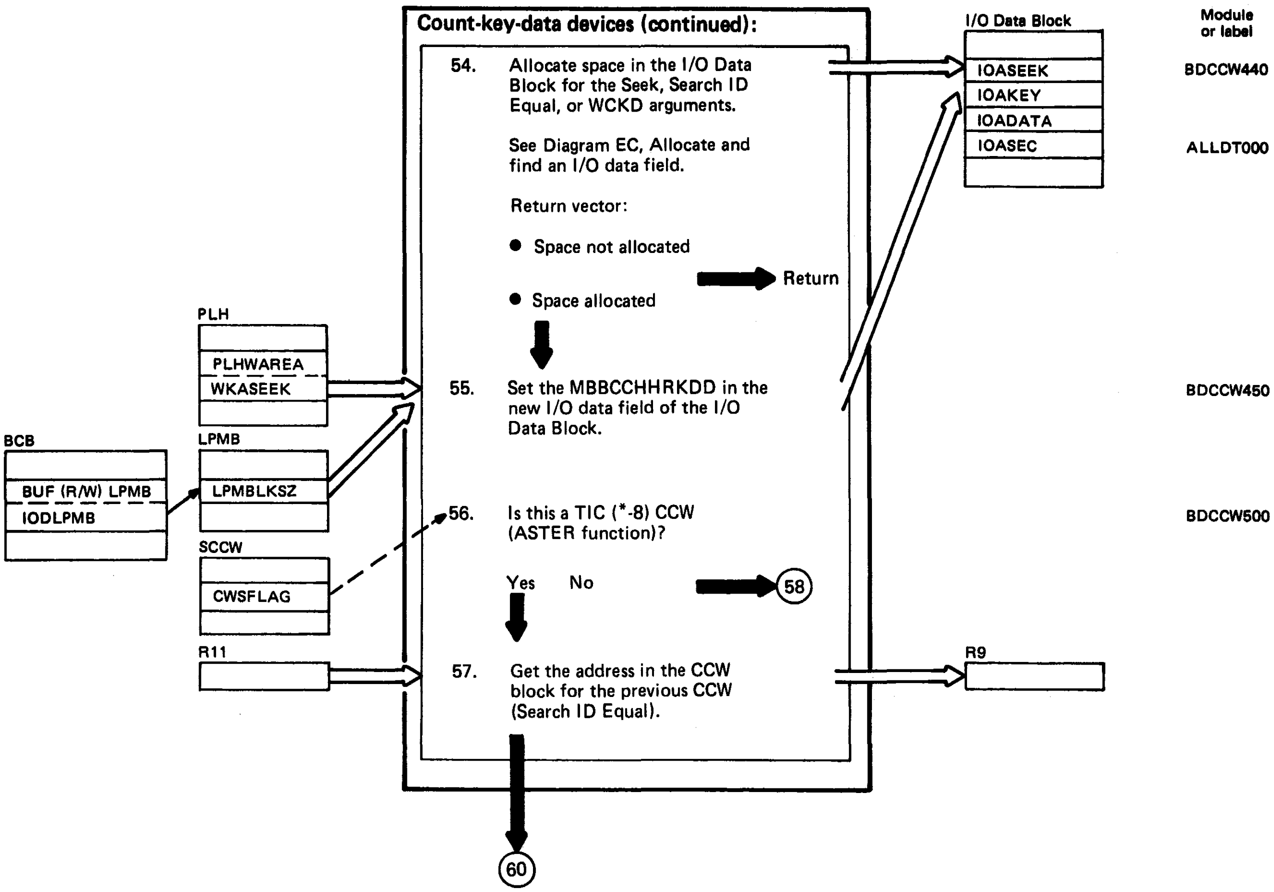
50. Get the sector value.

51. Is the track number for the current CCW the same as for the previous CCW?

52. Is the R of the CCHHR valid?

53. Update the R of the CCHHR to the R of the current CCW.

Diagram EF13. I/O Manager: Build CCW



Count-key-data devices (continued):

54. Allocate space in the I/O Data Block for the Seek, Search ID Equal, or WCKD arguments.

See Diagram EC, Allocate and find an I/O data field.

Return vector:

- Space not allocated → Return
- Space allocated

55. Set the MBBCCHHRKDD in the new I/O data field of the I/O Data Block.

56. Is this a TIC (*-8) CCW (ASTER function)?

Yes No → 58

57. Get the address in the CCW block for the previous CCW (Search ID Equal).

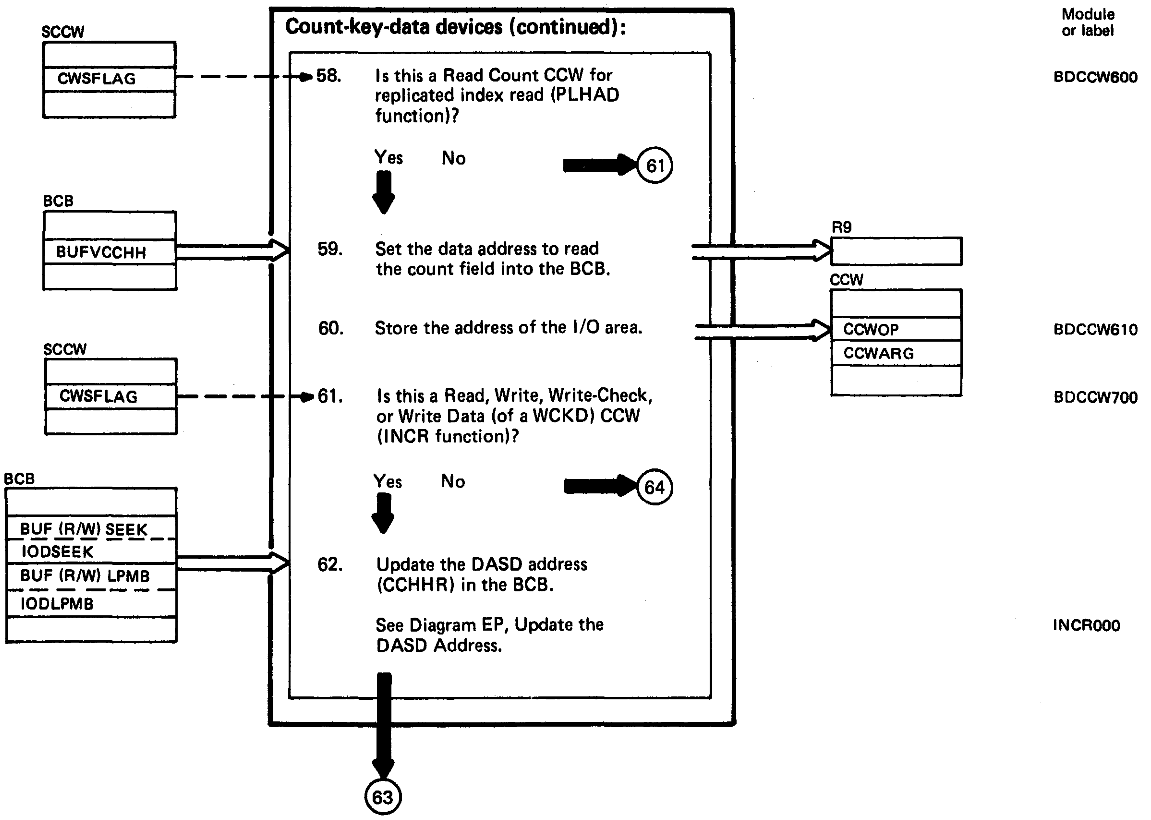
→ 60

Notes for Diagram EF13

54.-55. If a Seek, Write Count, or a Search ID Equal CCW (which does not follow a Seek CCW to the same track) is being processed, a new I/O argument field is allocated in the I/O Data Block. The MBBCCHHRKDD field (K=0) is initialized with the DASD address for the CCW and the physical blocksize.

56.-57. If a TIC CCW is being processed (Search ID Equal – TIC sequence), R9 is set to the address of the TIC CCW. It will be adjusted to point to the Search ID Equal CCW (see description of CWSASTER bit in CCW Skeleton).

Diagram EF14. I/O Manager: Build CCW



Count-key-data devices (continued):

58. Is this a Read Count CCW for replicated index read (PLHAD function)?

59. Set the data address to read the count field into the BCB.

60. Store the address of the I/O area.

61. Is this a Read, Write, Write-Check, or Write Data (of a WCKD) CCW (INCR function)?

62. Update the DASD address (CCHHR) in the BCB.

See Diagram EP, Update the DASD Address.

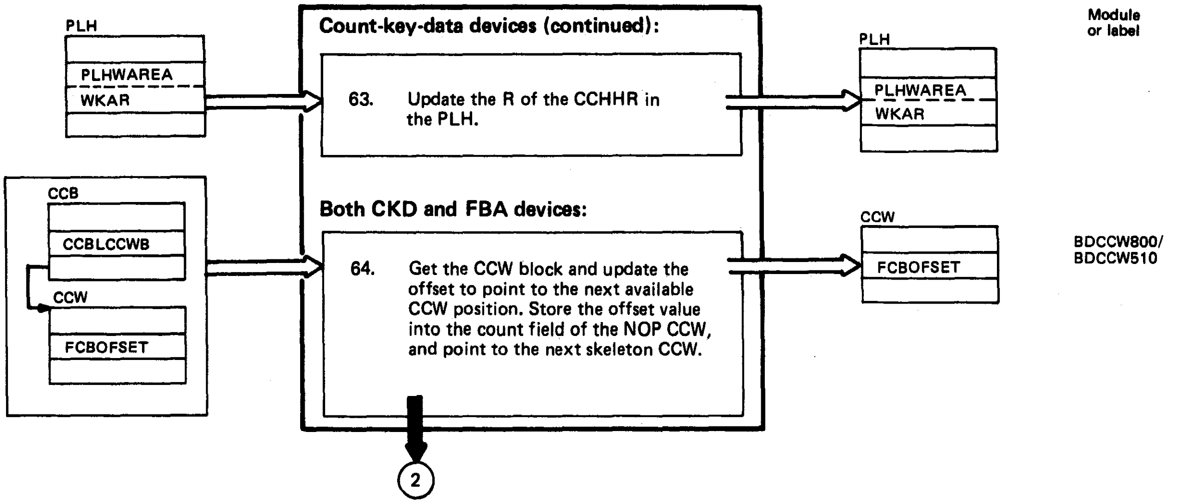
Notes for Diagram EF14

58.-59. If a Read Count CCW is being processed (for replicated index or sequence set), R9 is initialized to point to the BCB (BUFVCCHH) so that the count field can be read into the BCB.

60. The I/O argument address obtained in step 44 (for the buffer) or steps 54-59 from the I/O Data Block allocation (in R9) is placed in the CCW argument field.

62.-63. The Update DASD Address routine increments the CCHHR in the BCB to point to the next sequential record, and the R byte (WKAR) in the PLHWAREA field is incremented by one.

Diagram EF15. I/O Manager: Build CCW



63. Update the R of the CCHHR in the PLH.

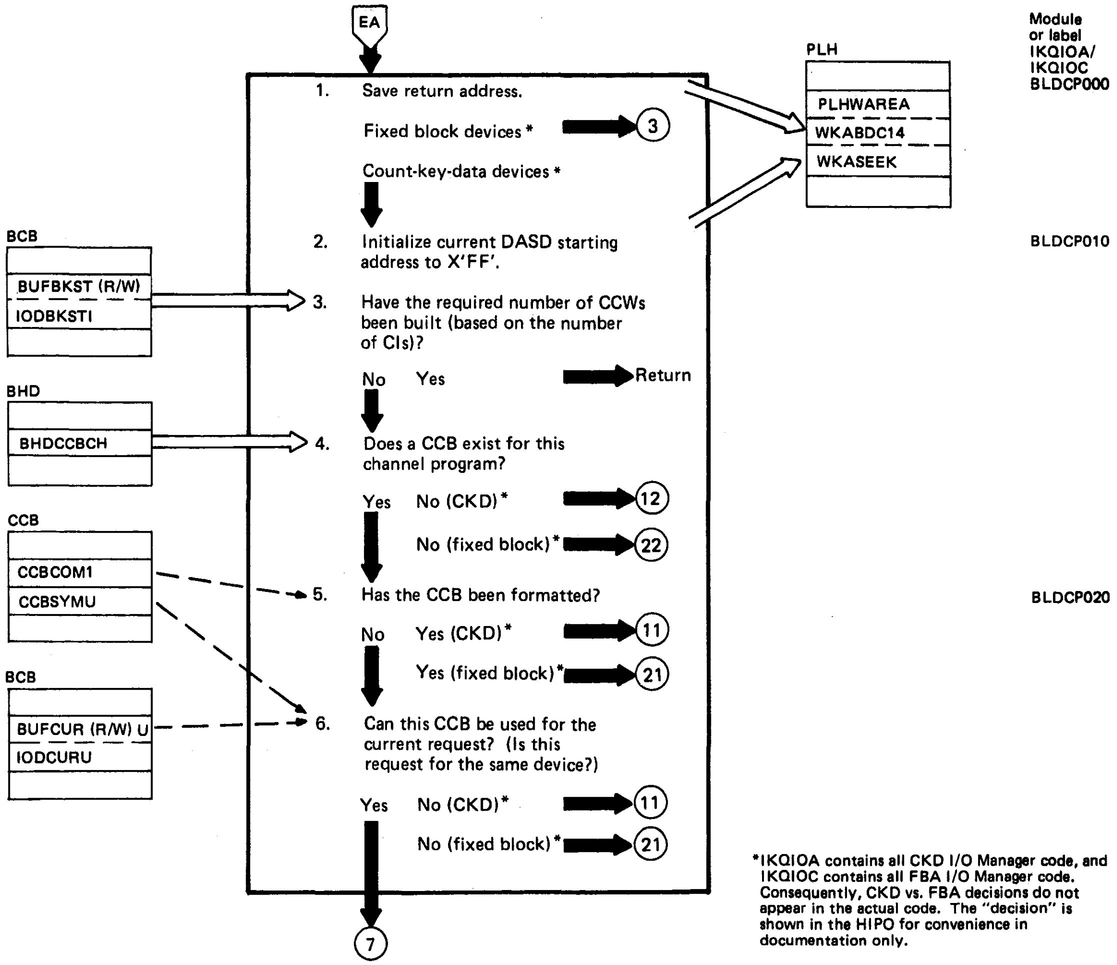
64. Get the CCW block and update the offset to point to the next available CCW position. Store the offset value into the count field of the NOP CCW, and point to the next skeleton CCW.

Notes for Diagram EF15

64. The offset in the CCW block is used as the count field for a NOP CCW at the end of the CCW chain. This is done so that the I/O Error Handler (IKQIOB) can process both CKD and fixed block devices.

    The pointer to the current CCW skeleton is incremented by 11 to point to the next CCW skeleton (if any) that may be associated with the current CCW skeleton.

Diagram EG1. I/O Manager: Build Channel Program



*IKQIOA contains all CKD I/O Manager code, and IKQIOC contains all FBA I/O Manager code. Consequently, CKD vs. FBA decisions do not appear in the actual code. The "decision" is shown in the HIPO for convenience in documentation only.

Notes for Diagram EG1

3. The return to the user is not taken the first time through this routine.

   The value referenced by IODBKSTI is either 1 (as set by RBA convert) or the value computed in step 31 of Diagram EA: I/O Manager Mainline. This value controls how many CCWs are built for the buffer being processed during this pass through BLDCP.

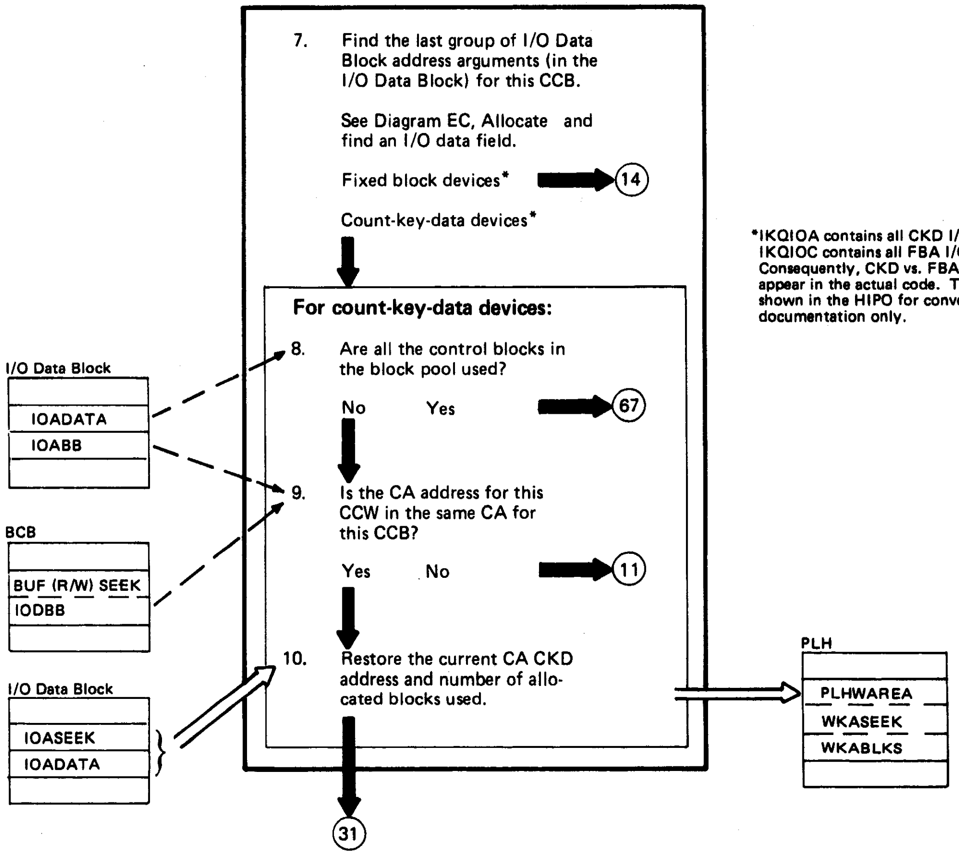
4. The first time this routine is called for a given set of buffers, no CCBs, CCWs, etc. will have been built.

   On succeeding passes, at least one channel program will have been started, and processing continues at BLDCP020.

5. If a CCB, CCW, etc. group has been completed and I/O started for some of the buffers in the set being processed (that is, for one CA), then bit CCBERROK in CCBCOM1 will be set on, and processing continues at BLDCP080.

6. A test is made to see if the buffer being processed is associated with the non-formatted CCB tested in step 5.

Diagram EG2. I/O Manager: Build Channel Program



7. Find the last group of I/O Data Block address arguments (in the I/O Data Block) for this CCB.

See Diagram EC, Allocate and find an I/O data field.

Fixed block devices*

Count-key-data devices*

For count-key-data devices:

8. Are all the control blocks in the block pool used?

No Yes

9. Is the CA address for this CCW in the same CA for this CCB?

Yes No

10. Restore the current CA CKD address and number of allocated blocks used.

Module or label

ALLDT000

*IKQIOA contains all CKD I/O Manager code, and IKQIOC contains all FBA I/O Manager code. Consequently, CKD vs. FBA decisions do not appear in the actual code. The "decision" is shown in the HIPO for convenience in documentation only.

BLDCP030

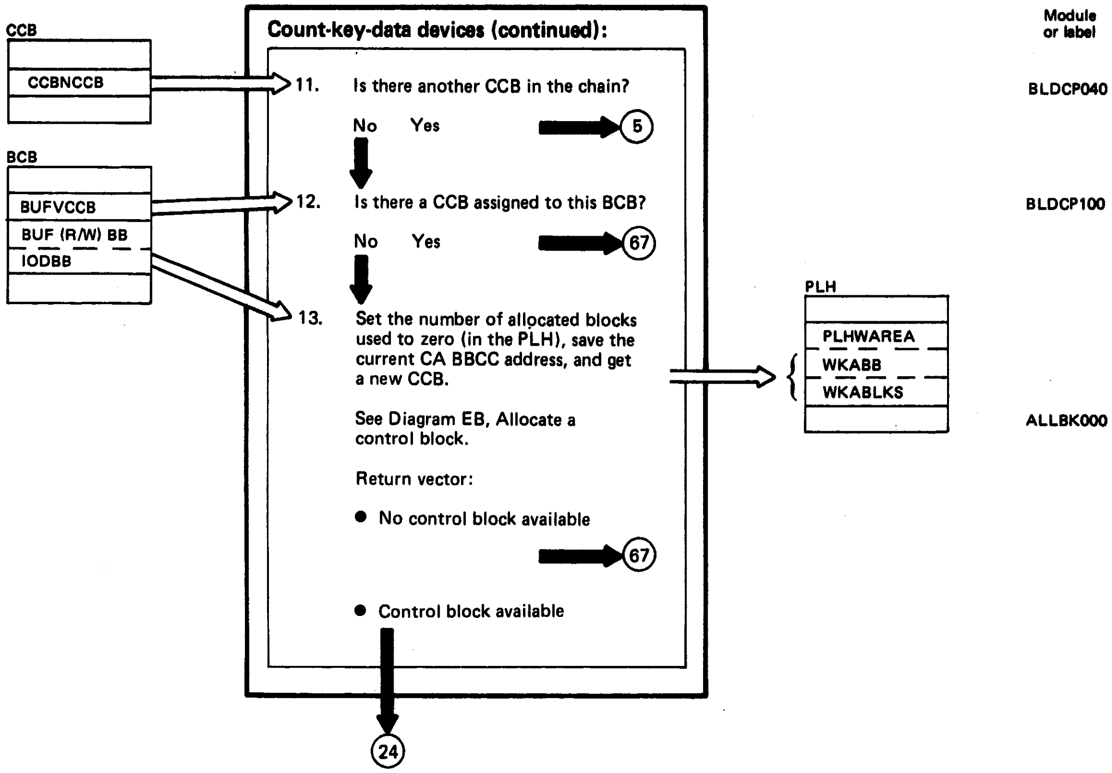
Notes for Diagram EG2

7. If the buffer being processed is associated with the non-formatted CCB tested in step 5, a search is made of the I/O Data Block (FCDB) for the last arguments stored in the block (during step 62 or 63). These arguments are: the number of allocated control blocks used, and the CA DASD address.

8. If the number of blocks allocated for the CCB exceeds 31, the channel program is started and waited for.

9.-10. A test is made to see if the buffer being processed is for the CA associated with the CCB (tested in step 5). If it is for that CCB, the CA address and the number of allocated blocks used are saved in the PLH for use in building the CCW(s) for this buffer.

Diagram EG3. I/O Manager: Build Channel Program



11. Is there another CCB in the chain?

12. Is there a CCB assigned to this BCB?

13. Set the number of allocated blocks used to zero (in the PLH), save the current CA BBCC address, and get a new CCB.

See Diagram EB, Allocate a control block.

Return vector:

- No control block available
- Control block available

Notes for Diagram EG3

11. If the CA address of the buffer being processed did not match the CA of the CCB (tested in step 5), the next CCB (if any) in the chain is checked to determine if it can be used for the buffer.

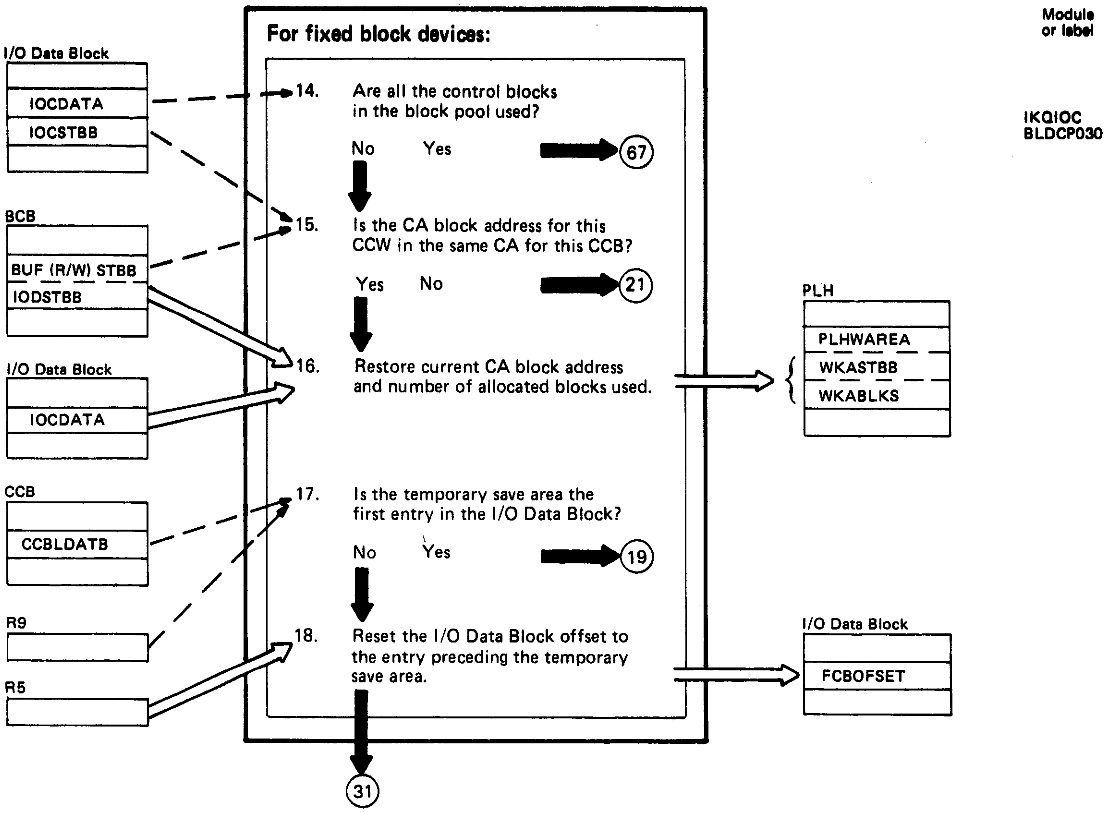
12. This step tests whether the buffer, for which no CCB can be found in the CCB chain, has a CCB assigned to it. The buffer has its CCB address stored in it (BUFVCCB) whenever a "piggy-back" I/O operation is performed. (A "piggy-back" operation occurs when one buffer is used for both writing and reading in the same channel program. The contents of the buffer are first written out, and then data is read into it during the same I/O operation.)

    If the buffer has a CCB address stored in it, then the read address is in a different CA then the write address, and the read is not handled in the same EXCP/WAIT operation.

13. If there is no CCB associated with the buffer being processed, the PLH is reinitialized with the CA address of this buffer, and the "number of allocated blocks in use" counter is set to zero.

    The allocate block routine is called to get a 64-byte block to be used as a CCB for the buffer. If no block is available, an EXCP and WAIT are issued for the current set of channel programs to free the control blocks currently in use.

Diagram EG4. I/O Manager: Build Channel Program



Module or label
For fixed block devices:
I/O Data Block
IOCDATA
IOCSTBB
BCB
BUF (R/W) STBB
IODSTBB
I/O Data Block
IOCDATA
CCB
CCBLDATB
R9
R5
14. Are all the control blocks in the block pool used?
No Yes
67
IKQIOC
BLDCP030
15. Is the CA block address for this CCW in the same CA for this CCB?
Yes No
21
16. Restore current CA block address and number of allocated blocks used.
PLH
PLHWAREA
WKASTBB
WKABLKS
17. Is the temporary save area the first entry in the I/O Data Block?
No Yes
19
18. Reset the I/O Data Block offset to the entry preceding the temporary save area.
I/O Data Block
FCBOFSET
31

Notes for Diagram EG4

14. See note for step 8.

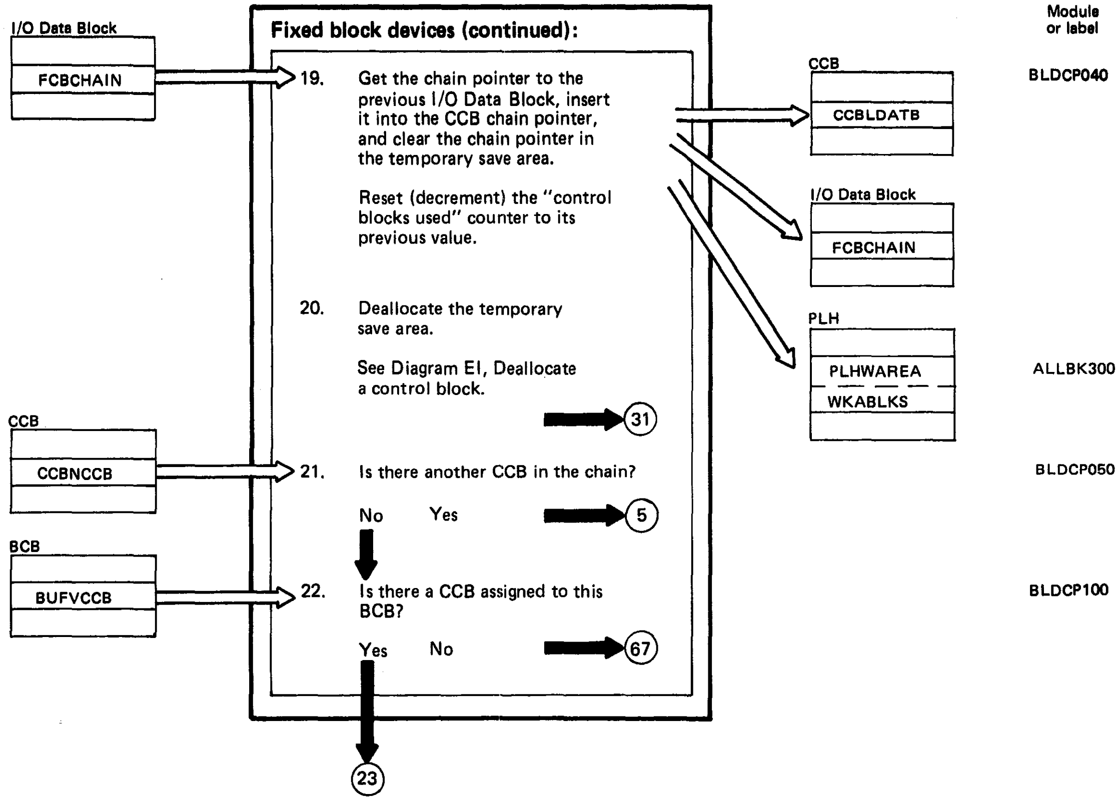
15.-16. See note for steps 9-10.

17. After the data in the temporary save area in the I/O Data Block is obtained, the save area is available for reuse, and the offset is repositioned to reflect the available space.

This must be done because if the next "allocate" function is a "find," the correct arguments must be obtained (not the data in the temporary save area). When the temporary save area is the first entry in the I/O Data Block, then the data block must be dequeued and the control blocks and offset restored to indicate the last entry in the previous block.

18. This step backs up the offset in the I/O Data Block (because the save area is not the first entry) and clears the indicator (in FCBALI) so it no longer indicates that the previous call to ALLDT was for a temporary save area.

Diagram EG5. I/O Manager: Build Channel Program



Fixed block devices (continued):

19. Get the chain pointer to the previous I/O Data Block, insert it into the CCB chain pointer, and clear the chain pointer in the temporary save area.

    Reset (decrement) the "control blocks used" counter to its previous value.

20. Deallocate the temporary save area.

    See Diagram EI, Deallocate a control block.

21. Is there another CCB in the chain?

22. Is there a CCB assigned to this BCB?

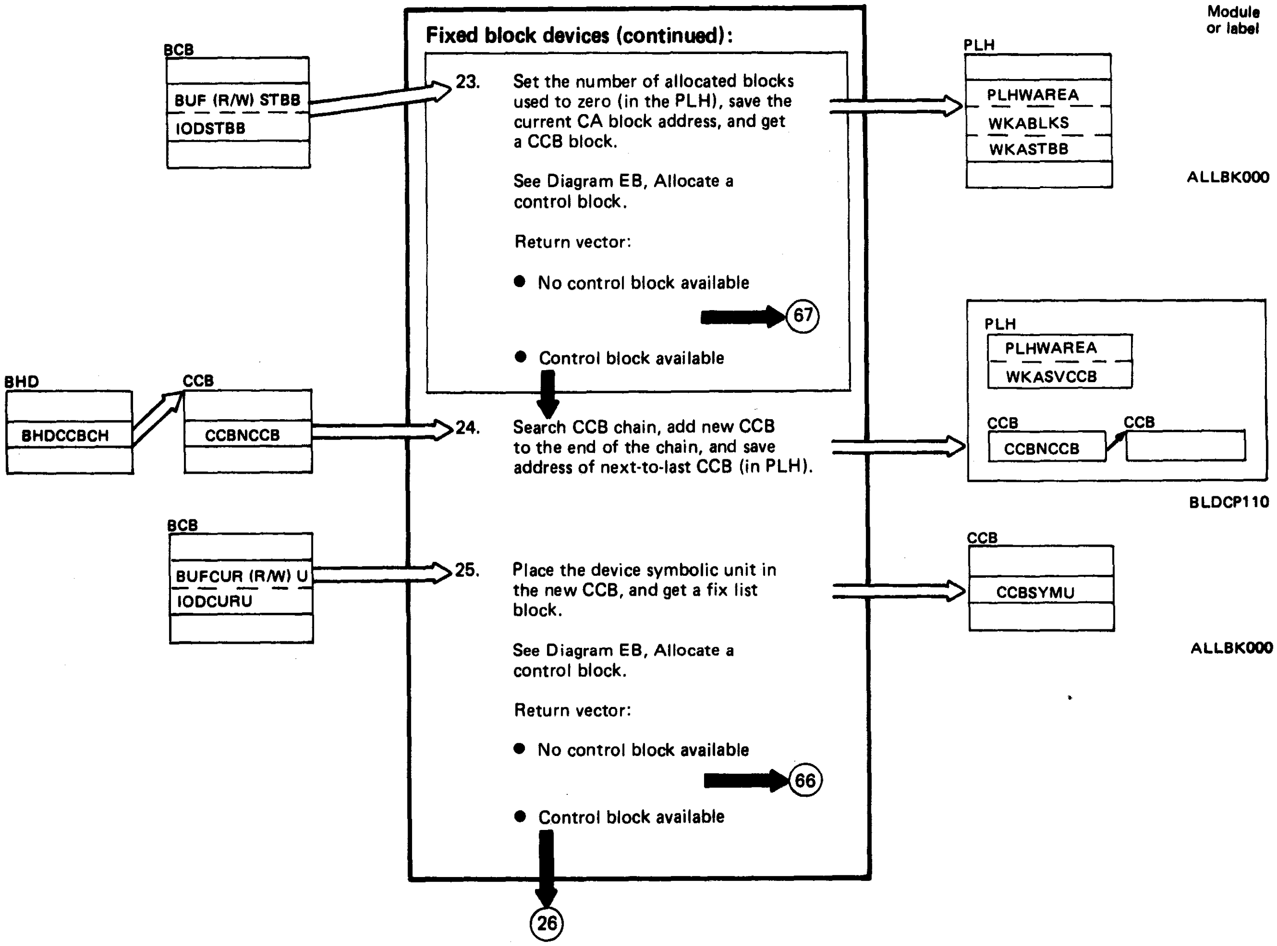
Notes for Diagram EG5

19.-20. The block allocated for the temporary save area points to the previous block used. Therefore the pointer is saved in the CCB, and the pointer field is cleared. The temporary save area block is returned to the list of available control blocks.

21. See note for step 11.

22. See note for step 12.

Diagram EG6. I/O Manager: Build Channel Program



Fixed block devices (continued):

23. Set the number of allocated blocks used to zero (in the PLH), save the current CA block address, and get a CCB block.

    See Diagram EB, Allocate a control block.

    Return vector:

    - No control block available
    - Control block available

24. Search CCB chain, add new CCB to the end of the chain, and save address of next-to-last CCB (in PLH).

25. Place the device symbolic unit in the new CCB, and get a fix list block.

    See Diagram EB, Allocate a control block.

    Return vector:

    - No control block available
    - Control block available

Notes for Diagram EG6

23. See note for step 13.

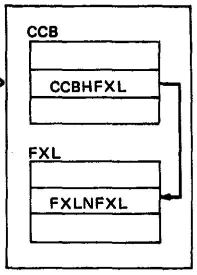
24.-25. If a block is allocated, it is added to the end of the CCB chain, the symbolic unit field is initialized, and the address of the CCB that points to the new block (or its BHD) is saved in the PLH.

In step 25, a Fix List is not obtained if execution is on a S/370.

NOTE: The address of the CCB that points to the new CCB is saved for the purpose of backing out of the chain whenever a Fix List, CCW block, or I/O Data Block cannot be initially allocated.

Diagram EG7. I/O Manager: Build Channel Program

26. Save the address of the Fix List (in the CCB), and set the Fix List end-of-chain indicator.



27. Get a new CCW block.

See Diagram EB, Allocate a control block.

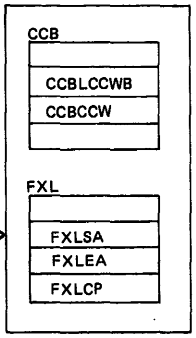
Return vector:

- No control block available → 65
- Control block available

28. Save the addresses of the last CCW block and the beginning of the first CCW slot in the CCB. Put an entry in the Fix List for the CCW block.

See Diagram EN, Make a Fix List entry.

29



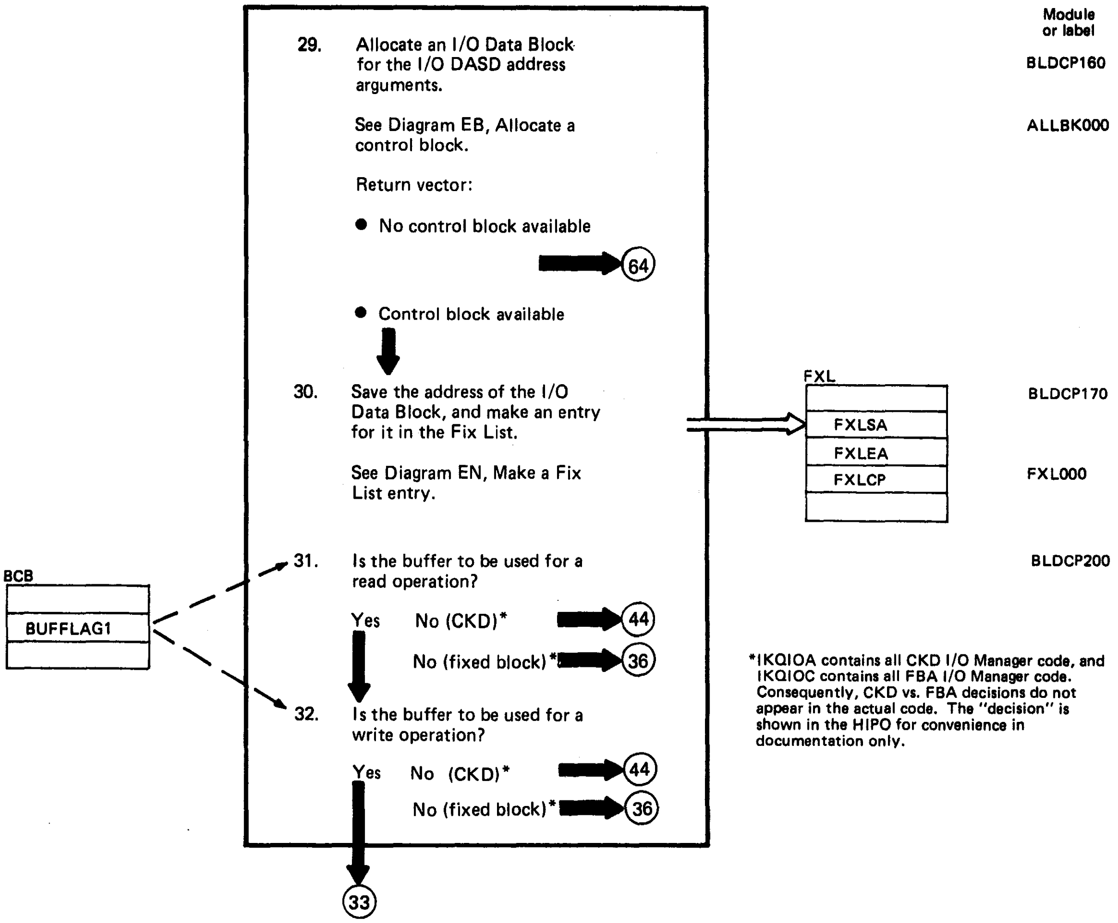
Module or label
BLDCP130
BLDCP140
ALLBK000
BLDCP150
FXL000

Notes for Diagram EG7

26.-30. A Fix List is allocated for the new CCB and initialized to indicate that it is the last Fix List in a Fix List chain. Next, a CCW and an I/O Data Block are allocated, and entries are made for them in the Fix List. Pointers to the Fix List, CCW, and I/O Data Block are stored in the CCB.

If a Fix List cannot be allocated, I/O is performed to free the currently allocated control blocks for reuse.

Diagram EG8. I/O Manager: Build Channel Program



29. Allocate an I/O Data Block for the I/O DASD address arguments.

See Diagram EB, Allocate a control block.

Return vector:

- No control block available
- Control block available

30. Save the address of the I/O Data Block, and make an entry for it in the Fix List.

See Diagram EN, Make a Fix List entry.

31. Is the buffer to be used for a read operation?

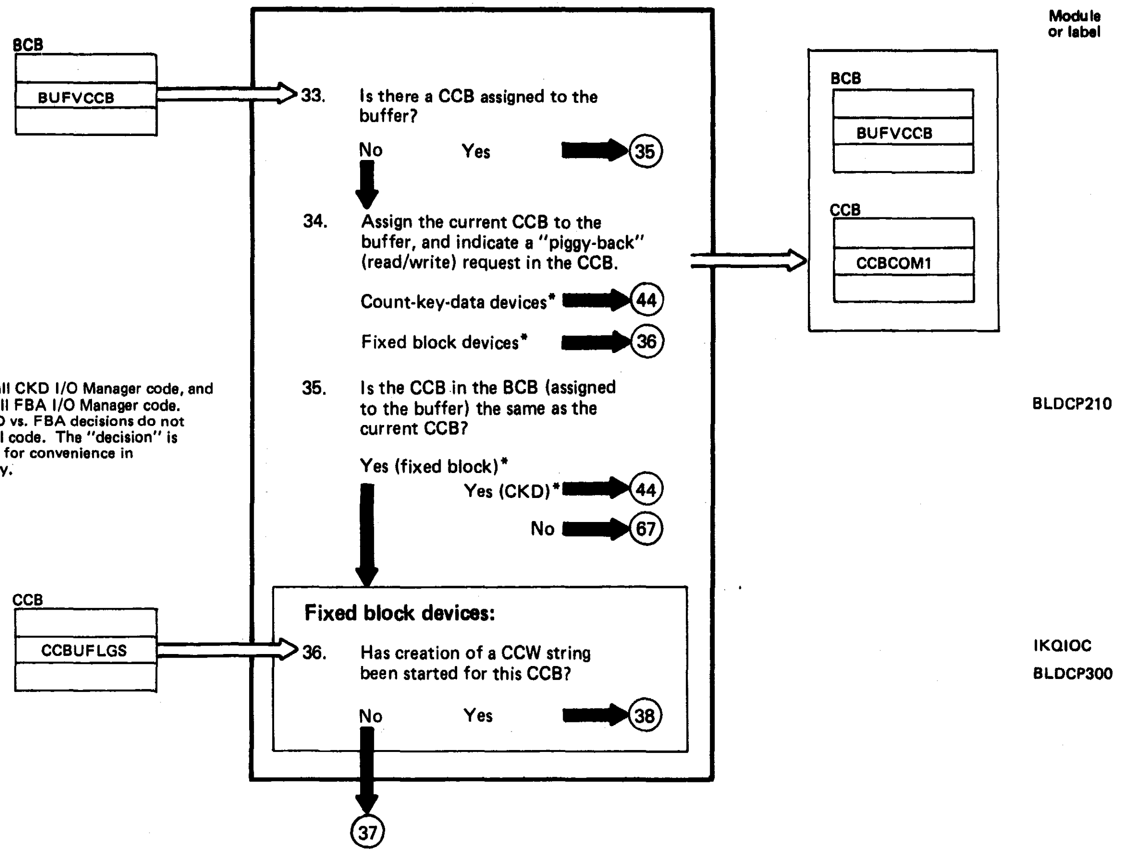
32. Is the buffer to be used for a write operation?

*IKQIOA contains all CKD I/O Manager code, and IKQIOC contains all FBA I/O Manager code. Consequently, CKD vs. FBA decisions do not appear in the actual code. The "decision" is shown in the HIPO for convenience in documentation only.

Notes for Diagram EG8

31.-32. These steps determine if the buffer is to be used for both a write and read I/O operation ("piggy-back" I/O).

Diagram EG9. I/O Manager: Build Channel Program


Module or label
BCB
BUFVCCB
33. Is there a CCB assigned to the buffer?
No
Yes
35
34. Assign the current CCB to the buffer, and indicate a "piggy-back" (read/write) request in the CCB.
Count-key-data devices*
44
Fixed block devices*
36
BCB
BUFVCCB
CCB
CCBCOM1
35. Is the CCB in the BCB (assigned to the buffer) the same as the current CCB?
BLDCP210
Yes (fixed block)*
Yes (CKD)*
44
No
67
CCB
CCBUFLGS
Fixed block devices:
36. Has creation of a CCW string been started for this CCB?
IKQIOC
BLDCP300
No
Yes
38
37

*IKQIOA contains all CKD I/O Manager code, and IKQIOC contains all FBA I/O Manager code. Consequently, CKD vs. FBA decisions do not appear in the actual code. The "decision" is shown in the HIPO for convenience in documentation only.

Notes for Diagram EG9

33.-34. A CCB is assigned to the buffer (if one is not already assigned), and the bit CCBUERR is set to indicate that this buffer can be used for a "piggy-back" operation. This occurs during the write pass for building the channel program.

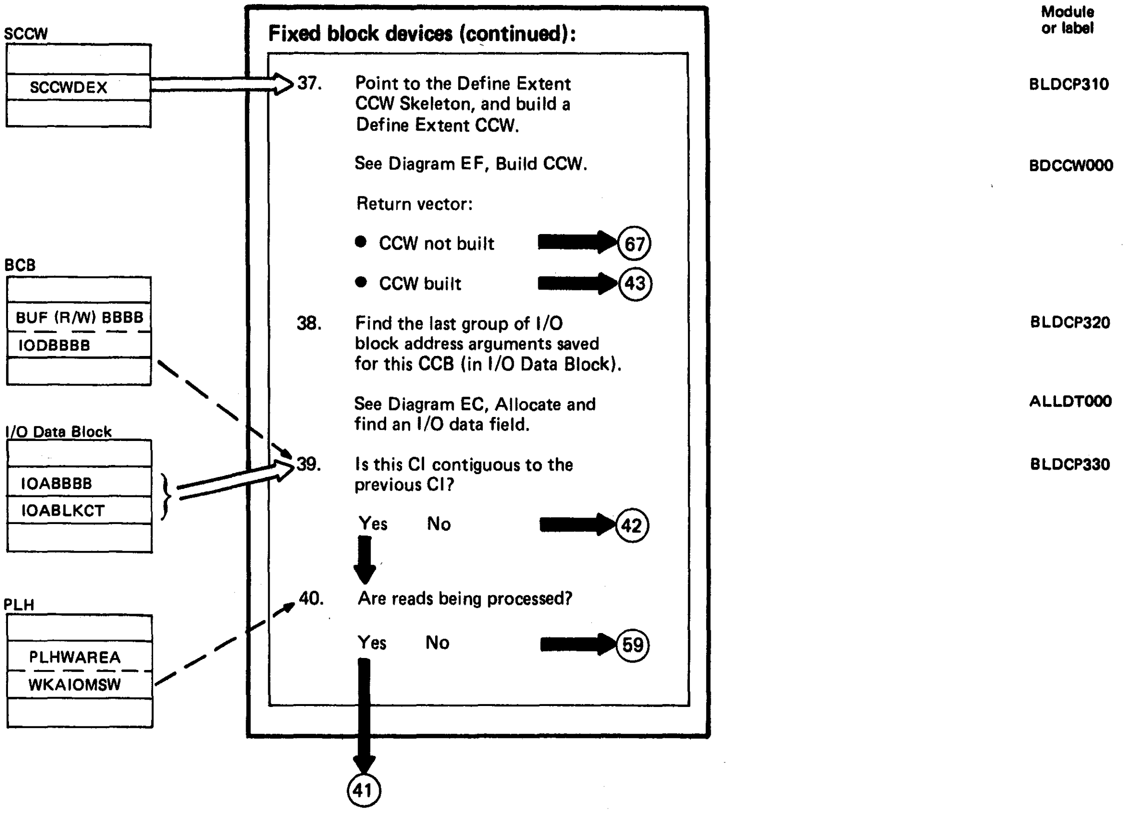
35. If a CCB for a write is already assigned to the buffer, it is compared to the CCB (for a read) currently being processed for this buffer. If they do not match, it means that the write and read do not apply to the same CA (or cylinder). Therefore the buffer cannot be used for both writing and reading in the same CCW chain ("piggy-back" I/O). I/O is executed for all writes and any existing reads in the current CCB chain.

NOTE: This condition can occur only if the BCB has previously been processed for a write operation and is to be used for a read operation also. For CKD devices, this occurs on a cylinder boundary because the DASD file protect feature inhibits cylinder-switching seek operations in the middle of a channel program.

36.-37. The first time a CCW string is started for a CCB, a Define Extent CCW must be built. The CCBRPS bit in CCBUFLGS indicates the beginning of a new CCW string.

A Locate CCW is built after the Define Extent CCW. If the Define Extent CCW was not built (due to lack of control blocks), the current chain of CCBs will be EXCPed (or started for I/O) to free any allocated control blocks for reuse.

Diagram EG10. I/O Manager: Build Channel Program



Fixed block devices (continued):

37. Point to the Define Extent CCW Skeleton, and build a Define Extent CCW. — BLDCP310

See Diagram EF, Build CCW. — BDCCW000

Return vector:

- CCW not built → 67
- CCW built → 43

38. Find the last group of I/O block address arguments saved for this CCB (in I/O Data Block). — BLDCP320

See Diagram EC, Allocate and find an I/O data field. — ALLDT000

39. Is this CI contiguous to the previous CI? — BLDCP330

Yes No → 42

40. Are reads being processed?

Yes No → 59

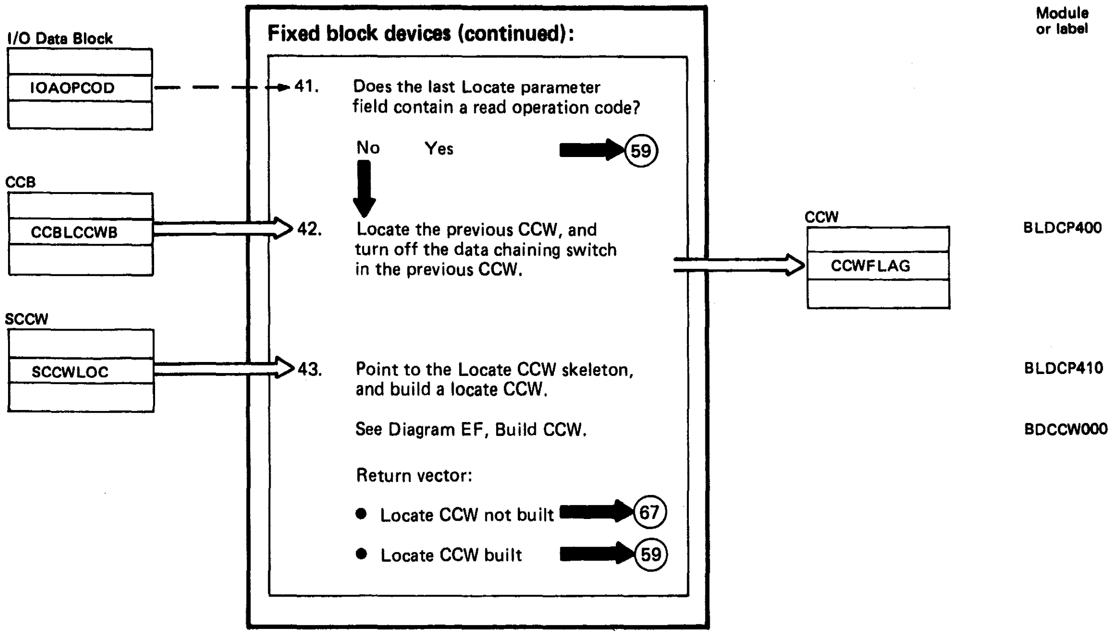
Notes for Diagram EG10

38.-41. During Write/Read CCW processing for fixed block devices, the last set of Locate arguments for this CCW chain are found to determine if the new Write/Read CCW can be added to the end of the current CCW string, or whether a new Locate CCW must precede the Write/Read CCW.

A Locate CCW is required if Write CCWs are being processed, and if the CI for the current buffer does not physically follow the CI associated with the last Write CCW in the chain.

If a Read CCW is being processed, a Locate CCW is required when the CI is not contiguous to the CI of the previous Read CCW, or if the previous CCW was a Write CCW (even when the CIs are contiguous).

Diagram EG11. I/O Manager: Build Channel Program

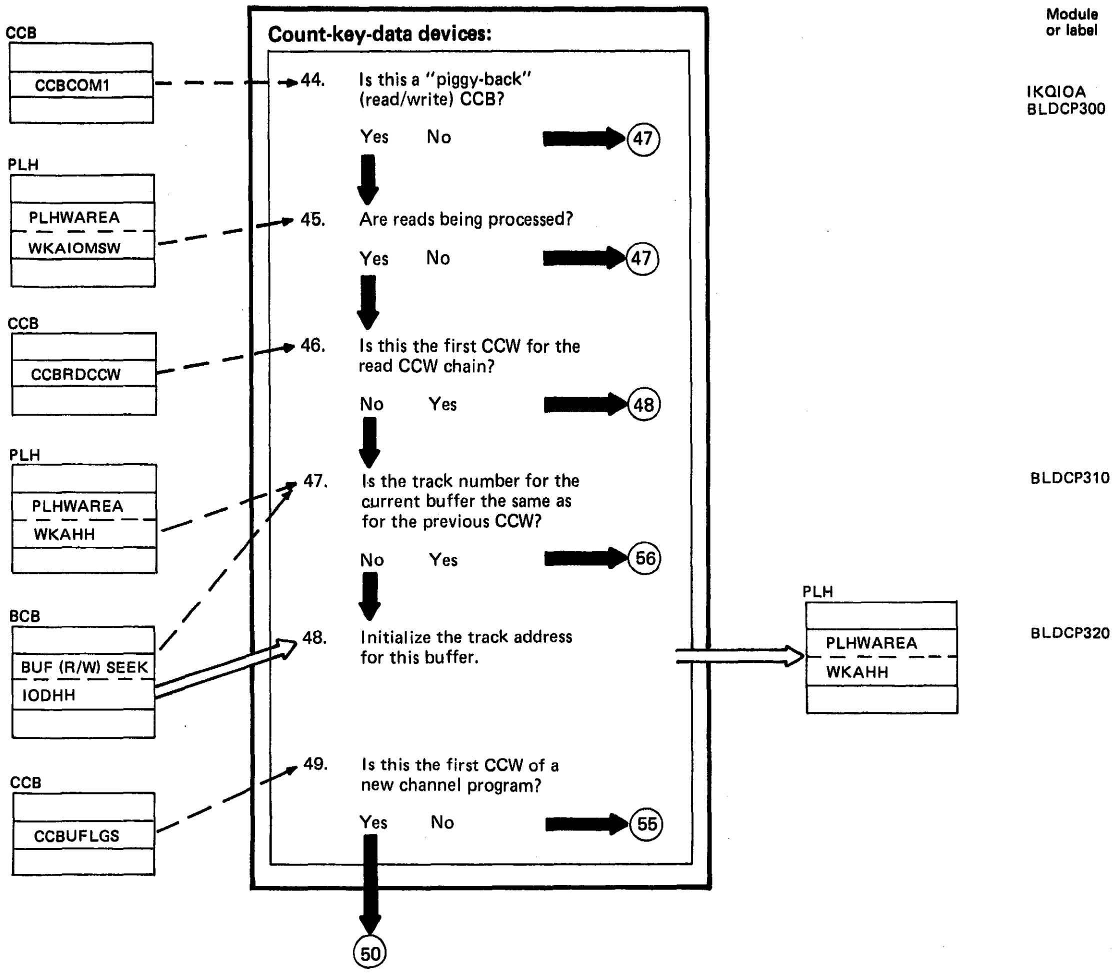

Module or label
I/O Data Block
IOAOPCOD
CCB
CCBLCCWB
SCCW
SCCWLOC
Fixed block devices (continued):
41. Does the last Locate parameter field contain a read operation code?
No
Yes
59
42. Locate the previous CCW, and turn off the data chaining switch in the previous CCW.
CCW
CCWFLAG
BLDCP400
43. Point to the Locate CCW skeleton, and build a locate CCW.
BLDCP410
See Diagram EF, Build CCW.
BDCCW000
Return vector:
● Locate CCW not built
67
● Locate CCW built
59

Notes for Diagram EG11

42. When a Locate CCW follows a Write/Read CCW, the data chaining switch is turned off in the Write/Read CCW to prevent an I/O error.

43. This step builds the Locate CCW. If the Locate CCW is not built (due to lack of control blocks), the current chain of CCBs is EXCPed (or started for I/O) to free the current blocks for reuse.

Diagram EG12. I/O Manager: Build Channel Program


CCB
CCBCOM1
PLH
PLHWAREA
WKAIOMSW
CCB
CCBRDCCW
PLH
PLHWAREA
WKAHH
BCB
BUF (R/W) SEEK
IODHH
CCB
CCBUFLGS
Count-key-data devices:
44. Is this a "piggy-back" (read/write) CCB?
Yes No
47
45. Are reads being processed?
Yes No
47
46. Is this the first CCW for the read CCW chain?
No Yes
48
47. Is the track number for the current buffer the same as for the previous CCW?
No Yes
56
48. Initialize the track address for this buffer.
49. Is this the first CCW of a new channel program?
Yes No
55
50
PLH
PLHWAREA
WKAHH
Module or label
IKQIOA
BLDCP300
BLDCP310
BLDCP320

Notes for Diagram EG12

44.-46. If the buffer being processed is for a "piggy-back" read operation, and if its CCW is the first Read CCW in the CCW chain, then a Head-Switch Seek CCW is forced into the CCW chain before the Read CCW is built. This is done for error retry during IKQIOB.

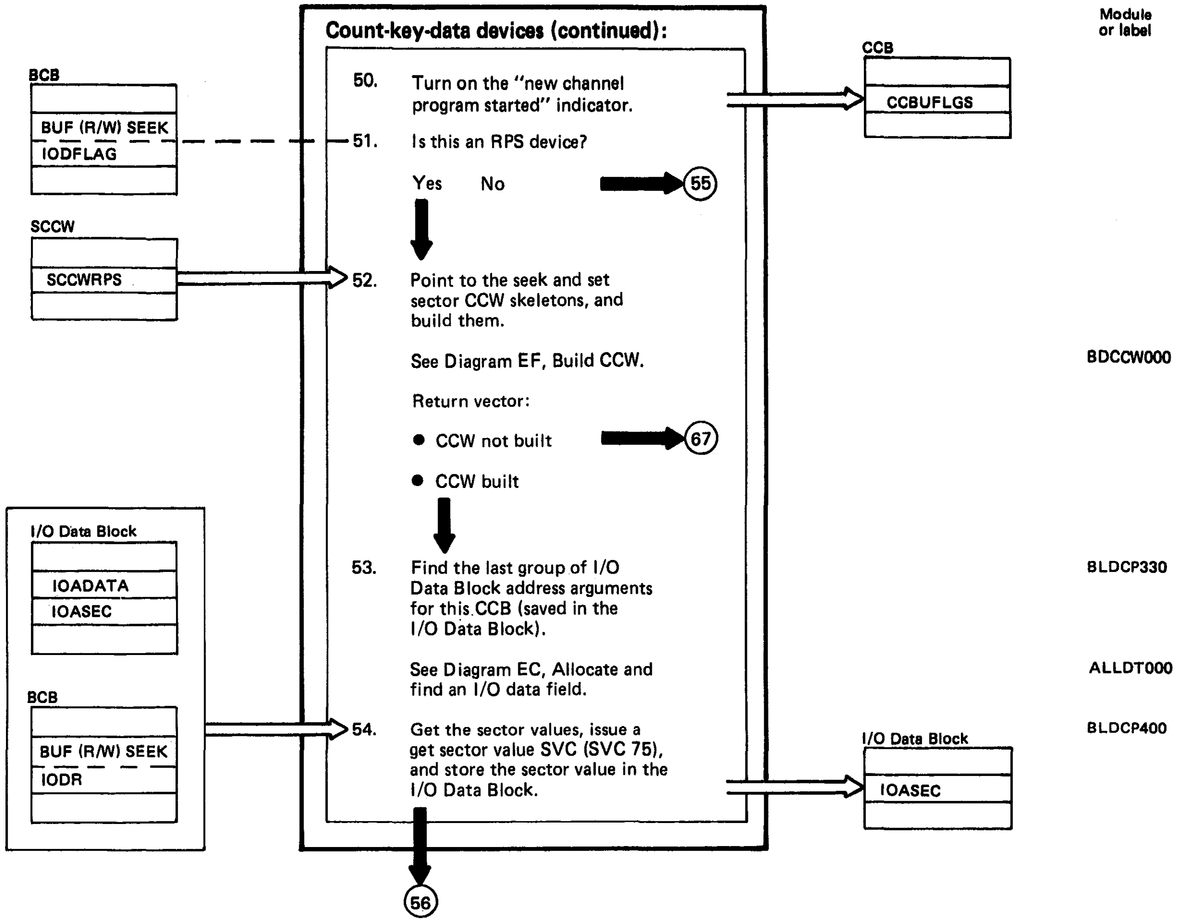
A CCW chain performing "piggy-back" I/O operations is split into the write portion and the read portion; each part is separately executed. Therefore, the read portion must start with a Seek CCW. (IKQIOB sets it to a "long seek" during retry.)

47.-48. Initially, a Seek CCW is required at the start of the CCW string, and the WKAHH field in PLHWAREA is set to X'FF' during BLDCP initialization.

After the first Seek CCW (or "piggy-back" Seek CCW) is to be built, WKAHH is set to the current head number. Whenever the head number of the CCW to be built differs from that of the previous CCW, a Head Seek CCW is inserted into the CCW chain before the Write/Read CCW is built.

49.-50. The CCBRPS bit was set to zero initially so that the first CCW in each CCW string will be a Seek CCW.

Diagram EG13. I/O Manager: Build Channel Program



Count-key-data devices (continued):

50. Turn on the "new channel program started" indicator.

51. Is this an RPS device?

Yes No

52. Point to the seek and set sector CCW skeletons, and build them.

See Diagram EF, Build CCW.

Return vector:

- ● CCW not built
- ● CCW built

53. Find the last group of I/O Data Block address arguments for this CCB (saved in the I/O Data Block).

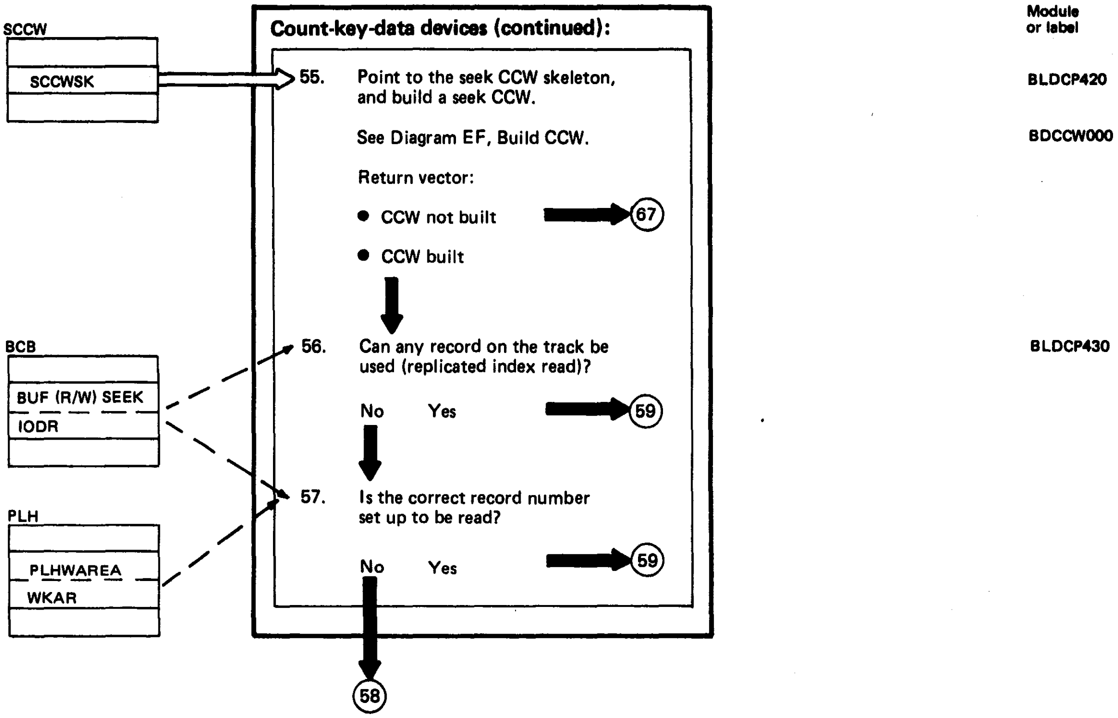
See Diagram EC, Allocate and find an I/O data field.

54. Get the sector values, issue a get sector value SVC (SVC 75), and store the sector value in the I/O Data Block.

Notes for Diagram EG13

51.-54. If the IODRPS bit in the BCB indicates an RPS device, a Seek CCW, followed by a Set Sector CCW, is built. The blocksize, record number, and device type are obtained, and the RPS sector value is obtained from the SECTVAL (SVC 75) routine.

Diagram EG14. I/O Manager: Build Channel Program



SCCW
SCCWSK
BCB
BUF (R/W) SEEK
IODR
PLH
PLHWAREA
WKAR
Count-key-data devices (continued):
55. Point to the seek CCW skeleton, and build a seek CCW.
See Diagram EF, Build CCW.
Return vector:
● CCW not built
67
● CCW built
56. Can any record on the track be used (replicated index read)?
No
Yes
59
57. Is the correct record number set up to be read?
No
Yes
59
58
Module or label
BLDCP420
BDCCW000
BLDCP430

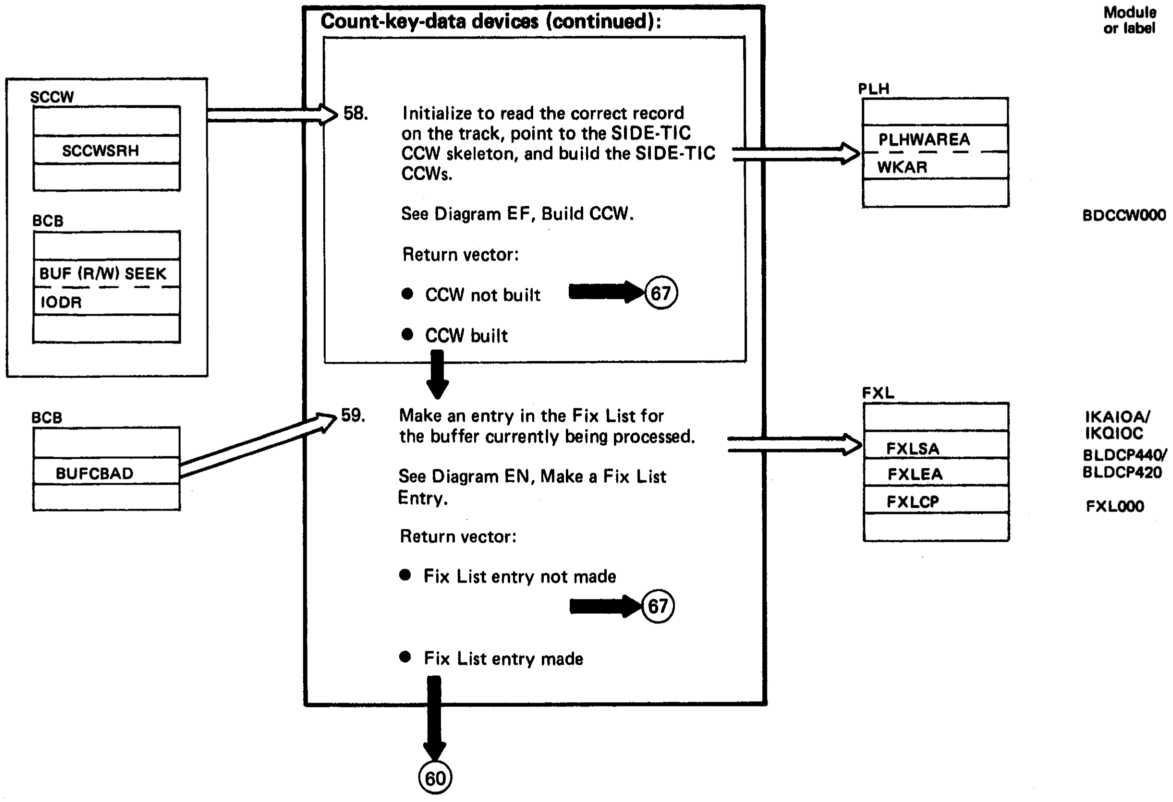
Notes for Diagram EG14

55. If the device is not an RPS type, only a Seek CCW is built.

56. If this is a replicated read operation for an index (or sequence set) record, the R number is not used because any record on the track can be read.

57. A SIDE-TIC is required before building the first Read CCW in the chain, or when the Read CCW to be built is for a record that is not contiguous to the record to be read by the preceding Read CCW.

Diagram EG15. I/O Manager: Build Channel Program


Count-key-data devices (continued):
Module or label
SCCW
SCCWSRH
BCB
BUF (R/W) SEEK
IODR
58. Initialize to read the correct record on the track, point to the SIDE-TIC CCW skeleton, and build the SIDE-TIC CCWs.
See Diagram EF, Build CCW.
Return vector:
● CCW not built
67
● CCW built
PLH
PLHWAREA
WKAR
BDCCW000
BCB
BUFCBAD
59. Make an entry in the Fix List for the buffer currently being processed.
See Diagram EN, Make a Fix List Entry.
Return vector:
● Fix List entry not made
67
● Fix List entry made
60
FXL
FXLSA
FXLEA
FXLCP
IKAIOA/
IKQIOC
BLDCP440/
BLDCP420
FXL000

Notes for Diagram EG15

58. This step determines whether a Search ID-TIC CCW sequence is required. A Write Data CCW always requires a Search ID Equal-TIC sequence or the start of any Write/Read CCW sequence in the CCW chain.

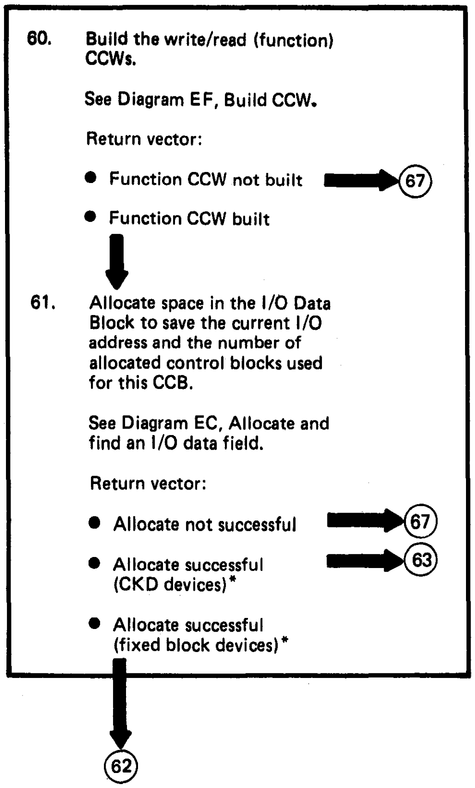
    The WKAR field in the PLHWAREA is updated to contain the record number of the CCW to be built. WKAR is set to X'FF' whenever a Seek CCW is built, or to X'00' when a Write Data CCW is built.

    For the Read, Write Check, and Write CKD CCWs, WKAR is set to the next record number to be read after the CCW is built.

59. An entry is made in the Fix List for the buffer currently being processed. The entry is made before the Write/Read CCW is built in case a control block is needed to extend the Fix List to accommodate the entry .

    If an entry is not made in the Fix List, I/O is started for the current chain of CCBs. There is no impact because the CCWs for that buffer have not been built yet.

Diagram EG16. I/O Manager: Build Channel Program



60. Build the write/read (function) CCWs.

See Diagram EF, Build CCW.

Return vector:

- Function CCW not built
- Function CCW built

61. Allocate space in the I/O Data Block to save the current I/O address and the number of allocated control blocks used for this CCB.

See Diagram EC, Allocate and find an I/O data field.

Return vector:

- Allocate not successful
- Allocate successful (CKD devices)*
- Allocate successful (fixed block devices)*

Module or label

BLDCP450/
BLDCP430

BDCCW000

BLDCP460/
BLDCP440

ALLDT000

*IKQIOA contains all CKD I/O Manager code, and IKQIOC contains all FBA I/O Manager code. Consequently, CKD vs. FBA decisions do not appear in the actual code. The "decision" is shown in the HIPO for convenience in documentation only.

Notes for Diagram EG16

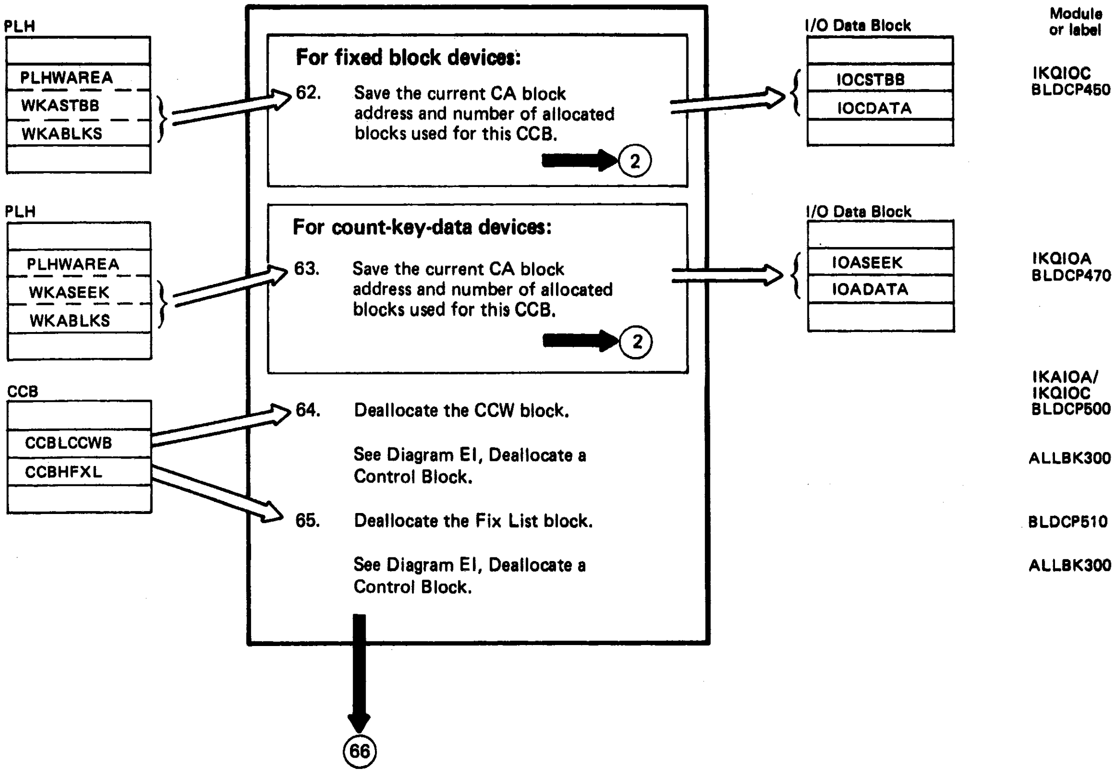
60. The Write/Read CCW is now built. The pointer to its skeleton was passed by the I/O Manager Mainline in register 15 (steps 41 and 60 of Diagram EA).

If the CCW was not built (due to lack of control blocks), I/O is performed for the current chain of CCBs to free existing control blocks for reuse.

61.-63. A temporary save area is obtained from the last I/O Data Block (FCDB) associated with this CCB. The save area is used to save the CA and the number of allocated blocks used by this CCB because the next buffer to be processed may be for another CCB.

If space cannot be found for the save area (due to lack of control blocks), I/O is performed for the current chain of CCBs to free existing control blocks for re-use.

Diagram EG17. I/O Manager: Build Channel Program



For fixed block devices:

62. Save the current CA block address and number of allocated blocks used for this CCB. — IKQIOC BLDCP450

For count-key-data devices:

63. Save the current CA block address and number of allocated blocks used for this CCB. — IKQIOA BLDCP470

64. Deallocate the CCW block. — IKAIOA/ IKQIOC BLDCP500

See Diagram EI, Deallocate a Control Block. — ALLBK300

65. Deallocate the Fix List block. — BLDCP510

See Diagram EI, Deallocate a Control Block. — ALLBK300

Notes for Diagram EG17

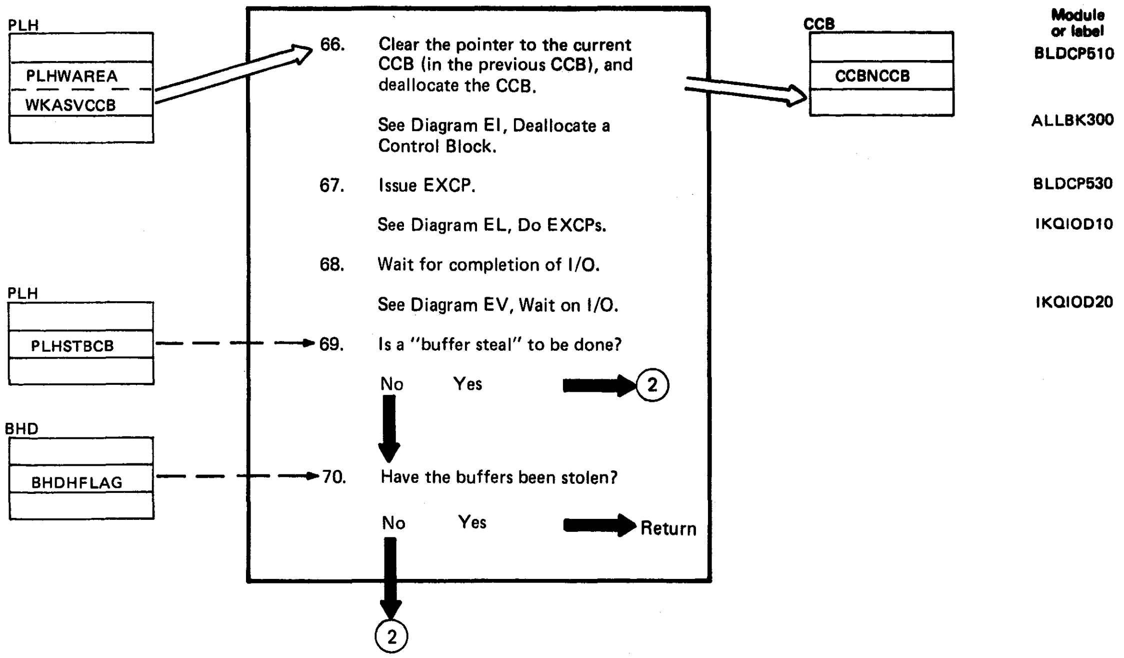
64.-71. These steps are entered at various points when no more control blocks are available for creating a CCB, CCW, I/O Data Block, or Fix List.

If a lack of available blocks is detected during initial setup of the CCB, the CCB is removed from the CCB chain, and all control blocks associated with it are returned to the control block queue free list.

If a lack of available blocks is detected at any other time, the partially built channel program currently being processed is EXCPed. Later, another channel program is built to complete I/O for the buffer currently being processed.

The EXCP/WAIT is then done for the remaining chain of CCBs in order to free blocks in use.

Diagram EG18. I/O Manager: Build Channel Program



66. Clear the pointer to the current CCB (in the previous CCB), and deallocate the CCB.

See Diagram EI, Deallocate a Control Block.

67. Issue EXCP.

See Diagram EL, Do EXCPs.

68. Wait for completion of I/O.

See Diagram EV, Wait on I/O.

69. Is a "buffer steal" to be done?

70. Have the buffers been stolen?

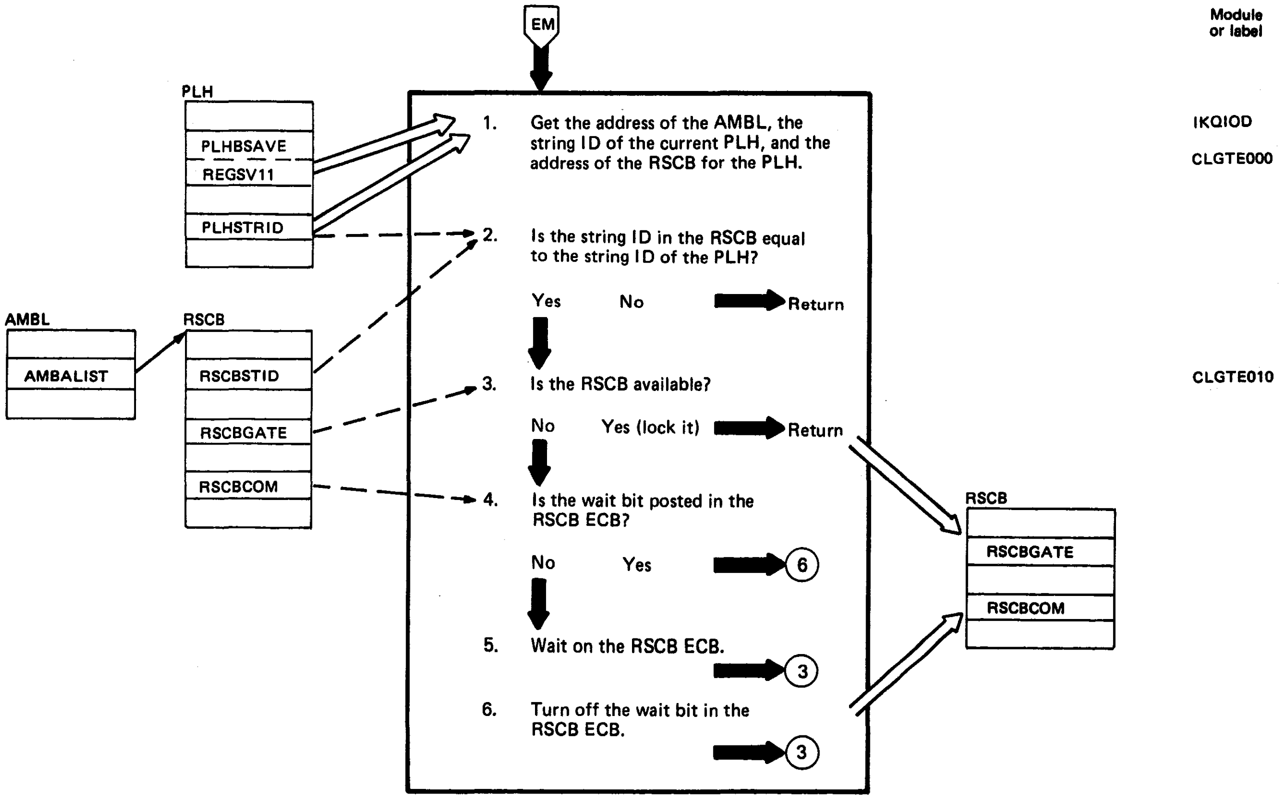
Notes for Diagram EG18

69. If the answer to this step is "yes", then buffers are being stolen from another task while it is also doing I/O Manager processing. The "buffer steal" bit (PLHSTBCB) is turned on by the Buffer Manager whenever the I/O Manager is requested to complete the I/O of another task. PLHSTBCB indicates that buffers are available for the steal.

    If the answer to this step is "no", this request to the I/O Manager is for normal processing, not a "buffer steal".

70. This step determines if buffers were stolen during the exit to the user's EXCPAD routine.

Diagram EH1. I/O Manager: Lock the RSCB ECB



1. Get the address of the AMBL, the string ID of the current PLH, and the address of the RSCB for the PLH.

2. Is the string ID in the RSCB equal to the string ID of the PLH?

   Yes No ➡ Return

3. Is the RSCB available?

   No Yes (lock it) ➡ Return

4. Is the wait bit posted in the RSCB ECB?

   No Yes ➡ (6)

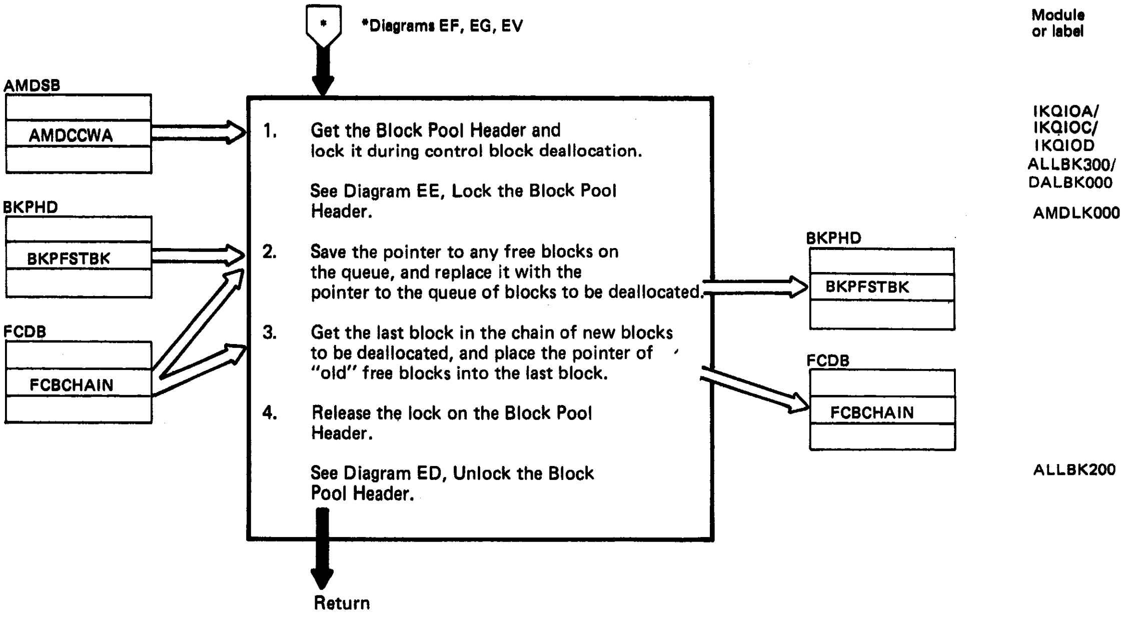
5. Wait on the RSCB ECB. ➡ (3)

6. Turn off the wait bit in the RSCB ECB. ➡ (3)

Notes for Diagram EH1

This routine functions like the "Lock the Block Pool Header" routine except that the RSCB ID is tested to determine if it is owned by the current PLH and is the resource lock for the EXCPAD Parameter List. Only the owner of the RSCB can lock the ECB.

Diagram EI1. I/O Manager: Deallocate a Control Block



1. Get the Block Pool Header and lock it during control block deallocation.

   See Diagram EE, Lock the Block Pool Header.

2. Save the pointer to any free blocks on the queue, and replace it with the pointer to the queue of blocks to be deallocated.

3. Get the last block in the chain of new blocks to be deallocated, and place the pointer of "old" free blocks into the last block.

4. Release the lock on the Block Pool Header.

   See Diagram ED, Unlock the Block Pool Header.

Diagram EJ1. I/O Manager: Dump Control Blocks

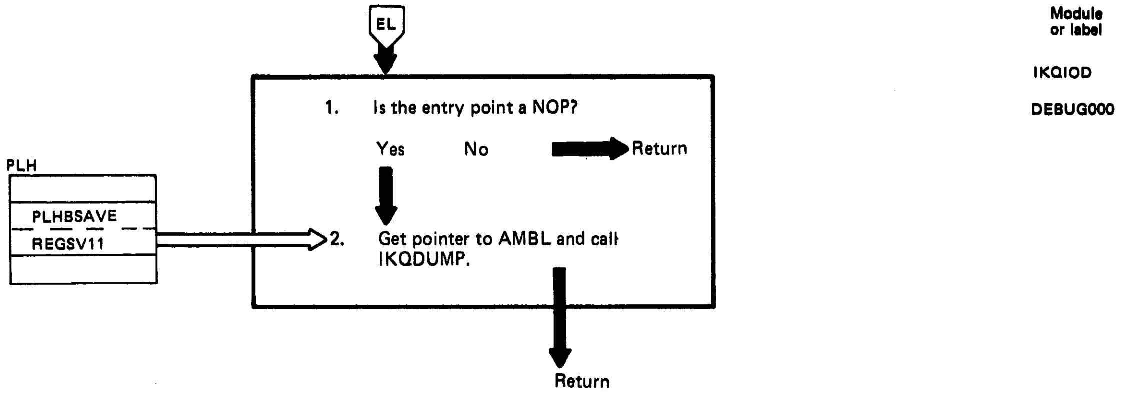
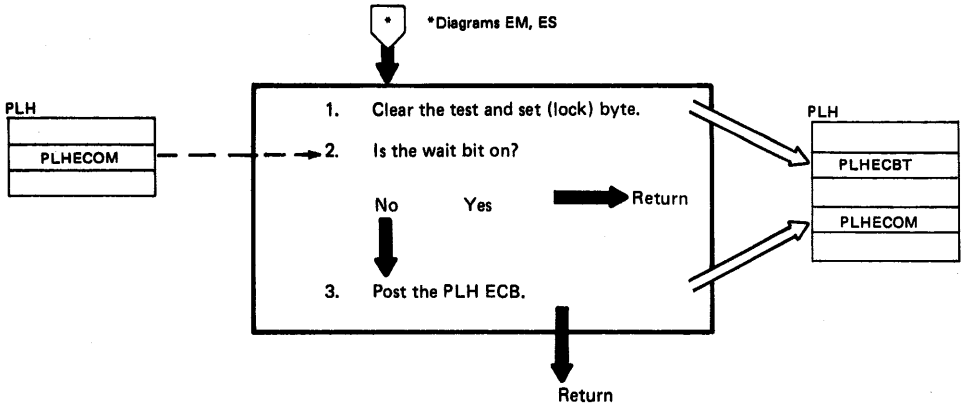

Module or label
EL
IKQIOD
DEBUG000
PLH
PLHBSAVE
REGSV11
1. Is the entry point a NOP?
Yes
No
Return
2. Get pointer to AMBL and call IKQDUMP.
Return

Diagram EK1. I/O Manager: Unlock the PLH ECB

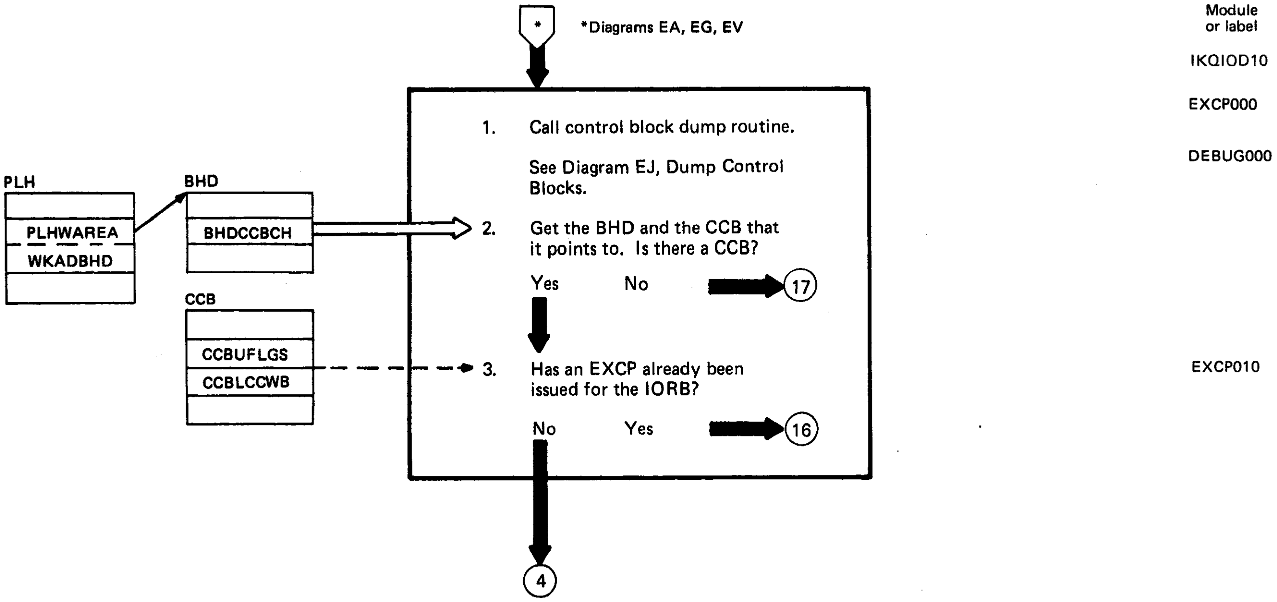


1. Clear the test and set (lock) byte.
2. Is the wait bit on?
3. Post the PLH ECB.

Module or label

IKQIOA/
IKQIOC/
IKQIOD
DNECB000

Diagram EL1. I/O Manager: Do EXCPs



1. Call control block dump routine.

   See Diagram EJ, Dump Control Blocks.

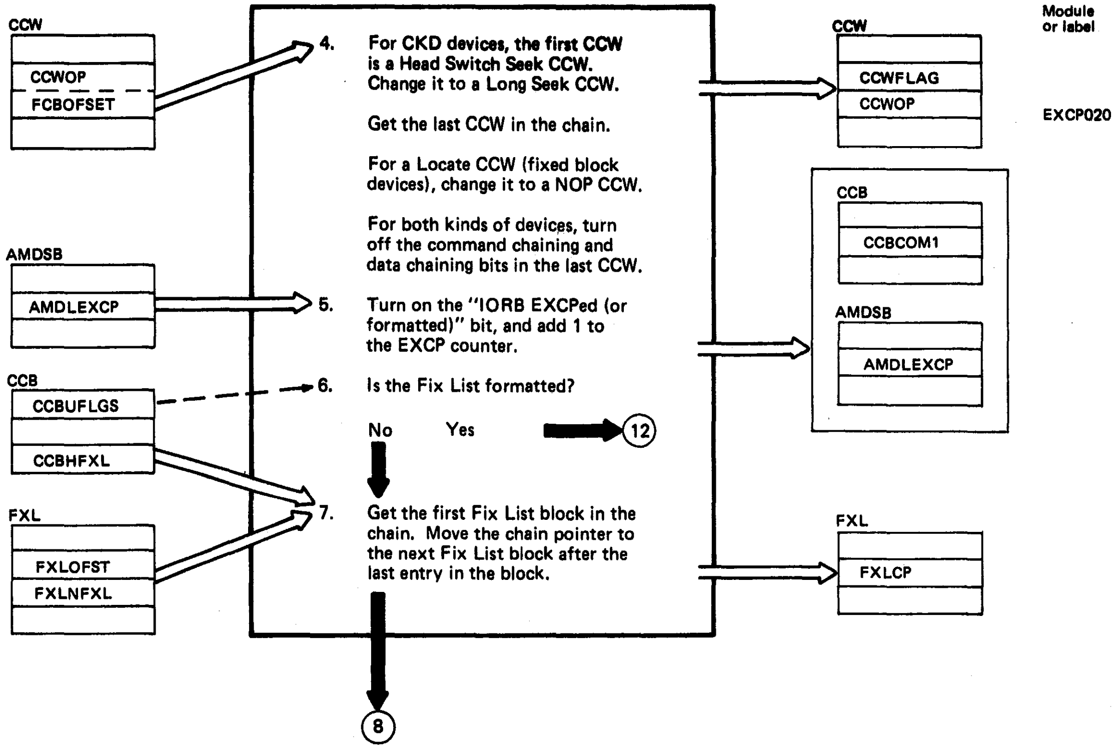
2. Get the BHD and the CCB that it points to. Is there a CCB?

3. Has an EXCP already been issued for the IORB?

Notes for Diagram EL1

1.-3. This routine is used for both count-key-data and fixed block devices. These steps search the CCB chain for any IORBs that have not had an EXCP issued for them.

Diagram EL2. I/O Manager: Do EXCPs



4. For CKD devices, the first CCW is a Head Switch Seek CCW. Change it to a Long Seek CCW.

Get the last CCW in the chain.

For a Locate CCW (fixed block devices), change it to a NOP CCW.

For both kinds of devices, turn off the command chaining and data chaining bits in the last CCW.

5. Turn on the "IORB EXCPed (or formatted)" bit, and add 1 to the EXCP counter.

6. Is the Fix List formatted?

7. Get the first Fix List block in the chain. Move the chain pointer to the next Fix List block after the last entry in the block.

Notes for Diagram EL2

4. For fixed block devices, the last CCW in the chain is found, and if it is a Locate CCW (X'43'), it is converted to a NOP CCW (X'03').

For CKD devices, the first CCW in the chain is converted from a Head Switch Seek CCW (X'1B') to a Long Seek CCW (X'07').

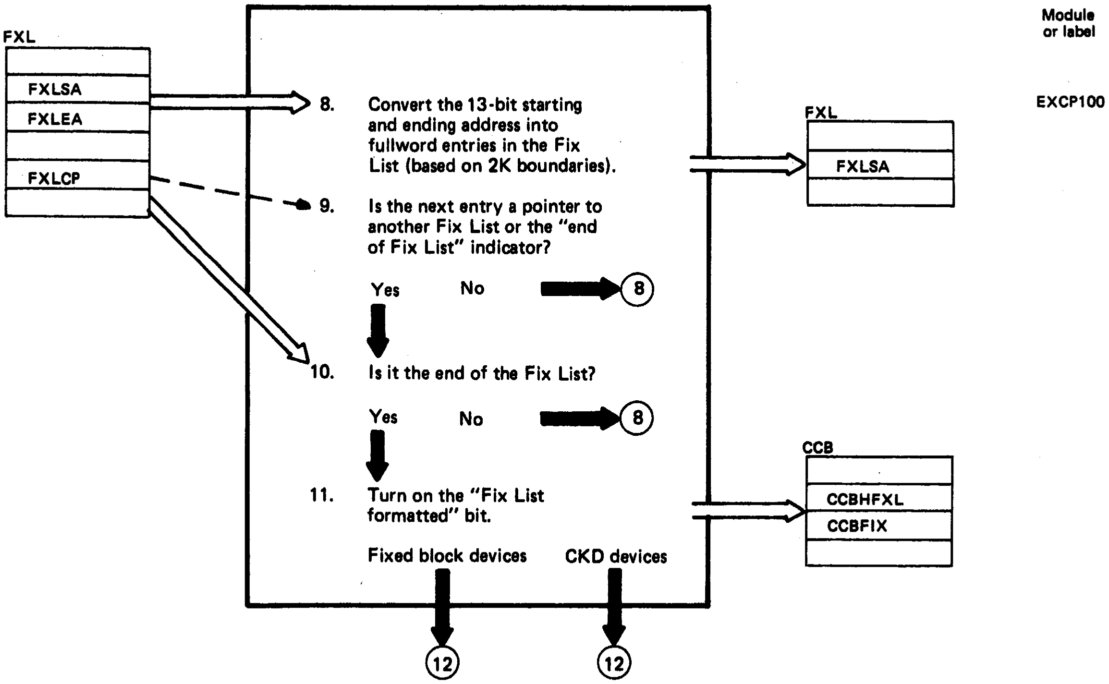
For both device classes, the command chaining and data chaining bits are turned off in the last CCW.

5. The bit CCBERROK is turned on to indicate that the IORB has had an EXCP issued for it. One is added to the EXCP counter in the AMDSB.

6.-11. If the Fix List has not yet been formatted, the "end of Fix List" indicator (or the pointer to the next Fix List entry in the chain) is set following the last entry in the Fix List pointed to by the IORB. (Each new Fix List block was inserted into the chain between the IORB and the Fix List block it pointed to. The last Fix List block used becomes the first block in the chain.)

The starting and ending addresses (2 bytes each) are converted to fullword addresses, adjusted to the beginning and end of their respective 2K addresses. After the Fix List entries have been adjusted, the format bit (CCBFIX) is turned on.

Diagram EL3. I/O Manager: Do EXCPs



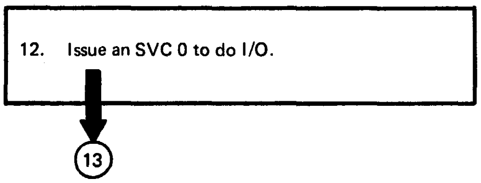
8. Convert the 13-bit starting and ending address into fullword entries in the Fix List (based on 2K boundaries).

9. Is the next entry a pointer to another Fix List or the "end of Fix List" indicator?

10. Is it the end of the Fix List?

11. Turn on the "Fix List formatted" bit.

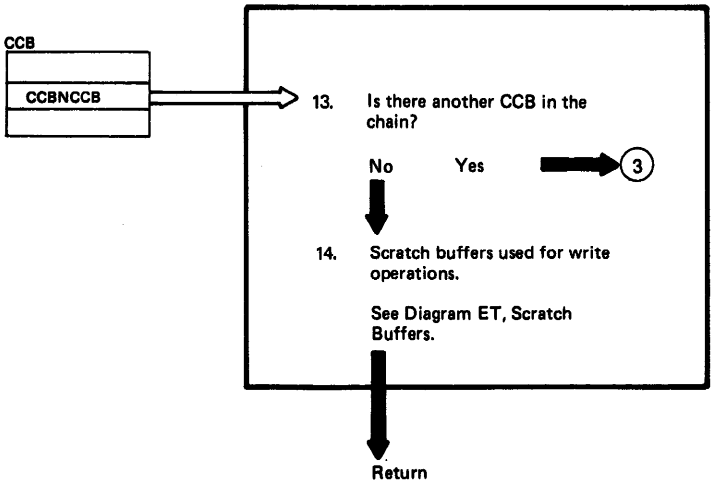
Diagram EL4. I/O Manager: Do EXCPs



12. Issue an SVC 0 to do I/O.

Module
or label
EXCP200

Diagram EL5. I/O Manager: Do EXCPs



13. Is there another CCB in the chain?

14. Scratch buffers used for write operations.

See Diagram ET, Scratch Buffers.

Module or label

EXCP210

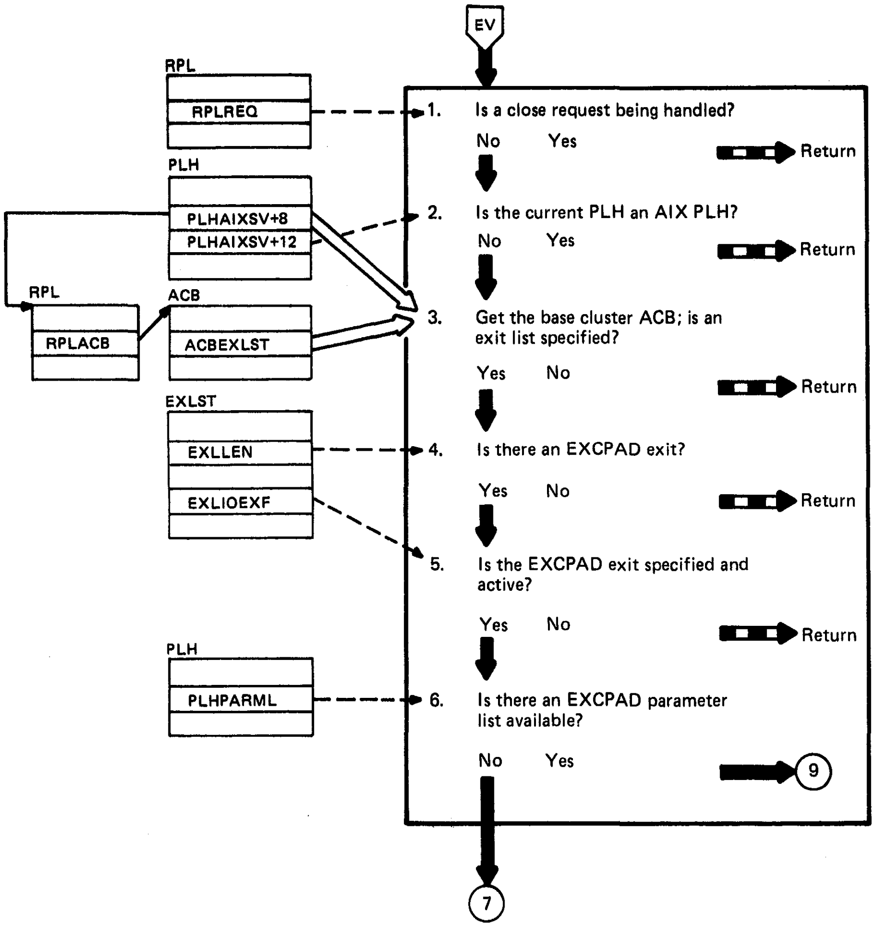
EXCP220

SCR000

Notes for Diagram EL5

13.-14. After all the IORBs in the chain have been EXCPed, the Scratch Buffers (SCR000) subroutine is invoked to invalidate the BCBs used by other PLH strings if they have the same RBA, and if a write was issued for that RBA by the EXCP routine.

Diagram EM1. I/O Manager: EXCPAD Exit Processing



1. Is a close request being handled?
   No Yes

2. Is the current PLH an AIX PLH?
   No Yes

3. Get the base cluster ACB; is an exit list specified?
   Yes No

4. Is there an EXCPAD exit?
   Yes No

5. Is the EXCPAD exit specified and active?
   Yes No

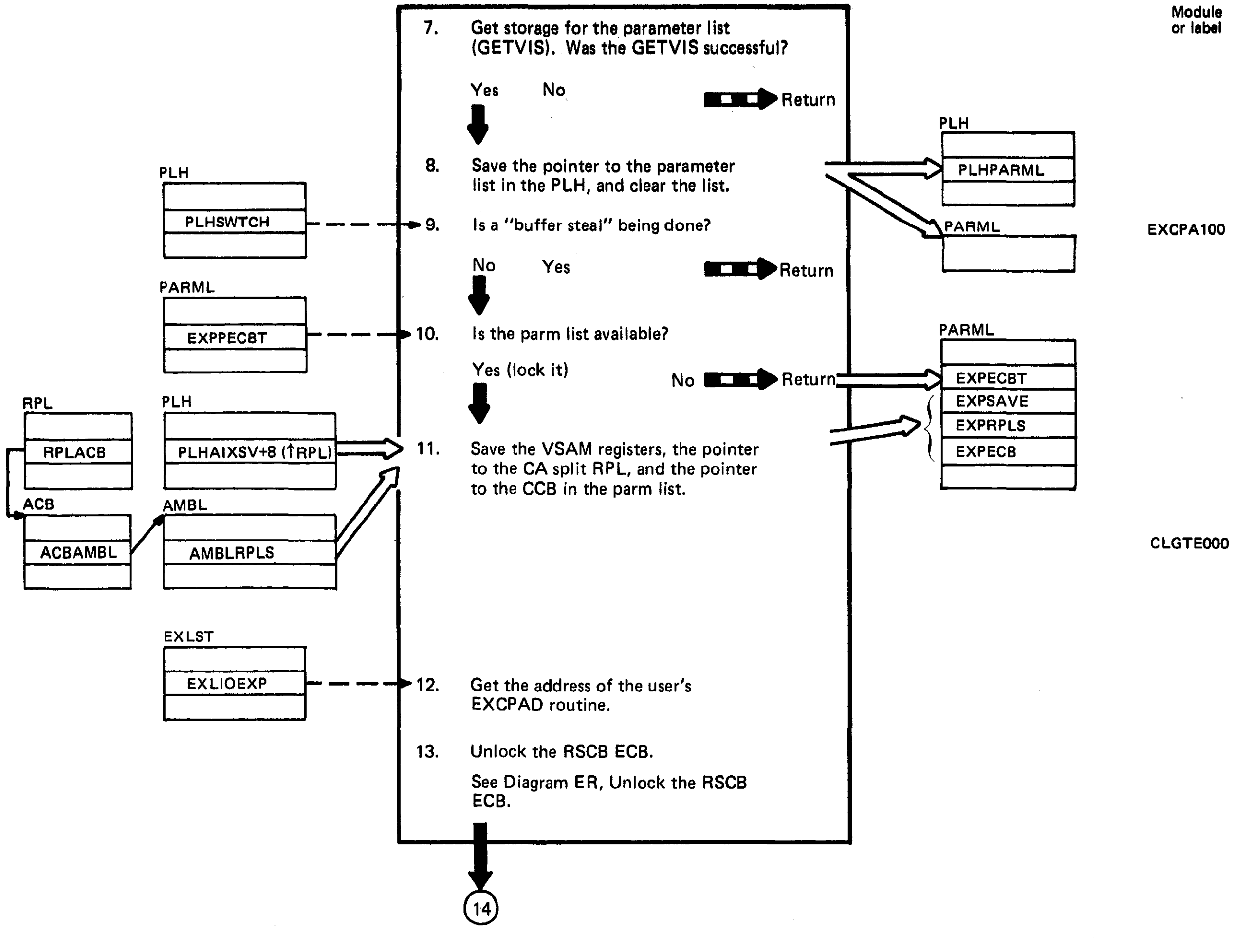
6. Is there an EXCPAD parameter list available?
   No Yes

Module or label

IKQIOD

EXCPA000

Diagram EM2. I/O Manager: EXCPAD Exit Processing



7. Get storage for the parameter list (GETVIS). Was the GETVIS successful?

8. Save the pointer to the parameter list in the PLH, and clear the list.

9. Is a "buffer steal" being done?

10. Is the parm list available?

11. Save the VSAM registers, the pointer to the CA split RPL, and the pointer to the CCB in the parm list.

12. Get the address of the user's EXCPAD routine.

13. Unlock the RSCB ECB.

See Diagram ER, Unlock the RSCB ECB.

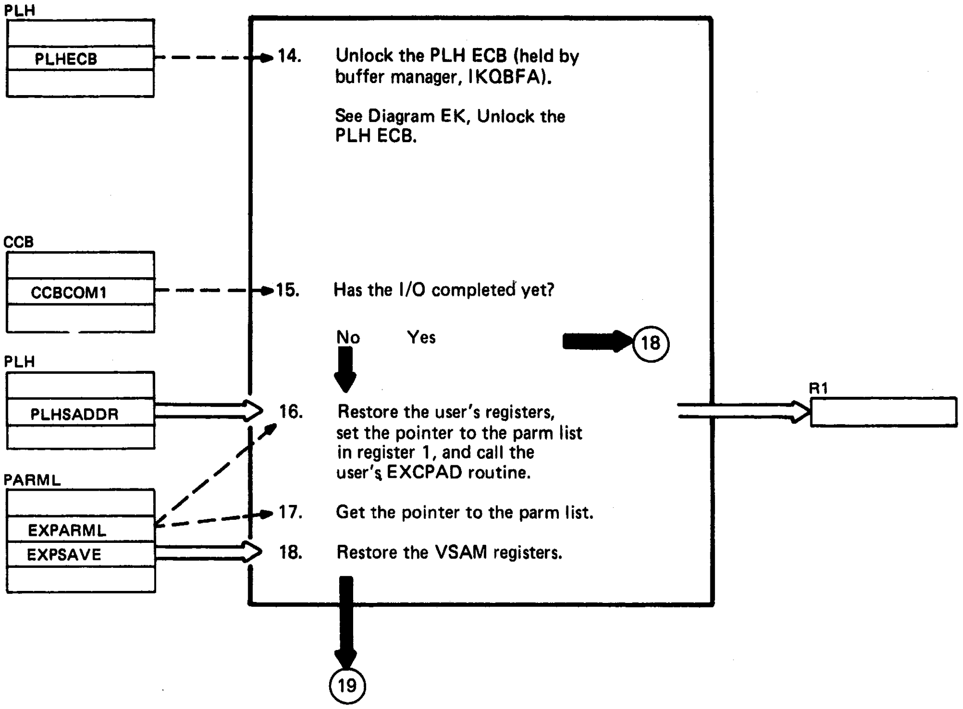
Notes for Diagram EM2

9. If the EXCPAD exit processing routine is entered from the WAIT routine while it is doing a "buffer steal", no exit is taken to the user's EXCPAD routine.

10. A test and set is issued against the EXCPAD parameter list ECB. If the ECB is not free, no exit is taken to the user's EXCPAD routine.

11.-14. The parameter list is initialized with the pointers to the calling RPL (user RPL), the pointer to the split RPL (used during CA split), and the CCB for which the WAIT will be issued. VSAM registers 2-15 are also saved in the parameter list. The RSCB ECB (if owned by the current PLH) and the PLH ECB are unlocked to enable a "buffer steal" to occur, if buffers are required by another PLH in the same string while the user is doing EXCPAD processing.

Diagram EM3. I/O Manager: EXCPAD Exit Processing



Module or label

14. Unlock the PLH ECB (held by buffer manager, IKQBFA).

    See Diagram EK, Unlock the PLH ECB. — DNECB000

15. Has the I/O completed yet?

16. Restore the user's registers, set the pointer to the parm list in register 1, and call the user's EXCPAD routine.

17. Get the pointer to the parm list. — EXCPA200

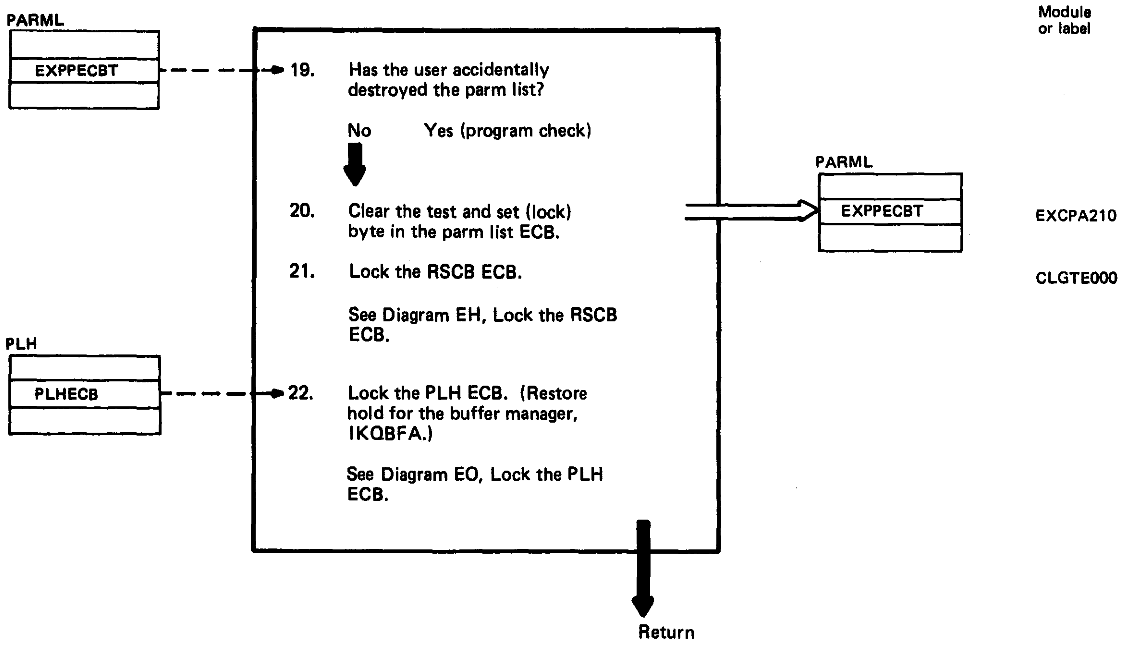
18. Restore the VSAM registers.

Notes for Diagram EM3

15.-16. If the I/O for the current CCB is complete, no exit is taken to the user's routine.

17.-19. Upon return from the user's exit, the parameter list is checked to see if it has been altered by the user (EXPPECBT≠X'FF'). If it has been altered, a program check is caused because it is not known if the VSAM registers (saved in parameter list) are still valid. To allow further VSAM processing at this point could cause unpredictable results.

Diagram EM4. I/O Manager: EXCPAD Exit Processing



19. Has the user accidentally destroyed the parm list?

20. Clear the test and set (lock) byte in the parm list ECB.

21. Lock the RSCB ECB.

See Diagram EH, Lock the RSCB ECB.

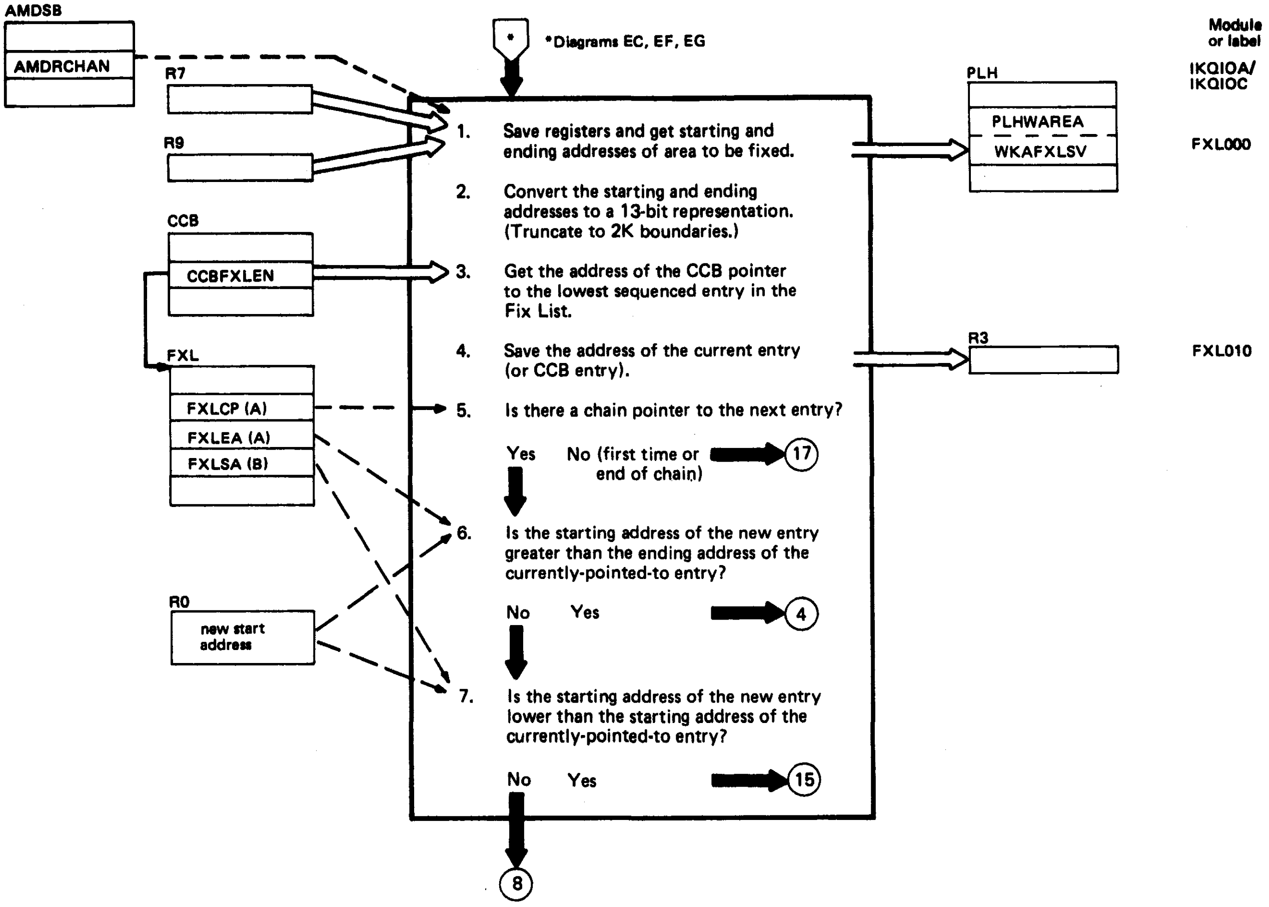
22. Lock the PLH ECB. (Restore hold for the buffer manager, IKQBFA.)

See Diagram EO, Lock the PLH ECB.

Notes for Diagram EM4

20.-22. The RSCB ECB and the PLH ECB are again locked for VSAM processing, and control returns to the WAIT routine.

Diagram EN1. I/O Manager: Make a Fix List Entry



1. Save registers and get starting and ending addresses of area to be fixed.

2. Convert the starting and ending addresses to a 13-bit representation. (Truncate to 2K boundaries.)

3. Get the address of the CCB pointer to the lowest sequenced entry in the Fix List.

4. Save the address of the current entry (or CCB entry).

5. Is there a chain pointer to the next entry?

   Yes No (first time or end of chain)

6. Is the starting address of the new entry greater than the ending address of the currently-pointed-to entry?

   No Yes

7. Is the starting address of the new entry lower than the starting address of the currently-pointed-to entry?

   No Yes

Notes for Diagram EN1

Each entry in the Fix List is rounded to upper and lower 2K boundaries. This minimizes the number of entries made to the table because most control blocks are within the same 2K page. Because the entries are adjusted to 2K boundaries, only two bytes are required for each entry (13 bits).

The CCB contains the pointers to the start of the Fix List (CCBHFXL) and to the lowest entry in the Fix List (CCBFXLEN). Each Fix List entry is 8 bytes; the first 4 bytes is used as a chain pointer to keep the entries sequenced in ascending order. The last 4 bytes contain the starting and ending entry addresses adjusted to 2K boundaries. The last entry in the Fix List is used as a chain pointer to another Fix List (if more than one is required per CCB), and the offset to the currently available position for the next entry.

1. If execution is on a S/370, a Fix List is not built. Control returns to the caller without any processing having been performed.

3.-5. The CCB pointer to the lowest entry is obtained and saved as a "back pointer" to the previous entry, in case the new entry becomes the lowest entry in the chain. If there is no entry in the Fix List, a new entry is made. If there are existing entries, the list is searched to determine where the new entry will be made. If the new entry is higher than existing entries, it is added to the end of the chain. The chain pointer of the last entry contains zeros to indicate the end of the entry chain.

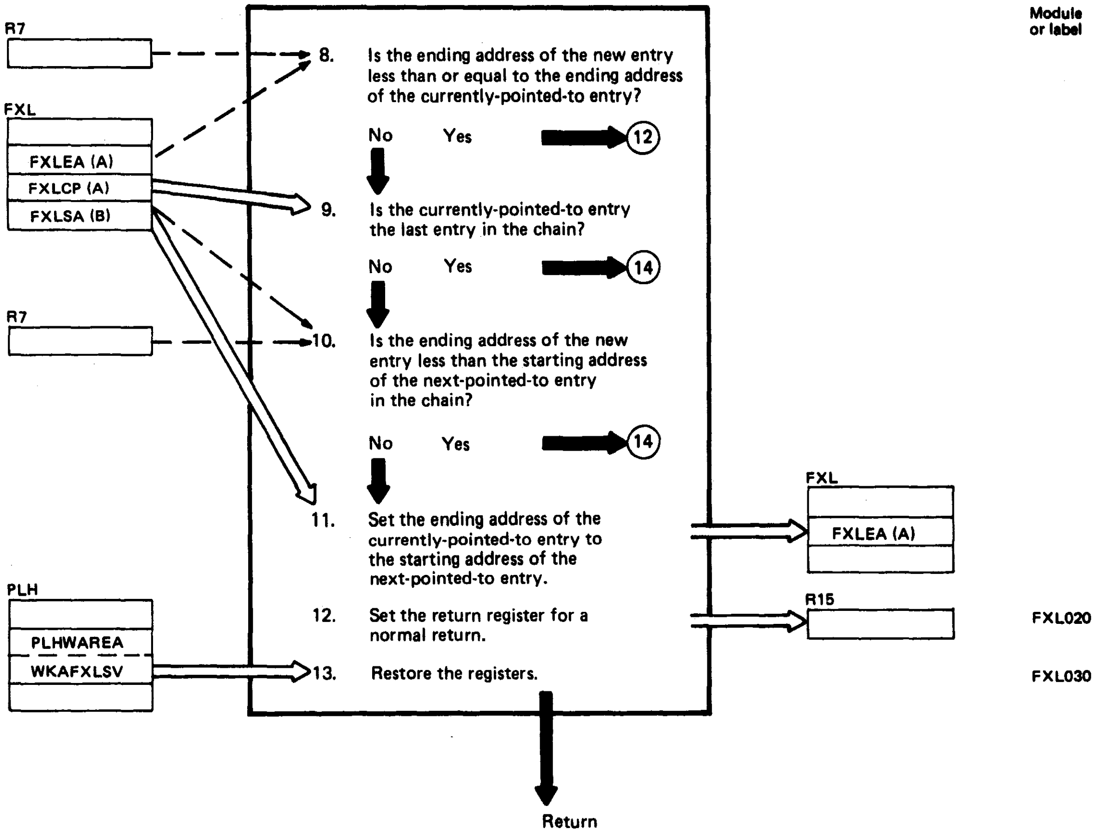
6. If there is an existing entry/entries in the Fix List, the starting address of the new entry is compared to the ending address of the current entry. If the new value is greater than the current entry, the next entry (if any) is obtained and checked.

7. The starting address of the new entry is then compared to the starting address of the current entry. If the new entry is less than the current entry, the ending address of the new entry will be compared to the starting address of the current entry (steps 15-24).

   If the new ending address is lower, the new entry is placed in the Fix List and chained from the previous entry (or CCB) to the current entry.

   If the new ending address is not lower than the starting address of the current entry, the starting address of the current entry is replaced with the starting address of the new entry because the two entries overlap (steps 15 and 16).

Diagram EN2. I/O Manager: Make a Fix List Entry



8. Is the ending address of the new entry less than or equal to the ending address of the currently-pointed-to entry?

9. Is the currently-pointed-to entry the last entry in the chain?

10. Is the ending address of the new entry less than the starting address of the next-pointed-to entry in the chain?

11. Set the ending address of the currently-pointed-to entry to the starting address of the next-pointed-to entry.

12. Set the return register for a normal return.

13. Restore the registers.

Notes for Diagram EN2

8. If the starting address of the new entry is not lower than that of the current entry, the ending address of the new entry is compared to the ending address of the current entry. If the new address is less than or equal to the current address, the new address is ignored because its address range is already included in the area spanned by the current entry.

9. If the ending address of the new entry is greater than that of the current entry, then the next entry in the entry chain is obtained (if one exists).

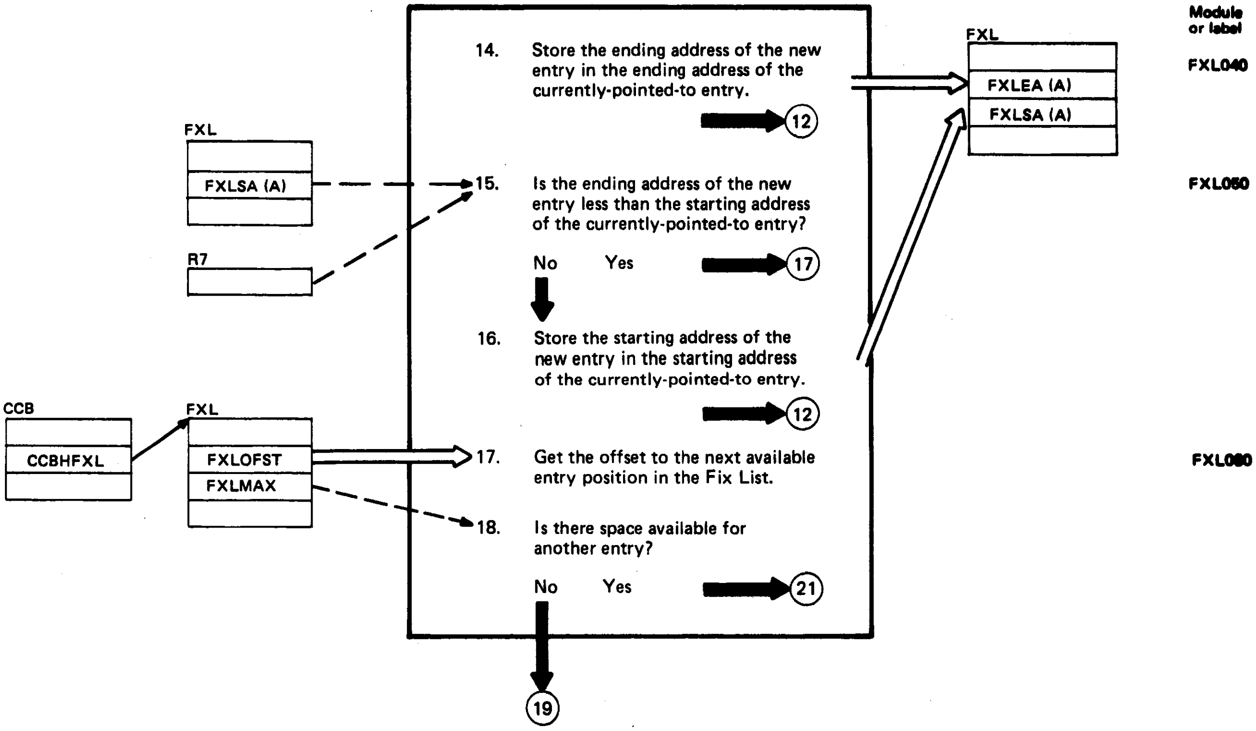
   If another entry does not exist, the ending address of the current entry is replaced by the ending address of the new entry because the two entries overlap (step 14).

10. If another entry exists in the entry chain, the ending address of the new entry is compared to the starting address of the next entry. If the new address is lower, the ending address of the current entry is replaced by the ending address of the new entry because the two entries overlap (step 14).

11. This new entry overlaps the address ranges of the current and next entries. This situation can occur if two earlier entries do not start and end on 2K boundaries except after rounding.

    The two en tries are in sequence in the entry chain, but they are not contiguous in storage. This can occur if the new entry begins in the same 2K block as the lower (current) entry, and ends in the same 2K block as the higher (next) entry.

Diagram EN3. I/O Manager: Make a Fix List Entry



14. Store the ending address of the new entry in the ending address of the currently-pointed-to entry.

15. Is the ending address of the new entry less than the starting address of the currently-pointed-to entry?

No Yes

16. Store the starting address of the new entry in the starting address of the currently-pointed-to entry.

17. Get the offset to the next available entry position in the Fix List.

18. Is there space available for another entry?

No Yes

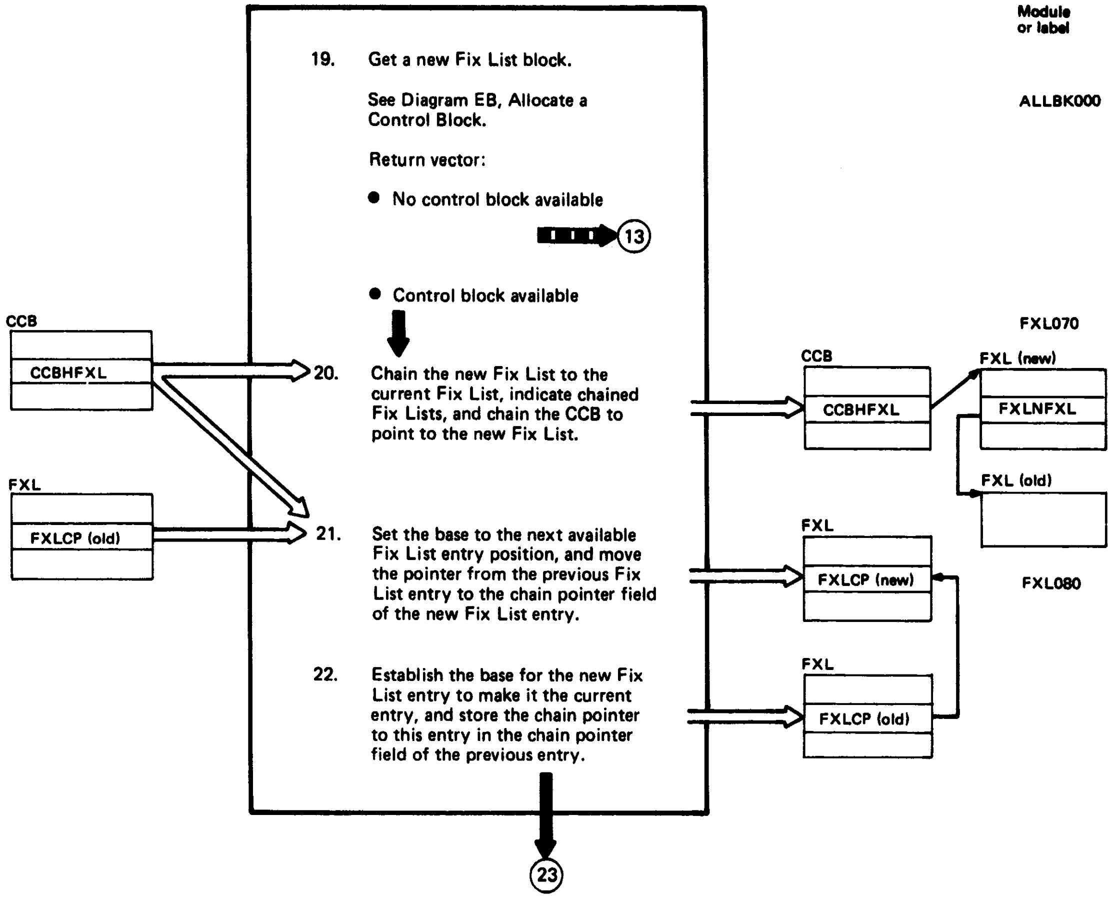
Notes for Diagram EN3

17.-20. When a new entry must be placed into the Fix List, the pointer to the currently used Fix List block is obtained from the CCB. If no room exists in the Fix List block, a new block is obtained from the ALLBK000 routine.

If a new block cannot be allocated, an error return is taken, and I/O is performed to free the currently used control blocks.

If a new block is obtained, it is chained to the existing Fix List blocks, and the CCB is chained to point to the new block.

Diagram EN4. I/O Manager: Make a Fix List Entry



19. Get a new Fix List block.

See Diagram EB, Allocate a Control Block.

Return vector:

- No control block available
- Control block available

20. Chain the new Fix List to the current Fix List, indicate chained Fix Lists, and chain the CCB to point to the new Fix List.

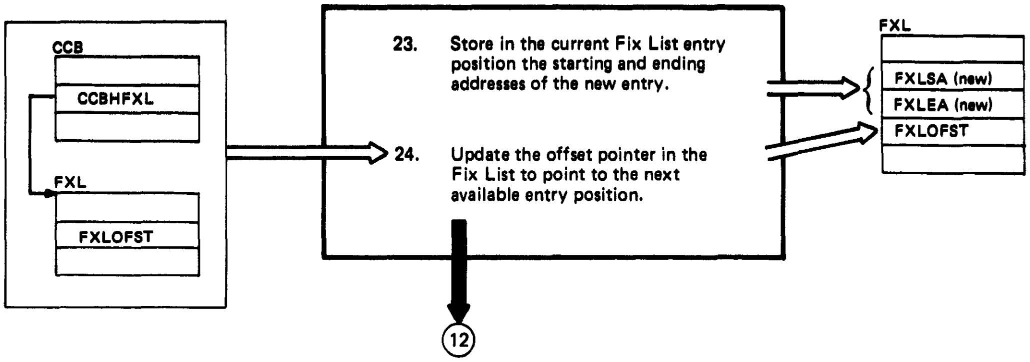
21. Set the base to the next available Fix List entry position, and move the pointer from the previous Fix List entry to the chain pointer field of the new Fix List entry.

22. Establish the base for the new Fix List entry to make it the current entry, and store the chain pointer to this entry in the chain pointer field of the previous entry.

Notes for Diagram EN4

21.-24. The address of the next available position in the Fix List block is determined, and the new entry is linked into the chain in the appropriate sequence. (The starting and ending addresses are stored in the next available position.) The offset is then updated to point to the next available position in the Fix List block.

Diagram EN5. I/O Manager: Make a Fix List Entry



23. Store in the current Fix List entry position the starting and ending addresses of the new entry.

24. Update the offset pointer in the Fix List to point to the next available entry position.

Diagram EO1. I/O Manager: Lock the PLH ECB

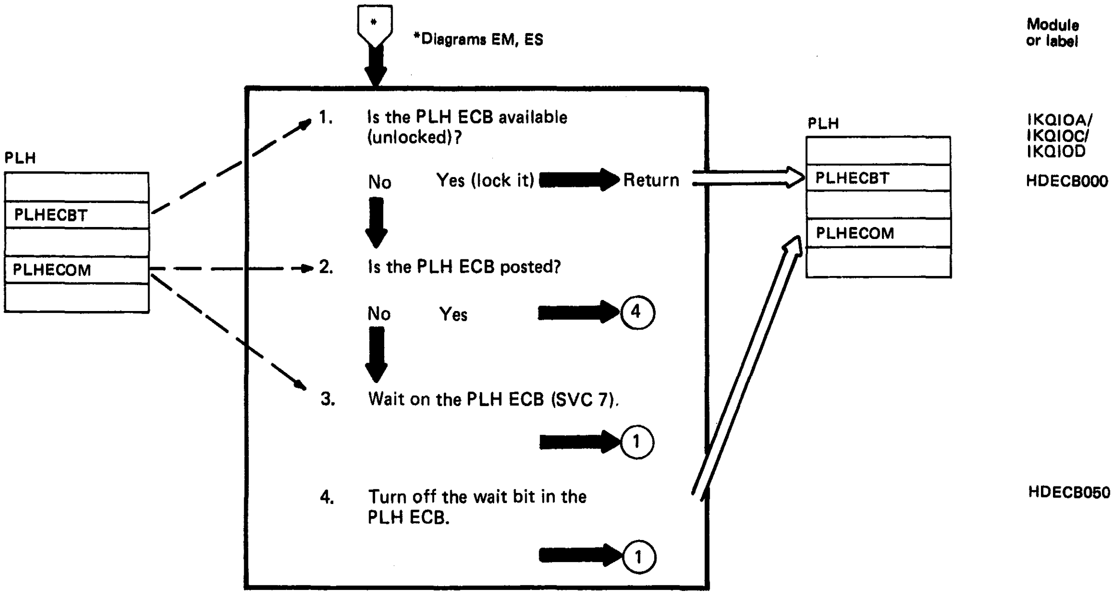
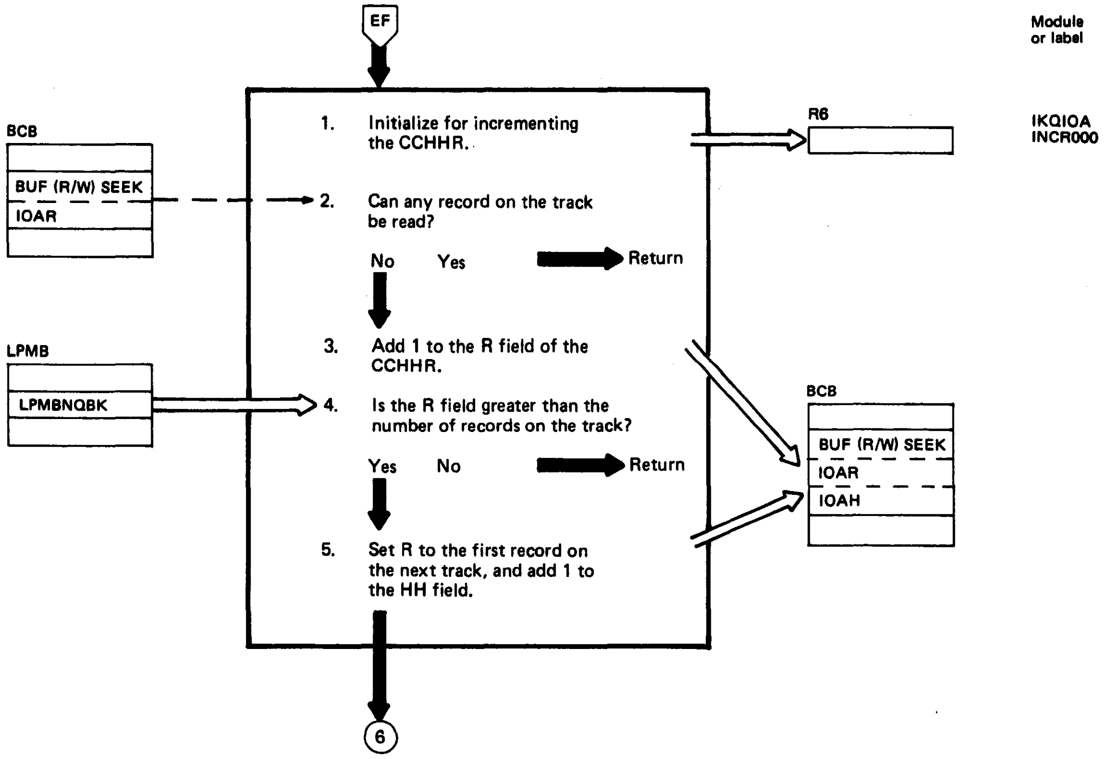

*
*Diagrams EM, ES
Module or label
PLH
PLHECBT
PLHECOM
1. Is the PLH ECB available (unlocked)?
No
Yes (lock it)
Return
2. Is the PLH ECB posted?
No
Yes
4
3. Wait on the PLH ECB (SVC 7).
1
4. Turn off the wait bit in the PLH ECB.
1
PLH
PLHECBT
PLHECOM
IKQIOA/
IKQIOC/
IKQIOD
HDECB000
HDECB050

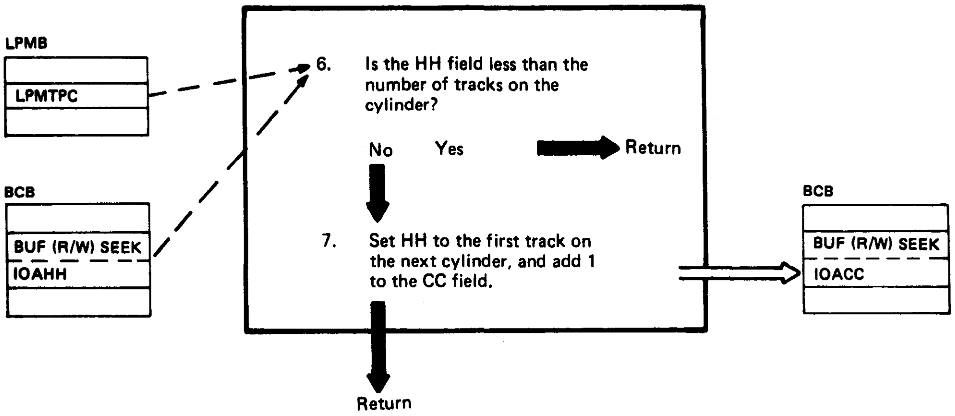
Diagram EP1. I/O Manager: Update the DASD Address


EF
Module or label
BCB
BUF (R/W) SEEK
IOAR
1. Initialize for incrementing the CCHHR.
R6
IKQIOA
INCR000
2. Can any record on the track be read?
No
Yes
Return
LPMB
LPMBNQBK
3. Add 1 to the R field of the CCHHR.
4. Is the R field greater than the number of records on the track?
Yes
No
Return
BCB
BUF (R/W) SEEK
IOAR
IOAH
5. Set R to the first record on the next track, and add 1 to the HH field.
6

Notes for Diagram EP1

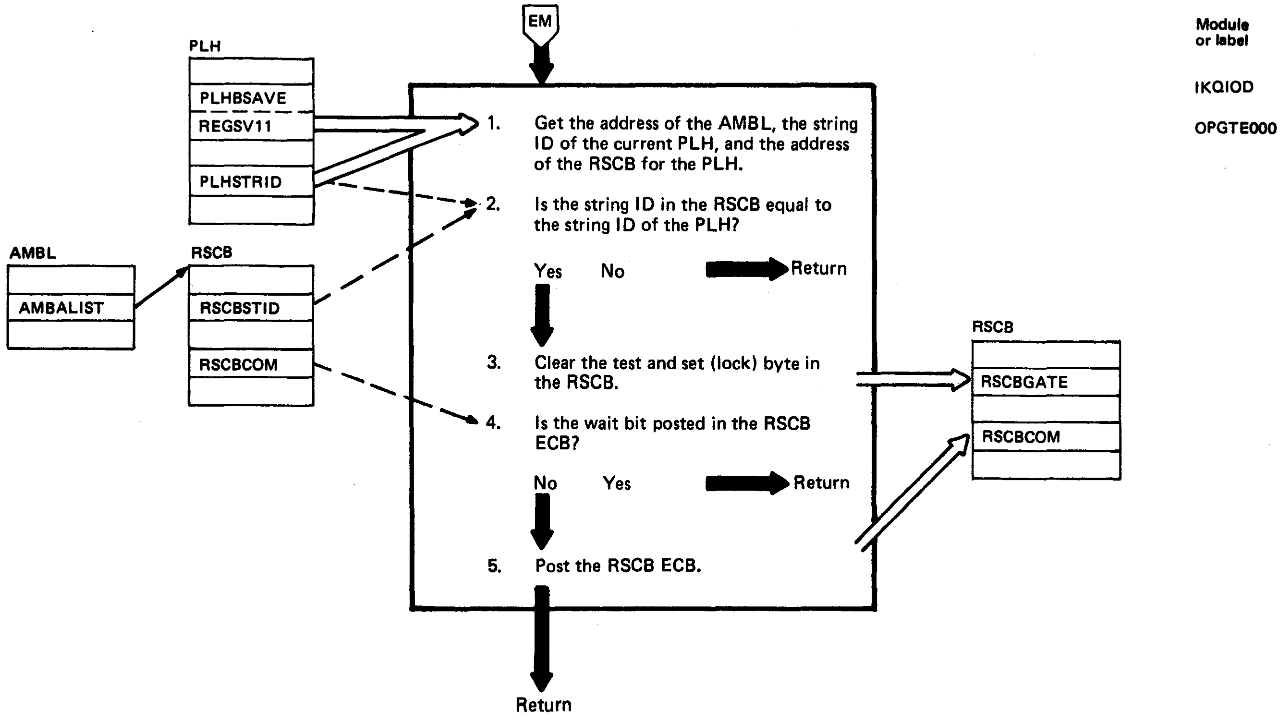
This routine is called whenever a Format Write, Write, Read, or Read Back Check CCW is built. It updates the CCHHR address, in the buffer associated with the CCW being built, to point to the next sequential data record in the data set.

Diagram EP2. I/O Manager: Update the DASD Address



LPMB
LPMTPC
BCB
BUF (R/W) SEEK
IOAHH
6. Is the HH field less than the number of tracks on the cylinder?
No
Yes
Return
7. Set HH to the first track on the next cylinder, and add 1 to the CC field.
Return
BCB
BUF (R/W) SEEK
IOACC

Diagram ER1. I/O Manager: Unlock the RSCB ECB



1. Get the address of the AMBL, the string ID of the current PLH, and the address of the RSCB for the PLH.

2. Is the string ID in the RSCB equal to the string ID of the PLH?

3. Clear the test and set (lock) byte in the RSCB.

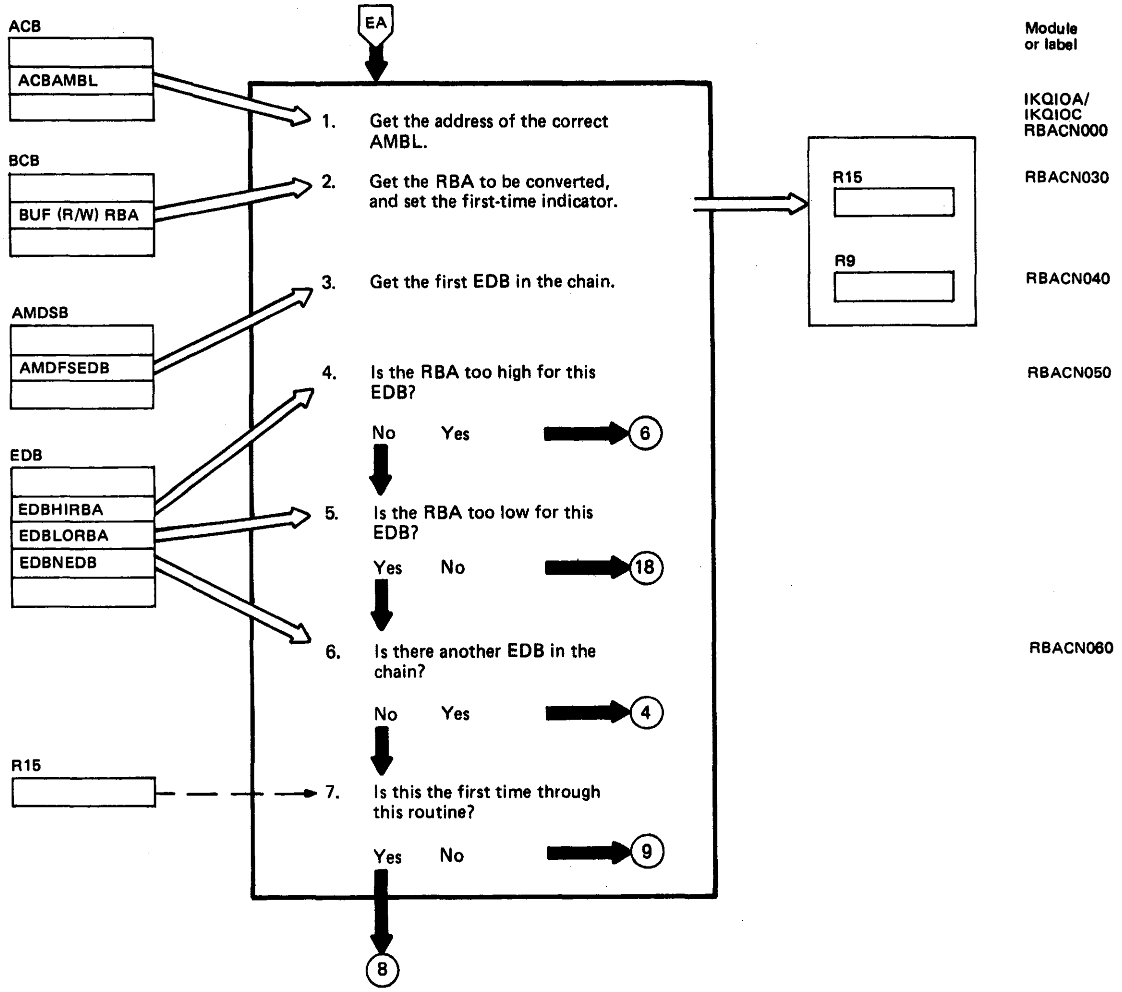
4. Is the wait bit posted in the RSCB ECB?

5. Post the RSCB ECB.

Notes for Diagram ER1

This routine functions like the "Unlock the Block Pool Header" routine except that the RSCB ID is tested to determine if it is owned by the current PLH and is the resource lock for the EXCPAD Parameter List. Only the owner of the RSCB can unlock it.

Diagram ES1. I/O Manager: Convert RBA

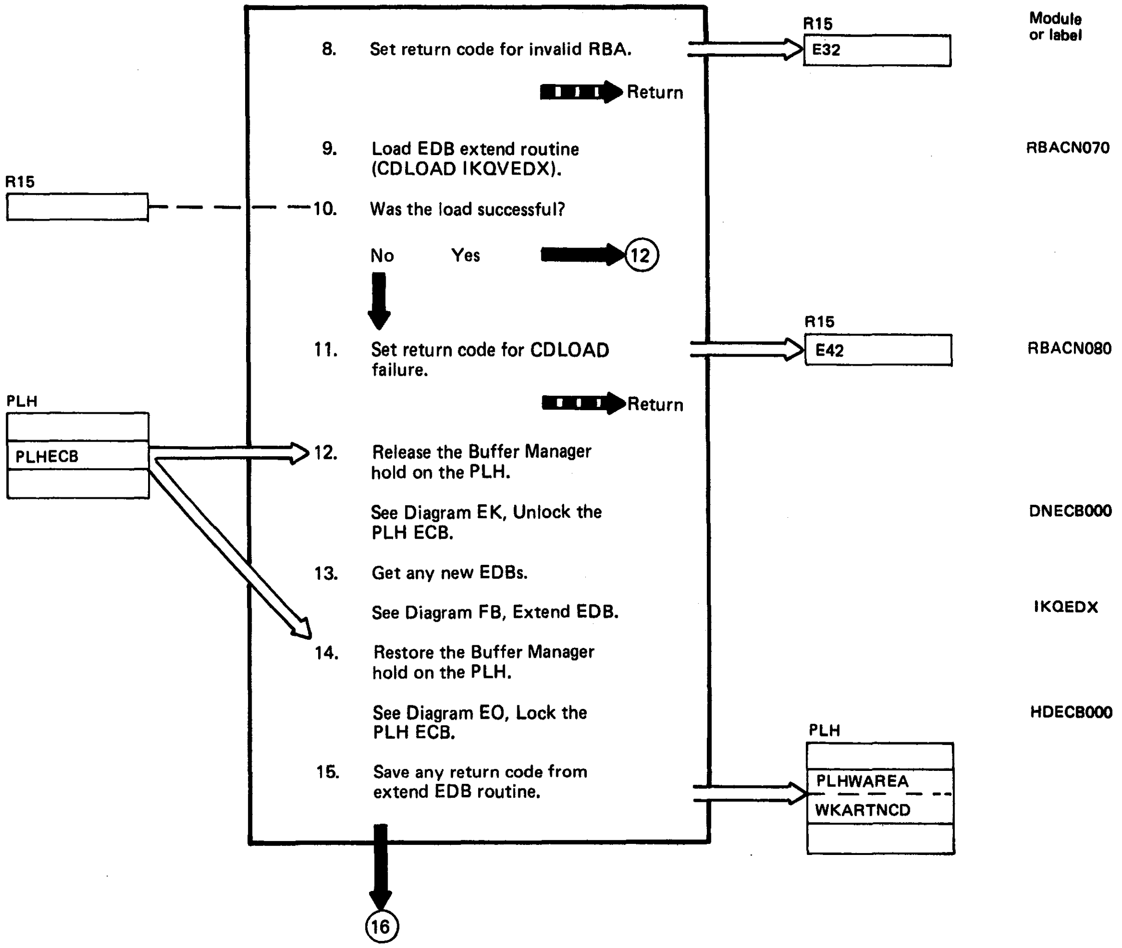

EA
ACB
ACBAMBL
BCB
BUF (R/W) RBA
AMDSB
AMDFSEDB
EDB
EDBHIRBA
EDBLORBA
EDBNEDB
R15
1. Get the address of the correct AMBL.
2. Get the RBA to be converted, and set the first-time indicator.
3. Get the first EDB in the chain.
4. Is the RBA too high for this EDB?
No Yes 6
5. Is the RBA too low for this EDB?
Yes No 18
6. Is there another EDB in the chain?
No Yes 4
7. Is this the first time through this routine?
Yes No 9
8
R15
R9
Module or label
IKQIOA/ IKQIOC RBACN000
RBACN030
RBACN040
RBACN050
RBACN060

Notes for Diagram ES1

1. The address of the AMBL is required for the extend EDB routine (IKQEDX) or the mount volume routine (IKQEOV).

2.-17. Register 15 is used as an indicator when searching EDBs for the correct RBA range for the RBA to be converted. If the RBA is not found in the current chain of EDBs, the extend EDB routine is called to ensure that all EDBs for the data set are available. After the EDBs have been extended, a second pass is made, searching for the proper RBA range. If a valid range is not found, register 15 is set to zero (set by the return from extend EDB routine), and an error exit is taken.

Diagram ES2. I/O Manager: Convert RBA



8. Set return code for invalid RBA.

9. Load EDB extend routine (CDLOAD IKQVEDX).

10. Was the load successful?

11. Set return code for CDLOAD failure.

12. Release the Buffer Manager hold on the PLH.

See Diagram EK, Unlock the PLH ECB.

13. Get any new EDBs.

See Diagram FB, Extend EDB.

14. Restore the Buffer Manager hold on the PLH.

See Diagram EO, Lock the PLH ECB.

15. Save any return code from extend EDB routine.

Notes for Diagram ES2

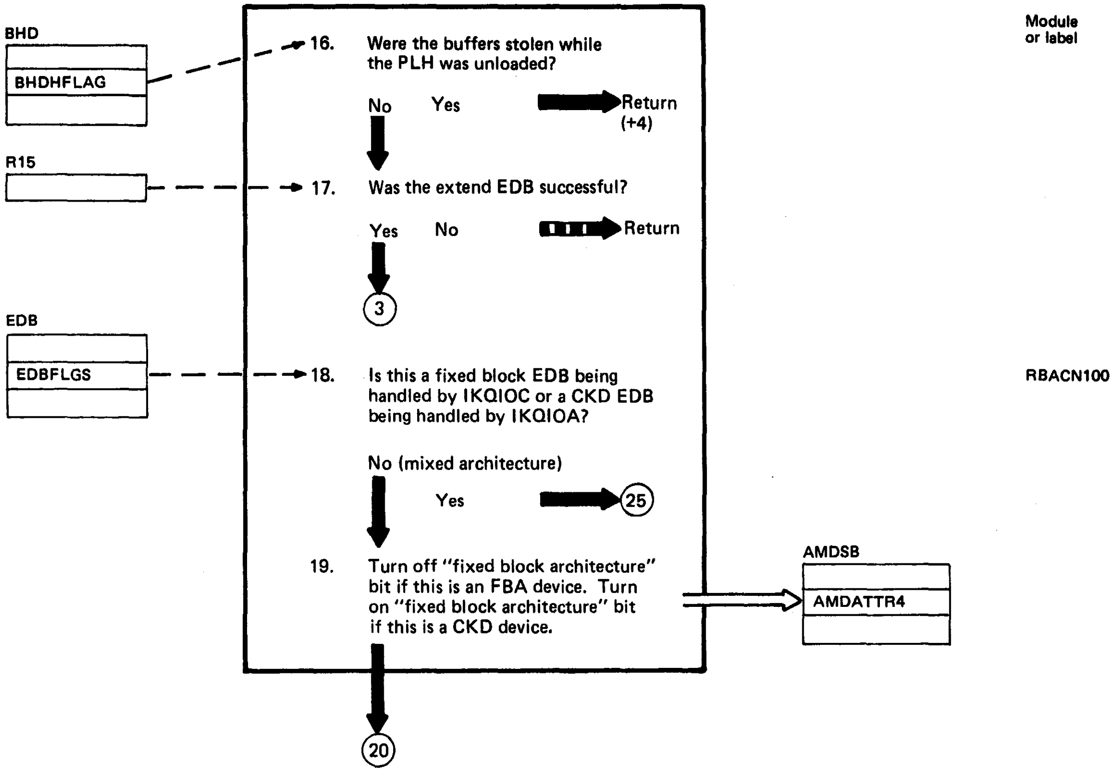
12. This step releases the Buffer Manager (IKQBFA) hold on the PLH ECB (dequeues it) while the extend EDB routine is processing.

14. This step restores the hold on the PLH ECB (enqueues it) for the Buffer Manager.

15.-16. Between the dequeue and enqueue of the PLH ECB, the buffers could have been stolen. Therefore, it is necessary to check if any buffers are still on the I/O queue.

NOTE: If this RBA conversion request was invoked by the Buffer Manager CA hold routine (IKQBFC), the request is retried because no buffers are on the queue at this point; that is, this is the only subroutine in the I/O Manager that does any processing for this buffer manager request.

Diagram ES3. I/O Manager: Convert RBA



Module or label
BHD
BHDHFLAG
R15
EDB
EDBFLGS
16. Were the buffers stolen while the PLH was unloaded?
No
Yes
Return (+4)
17. Was the extend EDB successful?
Yes
No
Return
3
RBACN100
18. Is this a fixed block EDB being handled by IKQIOC or a CKD EDB being handled by IKQIOA?
No (mixed architecture)
Yes
25
19. Turn off "fixed block architecture" bit if this is an FBA device. Turn on "fixed block architecture" bit if this is a CKD device.
AMDSB
AMDATTR4
20

Notes for Diagram ES3

17. After any new EDBs were successfully obtained, the EDB search is retried.

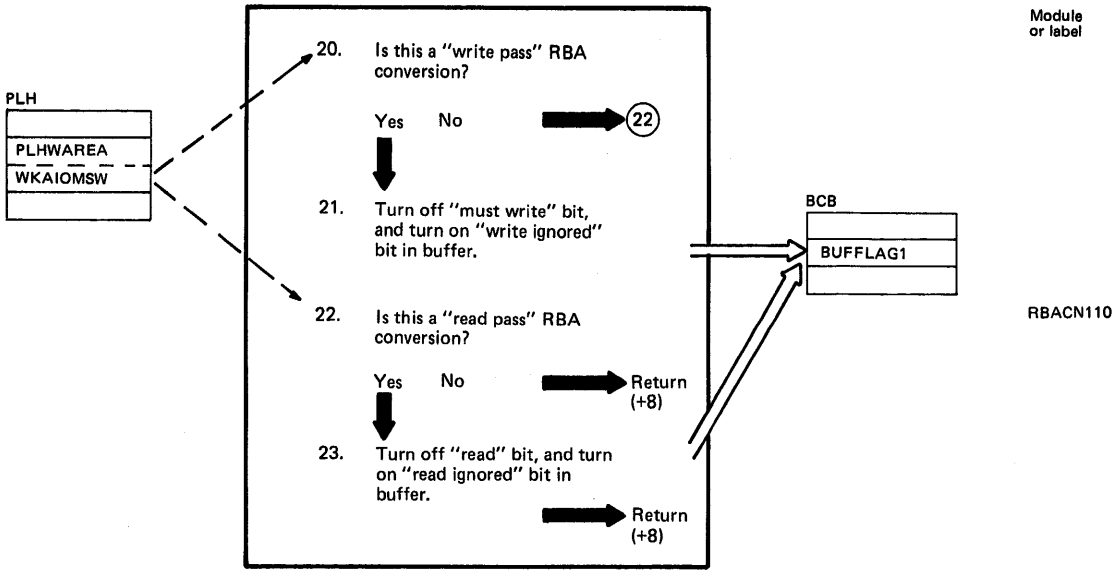
18.-23. The possibility exists that the wrong I/O Manager phase could be invoked when sequentially processing a KSDS index set because the index set is on a fixed block device, while the imbedded sequence set is on a count-key-data device (or vice versa). In this case, I/O is performed only for those buffers associated with the current I/O Manager phase (IKQIOA for CKD, IKQIOC for FBA).

The EDB is tested to determine if the proper I/O Manager phase has control, or if "RBA conversion only" is to be performed.

If neither of the above cases exists, the architecture bit in the AMDSB is turned off if the fixed block I/O Manager is in control, or turned on if the CKD I/O Manager is in control. This causes the Buffer Manager to redrive the I/O requests (that were not completed in the current pass) invoking the other I/O Manager.

NOTE: For "piggy-back" write/read requests, this could result in 3 or 4 passes if an extent boundary is crossed and processing goes from an index set record to a sequence set record (or vice versa).

Diagram ES4. I/O Manager: Convert RBA



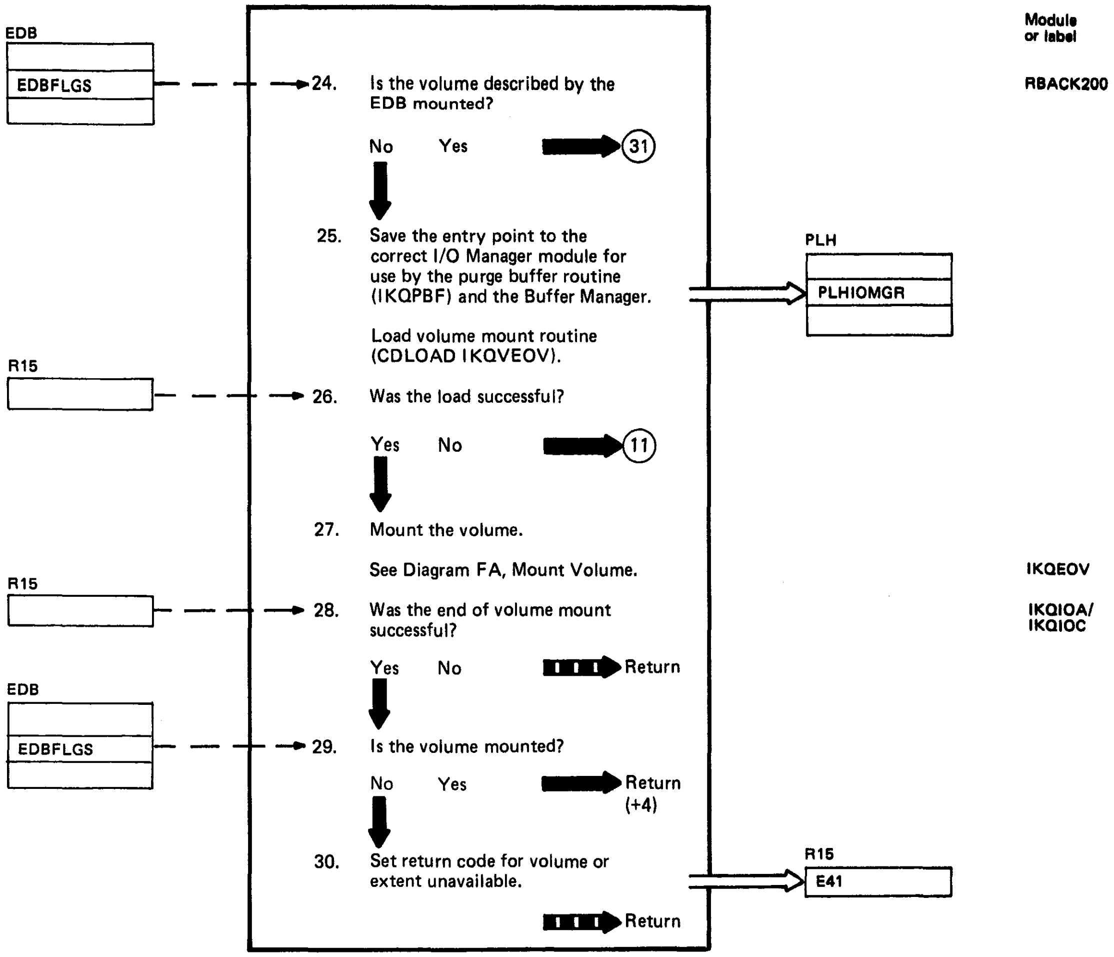
Module or label
PLH
PLHWAREA
WKAIOMSW
20. Is this a "write pass" RBA conversion?
Yes No
22
21. Turn off "must write" bit, and turn on "write ignored" bit in buffer.
22. Is this a "read pass" RBA conversion?
Yes No
Return (+8)
23. Turn off "read" bit, and turn on "read ignored" bit in buffer.
Return (+8)
BCB
BUFFLAG1
RBACN110

Notes for Diagram ES4

20.-21. If the buffer has a write RBA that must be handled by the other I/O Manager phase, then the "must write" bit (BUFCMW) is turned off to inhibit writes, and the "write ignored" bit (BUFWRIGN) is turned on to inform the Buffer Manager that the write for the buffer needs to be redriven.

22.-23. If the buffer has a read RBA that must be handled by the other I/O Manager phase, then the "read" bit (BUFCRRD) is turned off to inhibit reads, and the "read ignored" bit (BUFRDIGN) is turned on to inform the Buffer Manager that the read for the buffer needs to be redriven.

Diagram ES5. I/O Manager: Convert RBA



24. Is the volume described by the EDB mounted?

No / Yes → 31

25. Save the entry point to the correct I/O Manager module for use by the purge buffer routine (IKQPBF) and the Buffer Manager.

Load volume mount routine (CDLOAD IKQVEOV).

26. Was the load successful?

Yes / No → 11

27. Mount the volume.

See Diagram FA, Mount Volume.

28. Was the end of volume mount successful?

Yes / No → Return

29. Is the volume mounted?

No / Yes → Return (+4)

30. Set return code for volume or extent unavailable.

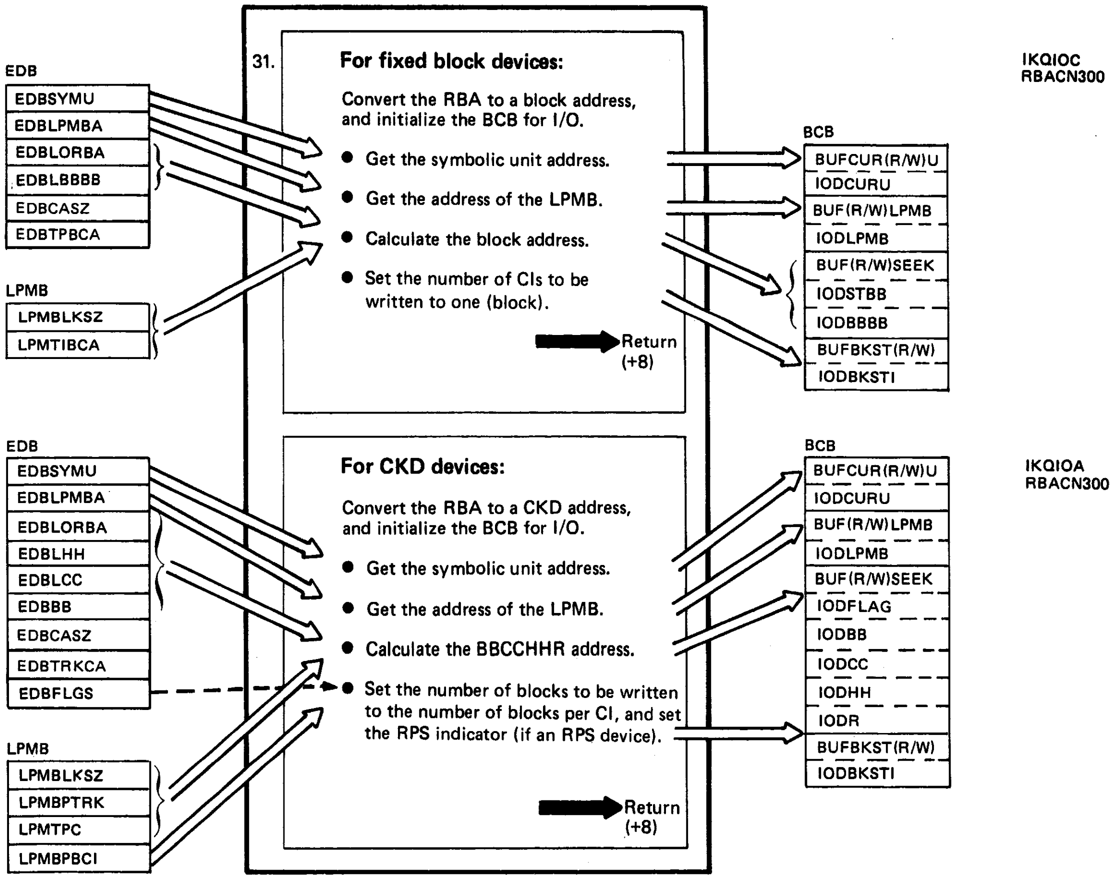
→ Return

Notes for Diagram ES5

24.-30. After the correct EDB has been located, it is tested to see if the volume is mounted. If it is not, the mount volume routine is called. If the correct volume is mounted, the RBA convert routine returns to the I/O Manager mainline routine to retry RBA conversion as described under step 21 of Diagram EA, I/O Manager: Mainline.

If the correct volume cannot be mounted, an error exit is taken.

Diagram ES6. I/O Manager: Convert RBA


EDB
EDBSYMU
EDBLPMBA
EDBLORBA
EDBLBBBB
EDBCASZ
EDBTPBCA
LPMB
LPMBLKSZ
LPMTIBCA
31.
For fixed block devices:
Convert the RBA to a block address, and initialize the BCB for I/O.
● Get the symbolic unit address.
● Get the address of the LPMB.
● Calculate the block address.
● Set the number of CIs to be written to one (block).
Return (+8)
IKQIOC
RBACN300
BCB
BUFCUR(R/W)U
IODCURU
BUF(R/W)LPMB
IODLPMB
BUF(R/W)SEEK
IODSTBB
IODBBBB
BUFBKST(R/W)
IODBKSTI
EDB
EDBSYMU
EDBLPMBA
EDBLORBA
EDBLHH
EDBLCC
EDBBB
EDBCASZ
EDBTRKCA
EDBFLGS
LPMB
LPMBLKSZ
LPMBPTRK
LPMTPC
LPMBPBCI
For CKD devices:
Convert the RBA to a CKD address, and initialize the BCB for I/O.
● Get the symbolic unit address.
● Get the address of the LPMB.
● Calculate the BBCCHHR address.
● Set the number of blocks to be written to the number of blocks per CI, and set the RPS indicator (if an RPS device).
Return (+8)
BCB
BUFCUR(R/W)U
IODCURU
BUF(R/W)LPMB
IODLPMB
BUF(R/W)SEEK
IODFLAG
IODBB
IODCC
IODHH
IODR
BUFBKST(R/W)
IODBKSTI
IKQIOA
RBACN300

Notes for Diagram ES6

31. For fixed block devices, block address conversion is as follows:

   a. Determine the relative displacement of the RBA within its extent (EDB) range, and divide it by the CA size. This step determines the relative CA number within the extent.

   b. Multiply the quotient by the total number of physical blocks in the CA (including any waste blocks at the end of the CA).

   c. Add the starting block number of the extent to the value from step b to get the CA starting block number.

   d. Save the CA starting block number in the BCB.

   e. Divide the remainder calculated in step a by the physical blocksize.

   f. Add the total number of physical blocks used for the imbedded sequence set, plus any unused blocks at the end of the sequence set to the value from step e. (If there is no imbedded sequence set, a zero value is added.)

   g. Store the result from step f in the starting block for the CI.

   For count-key-data devices, CCHHR address conversion is as follows: as follows:

   a. Determine the relative displacement of the RBA within its extent (EDB) range, and divide it by the CA size. This step determines the relative CA number within the extent.

   b. Divide the remainder from step a by the number of bytes per track.

   c. Multiply the quotient from step a by the number of tracks per CA.

   d. Add the quotient from step b to the value obtained in step c.

   e. Divide the remainder from step b by the physical blocksize.

   f. Add 1 to the quotient from step e, and store it in the R field of the MBBCCHHR.

   g. Divide the value obtained in step d by the number of tracks per cylinder.

   h. Add the starting track address of the extent (EDB) range to the value from step g, and store it in the HH field of the MBBCCHHR.

   i. Subtract the number of tracks per cylinder from the value obtained in step h. If the difference is positive or zero, store the value in the HH field of the MBBCCHHR, and add 1 to the quotient obtained in step g.

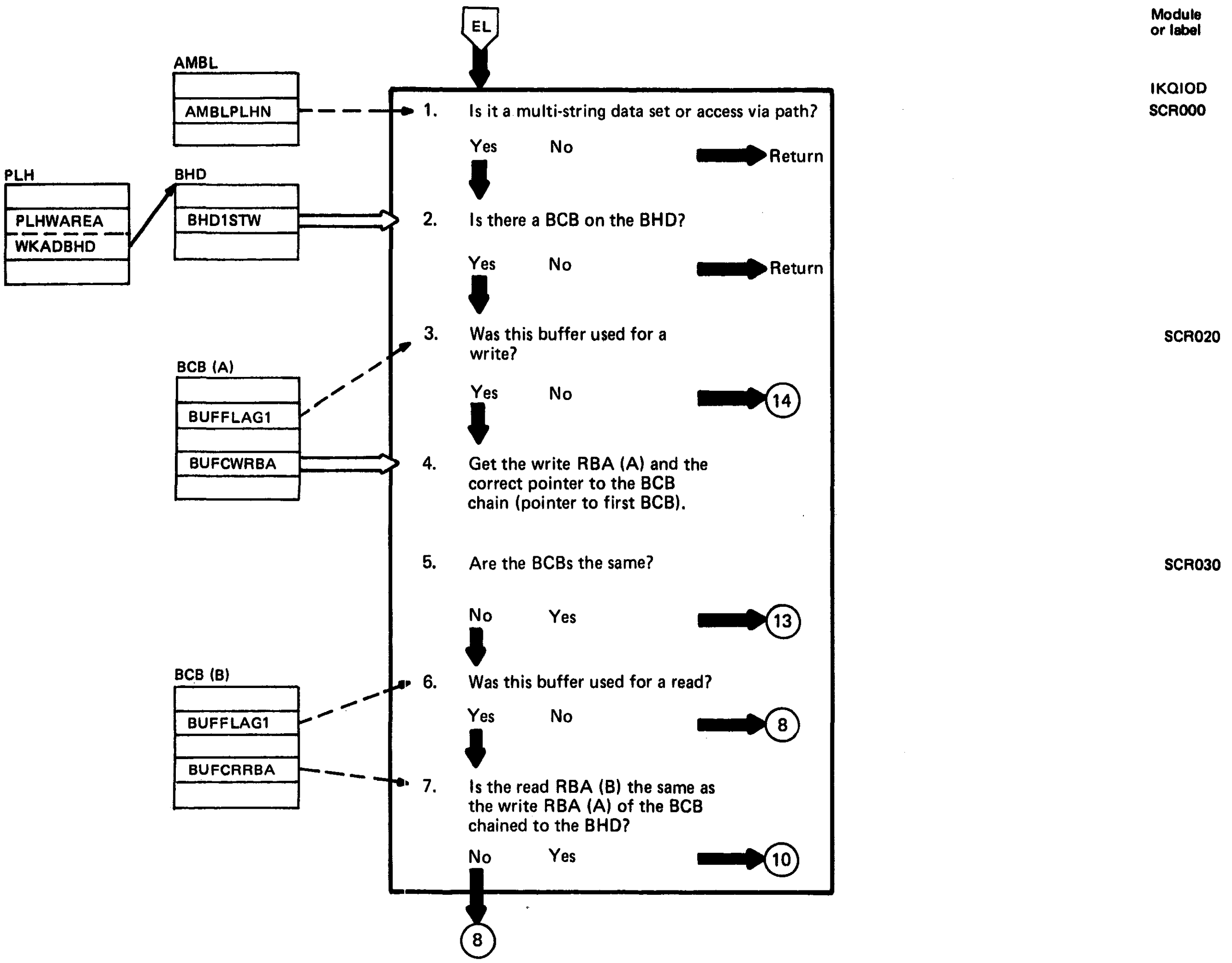
      If the difference is negative, do nothing to the MBBCCHHR or the quotient.

   j. Add the starting cylinder address of the extent (EDB) range to the quotient obtained in step g (plus step i, if applicable). Store the result in the CC field of the MBBCCHHR.

   k. Move the BB field from the EDB to the BB field of the MBBCCHHR.

      Set the M field to X'00' (not an RPS device) or to X'80' (RPS device).

Diagram ET1. I/O Manager: Scratch Buffers

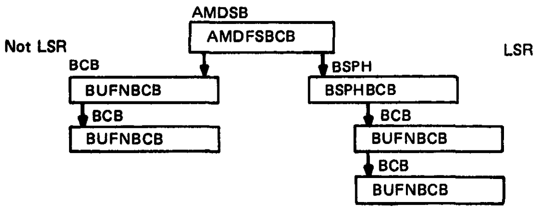

EL
Module or label
AMBL
AMBLPLHN
1. Is it a multi-string data set or access via path?
IKQIOD
SCR000
Yes
No
Return
PLH
PLHWAREA
WKADBHD
BHD
BHD1STW
2. Is there a BCB on the BHD?
Yes
No
Return
3. Was this buffer used for a write?
SCR020
BCB (A)
BUFFLAG1
BUFCWRBA
Yes
No
14
4. Get the write RBA (A) and the correct pointer to the BCB chain (pointer to first BCB).
5. Are the BCBs the same?
SCR030
No
Yes
13
BCB (B)
BUFFLAG1
BUFCRRBA
6. Was this buffer used for a read?
Yes
No
8
7. Is the read RBA (B) the same as the write RBA (A) of the BCB chained to the BHD?
No
Yes
10
8

Notes for Diagram ET1

This routine invalidates buffers (with the same read/write RBA) associated with other PLH strings for the same data set when the buffer for the current PLH has been written.

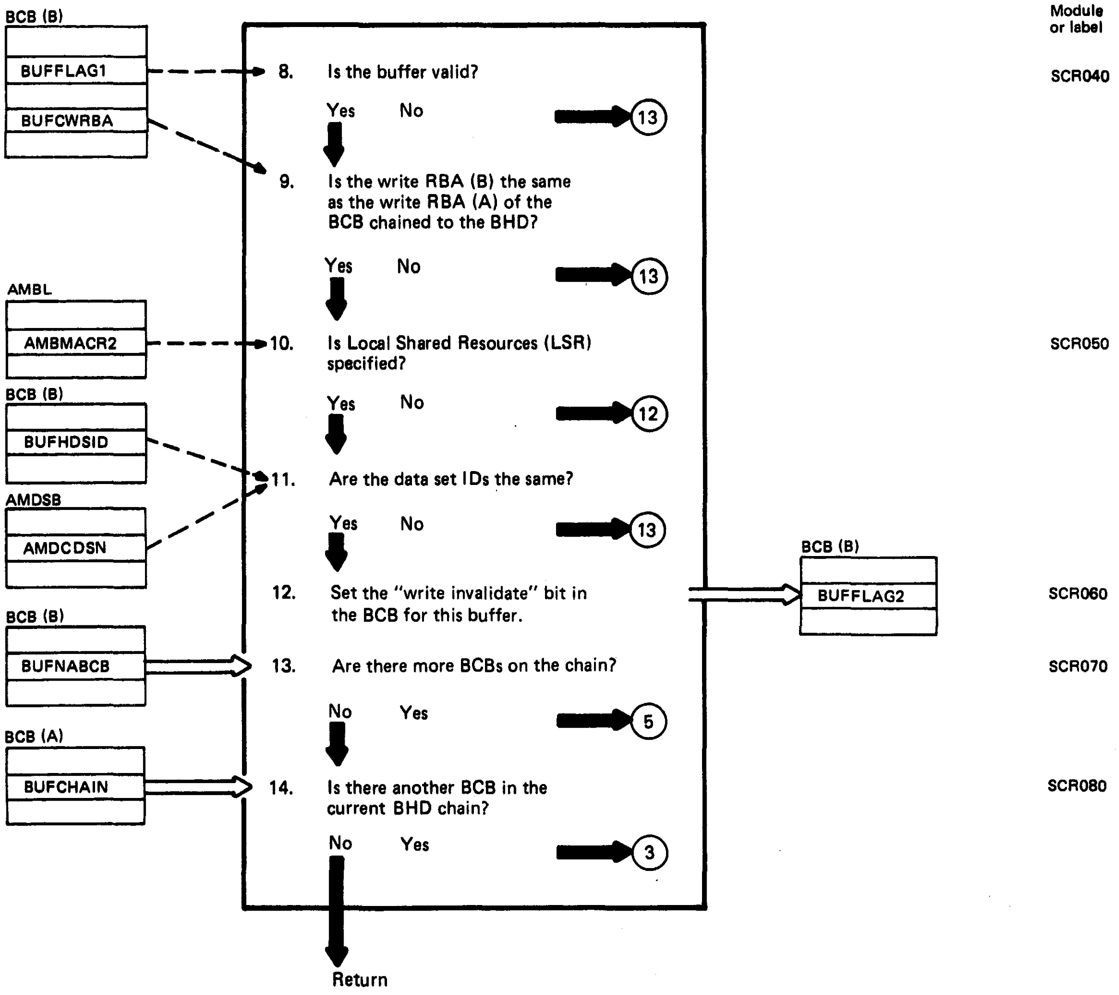
1.-3. If the AMBL indicates a multi-string data set and there are buffers on the BHD that have their write indicators on (BUFCMW and BUFCFMT in BUFFLAG1), the write RBA of each buffer on the queue is compared to the read/write RBAs of all buffers (for other PLHs) for the same AMDSB.

4.-5. The buffer pool header is obtained from the AMDSB (or the BSPH for local shared resources), and the BCB is compared to that obtained in step 3 or step 14. If they are the same, the buffer is skipped.



6.-9. If the string IDs do not match, the RBAs of the buffer just obtained are compared to the write RBA of the current buffer. If the read (or write) RBA does not match, or if the validity bit is off (BUFCVAL in BUFFLAG1), the next buffer in the pool is checked.

Diagram ET2. I/O Manager: Scratch Buffers

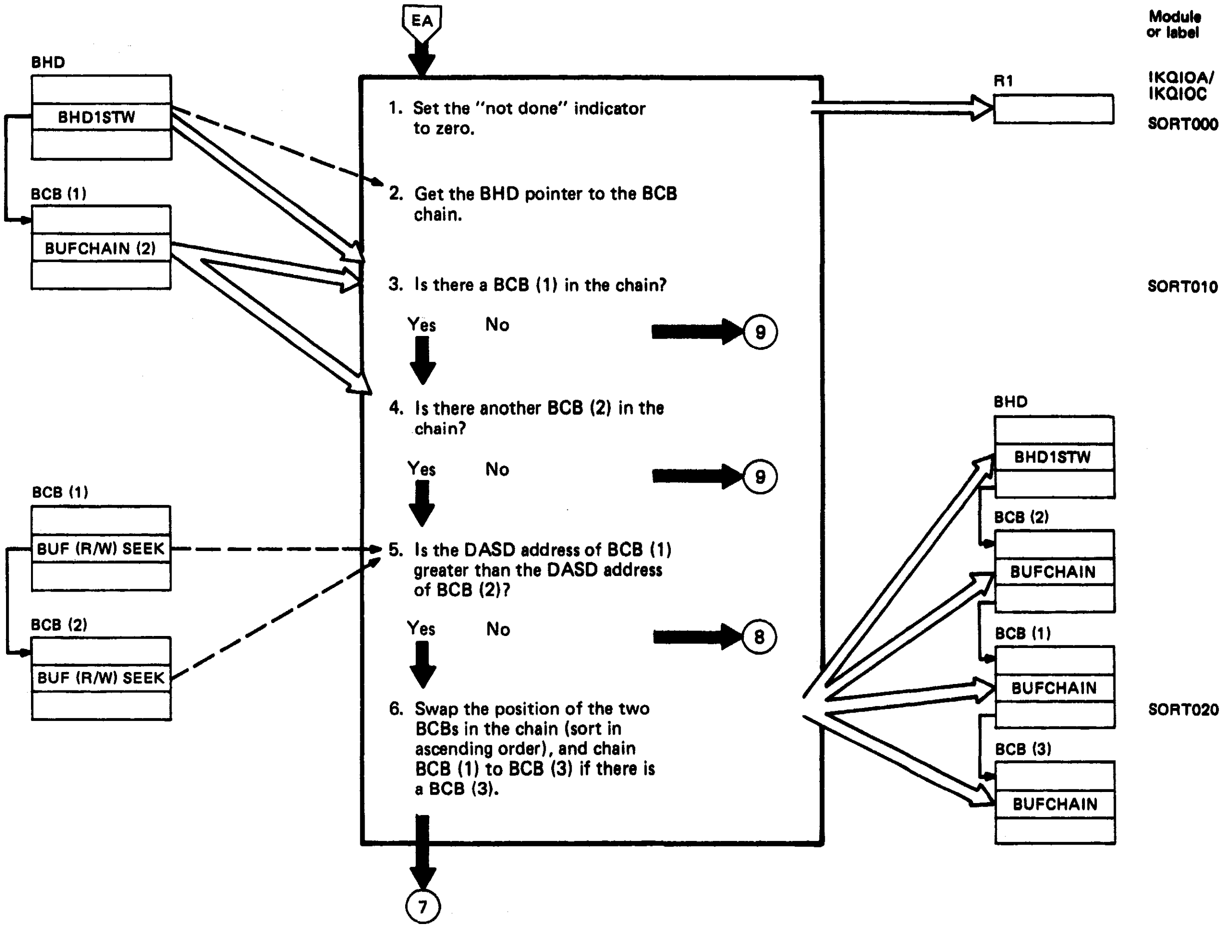

Module or label
BCB (B)
BUFFLAG1
BUFCWRBA
8. Is the buffer valid?
Yes No
13
SCR040
9. Is the write RBA (B) the same as the write RBA (A) of the BCB chained to the BHD?
Yes No
13
AMBL
AMBMACR2
10. Is Local Shared Resources (LSR) specified?
SCR050
Yes No
12
BCB (B)
BUFHDSID
AMDSB
AMDCDSN
11. Are the data set IDs the same?
Yes No
13
BCB (B)
BUFFLAG2
12. Set the "write invalidate" bit in the BCB for this buffer.
SCR060
BCB (B)
BUFNABCB
13. Are there more BCBs on the chain?
SCR070
No Yes
5
BCB (A)
BUFCHAIN
14. Is there another BCB in the current BHD chain?
SCR080
No Yes
3
Return

Notes for Diagram ET2

10.-12. If LSR was not specified, or if the read RBAs matched, or if the buffer was valid and the write RBAs matched, the write invalid bit is turned on in the buffer in the buffer pool (no LSR). For LSR, the buffer in the LSR pool must have its ID compared to the ID of the current AMDSB to determine if the buffer is for the same data set.

13.-14. Each buffer in the pool is compared to each buffer in the chain in a similar manner.

Diagram EU1. I/O Manager: Sort BCBs



1. Set the "not done" indicator to zero.

2. Get the BHD pointer to the BCB chain.

3. Is there a BCB (1) in the chain?

4. Is there another BCB (2) in the chain?

5. Is the DASD address of BCB (1) greater than the DASD address of BCB (2)?

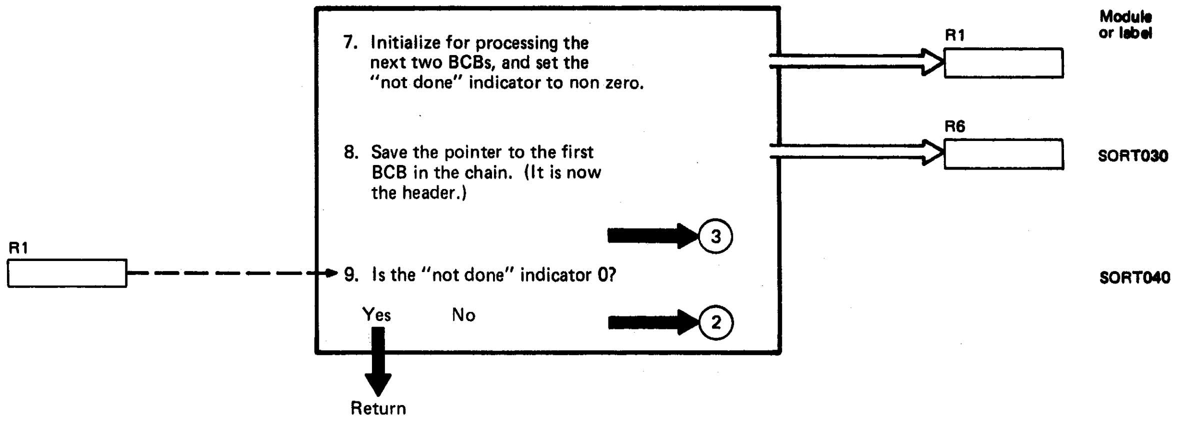
6. Swap the position of the two BCBs in the chain (sort in ascending order), and chain BCB (1) to BCB (3) if there is a BCB (3).

Notes for Diagram EU1

1.-4. The BHD pointer to the I/O queue is made to look like a BCB chain pointer. No sorting is done if there is only one (or none) BCB in the queue.

5.-8. The DASD (read/write) address of the first BCB is compared to the DASD (read/write) address of the second BCB (depending on whether reads or writes are being processed). The two BCBs are rechained in ascending sequence, based on their DASD addresses. The second BCB, if rechained, is now compared to the third BCB, if any, in a like manner.

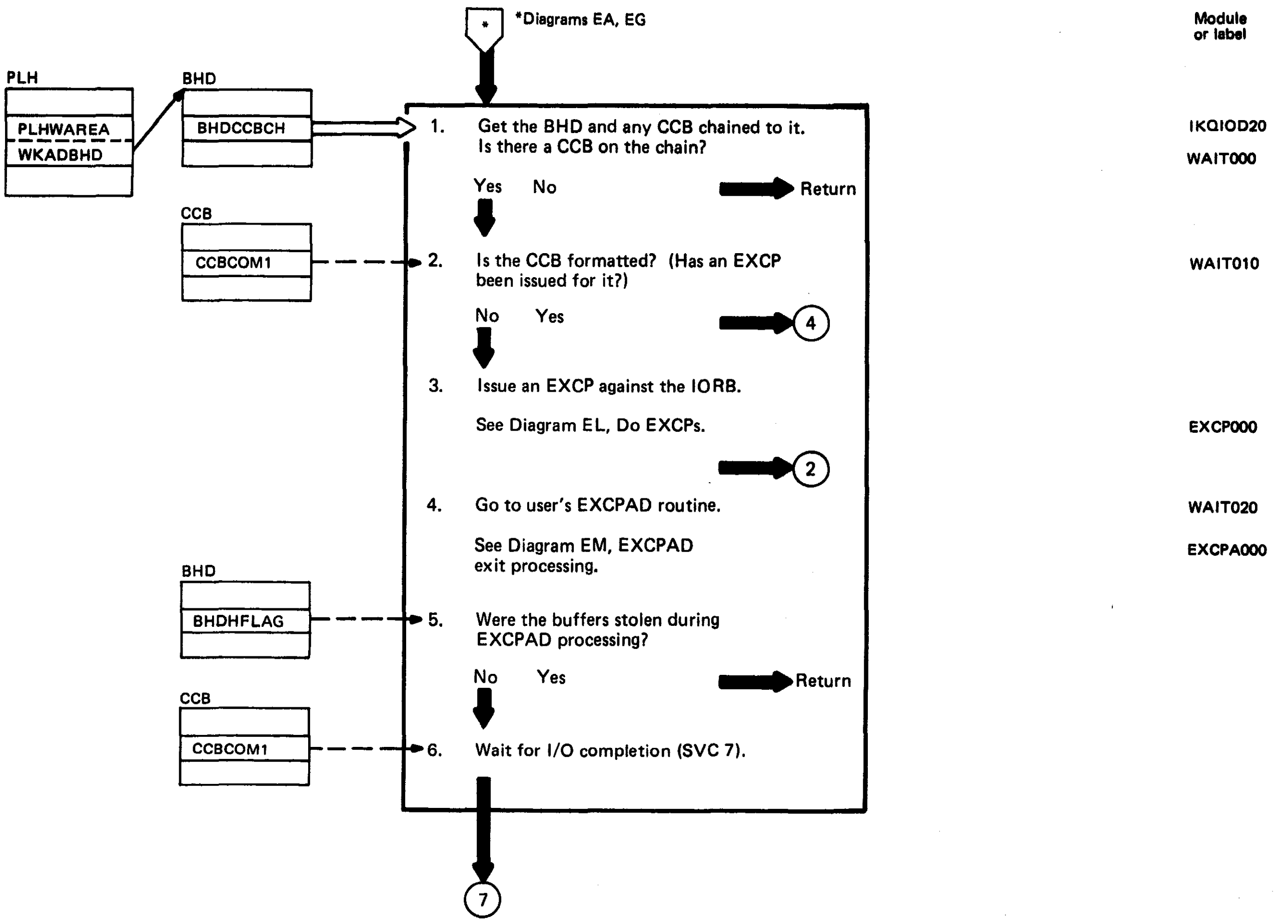
Diagram EU2. I/O Manager: Sort BCBs


Module or label
7. Initialize for processing the next two BCBs, and set the "not done" indicator to non zero.
R1
8. Save the pointer to the first BCB in the chain. (It is now the header.)
R6
SORT030
3
R1
9. Is the "not done" indicator 0?
SORT040
Yes
No
2
Return

Notes for Diagram EU2

9. After the first pass through the chain, the BCB with the highest DASD address is positioned at the end of the chain. Another pass is made through the chain to sort the other BCBs preceding it in the chain.

Diagram EV1. I/O Manager: Wait for I/O Completion


*Diagrams EA, EG
Module or label
PLH
PLHWAREA
WKADBHD
BHD
BHDCCBCH
1. Get the BHD and any CCB chained to it. Is there a CCB on the chain?
Yes No
Return
IKQIOD20
WAIT000
CCB
CCBCOM1
2. Is the CCB formatted? (Has an EXCP been issued for it?)
No Yes
4
WAIT010
3. Issue an EXCP against the IORB.
See Diagram EL, Do EXCPs.
EXCP000
2
4. Go to user's EXCPAD routine.
See Diagram EM, EXCPAD exit processing.
WAIT020
EXCPA000
BHD
BHDHFLAG
5. Were the buffers stolen during EXCPAD processing?
No Yes
Return
CCB
CCBCOM1
6. Wait for I/O completion (SVC 7).
7

Notes for Diagram EV1

1. This routine is used for both count-key-data and fixed block devices. The first CCB on the BHD chain is obtained for WAIT processing. There is always one CCB with an associated CCW, Fix List, and I/O Data Blocks left on the BHD queue, although the buffers may be gone. The CCB left on the queue is the one originally put on the queue.

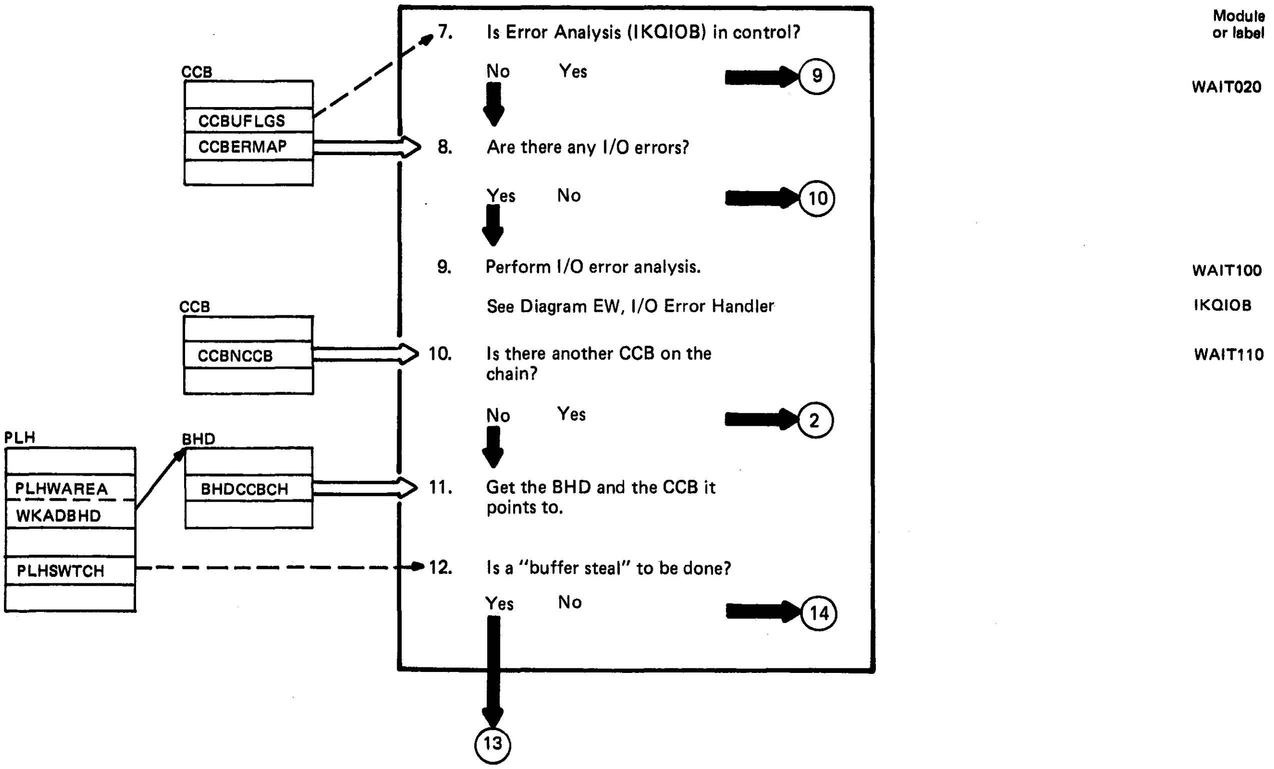
2.-3. The CCB (CCBERROK) is tested to determine if an EXCP is required.

4. Before the WAIT is issued, the user's EXCPAD routine will be called (if one was specified).

5. After return from the EXCPAD routine, the BHD is tested to determine if a buffer steal occurred. If no buffer steal occurred, the WAIT is issued if the IORB has not yet been posted I/O complete (CCBWAIT).

   NOTE: A "buffer steal" can occur during EXCPAD processing (while the PLH ECB is unlocked). A different task can enter the wait routine during processing and initiate a "buffer steal". The wait routine then waits on the IORBs of the task for which the EXCPAD exit has been taken. Therefore, when control returns from the EXCPAD exit, I/O must be restarted for the task that had its buffers stolen. Because there are no buffers on the queue, the I/O Manager returns to the Buffer Manager to restart the I/O requests.

Diagram EV2. I/O Manager: Wait for I/O Completion


Module or label
CCB
CCBUFLGS
CCBERMAP
7. Is Error Analysis (IKQIOB) in control?
No Yes
9
WAIT020
8. Are there any I/O errors?
Yes No
10
9. Perform I/O error analysis.
WAIT100
See Diagram EW, I/O Error Handler
IKQIOB
CCB
CCBNCCB
10. Is there another CCB on the chain?
WAIT110
No Yes
2
PLH
PLHWAREA
WKADBHD
PLHSWTCH
BHD
BHDCCBCH
11. Get the BHD and the CCB it points to.
12. Is a "buffer steal" to be done?
Yes No
14
13

Notes for Diagram EV2

7. If the "error analysis in control" indicator (CCBUEAIC) is set on (set on by I/O Error Handler, IKQIOB), control returns to IKQIOB to complete error analysis. This occurs during error handling for a "piggy-back" I/O operation.

   When an error occurs involving a "piggy-back" I/O operation, the write portion of the CCW string must be separated from the read portion. The first call to the I/O Error Handler causes the CCW string to be split. The I/O Error Handler turns on the CCBUEAIC bit in the CCB to indicate that error analysis is in control. It then returns to the WAIT routine to retry the failing portion of the CCW string and allow VSE ERP to process the error.

8. The IORB is tested for any I/O errors; if none have occurred, the next CCB in the chain is processed in a similar manner.

9. If any errors were encountered, the I/O Error Handler is called.

   NOTE: The bit CCBUERR (set in the CCB by BLDCP) not only indicates a "piggy-back" I/O operation, but also inhibits VSE ERP from retrying the error. Retry is not desirable the first time because if the error occurred during the read operation, then retry will rewrite the wrong data on the device when the Write CCWs are retried.

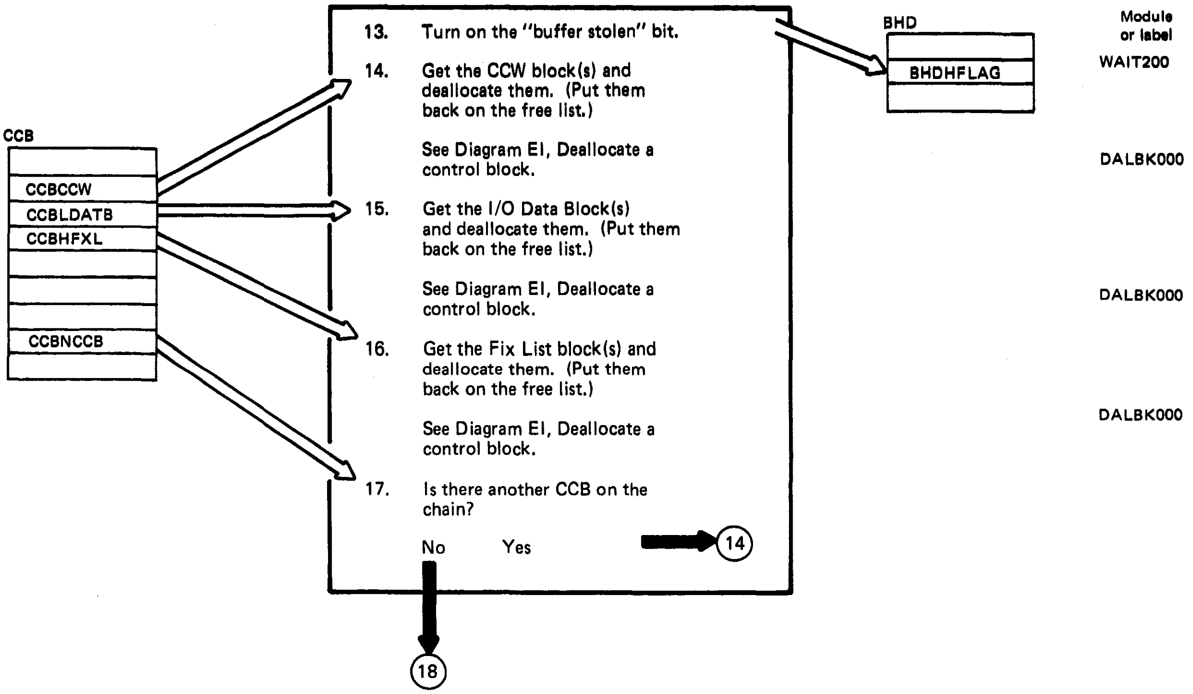
   After the first pass is made on the failing CCW string, the I/O Error Handler is called a second time to complete the error analysis.

   If the error occurred during the Write CCWs, the I/O Error Handler will be called three times, once to retry the Write CCWs, once to retry the Read CCWs, and once to complete the error analysis.

11.-18. After all IORBs have been tested for I/O errors and any I/O errors have been processed (or after a "buffer steal" has been done), control blocks used for I/O are deallocated.

12. If a "buffer steal" is to be done, all CCBs in the chain and their associated control blocks are deallocated and returned to the control block free queue. For a "buffer steal," the CCB chain pointer in the BHD is cleared to indicate the end of the CCB chain.

Diagram EV3. I/O Manager: Wait for I/O Completion



13. Turn on the "buffer stolen" bit.

14. Get the CCW block(s) and deallocate them. (Put them back on the free list.)

    See Diagram EI, Deallocate a control block.

15. Get the I/O Data Block(s) and deallocate them. (Put them back on the free list.)

    See Diagram EI, Deallocate a control block.

16. Get the Fix List block(s) and deallocate them. (Put them back on the free list.)

    See Diagram EI, Deallocate a control block.

17. Is there another CCB on the chain?

    No Yes

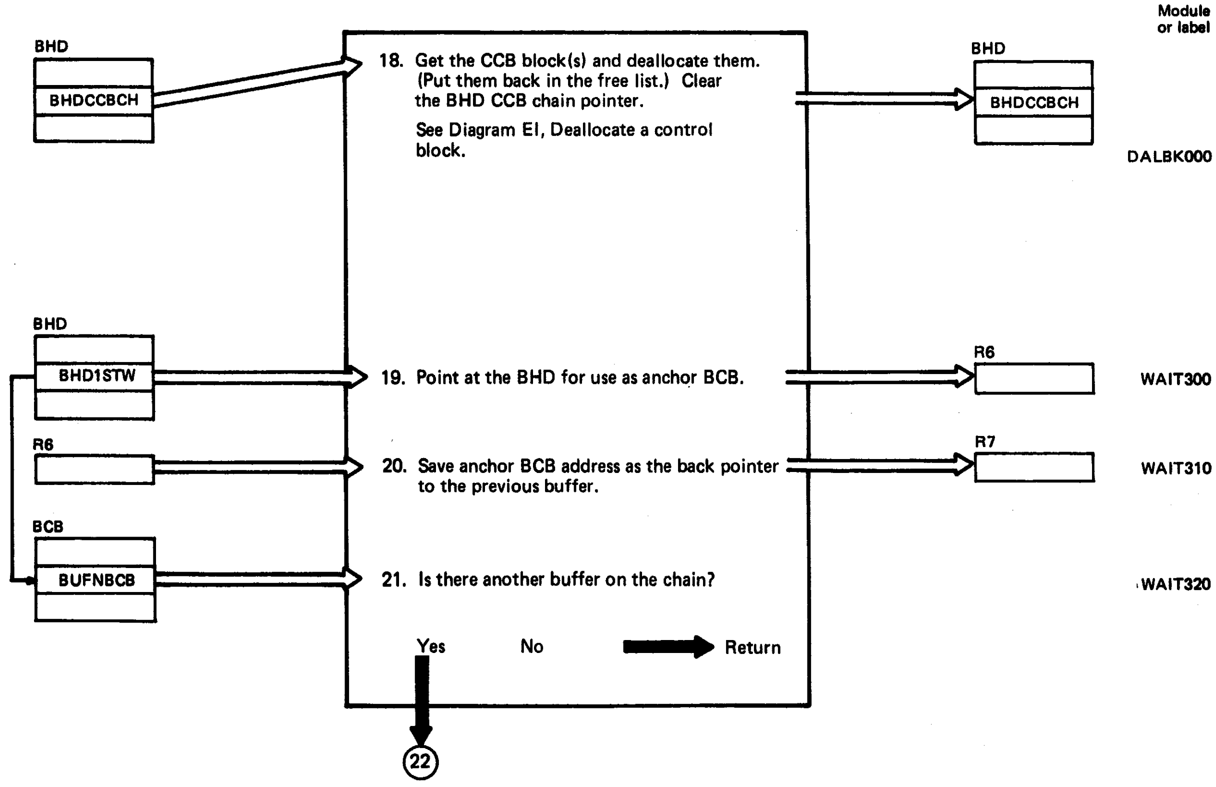
Notes for Diagram EV3

16. If a Fix List was not built (because execution is on a S/370), no deallocation takes place.

19.-29. The buffer queues are searched for any buffer used for a replicated index read operation, and the count field (CCHH) of the record read is compared to the CCHH that was supposed to have been read. (The count field is read into the BCB.) This procedure checks the possibility of a bad seek occurring, particularly on a 2314 device.

If a bad seek occurred, the error code (E65 – index error; E66 – sequence set error) is placed in the PLH, and the buffer in error is removed from the I/O queue (either scheduled or non-scheduled queue) and placed on the free queue. If this is the first buffer on the queue, then the second buffer on the I/O queue becomes the first buffer on the queue, and the pointer in either the scheduled or non-scheduled queue must be updated accordingly to match the pointer on the I/O queue.

Diagram EV4. I/O Manager: Wait for I/O Completion



18. Get the CCB block(s) and deallocate them. (Put them back in the free list.) Clear the BHD CCB chain pointer.

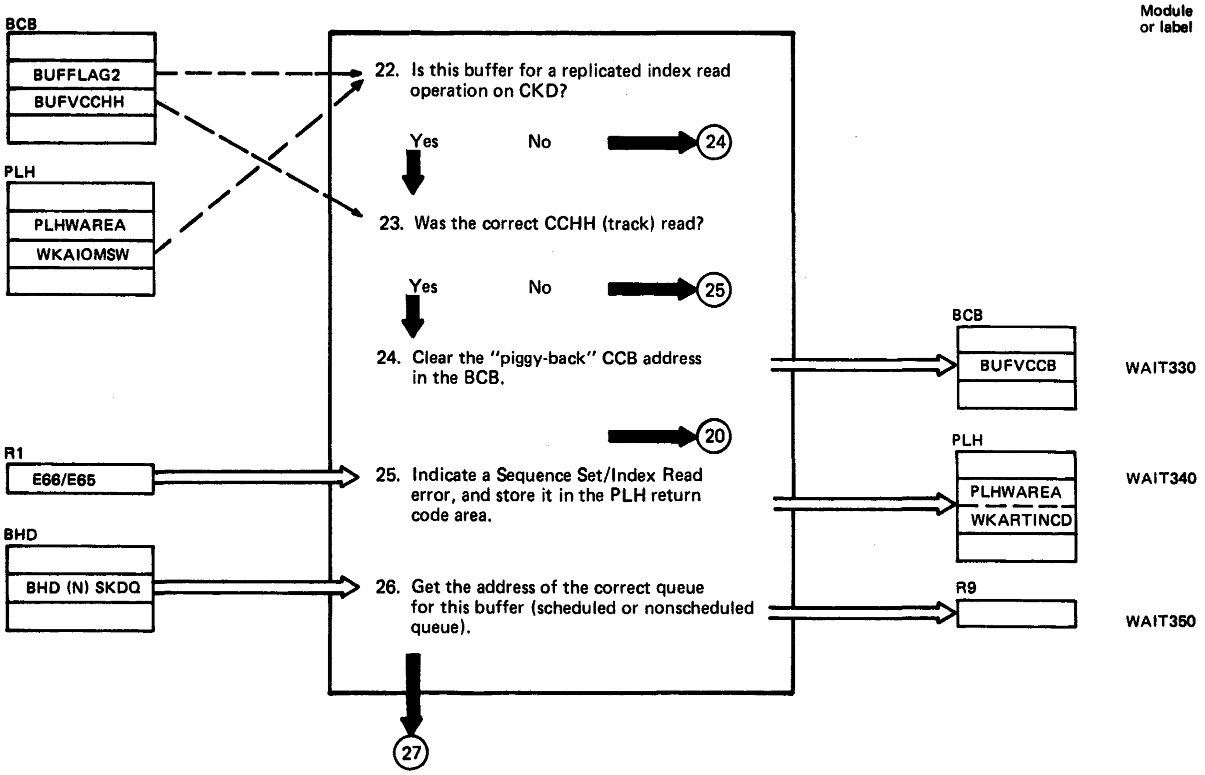
    See Diagram EI, Deallocate a control block.

19. Point at the BHD for use as anchor BCB.

20. Save anchor BCB address as the back pointer to the previous buffer.

21. Is there another buffer on the chain?

Diagram EV5. I/O Manager: Wait for I/O Completion



22. Is this buffer for a replicated index read operation on CKD?

23. Was the correct CCHH (track) read?

24. Clear the "piggy-back" CCB address in the BCB.

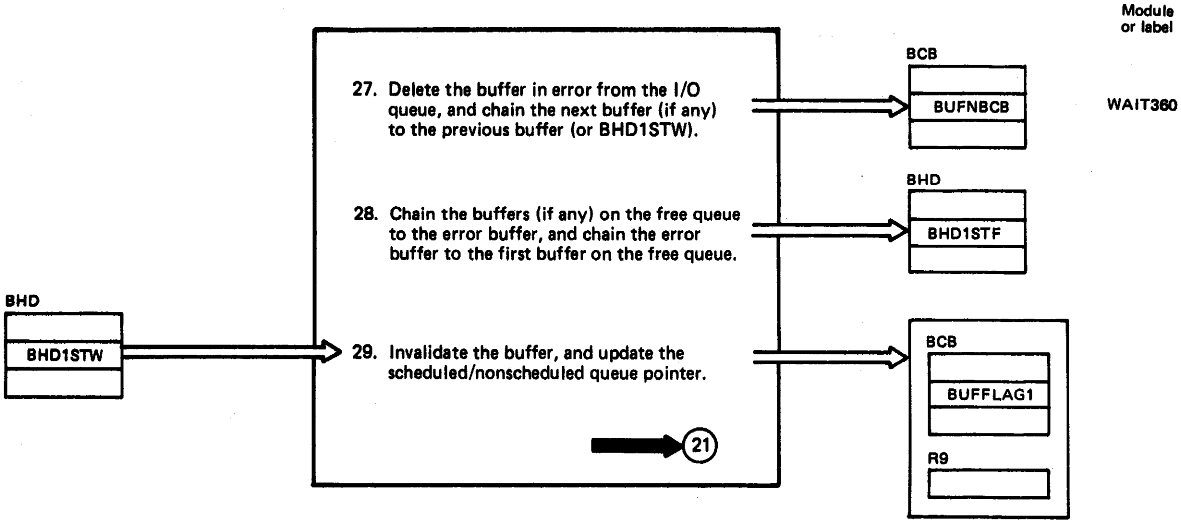
25. Indicate a Sequence Set/Index Read error, and store it in the PLH return code area.

26. Get the address of the correct queue for this buffer (scheduled or nonscheduled queue).

Notes for Diagram EV5

24. The BUFVCCB field must be cleared when I/O has been done for only some of the buffers. In this case, the write for a "piggy-back" buffer would have been done, but not the read. The BLDCP will redrive the buffer, but it will no longer be used for a "piggy-back" I/O operation.

Diagram EV6. I/O Manager: Wait for I/O Completion

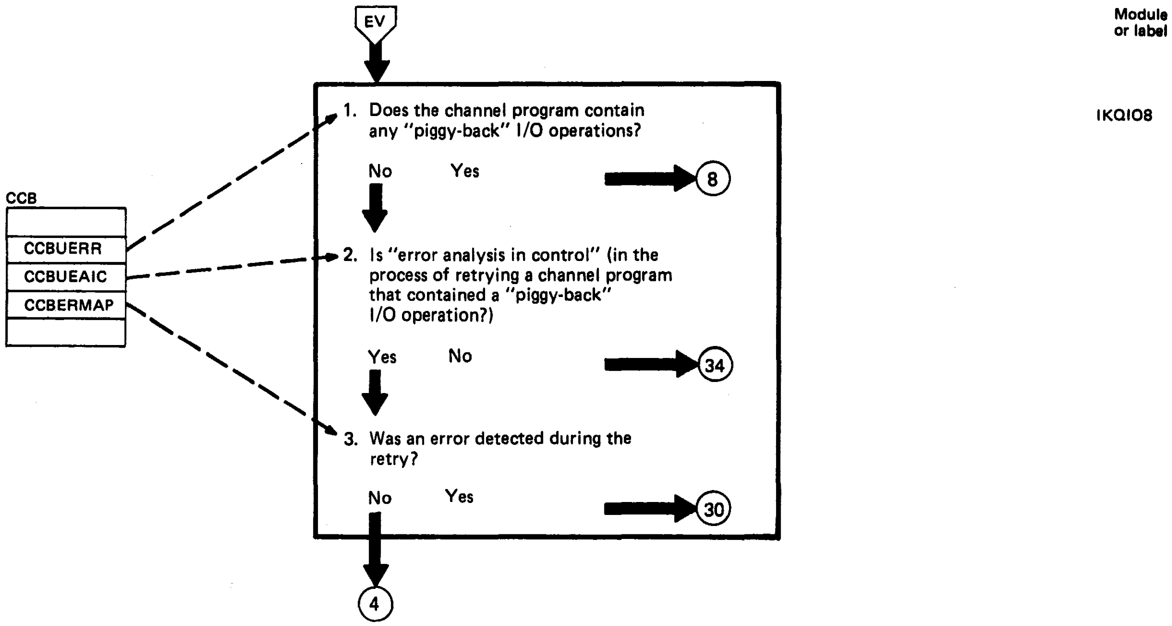


27. Delete the buffer in error from the I/O queue, and chain the next buffer (if any) to the previous buffer (or BHD1STW).

28. Chain the buffers (if any) on the free queue to the error buffer, and chain the error buffer to the first buffer on the free queue.

29. Invalidate the buffer, and update the scheduled/nonscheduled queue pointer.

Diagram EW1. I/O Error Handler


EV
Module or label
1. Does the channel program contain any "piggy-back" I/O operations?
IKQIO8
No
Yes
8
CCB
CCBUERR
CCBUEAIC
CCBERMAP
2. Is "error analysis in control" (in the process of retrying a channel program that contained a "piggy-back" I/O operation?)
Yes
No
34
3. Was an error detected during the retry?
No
Yes
30
4

Notes for Diagram EW1

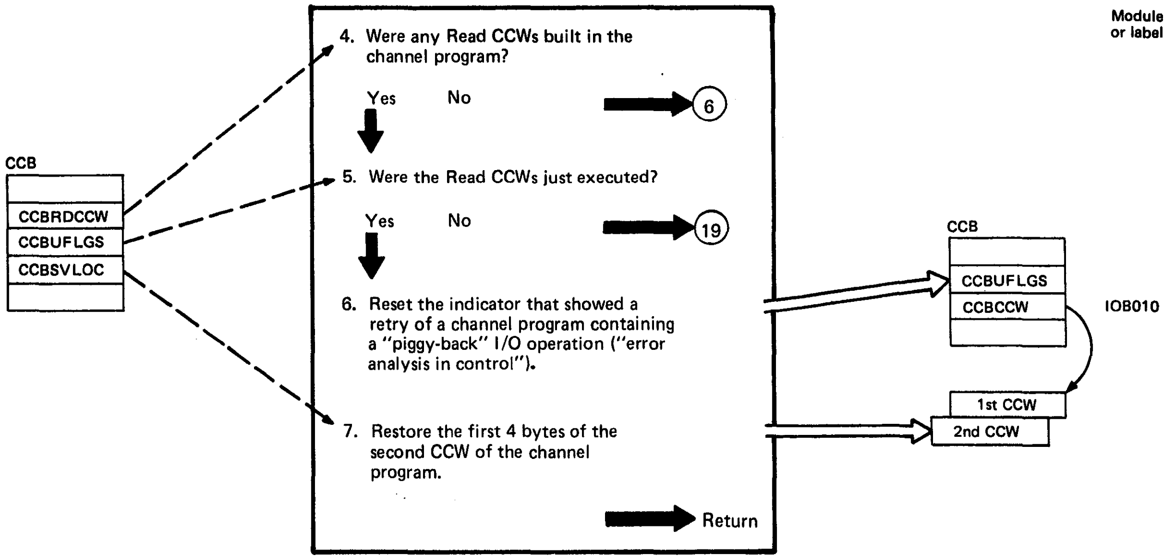
Note that the control block passed to the supervisor at EXCP time is the IORB, but VSAM has retained the use of labels beginning with the characters "CCB".

1. IKQIOB is called by the wait routine of the I/O Manager (IKQIOA or IKQIOC) when an IORB is found that has an error or that is in the process of retry. The channel programs that are retried are those that have at least one buffer that is written out of by the first part of the channel program, and read back into by the second part of the channel program ("piggy-back" I/O). For these channel programs, the flag CCBUERR was set on by the I/O Manager. This flag causes the supervisor to suppress error recovery for these channel programs. Error recovery is suppressed because, in general, the supervisor may retry a channel program from the beginning. Retry from the beginning could cause the wrong contents of the buffer to be written out.

   The method of retry consists of splitting the channel program into two parts, the first containing the Write CCWs and the second containing the Read CCWs. Each of these is executed as an independent channel program, with the CCBUERR bit off.

   Any channel program received in error by IKQIOB (CCBUERR off) is scanned from the point the error occurred to determine which buffers are affected by the error.
2. This step tests whether error analysis is in control.

Diagram EW2. I/O Error Handler



4. Were any Read CCWs built in the channel program?
5. Were the Read CCWs just executed?
6. Reset the indicator that showed a retry of a channel program containing a "piggy-back" I/O operation ("error analysis in control").
7. Restore the first 4 bytes of the second CCW of the channel program.

Notes for Diagram EW2

4. CCBRDCCW will have been stored by the I/O Manager the first time a CCW related to read operations is built. This will be a Head Seek CCW for CKD devices, or a Locate CCW for fixed block devices.

   It is possible that a channel program flagged with the CCBUERR bit may not actually contain any Read CCWs. This can occur if:
   - The channel program is broken due to the Read CCWs being on a different cylinder (CKD devices) or CA (fixed block devices) from the Write CCWs; or
   - There is a large number of Write CCWs; or
   - The partition is running out of storage in which to build channel programs and their related control blocks.

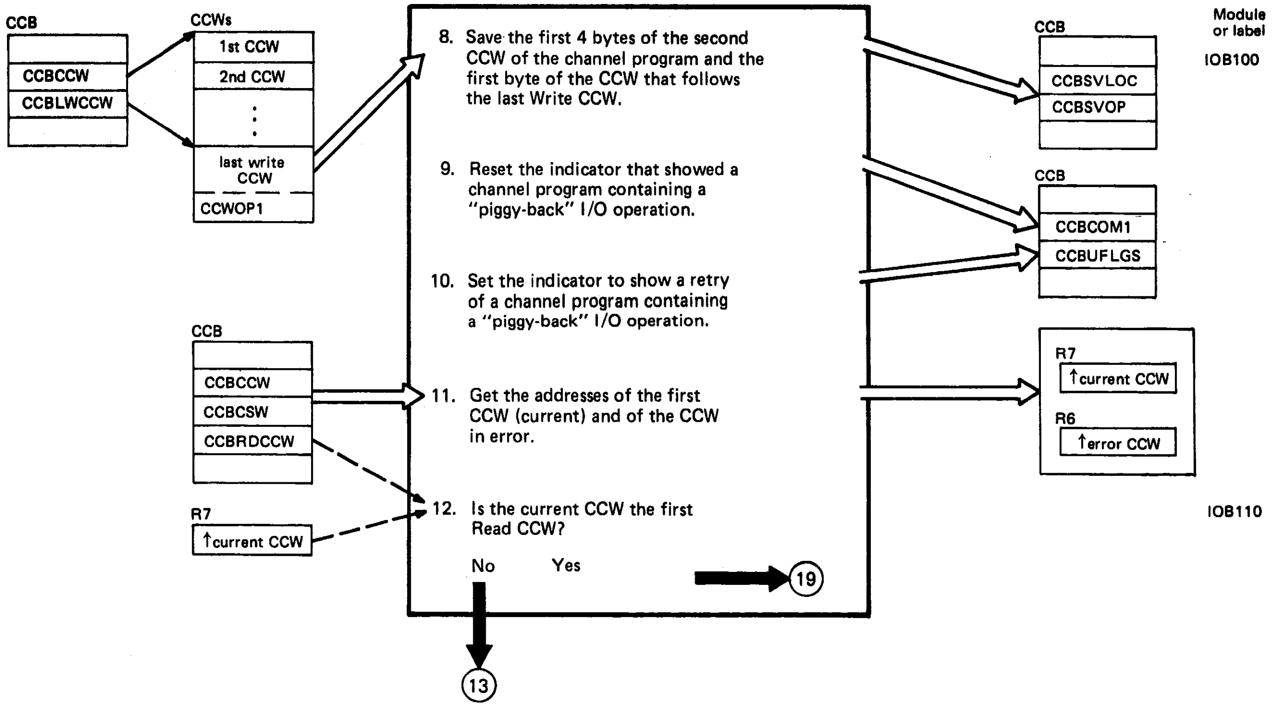
5. During the retry, the Write CCWs are EXCPed and completed before the Read CCWs are EXCPed. To distinguish between these two separate EXCPs, the bit CCBURDCW is set when the EXCP is being done for read operations.

6. The "error analysis in control" indicator is reset because the retry has been successfully completed for both the Write and Read CCW portions of the channel program.

7.-8. The first CCW of the complete channel program is used during the EXCP of both write and read operations. (This CCW is the Full Seek CCW for CKD devices or the Define Extent CCW for fixed block devices.) In order to use this CCW during the read EXCP, the first 4 bytes of the second CCW are saved. The second CCW is then overlaid, and it is changed to a TIC CCW during the read operations.

The CCW op code of the CCW following the last Write CCW is saved so that it can be changed later to a NOP CCW. (This is required if the Write Check option is used with fixed block devices.)

Diagram EW3. I/O Error Handler



8. Save the first 4 bytes of the second CCW of the channel program and the first byte of the CCW that follows the last Write CCW.

9. Reset the indicator that showed a channel program containing a "piggy-back" I/O operation.

10. Set the indicator to show a retry of a channel program containing a "piggy-back" I/O operation.

11. Get the addresses of the first CCW (current) and of the CCW in error.

12. Is the current CCW the first Read CCW?

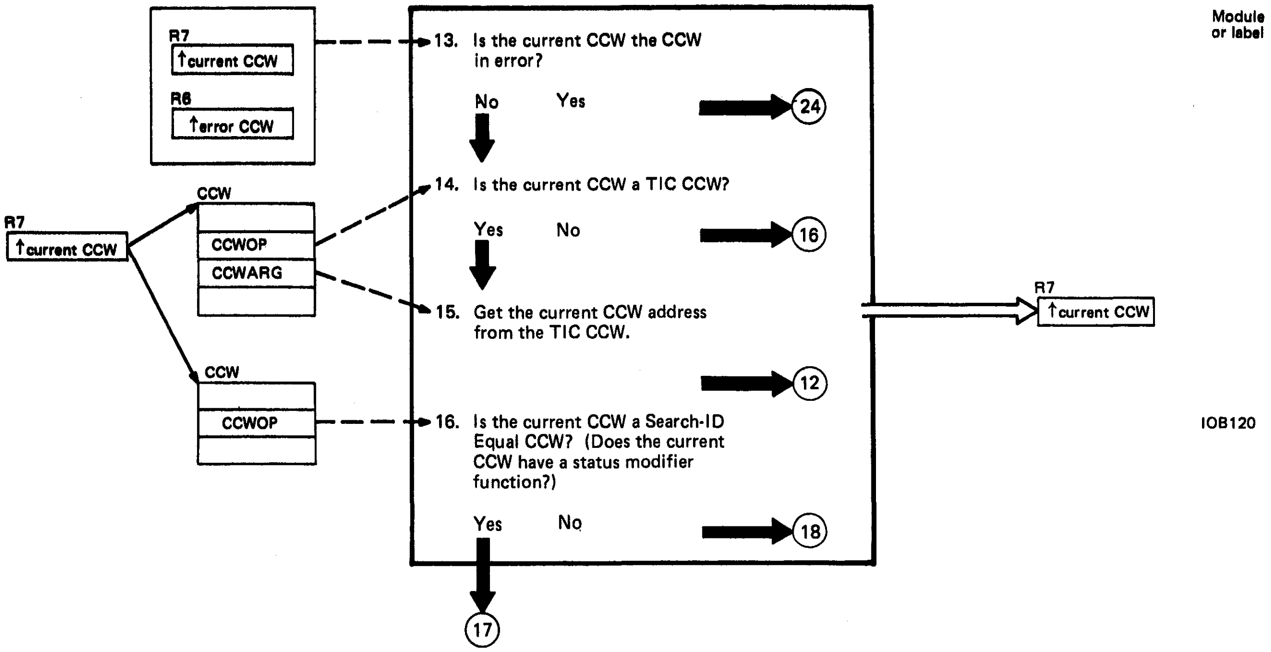
No Yes

Notes for Diagram EW3

9.-10. The bit CCBUERR is reset so that the system ERP routines will retry any I/O errors that may occur when the channel program is redriven. The bit CCBUEAIC is set to indicate that error analysis is in control (the channel program has been split into its write and read portions), and that the I/O Error Handler is to get control after each redrive of the channel program by the I/O Manager Wait routine.

11.-18. This loop determines whether the error occurred during the Write CCWs or the Read CCWs. If the error occurred during the Write CCWs, then the full retry scheme must be used. If the error occurred during the Read CCWs, then the write operations have been completed successfully, and only the read operations need to be retried.

Diagram EW4. I/O Error Handler



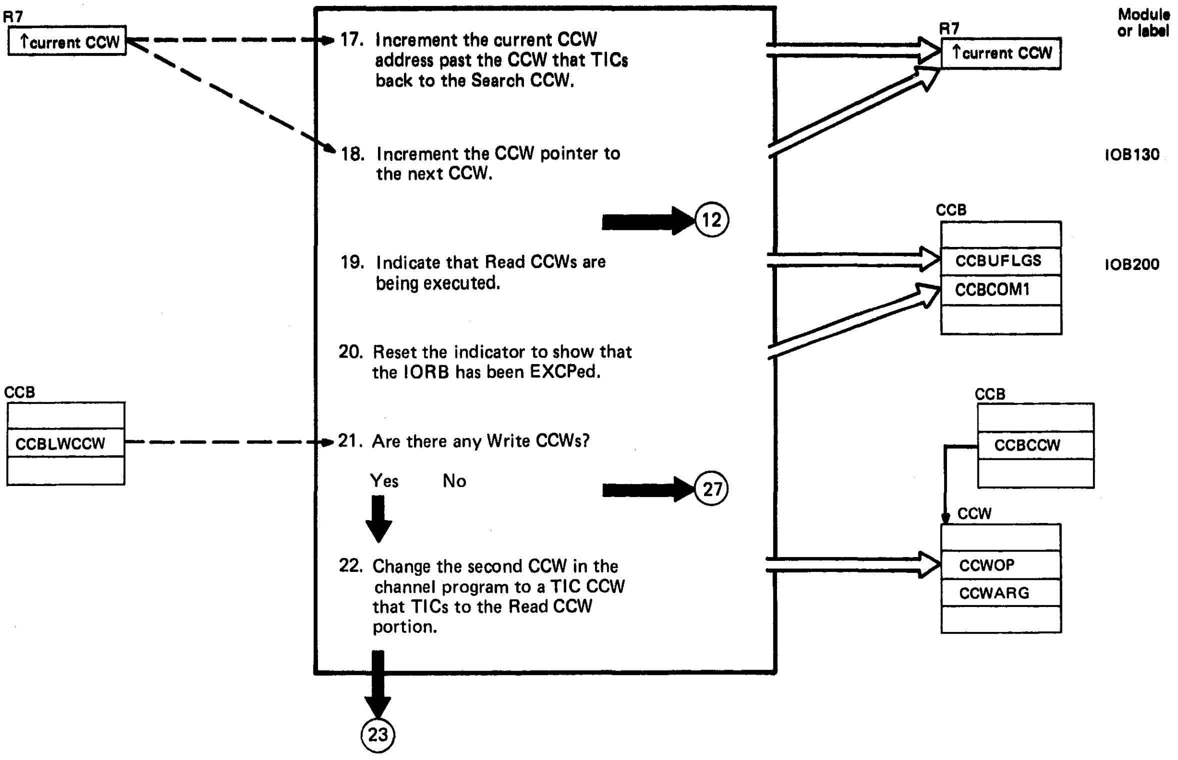
13. Is the current CCW the CCW in error?

14. Is the current CCW a TIC CCW?

15. Get the current CCW address from the TIC CCW.

16. Is the current CCW a Search-ID Equal CCW? (Does the current CCW have a status modifier function?)

Diagram EW5. I/O Error Handler



17. Increment the current CCW address past the CCW that TICs back to the Search CCW.

18. Increment the CCW pointer to the next CCW.

19. Indicate that Read CCWs are being executed.

20. Reset the indicator to show that the IORB has been EXCPed.

21. Are there any Write CCWs?

22. Change the second CCW in the channel program to a TIC CCW that TICs to the Read CCW portion.

Notes for Diagram EW5

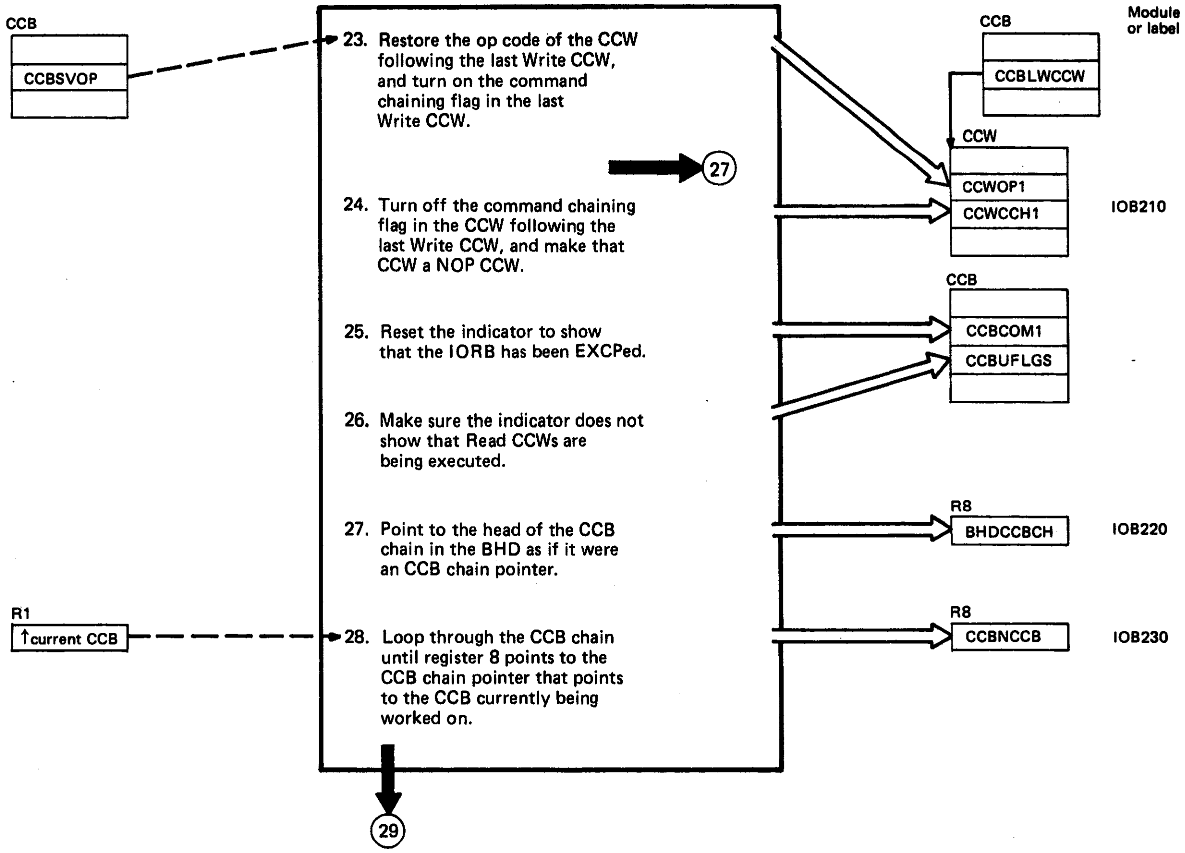
19. The bit CCBURDCW is set to indicate that the read portion of the channel program is being redriven.

20. The bit CCBERROK is set on by the EXCP subroutine within the I/O Manager just before the EXCP instruction is executed. This flag is used internally by VSAM to indicate that the channel program has been built ("formatted") and the EXCP has been issued. It must be reset at this point to indicate the channel programs must have the EXCP reissued.

21. This test must be made because the buffer requiring a "piggy-back" I/O operation may have only been started for the Write CCW and not read because the read has a different cylinder address (CKD) or CA (fixed block) than the write. As a result, when the I/O Manager redrives the BLDCP routine to handle the read operations, the bit CCBUERR will still be turned on even though there are no Write CCWs in the channel program. In this case, there is no need to modify or restore the channel program.

22. In the case where there are Write CCWs, then the Seek (CKD) or Define Extent (fixed block) CCW, at the beginning of the channel program, must be followed by a TIC CCW to the Read CCW part of the channel program.

Diagram EW6. I/O Error Handler



23. Restore the op code of the CCW following the last Write CCW, and turn on the command chaining flag in the last Write CCW.

24. Turn off the command chaining flag in the CCW following the last Write CCW, and make that CCW a NOP CCW.

25. Reset the indicator to show that the IORB has been EXCPed.

26. Make sure the indicator does not show that Read CCWs are being executed.

27. Point to the head of the CCB chain in the BHD as if it were an CCB chain pointer.

28. Loop through the CCB chain until register 8 points to the CCB chain pointer that points to the CCB currently being worked on.

Notes for Diagram EW6

23. This step restores the channel program so that the CCW following the last Write CCW will be correct for execution during read operations (it will no longer be a NOP CCW).

24. This step breaks the channel program at the end of the write operations for the retry.

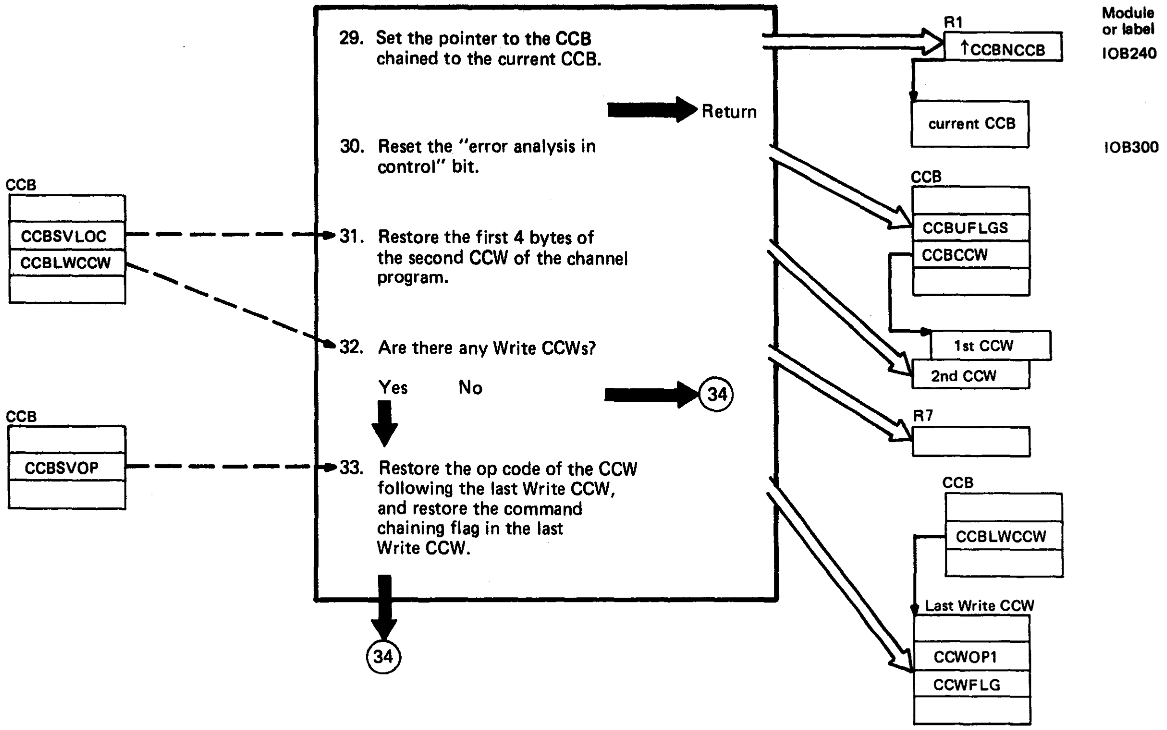
25. See note for step 20.

26. The bit CCBURDCW is reset so that the Read CCW portion of the channel program will be driven if the retry of the Write CCWs was successful.

27. Register 8 is initialized to contain the address of BHDCCBCH field minus the displacement of the CCBNCCB field from the beginning of the CCB control block. This is done so that the CCB control block pointer in the BHD will appear to be part of the CCB chain for search purposes.

28.-29. Find the preceding CCB control block in the chain. This is done because the I/O Manager Wait routine will load the pointer to the next CCB upon return from the I/O Error Handler. Therefore, register 1 must be initialized to ensure the current CCB block is the one that is obtained.

Diagram EW7. I/O Error Handler



29. Set the pointer to the CCB chained to the current CCB.

Return

30. Reset the "error analysis in control" bit.

31. Restore the first 4 bytes of the second CCW of the channel program.

32. Are there any Write CCWs?

Yes No

33. Restore the op code of the CCW following the last Write CCW, and restore the command chaining flag in the last Write CCW.

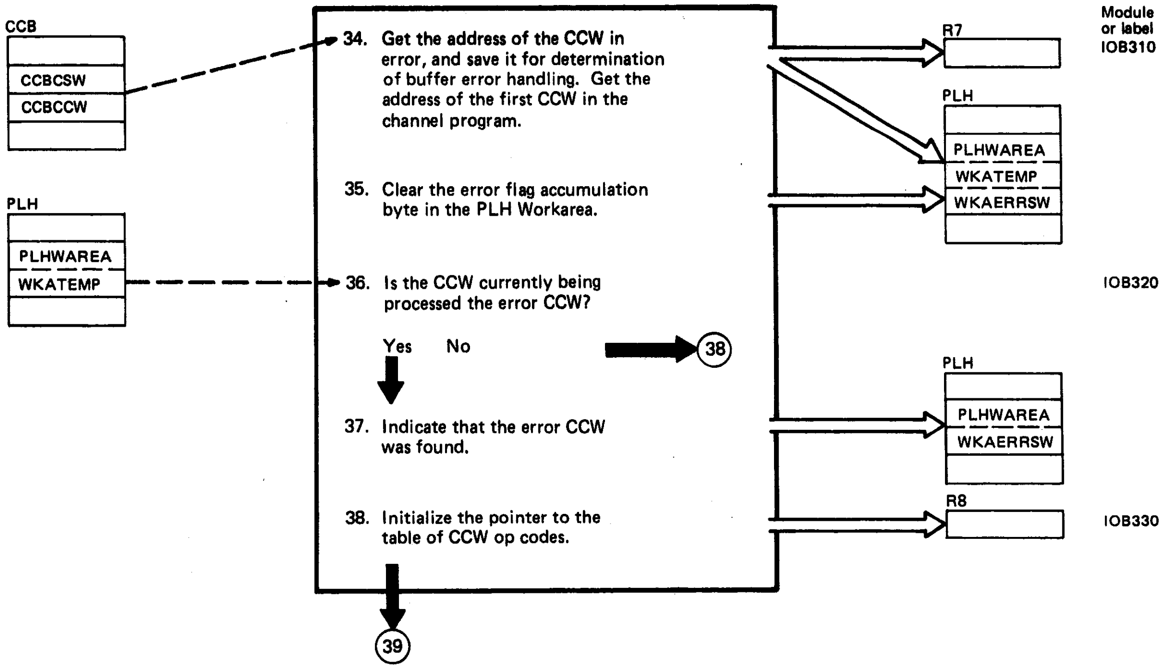
Notes for Diagram EW7

30.-31. See notes for steps 6 and 7.

32. See note for step 21.

33. See note for step 22.

Diagram EW8. I/O Error Handler



34. Get the address of the CCW in error, and save it for determination of buffer error handling. Get the address of the first CCW in the channel program.

35. Clear the error flag accumulation byte in the PLH Workarea.

36. Is the CCW currently being processed the error CCW?

Yes No

37. Indicate that the error CCW was found.

38. Initialize the pointer to the table of CCW op codes.

Notes for Diagram EW8

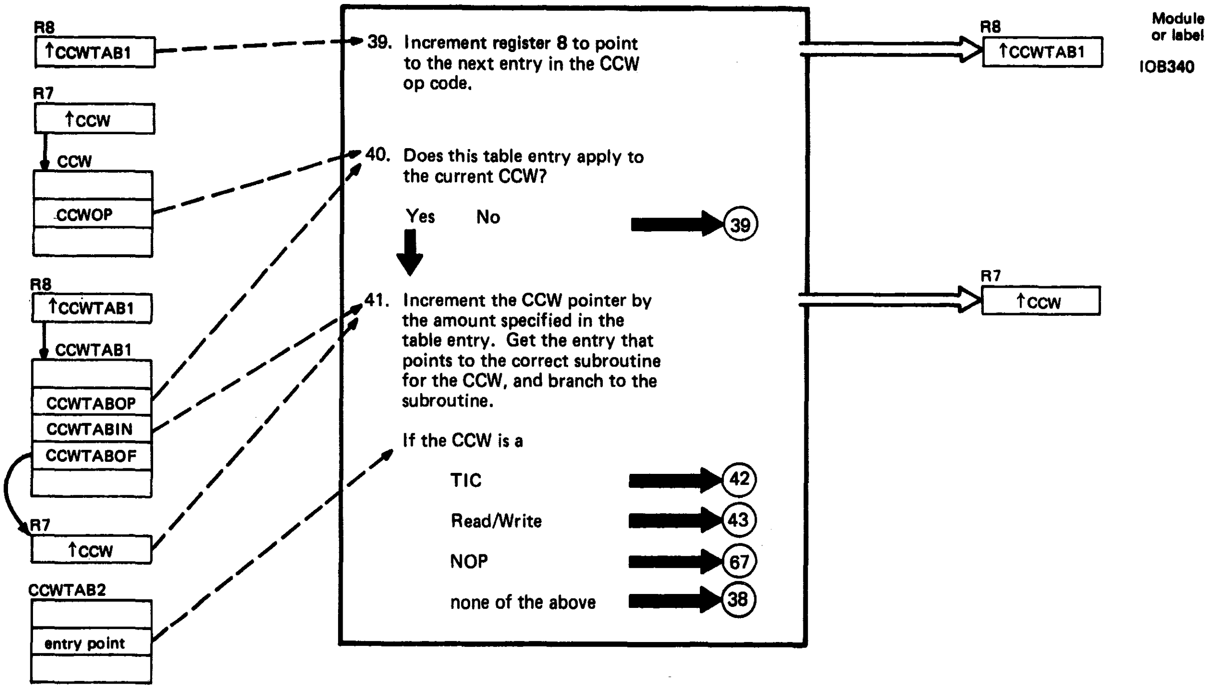
35. WKAERRSW uses the same bit definitions as BUFERFLG in the BCB.

36.-37. The "error CCW found" bit (IOMERP2) is used to control the resetting of the "write" bits in those buffers that have been successfully written before the error occurs.

NOTE: This is done because the Buffer Manager will attempt to redrive the remaining buffers whose I/O did not complete due to the error. For successful write operations, however, the "write" bits are turned off so that the buffers will not be rewritten.

38. The pointer to the table is backed up by the length of an entry to establish correct positioning because the next step will increment the pointer to the next entry.

Diagram EW9. I/O Error Handler



39. Increment register 8 to point to the next entry in the CCW op code.

40. Does this table entry apply to the current CCW?

41. Increment the CCW pointer by the amount specified in the table entry. Get the entry that points to the correct subroutine for the CCW, and branch to the subroutine.

If the CCW is a

TIC

Read/Write

NOP

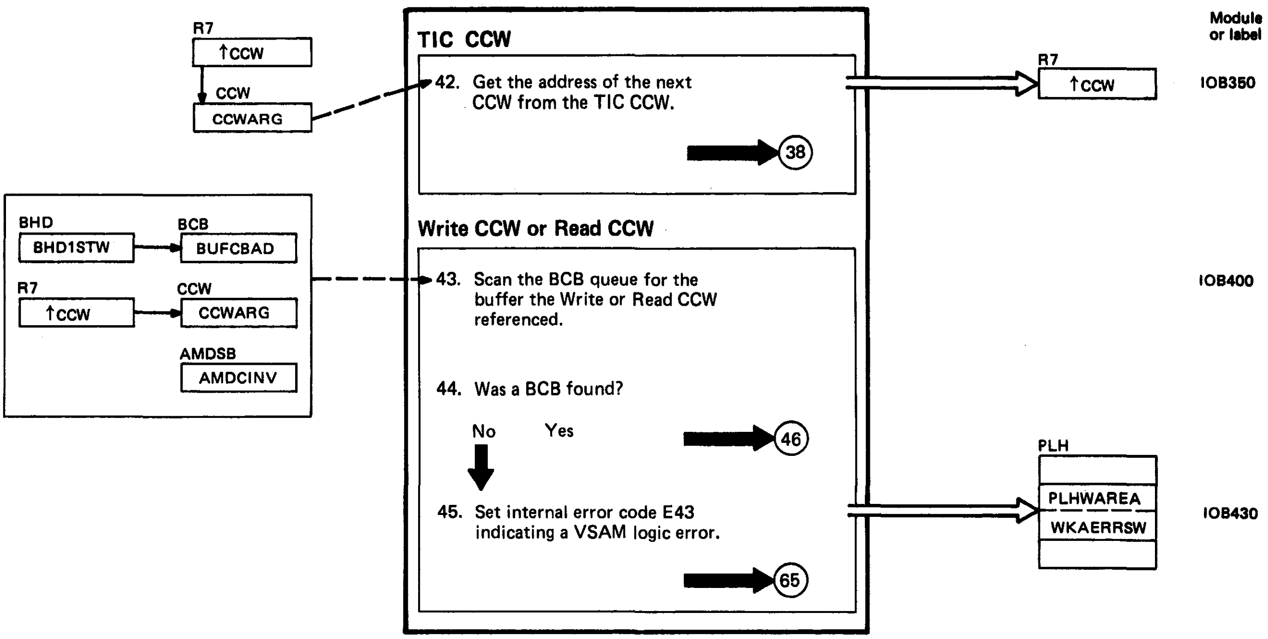
none of the above

Notes for Diagram EW9

39.-40. The table CCWTAB1 is searched until an entry is found to match the current CCW. The entry contains a displacement into the vector branch table (CCWTAB2) to determine the correct subroutine to be used for handling the current CCW.

41. This step sets the pointer to the current CCW plus the value specified in the CCWTAB1 table. For TIC, Read, and Write Data CCWs, there is no change. For the other CCWs, the pointer is positioned to the next CCW (except for SIDE and TIC, then positioning is to the CCW following the TIC). Step 66 will then increment the pointer to the next CCW.

Diagram EW10. I/O Error Handler


Module or label
R7
↑CCW
CCW
CCWARG
TIC CCW
42. Get the address of the next CCW from the TIC CCW.
38
R7
↑CCW
IOB350
BHD
BHD1STW
BCB
BUFCBAD
R7
↑CCW
CCW
CCWARG
AMDSB
AMDCINV
Write CCW or Read CCW
43. Scan the BCB queue for the buffer the Write or Read CCW referenced.
IOB400
44. Was a BCB found?
No
Yes
46
45. Set internal error code E43 indicating a VSAM logic error.
65
PLH
PLHWAREA
WKAERRSW
IOB430

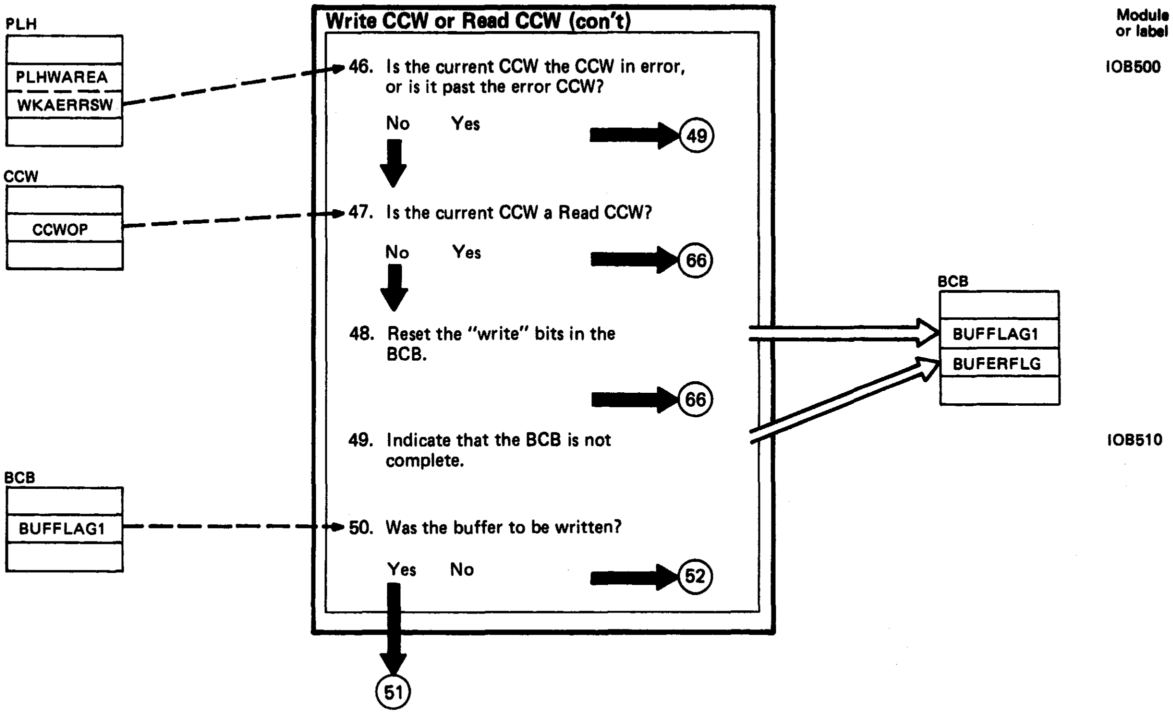
Notes for Diagram EW10

42. When a TIC CCW is encountered, the address specified in the TIC CCW is used as the pointer to the current CCW.

43. This check is made:
BUFCBAD≤CCWARG<BUFCBAD+AMDCINV
If this condition is satisfied, the buffer is referenced by the current CCW.

44.-45. A "no" condition should not occur here; if it does, error code E43 (VSAM logic error) is issued.

Diagram EW11. I/O Error Handler



PLH
PLHWAREA
WKAERRSW
CCW
CCWOP
BCB
BUFFLAG1
Write CCW or Read CCW (con't)
46. Is the current CCW the CCW in error, or is it past the error CCW?
No
Yes
49
47. Is the current CCW a Read CCW?
No
Yes
66
48. Reset the "write" bits in the BCB.
66
49. Indicate that the BCB is not complete.
50. Was the buffer to be written?
Yes
No
52
51
BCB
BUFFLAG1
BUFERFLG
Module or label
IOB500
IOB510

Notes for Diagram EW11

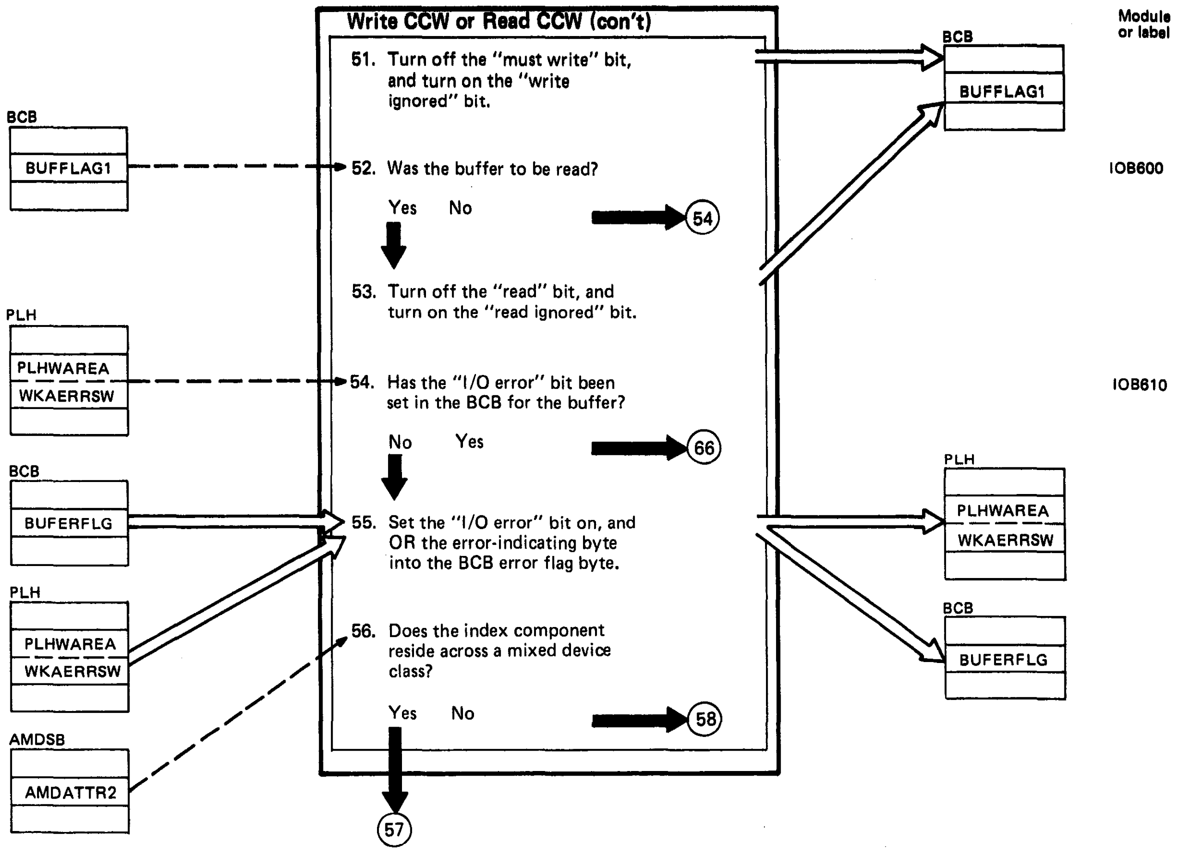
46.-48. All buffers associated with CCWs preceding the error CCW can be placed on the free queue, and all buffers associated with CCWs following the error CCW will be redriven.

If the error CCW was a Read CCW, then any buffers that were to be written were processed successfully, and the "write" bits (BUFCMW, BUFCFMT, and BUFPFMT) are reset so that they will not be rewritten during the redrive of the I/O operation.

49. The "BCB not complete" bit (BUFENTCM) is set to indicate that the buffer has not had any I/O done for it. It will be redriven by the Buffer Manager.

50.-51. If the buffer was to be written, the "must write" bit (BUFCMW) is turned off, and the "write ignored" bit (BUFWRIGN) is turned on. This causes the Buffer Manager to redrive the write operation for this buffer and ensures that any old writes for the current CCW chain are completed before returning to the user.

Diagram EW12. I/O Error Handler



Write CCW or Read CCW (con't)

51. Turn off the "must write" bit, and turn on the "write ignored" bit.

52. Was the buffer to be read?

Yes No

53. Turn off the "read" bit, and turn on the "read ignored" bit.

54. Has the "I/O error" bit been set in the BCB for the buffer?

No Yes

55. Set the "I/O error" bit on, and OR the error-indicating byte into the BCB error flag byte.

56. Does the index component reside across a mixed device class?

Yes No

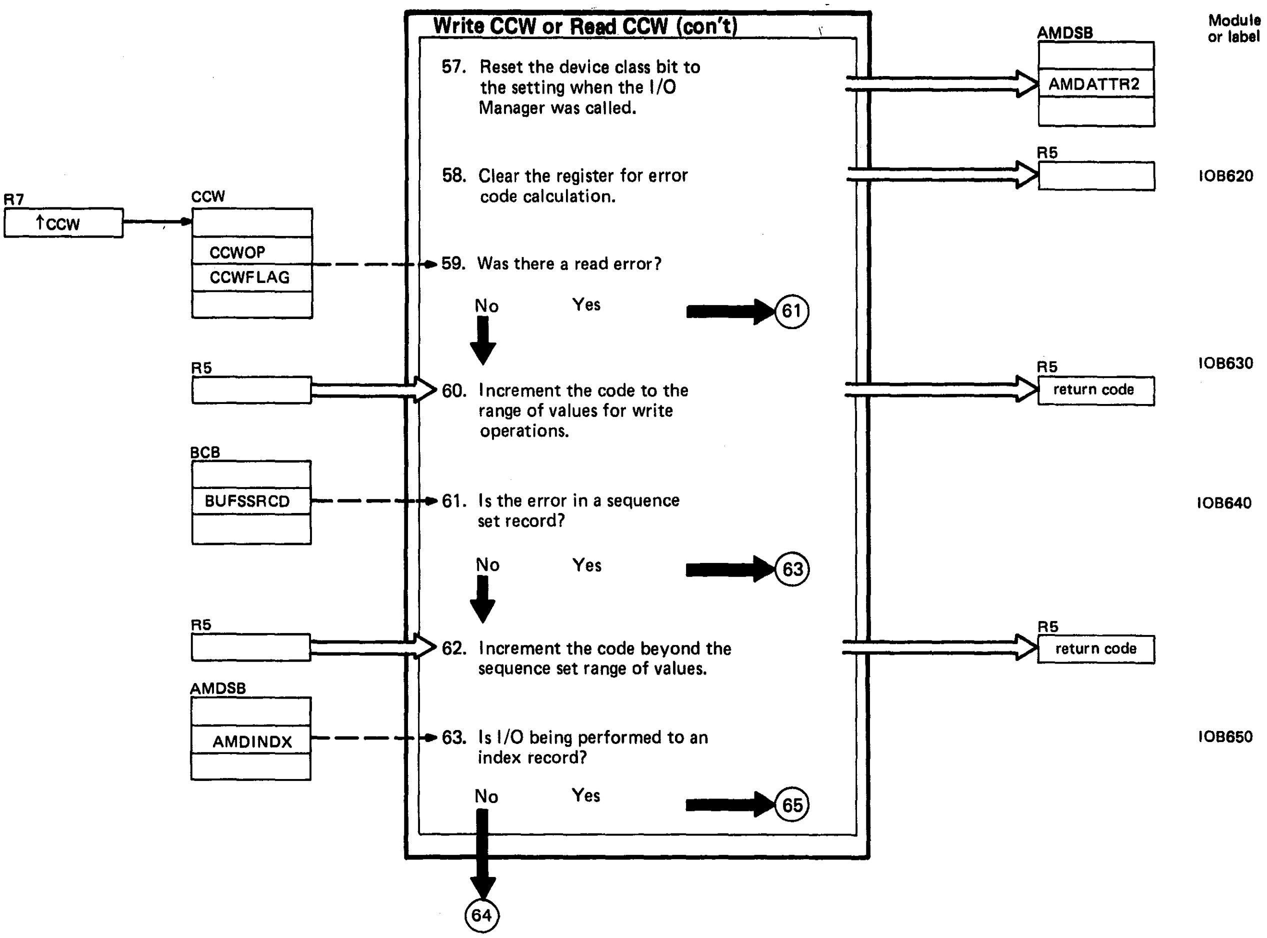
Notes for Diagram EW12

52.-53. If the buffer was to be used for a read, the "read" bit (BUFCRRD) is turned off, and the "read ignored" bit (BUFRDIGN) is turned on. This causes the Buffer Manager to redrive the read operation for the buffer and ensures that any reads for the current CCW chain are completed before returning to the user.

54.-55. If the buffer is the one associated with the error CCW, then the "I/O error" bit (BUFEIOER) is turned on, and the error indication(s) is ORed into the BCB.

56.-57. If the index component crosses different device classes (high-level index on CKD and sequence set on fixed block, or vice versa), then the device class bit (AMDARCH) is set to the original value when the I/O Manager was called.

Diagram EW13. I/O Error Handler



57. Reset the device class bit to the setting when the I/O Manager was called.

58. Clear the register for error code calculation.

59. Was there a read error?

60. Increment the code to the range of values for write operations.

61. Is the error in a sequence set record?

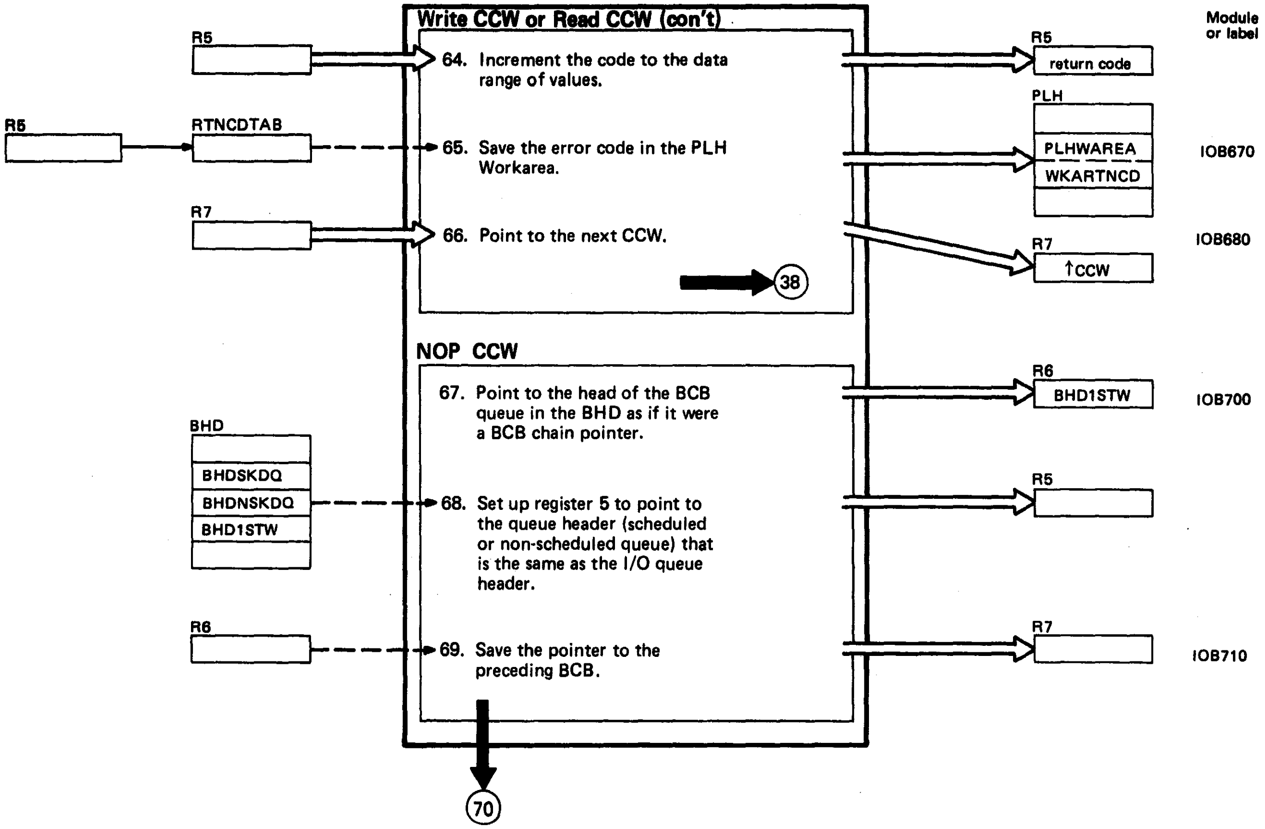
62. Increment the code beyond the sequence set range of values.

63. Is I/O being performed to an index record?

Notes for Diagram EW13

58.-64. These steps compute an index into a table of return code values. The first half of the table contains read errors; the second half contains write errors. Within each half the error codes appear in the order: sequence set, index (not sequence set), data.

Diagram EW14. I/O Error Handler



Write CCW or Read CCW (con't)

64. Increment the code to the data range of values.

65. Save the error code in the PLH Workarea.

66. Point to the next CCW.

NOP CCW

67. Point to the head of the BCB queue in the BHD as if it were a BCB chain pointer.

68. Set up register 5 to point to the queue header (scheduled or non-scheduled queue) that is the same as the I/O queue header.

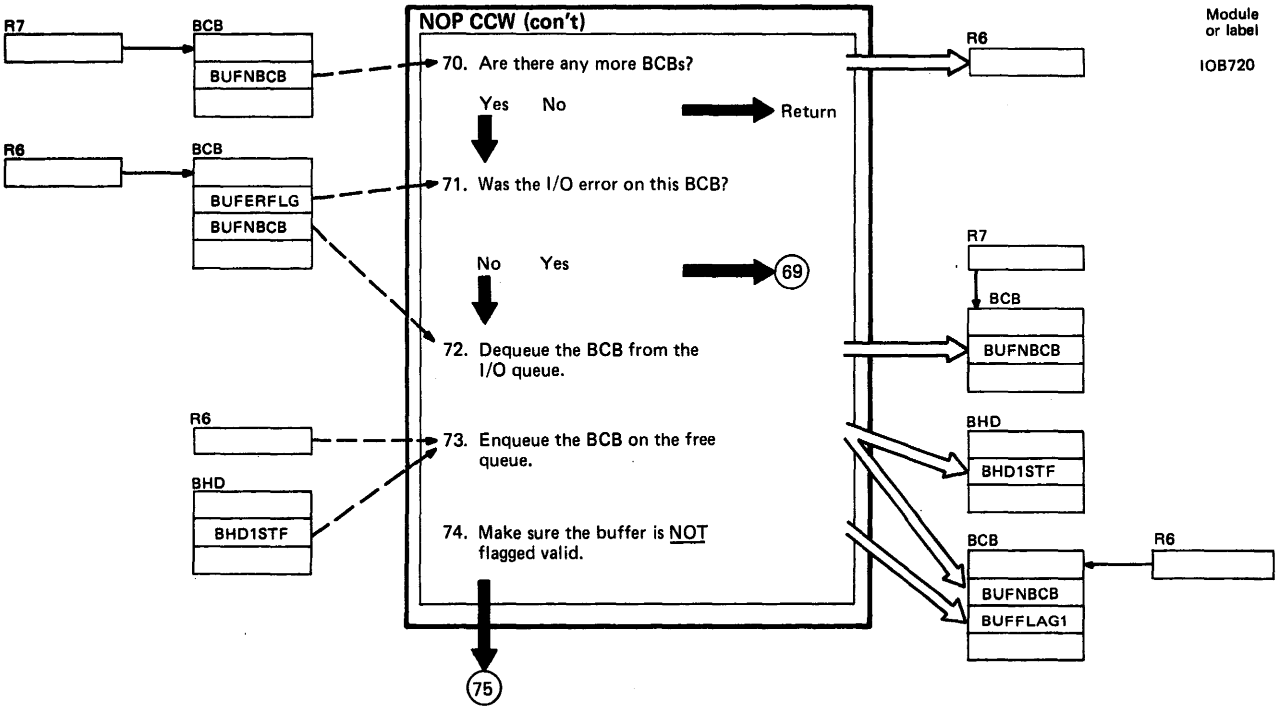
69. Save the pointer to the preceding BCB.

Notes for Diagram EW14

67.-77. The NOP CCW indicates the "end of CCW chain" (all CCWs have been processed). This routine updates the status of the BCBs on the I/O queue when an error has occurred. If a read or write error occurred on a BCB, then that BCB is moved to the free queue. If the BCB is flagged "not complete", but there was no read or write error, then that BCB is left on the I/O queue but is not marked valid.

67. Register 8 is initialized to contain the address of the BHD1STW minus the displacement of the BUFCHAIN field from the beginning of the BCB. This is done so that the BCB pointer in the BHD will appear to be part of the BCB chain for search purposes.

Diagram EW15. I/O Error Handler



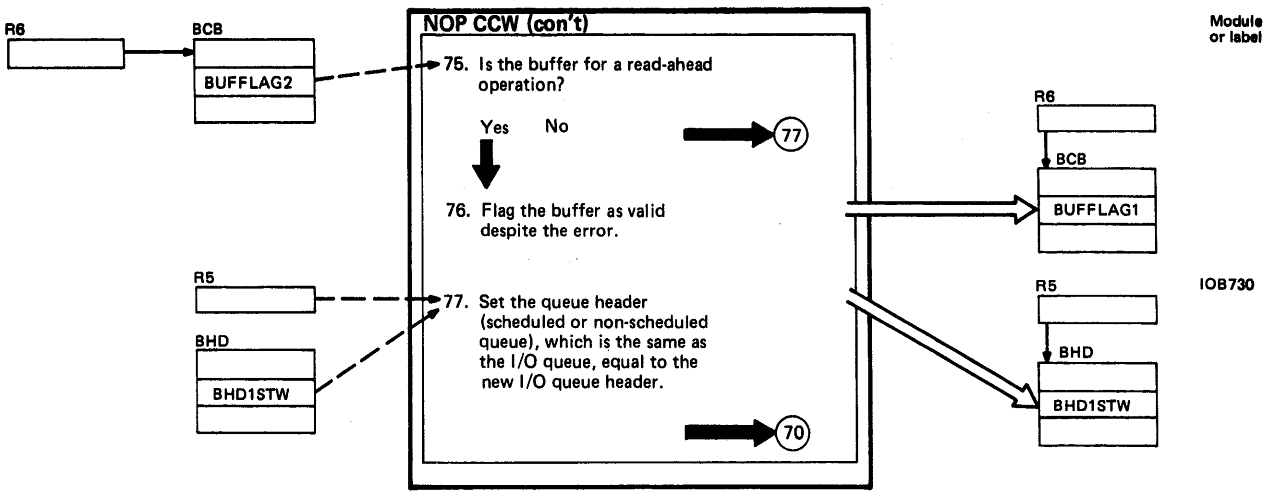
NOP CCW (con't)
Module or label
IOB720
R7
BCB
BUFNBCB
R6
BCB
BUFERFLG
BUFNBCB
70. Are there any more BCBs?
Yes
No
Return
71. Was the I/O error on this BCB?
No
Yes
69
72. Dequeue the BCB from the I/O queue.
R6
BHD
BHD1STF
73. Enqueue the BCB on the free queue.
74. Make sure the buffer is NOT flagged valid.
75
R6
R7
BCB
BUFNBCB
BHD
BHD1STF
BCB
BUFNBCB
BUFFLAG1
R6

Notes for Diagram EW15

70. Register 6 points to the next BCB (if one exists). Otherwise, control returns to the I/O Manager. (See Diagram EV, I/O Manager: Wait for I/O Completion.)

73. The "not complete" flag (BUFENTCM) and "write or read" error flag (BUFEIOER) were set during the scan of the CCW chain in steps 49-54.

Diagram EW16. I/O Error Handler



NOP CCW (con't)

75. Is the buffer for a read-ahead operation?

76. Flag the buffer as valid despite the error.

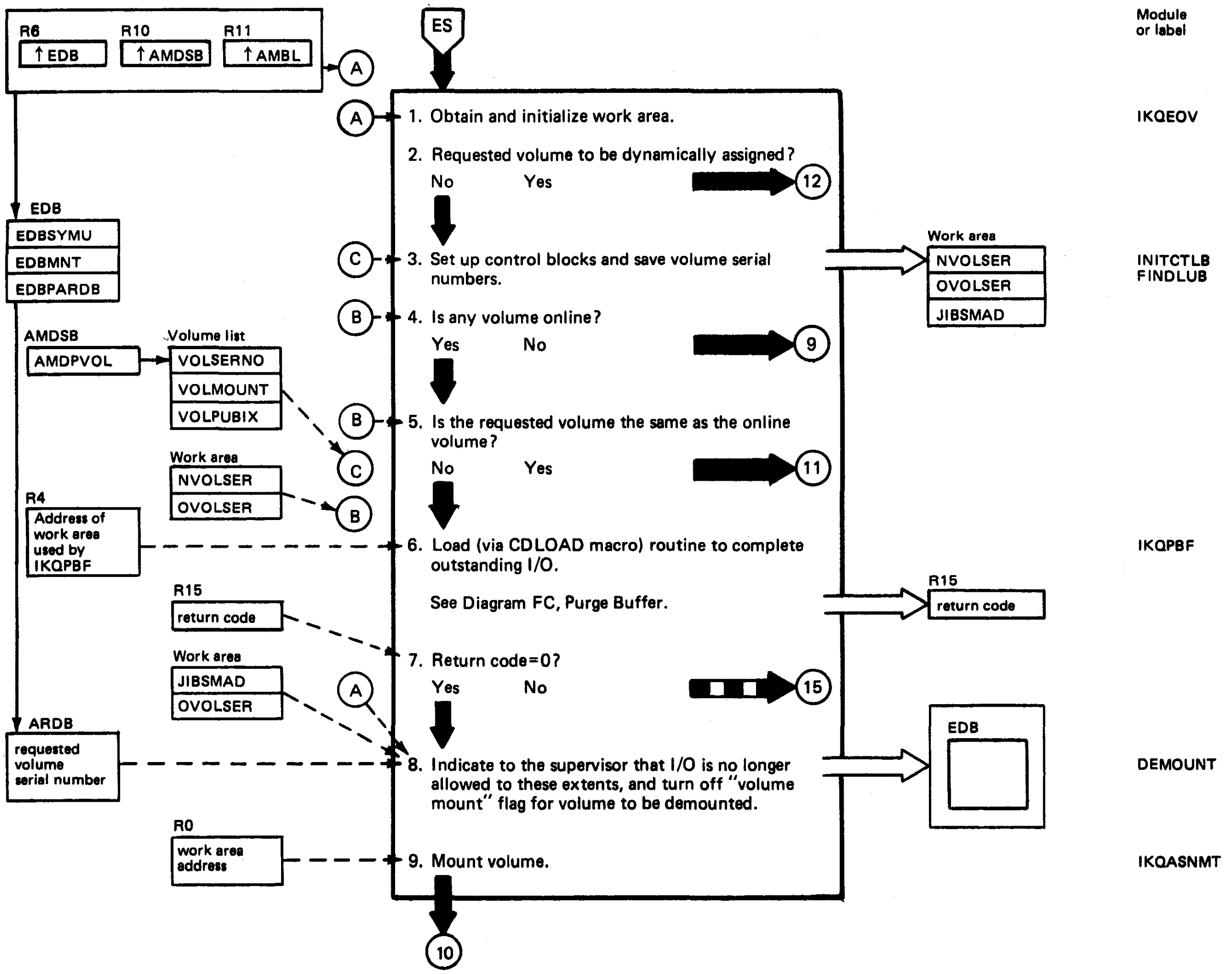
77. Set the queue header (scheduled or non-scheduled queue), which is the same as the I/O queue, equal to the new I/O queue header.

Notes for Diagram EW16

76. Errors on read-ahead buffers are checked for by the Get Next routine (IKQGNX00) or by the Read-Ahead Interface routine (RDAHI) of the Buffer Manager (IKQBFA00).

77. The header of the scheduled or non-scheduled queue (whichever was passed on entry to the I/O Manager as the I/O queue) is updated from the I/O queue header. (The I/O queue header may have been changed by moving buffers to the free queue.)

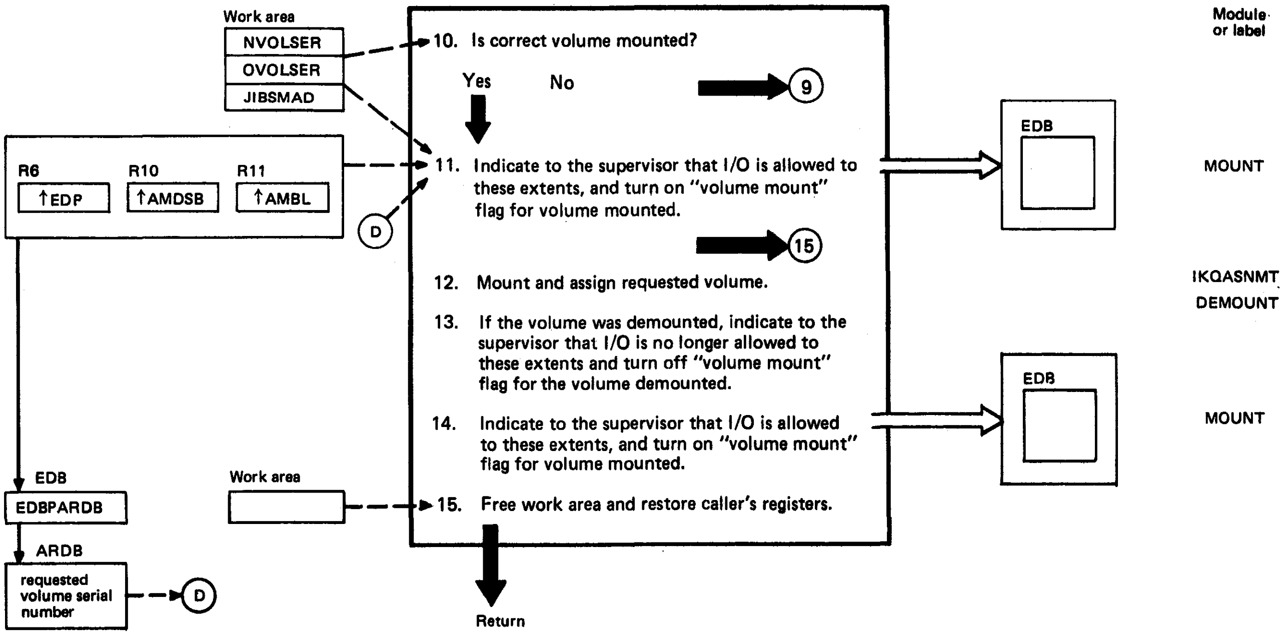
Diagram FA1. Mount Volume



1. Obtain and initialize work area.
2. Requested volume to be dynamically assigned?
   No Yes
3. Set up control blocks and save volume serial numbers.
4. Is any volume online?
   Yes No
5. Is the requested volume the same as the online volume?
   No Yes
6. Load (via CDLOAD macro) routine to complete outstanding I/O.

   See Diagram FC, Purge Buffer.
7. Return code=0?
   Yes No
8. Indicate to the supervisor that I/O is no longer allowed to these extents, and turn off "volume mount" flag for volume to be demounted.
9. Mount volume.

Diagram FA2. Mount Volume



10. Is correct volume mounted?

11. Indicate to the supervisor that I/O is allowed to these extents, and turn on "volume mount" flag for volume mounted.

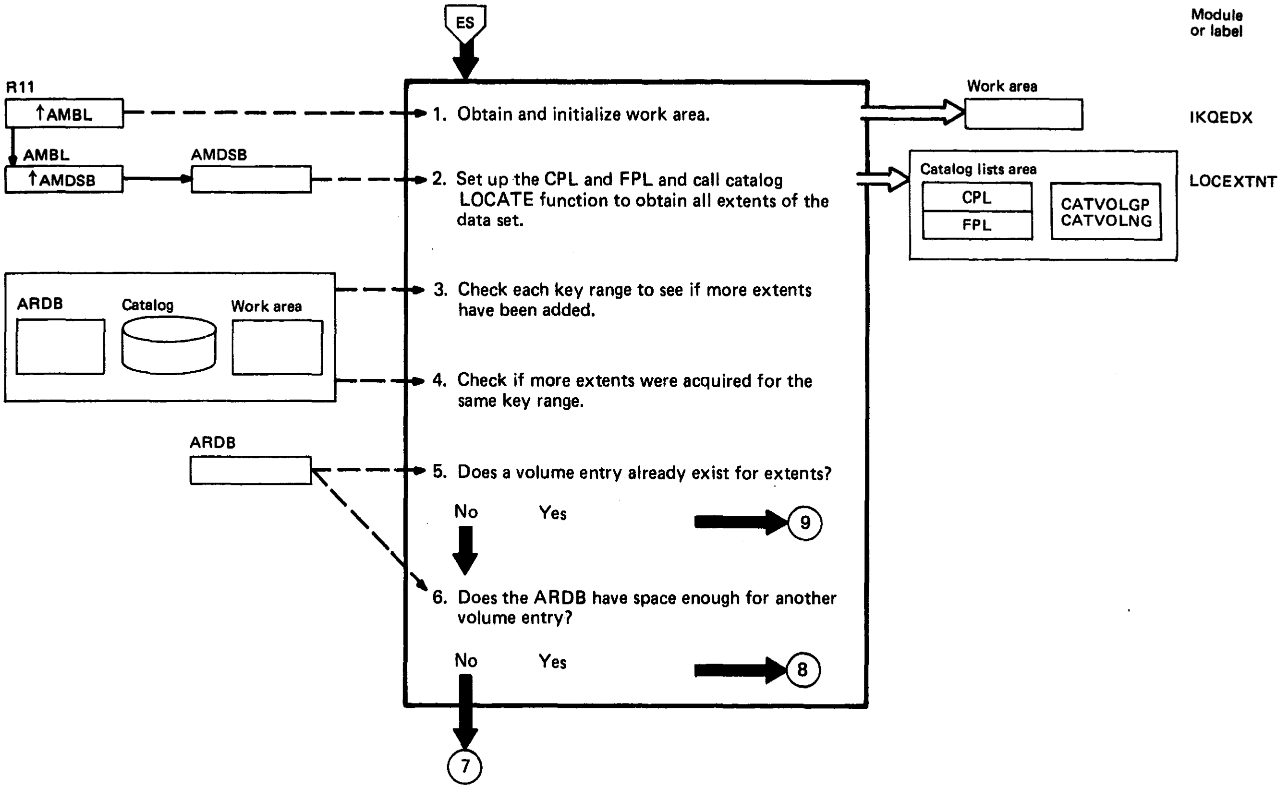
12. Mount and assign requested volume.

13. If the volume was demounted, indicate to the supervisor that I/O is no longer allowed to these extents and turn off "volume mount" flag for the volume demounted.

14. Indicate to the supervisor that I/O is allowed to these extents, and turn on "volume mount" flag for volume mounted.

15. Free work area and restore caller's registers.

Diagram FB1. Extend EDB



1. Obtain and initialize work area.
2. Set up the CPL and FPL and call catalog LOCATE function to obtain all extents of the data set.
3. Check each key range to see if more extents have been added.
4. Check if more extents were acquired for the same key range.
5. Does a volume entry already exist for extents?
6. Does the ARDB have space enough for another volume entry?

Notes for Diagram FB1

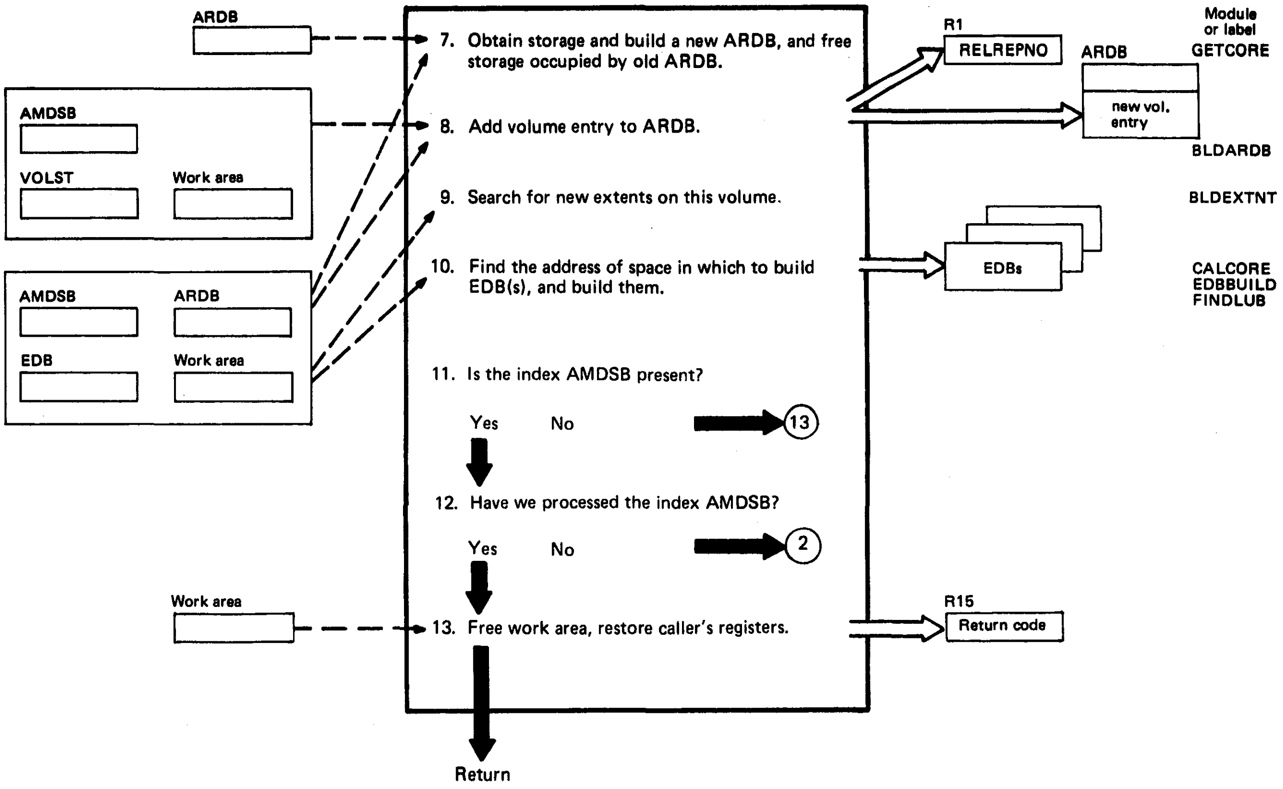
2. LOCEXTNT is called and, in turn, calls the catalog LOCATE function to obtain all extents of the data set. If a nonzero code is returned in register 15 by the catalog LOCATE routine, an error code is set in the work area by LOCEXTNT to specify what error condition occurred before exiting to the mainline routine.

   Steps 2-11 are processed twice if both data and index AMDSBs exist — first for the data, then for the index.

3.-4. Each of these steps constitutes a small loop within the major loop for each AMDSB.

   After each of the checks in steps 3 and 4, steps 5 through 10 are processed. That is, whenever extents are found that belong to the key range being processed, associated ARDBs, EDBs, and volume entries must be created for them, if they do not already exist.

Diagram FB2. Extend EDB

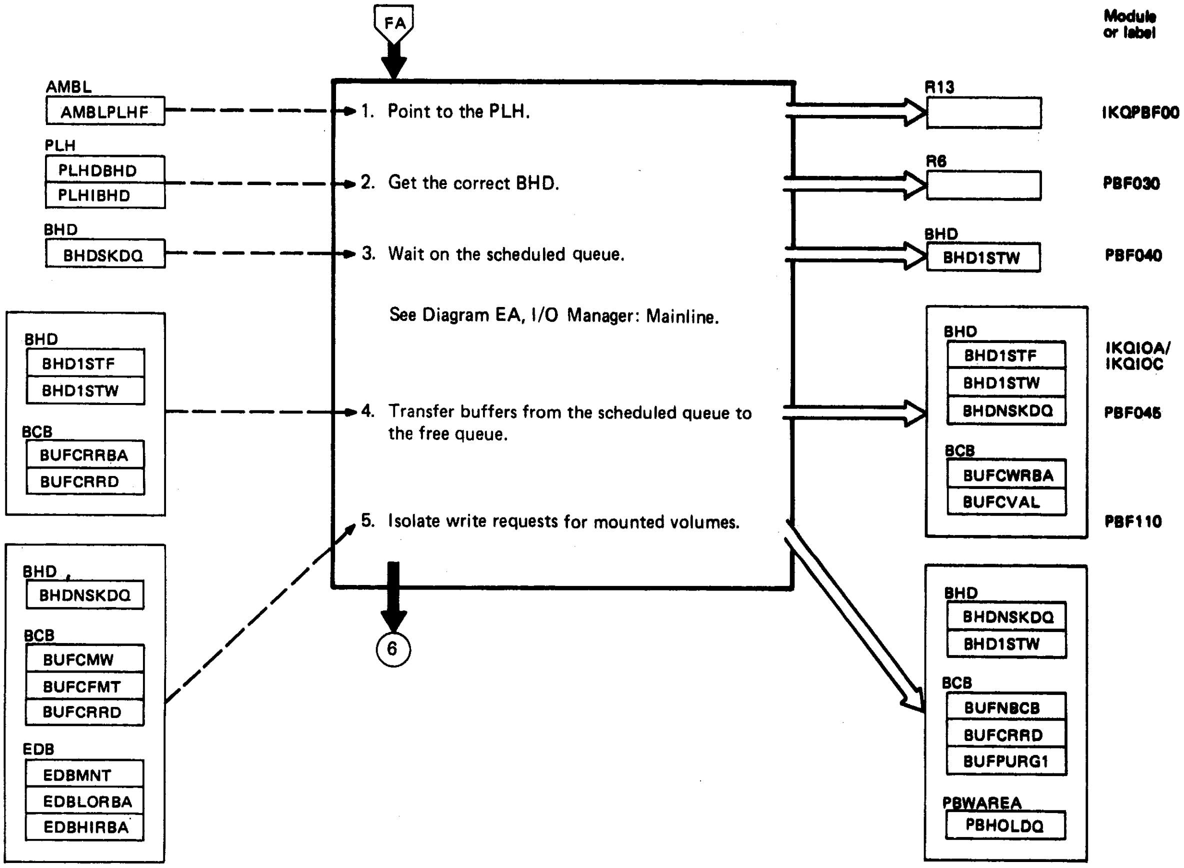

ARDB
AMDSB
VOLST
Work area
AMDSB
ARDB
EDB
Work area
Work area
7. Obtain storage and build a new ARDB, and free storage occupied by old ARDB.
8. Add volume entry to ARDB.
9. Search for new extents on this volume.
10. Find the address of space in which to build EDB(s), and build them.
11. Is the index AMDSB present?
Yes
No
13
12. Have we processed the index AMDSB?
Yes
No
2
13. Free work area, restore caller's registers.
Return
R1
RELREPNO
ARDB
new vol. entry
EDBs
R15
Return code
Module or label
GETCORE
BLDARDB
BLDEXTNT
CALCORE
EDBBUILD
FINDLUB

Notes for Diagram FB2

7. If there isn't enough space in the ARDB to add a volume entry, GETCORE is called to obtain storage, copy the old ARDB, and free the storage occupied by the old ARDB.

10. To locate and build EDB(s) for the new extents, BLDEXTNT is called and, in turn, calls CALCORE to get the address of space in which to build the EDBs, EDBBUILD to build the EDBs for the new extents, and FINDLUB to find the LUB index in order to turn on the mount flag.

Diagram FC1. Purge Buffer



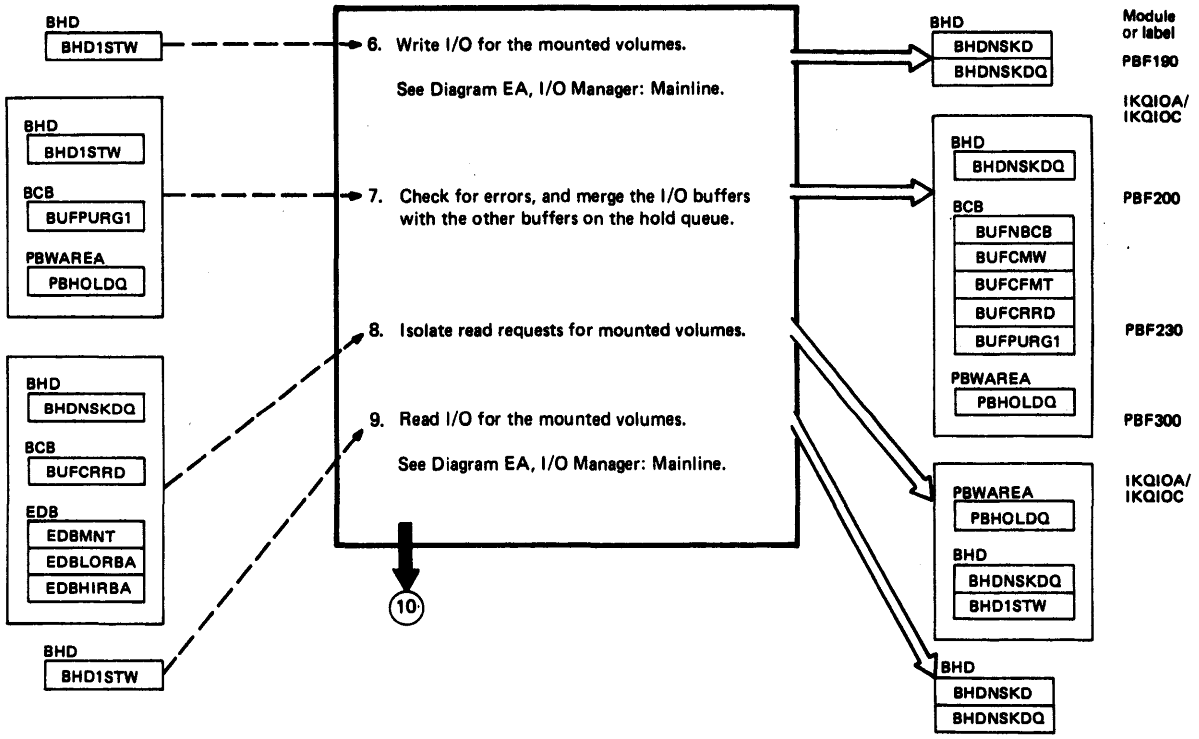
Notes for Diagram FC1

1. On entry, I/O Manager status from the PLH is saved in a work area (PBWAREA) provided to Purge Buffer by its caller (EOV). This save is necessary because the I/O Manager will be invoked recursively by Purge Buffer. (The I/O Manager calls EOV, which calls Purge Buffer, which in turn calls the I/O Manager.)

   The design of Purge Buffer assumes single string processing. This restriction is enforced by the open routines. For this reason, Purge Buffer only concerns itself with a single PLH.
2. The first time through this routine, the index BHD is gotten (if there is one). In all other cases the data BHD is gotten.
3. The scheduled queue contains I/O that has been started but not waited on (if any).
4. The free queue contains buffers that are available for use, regardless of whether their contents are valid.

   While transferring buffers from the scheduled queue to the free queue, any buffer for which a read has been done gets its current RBA updated to the RBA of the control interval read and is marked valid (BUFCVAL).
5. A queue containing only requests (that have not been started) for write I/O to mounted volumes is constructed. This queue is a subset of the requests that had been on the nonscheduled queue (queue of I/O requests not yet started). The remaining requests from the nonscheduled queue are placed on a temporary Purge Buffer queue (the hold queue, PBHOLDQ).

Diagram FC2. Purge Buffer



6. Write I/O for the mounted volumes.

   See Diagram EA, I/O Manager: Mainline.

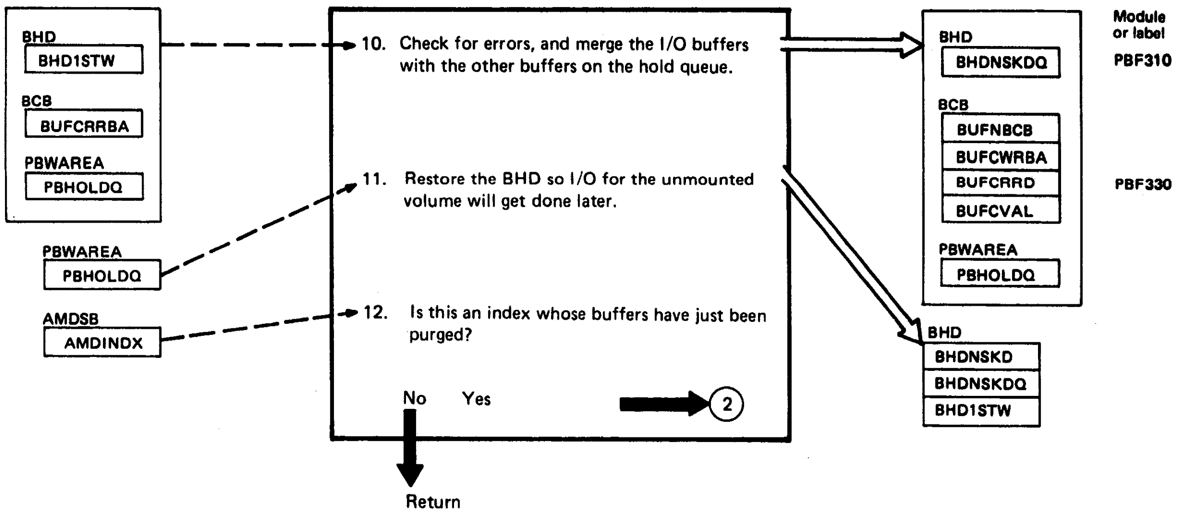
7. Check for errors, and merge the I/O buffers with the other buffers on the hold queue.

8. Isolate read requests for mounted volumes.

9. Read I/O for the mounted volumes.

   See Diagram EA, I/O Manager: Mainline.

Diagram FC3. Purge Buffer



10. Check for errors, and merge the I/O buffers with the other buffers on the hold queue.

11. Restore the BHD so I/O for the unmounted volume will get done later.

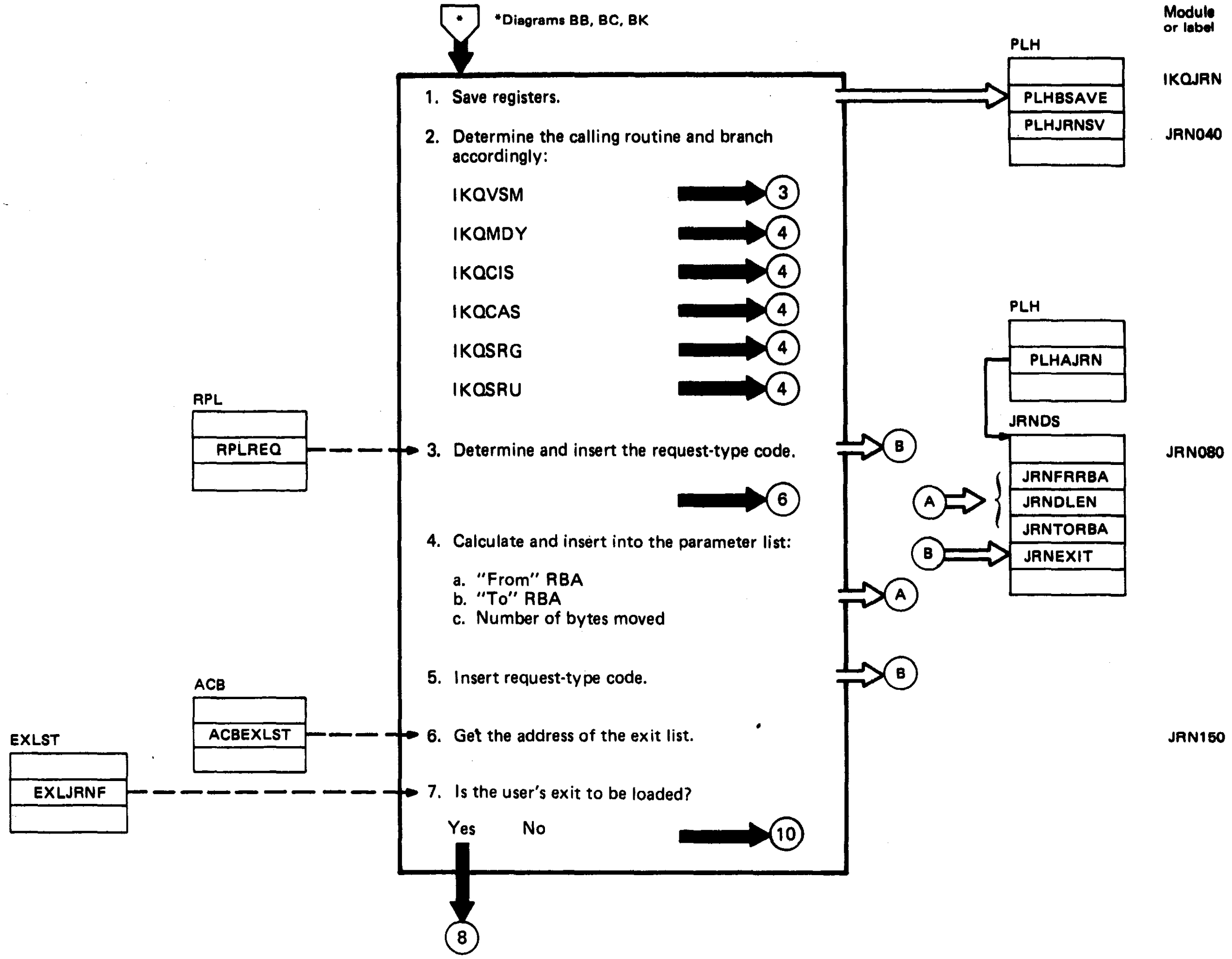
12. Is this an index whose buffers have just been purged?

Notes for Diagram FC3

11. The nonscheduled queue, as restored, will contain all buffers that were previously on it, but the flags indicating that I/O is requested will have been turned off for requests to volumes that were mounted before the EOV call. The I/O Manager ignores buffers with no I/O requests on them.

12. If an index has just been processed, it is necessary to make another pass to process the data component.

Diagram FD1. JRNAD Exit: Journal a Transaction



1. Save registers.

2. Determine the calling routine and branch accordingly:

IKQVSM → 3
IKQMDY → 4
IKQCIS → 4
IKQCAS → 4
IKQSRG → 4
IKQSRU → 4

3. Determine and insert the request-type code.

→ 6

4. Calculate and insert into the parameter list:

a. "From" RBA
b. "To" RBA
c. Number of bytes moved

5. Insert request-type code.

6. Get the address of the exit list.

7. Is the user's exit to be loaded?

Yes → 8 No → 10

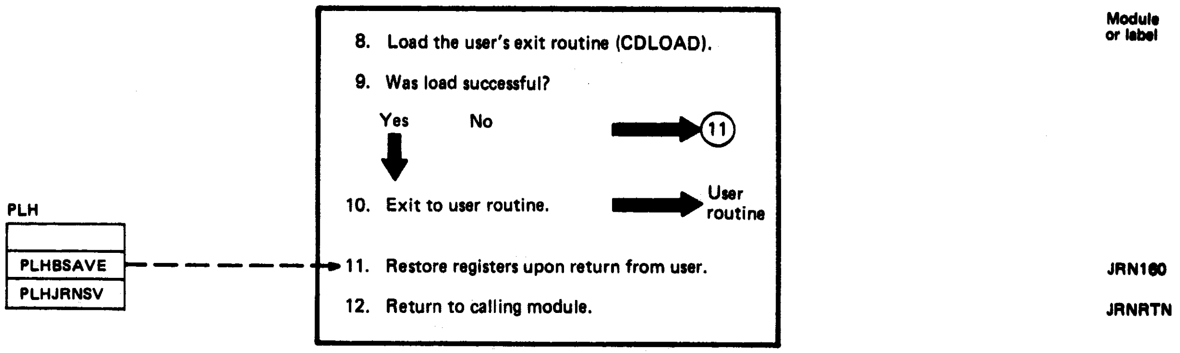
Notes for Diagram FD1

1. Registers 14, 0, and 1 are saved in field PLHJRNSV. Registers 2-12 are saved in field PLHBSAVE.
2. IKQAIX also calls JRNAD, but it uses the IKQVSM calling interface.

   Depending on the caller, the following parts of step 4 are executed:

IKQMDY	4a, 4b, and 4c
IKQCIS	4a and 4b
IKQCAS	4a (first pass) 4b (second pass)
IKQSRG	4a
IKQSRU	4c

Diagram FD2. JRNAD Exit: Journal a Transaction



8. Load the user's exit routine (CDLOAD).

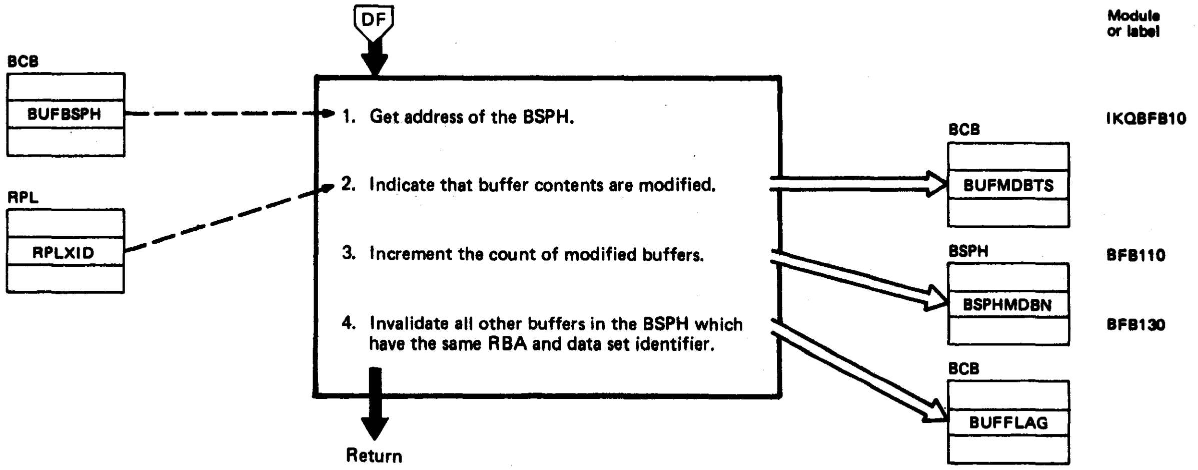
9. Was load successful?

10. Exit to user routine.

11. Restore registers upon return from user.

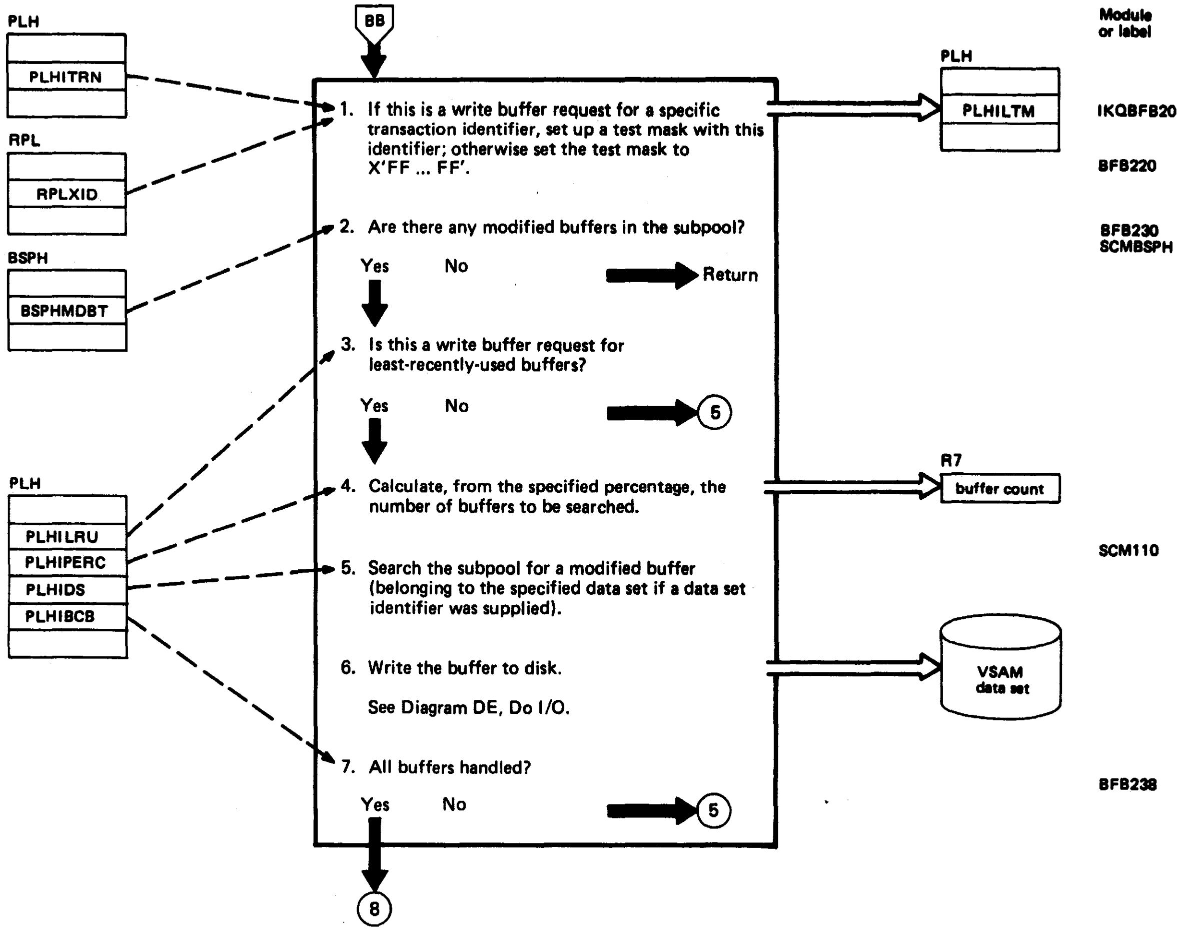
12. Return to calling module.

Diagram FE1. Defer Writing of Buffers



1. Get address of the BSPH.
2. Indicate that buffer contents are modified.
3. Increment the count of modified buffers.
4. Invalidate all other buffers in the BSPH which have the same RBA and data set identifier.

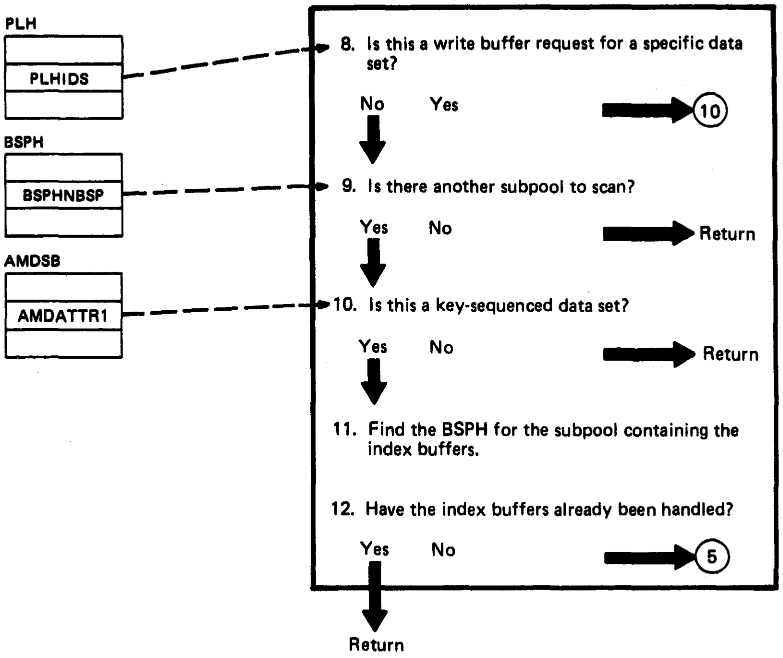
Diagram FF1. WRTBFR: Write Deferred Buffers



1. If this is a write buffer request for a specific transaction identifier, set up a test mask with this identifier; otherwise set the test mask to X'FF ... FF'.
2. Are there any modified buffers in the subpool?
3. Is this a write buffer request for least-recently-used buffers?
4. Calculate, from the specified percentage, the number of buffers to be searched.
5. Search the subpool for a modified buffer (belonging to the specified data set if a data set identifier was supplied).
6. Write the buffer to disk.

   See Diagram DE, Do I/O.
7. All buffers handled?

Diagram FF2. WRTBFR: Write Deferred Buffers



8. Is this a write buffer request for a specific data set?

9. Is there another subpool to scan?

10. Is this a key-sequenced data set?

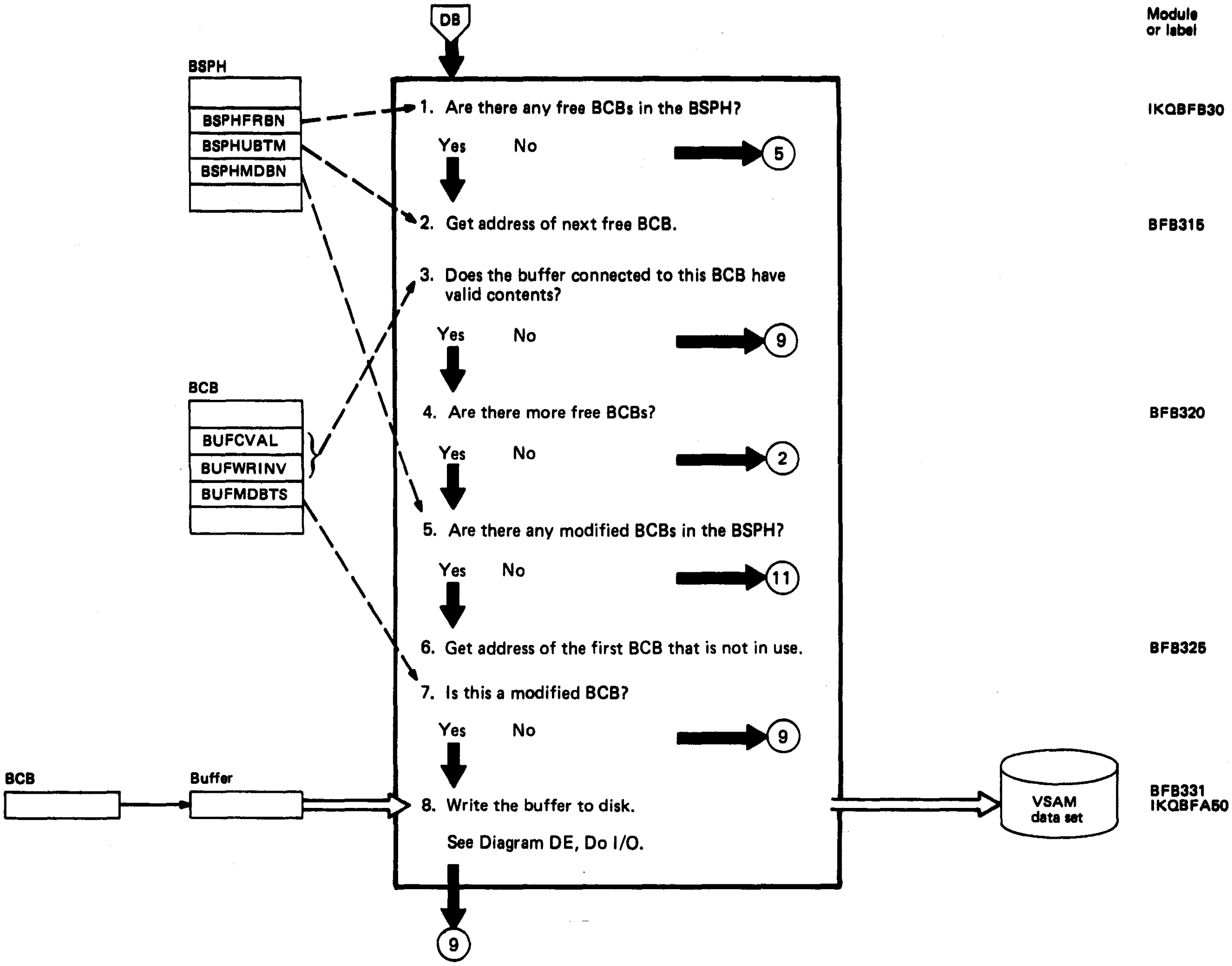
11. Find the BSPH for the subpool containing the index buffers.

12. Have the index buffers already been handled?

Module or label

BFB240

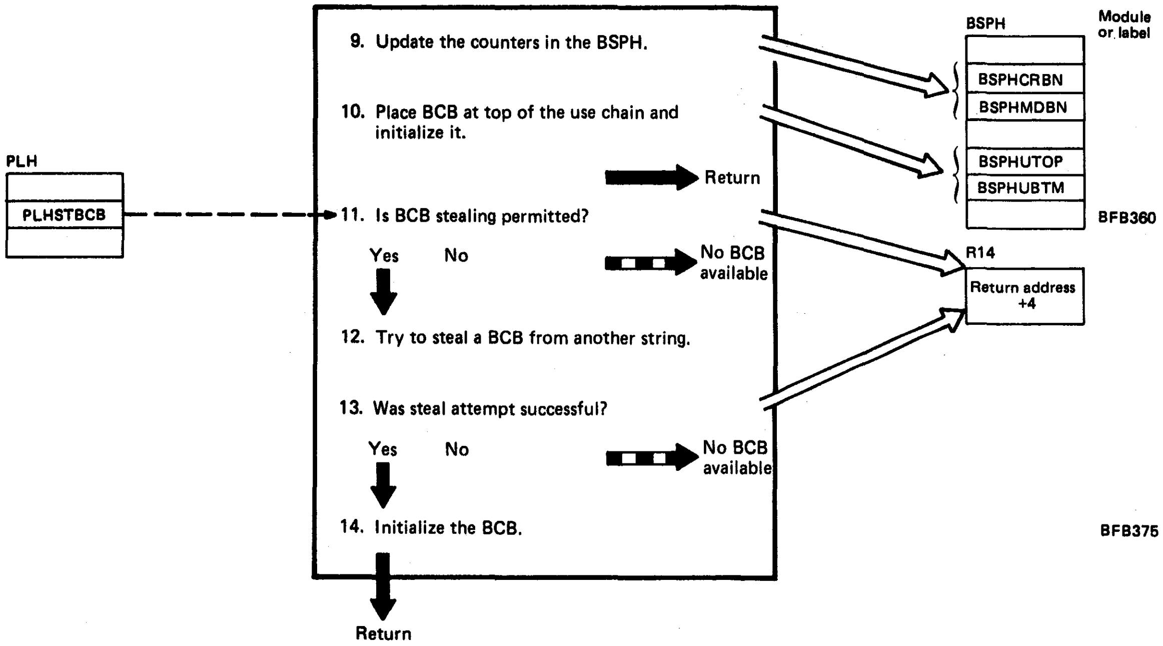
Diagram FG1. Get a Scratch Buffer from the Resource Pool



1. Are there any free BCBs in the BSPH?
2. Get address of next free BCB.
3. Does the buffer connected to this BCB have valid contents?
4. Are there more free BCBs?
5. Are there any modified BCBs in the BSPH?
6. Get address of the first BCB that is not in use.
7. Is this a modified BCB?
8. Write the buffer to disk.

   See Diagram DE, Do I/O.

Diagram FG2. Get a Scratch Buffer from the Resource Pool



9. Update the counters in the BSPH.

10. Place BCB at top of the use chain and initialize it.

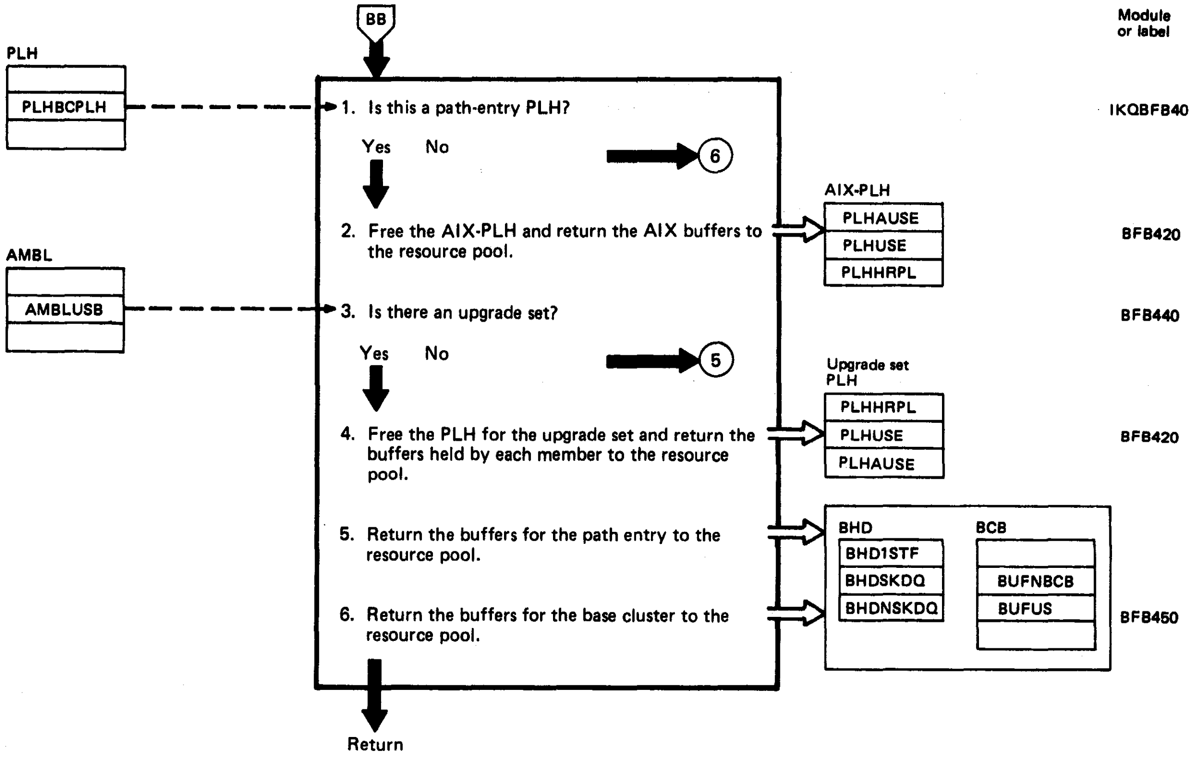
11. Is BCB stealing permitted?

12. Try to steal a BCB from another string.

13. Was steal attempt successful?

14. Initialize the BCB.

Diagram FH1. Return a Buffer to the Resource Pool



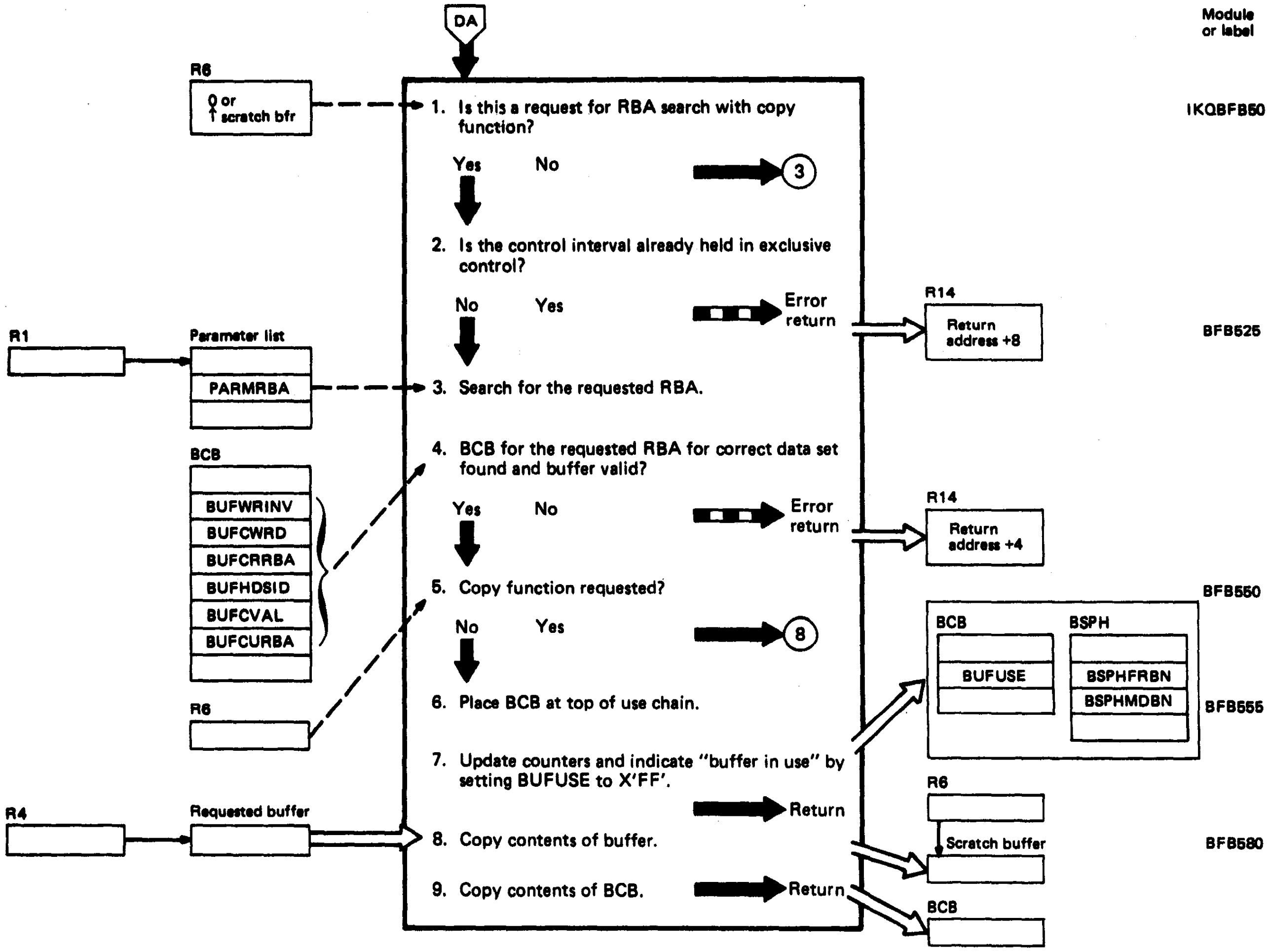
1. Is this a path-entry PLH?
   Yes No → 6
2. Free the AIX-PLH and return the AIX buffers to the resource pool.
3. Is there an upgrade set?
   Yes No → 5
4. Free the PLH for the upgrade set and return the buffers held by each member to the resource pool.
5. Return the buffers for the path entry to the resource pool.
6. Return the buffers for the base cluster to the resource pool.

Return

Notes for Diagram FH1

2, 4, 5, and 6. If the data set is share option 4 (AMDSHR on), the buffers are invalidated.

Diagram FI1. Search Resource Pool for Requested RBA



DA
Module or label
R6
0 or 1 scratch bfr
1. Is this a request for RBA search with copy function?
Yes
No
3
IKQBFB50
2. Is the control interval already held in exclusive control?
No
Yes
Error return
R14
Return address +8
BFB525
R1
Parameter list
PARMRBA
3. Search for the requested RBA.
BCB
4. BCB for the requested RBA for correct data set found and buffer valid?
BUFWRINV
BUFCWRD
BUFCRRBA
BUFHDSID
BUFCVAL
BUFCURBA
Yes
No
Error return
R14
Return address +4
5. Copy function requested?
No
Yes
8
BFB550
BCB
BSPH
BUFUSE
BSPHFRBN
BSPHMDBN
BFB555
R6
6. Place BCB at top of use chain.
7. Update counters and indicate "buffer in use" by setting BUFUSE to X'FF'.
Return
R6
R4
Requested buffer
8. Copy contents of buffer.
Scratch buffer
BFB580
9. Copy contents of BCB.
Return
BCB

Section 3. Program Organization

VSAM program listings are the key to VSAM's organization. You get into the listings from the method of operation diagrams. Once you have located the module or routine name that interests you in the diagrams, you are ready to turn to the listing to find the additional information you require.

Module Prologues

Each VSAM module listing begins with a description of the module, called the module prologue. The information contained in VSAM prologues is described in the topics that follow.

Module name: The external procedure name of the module (for example, IKQIOA).

Descriptive name: The English name of the module (for example, I/O Manager).

Status: The version and release level of the module.

Function: A brief step-by-step explanation of the functions performed by this module. Function is divided into steps so that you may more easily locate the routine responsible for each step.

Notes: A generalized heading that includes (1) any dependencies, for example, CPU model or features, that will affect the operation of this module, (2) any restrictions that apply to this module, (3) symbols used to represent registers and register usage, (4) symbolic name of the maintenance area for this module and whether the maintenance area is used or reserved, and (5) any special terms and acronyms that are used within this module that are not necessarily used elsewhere in the documentation.

Module Type: A description of the type of this module (for example, procedure or macro) the name of the compiler used/required to create this module, the amount of storage required by this module for executable code and associated data, and the attributes of the module (for example, reentrant or read-only).

Entry point: The name of the point at which control can enter this module, the conditions of entry, the calling sequence by which control was given, including any parameters passed and the names of modules that may enter at this entry point.

Input: A description of anything this module gets or references, such as registers, control blocks, or data. The means by which this module gains access to the input is included.

Output: A description of registers, control blocks, and data areas at output; any messages issued as a result of this module's processing are included.

Exit-normal: A description of conditions at and reasons for normal exit from this module, including the names of modules called by this module.

Exit-error: A description of conditions at and reasons for any error exit from this module.

External references: A list of modules, data areas, etc., defined outside of or accessible outside of this module.

Tables: A list of all local tables and work areas, that is, data areas built and used only within this module.

Macros: A description of system macros used by this module.

Change activity: A list of any change activity to this module.

Routine Prologues

The numbered steps in the module prologue FUNCTION heading are your link to the routine prologues. Routine prologues contain (1) an expanded description of the processing steps shown in the module prologues, (2) input to the routine, and (3) output from the routine.

Program Structures and Catalog Program Flowcharts

The following group of program structures shows how the VSAM program is organized. These structures link modules together from the time a macro instruction is issued by the user program to the time that control exits from VSAM. The structures are ordered by user-issued macro instructions and the verify function in a way similar to the organization of method of operation diagrams. In addition, program structures are also shown for significant subfunctions required to complete processing of a macro instruction. The subfunctions included in this volume are buffer and I/O management.

Figure 3.1 shows the symbols used on the structures and describes their meanings.

———————	Indicates that a module is called and returns to calling module
———————▶	Indicates that a module does not return to calling module
– – – – – –	Indicates that a module is called under certain conditions and then returns to calling module
– – – – – –▶	Indicates that a module is called under certain conditions and does not return to calling module
UPPER CASE	Indicates that a module is executed and calls one or more modules before returning
lower case	Indicates that a module is executed and then returns to the calling module

Figure 3.1 **Graphic symbols used in program structures**

Figure 3.2 Program structure to process POINT (part 1 of 2)

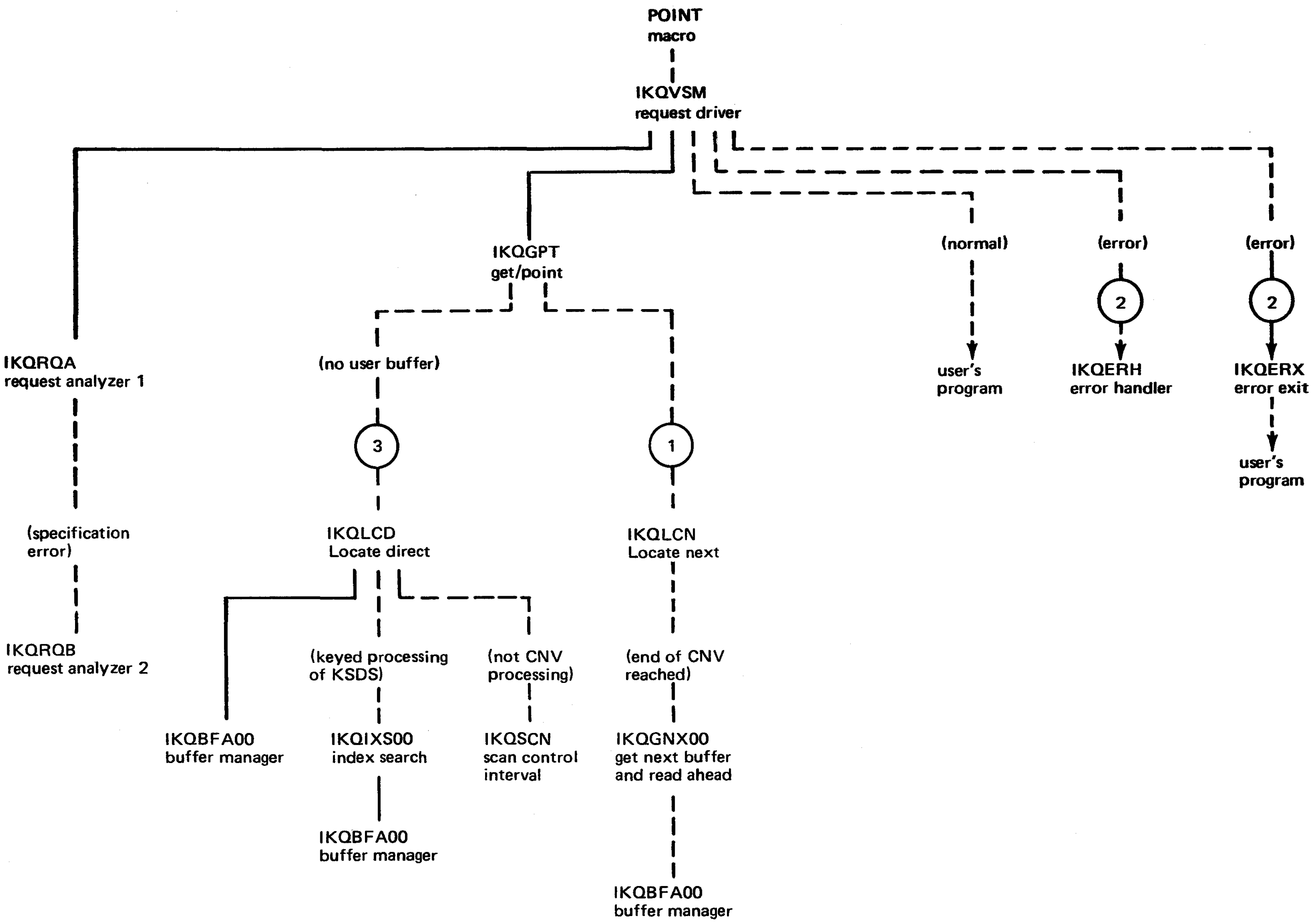


Figure 3.2 Program structure to process POINT (part 2 of 2)

(1.) IKQLCN is called
- if no user buffer and
  - a) IKQLCD didn't find the record and reached end of Control interval whenever:
    - FWD, KGE or
    - FWD, GEN or
    - BWD, LRD
  - b) BWD, LRD with ADR processing or keyed processing of RRDS

(2.) Possible logical errors are:
- ADR: Invalid RBA
- Keyed: no record found
end of data

(3.) IKQLCD is not called
- if user buffer
- if BWD, LRD, ADR processing
- if BWD, LRD keyed processing of RRDS

Figure 3.3 Program structure to process GET (part 1 of 2)

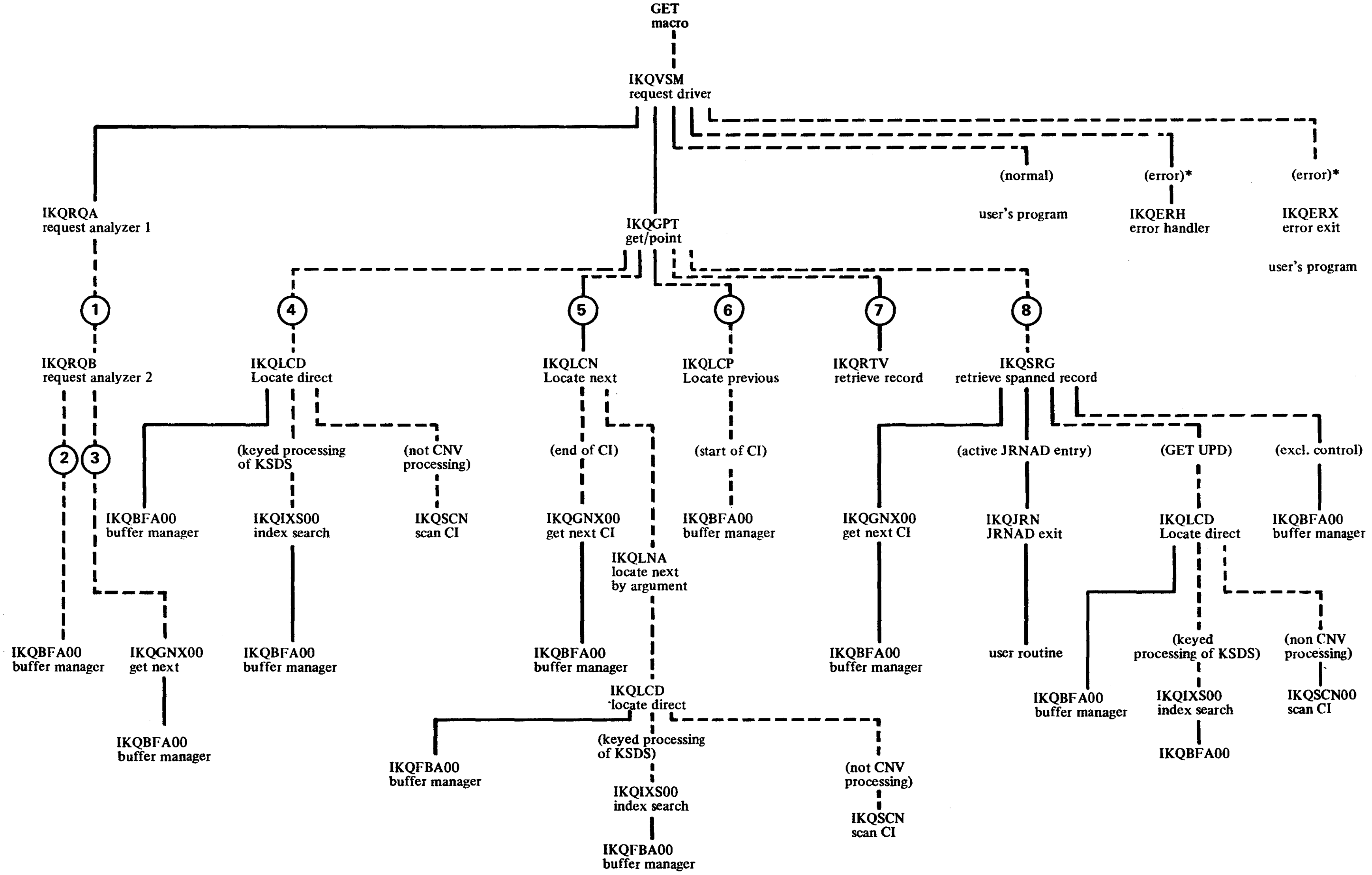


Figure 3.3 Program structure to process GET (part 2 of 2)

GET macro

1. IKQRQB is called
   - for initial positioning of the PLH for sequential forward processing
   - if specification errors were detected

2. IKQBFA00 is called
   - to read the first control interval of a data set

3. IKQGNX is called
   - if the first control interval is empty

4. IKQLCD is called
   - for direct and skip sequential processing, except for the retrieval of the last record (LRD) during addressed or keyed processing of an RRDS
   - for sequential processing
     - · if restart is required
     - · if exclusive control was required but not obtained
     - · if previous request resulted in an error or was to end of data
   - for user buffer processing
   - during sequential backward processing, of a KSDS, if a transition to the previous sequence set record is required

5. IKQLCN is called
   - for sequential forward processing, if the PLH is positioned to the last record
   - for skip sequential or direct forward processing, if the record could not be found by IKQLCD and the end of a control interval was reached and KGE or GEN was specified
   - for overlapped advance of the PLH, if the request is not for update and not LOC and UBF, or not BWD

   Note: IKQLNA is called if IKQLCN has been called for a SHAREOPTIONS(4) data set, but it was found that the SHAREOPTIONS 4 lock on the control area has been active for an excessively long time. IKQLNA is called to do the locate next function in such a way that the lock will be released and reacquired.

6. IKQLCP is called
   - for sequential backward processing, if the PLH is positioned
   - for direct backward LRD processing or keyed processing of an RRDS
   - for direct backward LRD keyed processing of a KSDS, if IKQLCD located an empty control interval

7. IKQRTV is called
   - for retrieval of non-spanned records, if user buffer processing is not specified

8. IKQSRG is called
   - for retrieval of spanned records

* Possible logical errors -
E 32 invalid RBA
E 33 no record found
E 34 end of data
E 35 user area too small
E 36 sequence error
E 44 exclusive control error
E 46 locate mode for spanned record GET
E 47 incinsistent spanned record

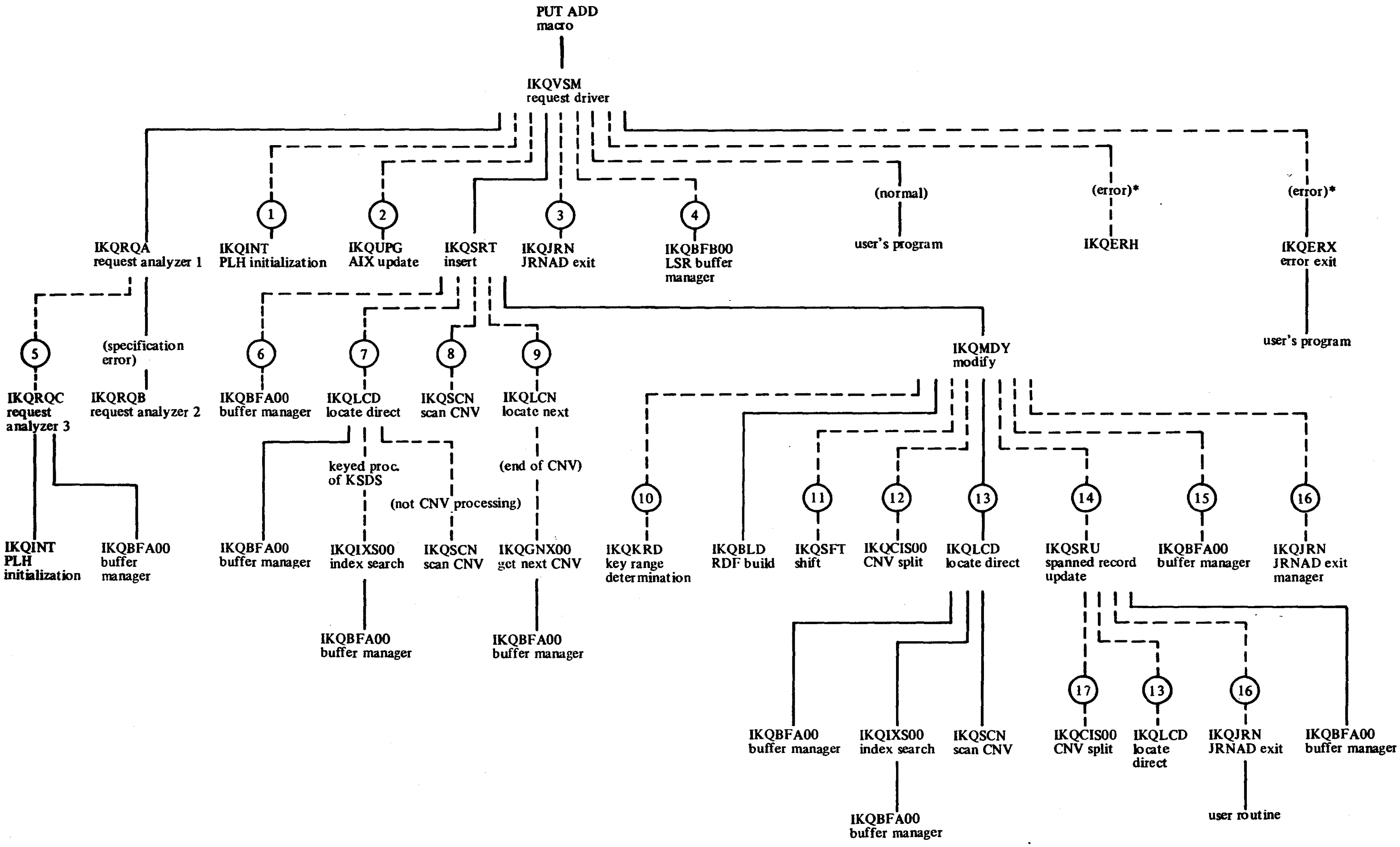


Figure 3.4 Program structure to process PUT (part 1 of 4)

Figure 3.4 Program structure to process PUT (part 2 of 4)

PUT ADD

① IKQINT is called – if LSR is specified, to initialize a PLH

② IKQUPG is called – if an upgrade set exists

③ IKQJRN is called – if the JRNAD exit is active, to inform the user of data set changes

④ IKQBFB is called – if LSR is specified, to return control blocks to the buffer pool

⑤ IKQRQC is called
- if path processing and LSR are specified, to assign a PLH to the base cluster
- if an upgrade set exists and LSR is specified, to assign BCBs

⑥ IKQBFA00 is called – for addressed processing

⑦ IKQLCD is called
- for keyed processing of an RRDS
  - if direct
  - if skip sequential
  - if sequential, after exceptional conditions or to obtain exclusive control
- for keyed processing of a KSDS
  - if direct
  - after exceptional condition
  - if PLH is not positioned
  - to obtain exclusive control

⑧ IKQSCN is called – in order to find the correct insertion point for keyed processing of a KSDS

⑨ IKQLCN is called – whenever the PLH is positioned to the previous record for keyed sequential processing of a KSDS or an RRDS

⑩ IKQKRD is called – to determine keyrange changes for a keyrange data set

⑪ IKQSFT is called – in order to make room for the record when no CNV split is necessary

⑫ IKQCIS00 is called – if a control interval split (or pseudo split) is necessary
  - if actual CNV free space is too short for changes (real split)
  - for CNV insert processing (pseudo split) except for an ESDS (see Note)
  - if first data load request (pseudo split)
  - if keyrange change (pseudo split)
  - if RRDS and insertion beyond preformatted limit (pseudo split)

Note: For CNV insert processing of an ESDS, IKQMDY carries out the pseudo-split internally.

⑬ IKQLCD is called – in order to reposition the PLH when insertion is to be retried after a CNV split. For keyed non-load processing only.

⑭ IKQSRU is called – in order to insert spanned records

⑮ IKQBFA00 is called – if immediate writing is required for user buffer processing or direct requests without the option NSP

⑯ IKQJRN is called – if the JRNAD exit is active, to inform the user of data set changes

⑰ IKQCIS00 is called
- if there are not enough CNVs in the control area to accept the spanned record
- for the insertion of each segment

*Logical errors – E 36 sequence error
E 37 duplicate record
E 43 VSAM internal logic error

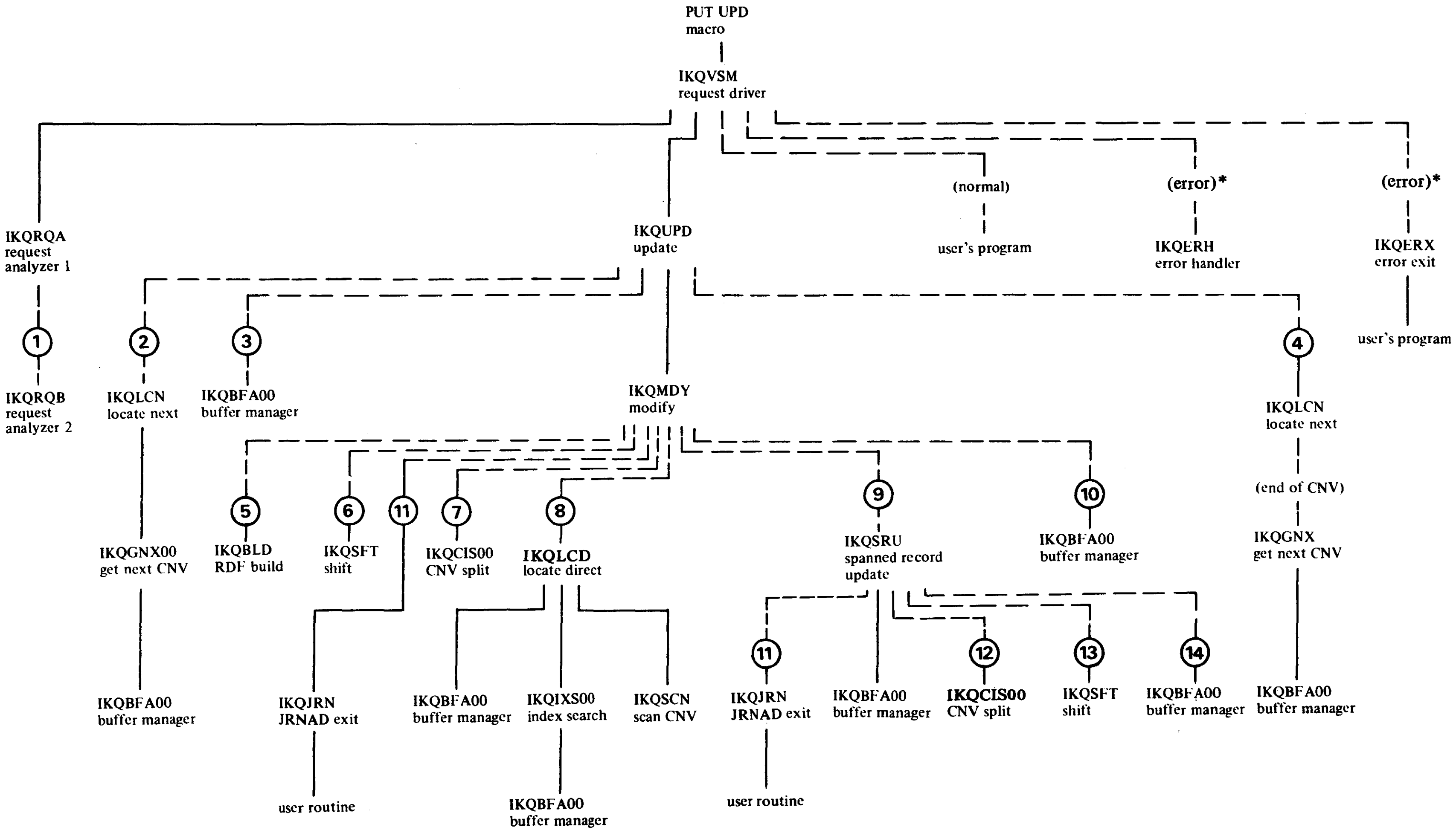


Figure 3.4 **Program structure to process PUT (part 3 of 4)**

Figure 3.4 Program structure to process PUT (part 4 of 4)

PUT UPD

① IKQRQB is called – for the initial positioning of the PLH for a sequential standalone update (user buffer processing only)
– if a specification error was detected

② IKQLCN is called – for sequential standalone update (user buffer processing only), if the PLH is positioned to the previous CNV

③ IKQBFA00 is called – in order to get the CNV to be updated for a direct standalone update

④ IKQLCN is called – in order to advance the PLH overlapped for a sequential forward request without UBF or LOC specified

⑤ IKQBLD is called – if an update to a non-spanned record causes a length change

⑥ IKQSFT is called – if a record with changed length fits into the CNV

⑦ IKQCIS00 is called – if a record with changed length does not fit into the CNV

⑧ IKQLCD is called – in order to reposition the PLH when the update is to be retried after a CNV split–for keyed processing only

⑨ IKQSRU is called – if the old record or the new record is spanned

⑩ IKQBFA00 is called – if immediate writing is required for user buffer processing or for direct requests without the option NSP

⑪ IKQJRN is called – if the JRNAD exit is active, in order to inform the user of changes to the data set

⑫ IKQCIS00 is called – if more control intervals are needed than are allocated

⑬ IKQSFT is called – in order to reorganize the sequence set record during CI space reclamation, if control intervals become free during update with length change

⑭ IKQBFA00 is called – to write freed data CNVs and changed sequence set records during CI space reclamation

*Logical error – End of data (for sequential standalone update)

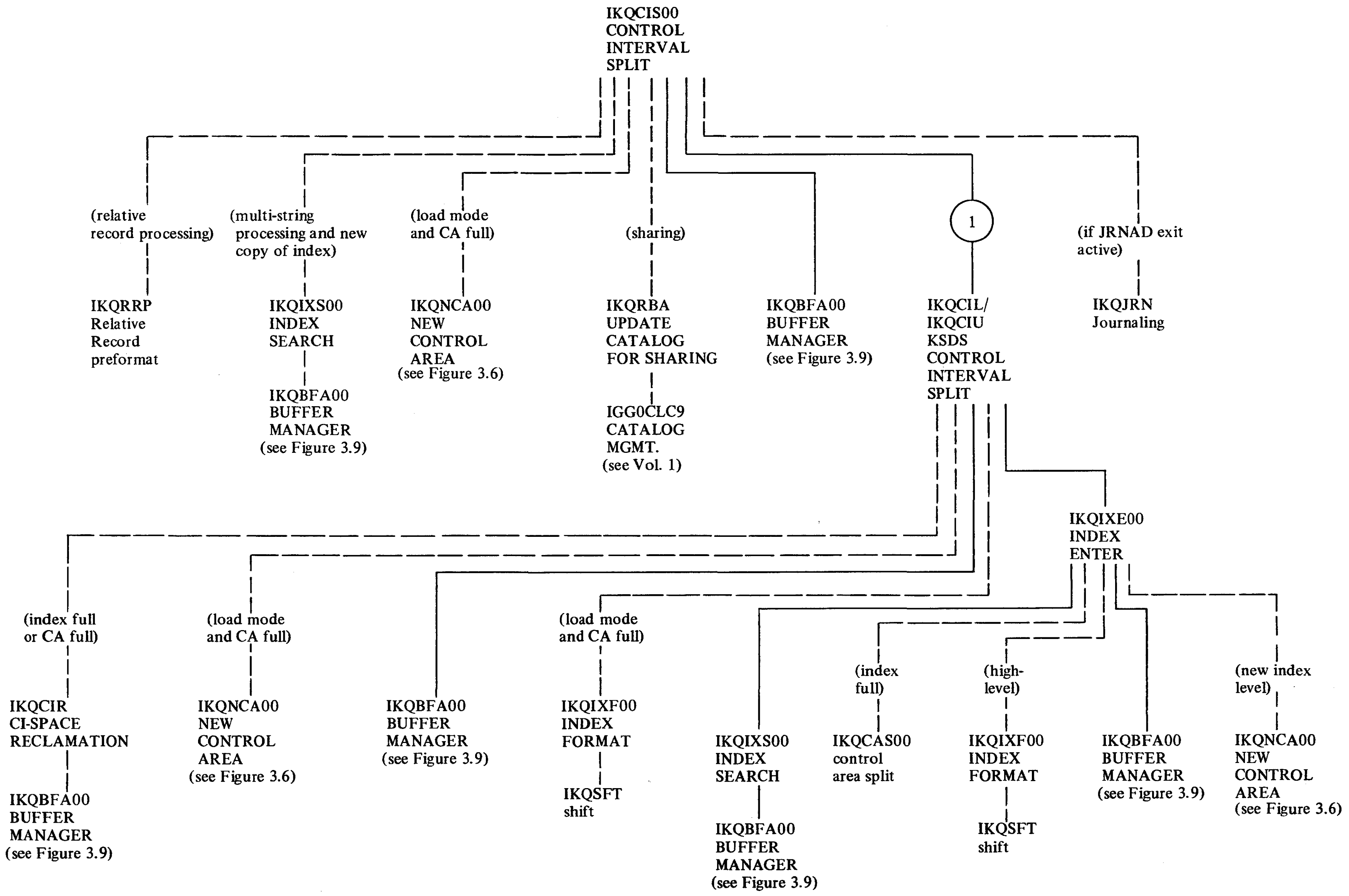


Figure 3.5 Program structure to process a control interval split (part 1 of 2)

IKQCIS00

1 IKQCIL is called for splits caused by

- load mode,
- resume load,
- mass insert.

IKQCIU is called for splits caused by

- update with length change,
- insert in front of a record already in the control interval,
- any direct mode insert, other than to the end of the data set.

Figure 3.5 Program structure to process a control interval split (part 2 of 2)

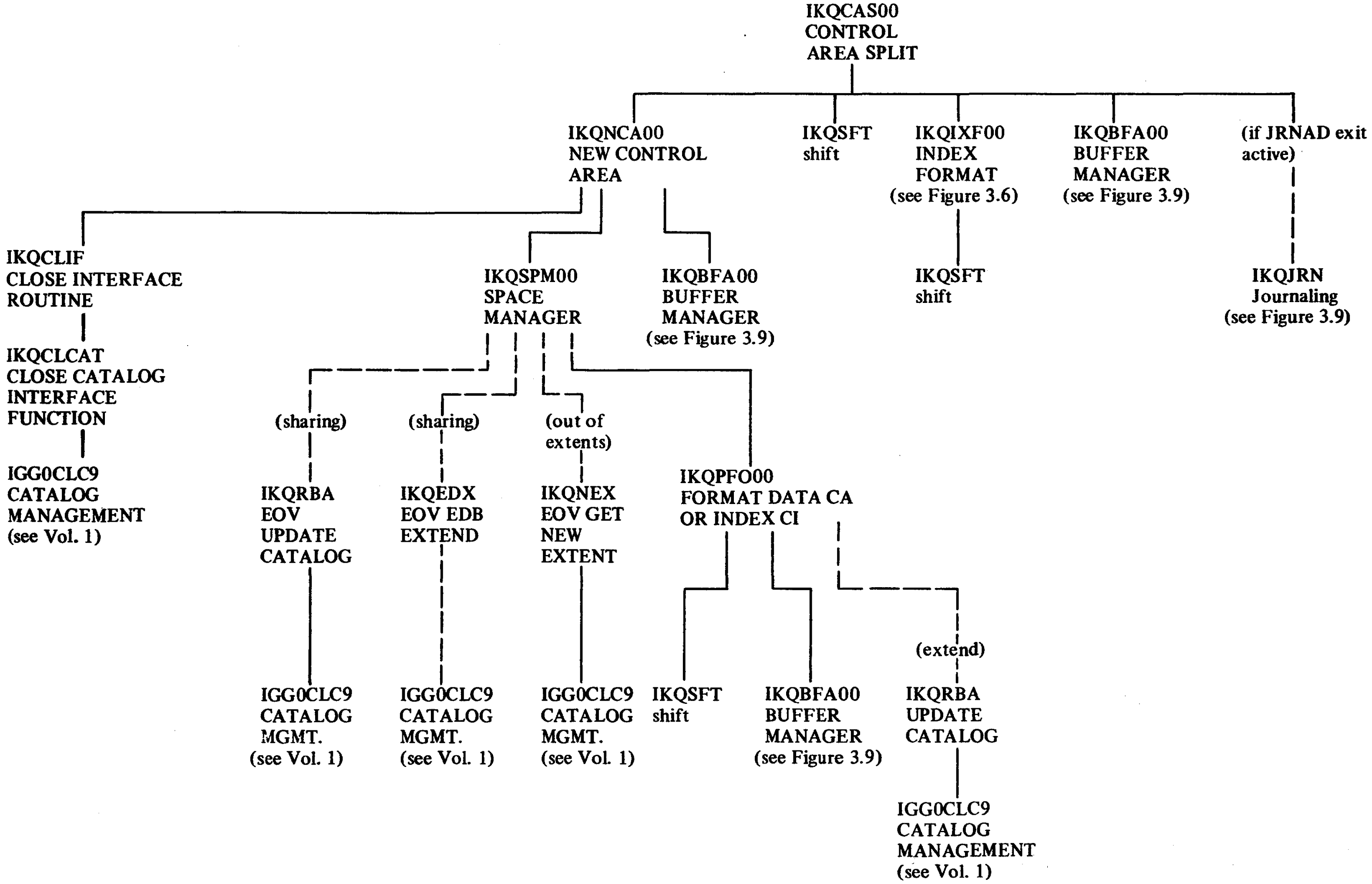


Figure 3.6 Program structure to process a control area split

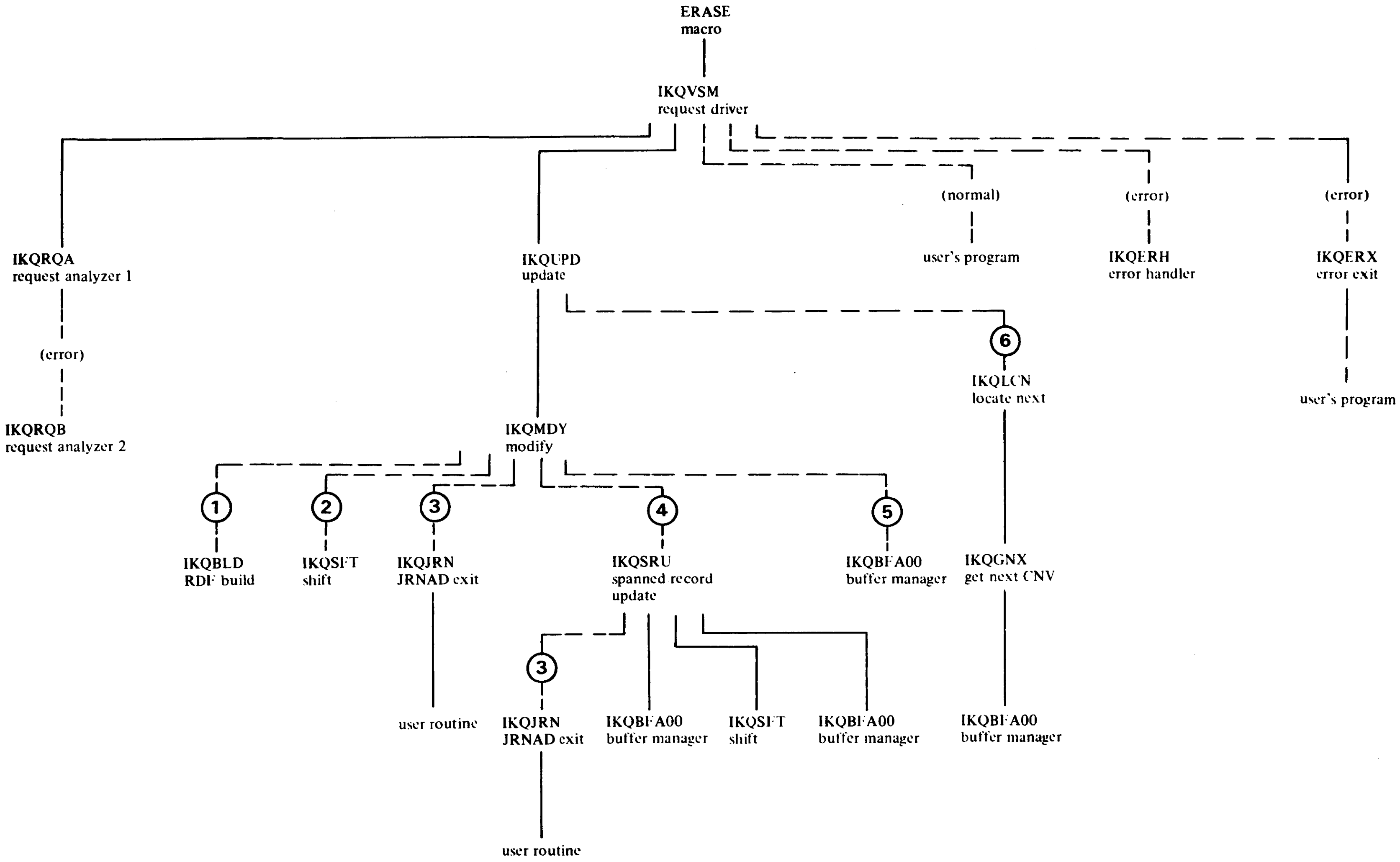


Figure 3.7 Program structure to process ERASE (part 1 of 2)

ERASE

1. IKQBLD is called - if the ERASE request is for a KSDS
2. IKQSFT is called - if the ERASE request is for a KSDS
3. IKQJRN is called - if the JRNAD exit is active
4. IKQSRU is called - if a spanned record is to be erased
5. IKQBFA00 is called - for direct requests without the NSP option
6. IKQLCD is called - for sequential and skip sequential requests during forward processing, whenever the end of a control interval is reached

Figure 3.7 Program structure to process ERASE (part 2 of 2)

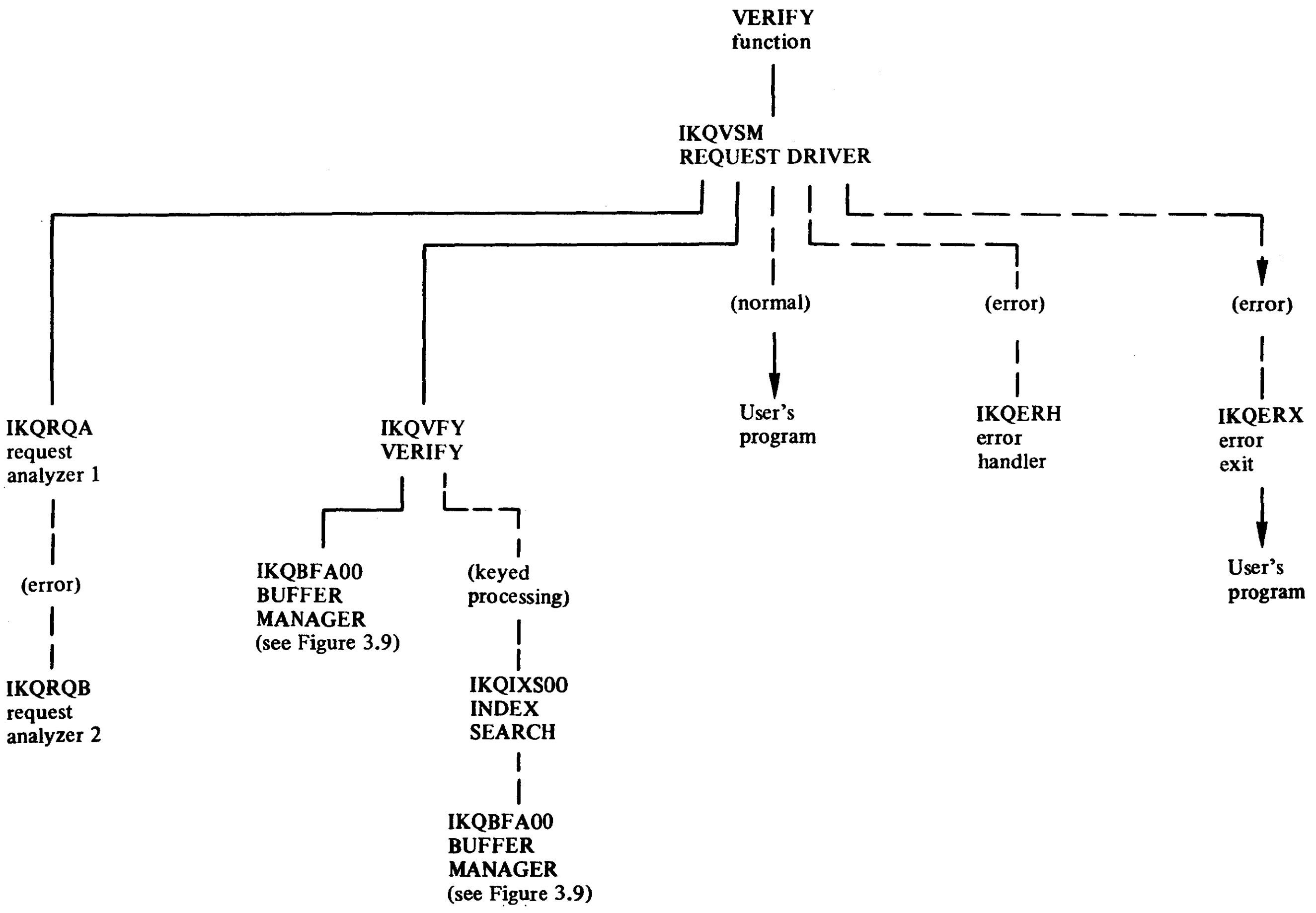


Figure 3.8 Program structure to process VERIFY

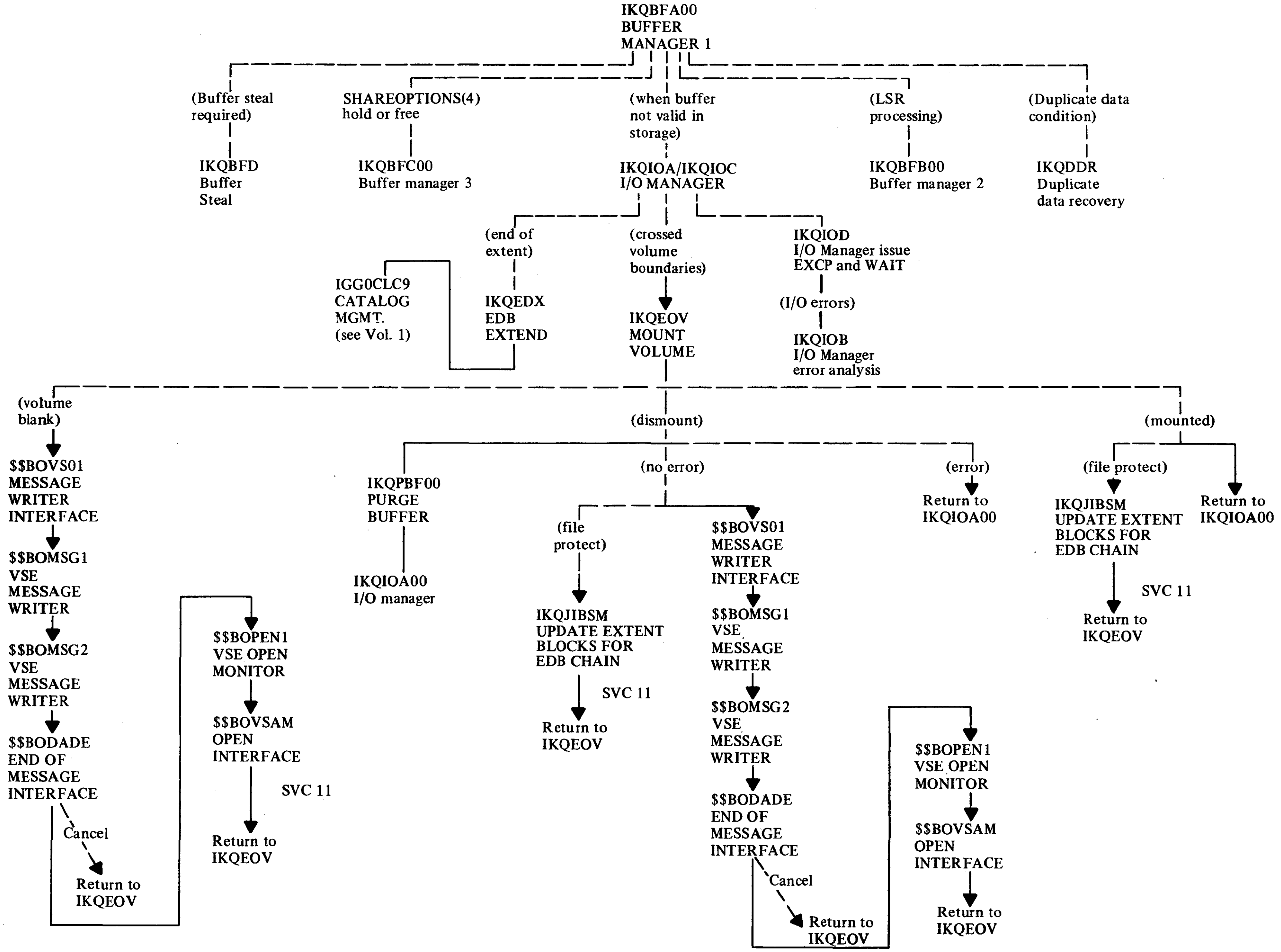


Figure 3.9 **Program structure to process buffer and I/O management**

Section 4. Directory

This section contains the following cross-reference material:

- VSAM Phase-to-Module Index
- Component Index
- Module Directory
- Routine Directory
- Data Area Directory

VSAM Phase-to-Module Index

The core image library contains the VSAM phases. Their names are identifiable by IKQV,$S, or $$B. Packaged within the phases are the VSAM modules, identifiable by the leading characters IKQ, IGG0, $S or $$B. Two service aid phases, IKQVDU and IKQVEDA are not included in the link-edit of VSAM and must be placed in the core image library by executing a job described in *Service Aids*.

The following list includes the phase names and the names of the modules included within each phase.

Phase name	Module name(s)			
IKQFTIND	IKQFTIND			
	IKQFT1			
	IKQFT2			
	IKQFT3			
IKQVASMT	IKQASNMT			
	IKQMTMSG			
IKQVBRP	IKQBRP			
IKQVCAT	IGG0CLAB	IGG0CLAX	IGG0CLBS	IGG0CLEG
	IGG0CLAC	IGG0CLAY	IGG0CLBT	IGG0CLES
	IGG0CLAD	IGG0CLAZ	IGG0CLBU	IGG0CLET
	IGG0CLAE	IGG0CLA6	IGG0CLBW	IGG0CLEX
	IGG0CLAF	IGG0CLA7	IGG0CLBX	IGG0CLEZ
	IGG0CLAG	IGG0CLA8	IGG0CLBY	IGG0CLFA
	IGG0CLAH	IGG0CLBA	IGG0CLB8	IGG0CLFB
	IGG0CLAJ	IGG0CLBB	IGG0CLCA	IGG0CLFC
	IGG0CLAK	IGG0CLBC	IGG0CLCB	IGG0CLFD
	IGG0CLAL	IGG0CLBD	IGG0CLCD	IGG0CLFE
	IGG0CLAN	IGG0CLBE	IGG0CLCG	IGG0CLFF
	IGG0CLAP	IGG0CLBF	IGG0CLCL	IGG0CLFH
	IGG0CLAQ	IGG0CLBG	IGG0CLCO	IGG0CLFQ
	IGG0CLAR	IGG0CLBH	IGG0CLCP	IKQALL00
	IGG0CLAS	IGG0CLBL	IGG0CLCR	IKQPOP00
	IGG0CLAT	IGG0CLBM	IGG0CLCS	IKQRDS00
	IGG0CLAU	IGG0CLBN	IGG0CLCX	IKQREN00
	IGG0CLAV	IGG0CLBQ	IGG0CLCY	IKQSCR00
	IGG0CLAW	IGG0CLBR	IGG0CLC9	IKQVTC00
				IKQWDS00
IKQVCLC	IKQCLCAT			
IKQVCLIF	IKQCLIF			
IKQVCLOC	IKQCLOCL			
IKQCLOS	IKQCLO			
IKQVCLOV	IKQCLOVY			
IKQVDCN	IKQDCN			
IKQVDNT	IKQDNT			
IKQVDRP	IKQDRP			
IKQVDTPE	IKQVDTPE			
IKQVDU	IKQVDU			
IKQVDUMP	IKQDUMP			
	IKQDUMPC			
IKQVEDA	IKQVEDA			
IKQVEDX	IKQEDX			
IKQVEOV	IKQEOV			

Figure 4.1 VSAM phase-to-module index (part 1 of 2)

Phase name	Module name(s)			
IKQVGEN	IKQGEN			
IKQVJIBS	IKQJIBSM			
IKQVLAB	IKQLAB			
IKQVCLRD	IKQCLRDD			
IKQVMSG	IKQOCMSG			
IKQVNEX	IKQNEX			
IKQVOPEN	IKQOPN	IKQOPNNC		
	IKQOPNAB	IKQOPNOV		
	IKQOPNAI	IKQOPNPV		
	IKQOPNCT	IKQOPNRD		
	IKQOPNDO	IKQOPNRP		
	IKQOPNDS	IKQOPNUC		
	IKQOPNHC	IKQOPNUS		
IKQVPBF	IKQPBF00			
IKQVRBA	IKQRBA			
IKQVRM	IKQAIX	IKQERX	IKQKRD	IKQRTV
	IKQBFA00	IKQGCI	IKQLCD	IKQSCN
	IKQBFB00	IKQGNX00	IKQLCN	IKQSFT
	IKQBFC00	IKQGPT	IKQLCP	IKQSPM00
	IKQBFD	IKQINT	IKQLNA	IKQSRG
	IKQBLD	IKQIOA	IKQMDY	IKQSRT
	IKQCAS00	IKQIOB	IKQNCA00	IKQSRU
	IKQCIL	IKQIOC	IKQPF000	IKQUPD
	IKQCIR	IKQIOD	IKQRCL00	IKQUPG
	IKQCIS00	IKQIXE00	IKQRQA	IKQVFY
	IKQCIU	IKQIXF00	IKQRQB	IKQVSM
	IKQDDR	IKQIXS00	IKQRQC	
	IKQERH	IKQJRN	IKQRRP	
IKQVRT	IKQVRT			
IKQVSCAT	IKQSCAT			
IKQVSHR	IKQOCSHR			
IKQVSTM	IKQSTM			
IKQVTMS	IKQTMSD			
	IKQTMSF			
$SVAVSAM	$SVAVSAM			
$$BACLOS	$$BACLOS			
$$BCLCRA	$$BCLCRA			
$$BCVSAM	$$BCVSAM			
$$BCVS02	$$BCVS02			
$$BCVS03	$$BCVS03			
$$BCVS04	$$BCVS04			
$$BODADE	$$BODADE			
$$BODADS	$$BODADS			
$$BOVSAM	$$BOVSAM			
$$BOVS01	$$BOVS01			
$$BTCLOS	$$BTCLOS			

Figure 4.1 VSAM phase-to-module index (part 2 of 2)

Component Index

VSAM is logically grouped into components, each of which consists of several modules.

Component	Module name	Module function
Rec. Mgmt	IKQAIX	Alternate index routine
	IKQBFA00	Control buffers and their contents
	IKQBFB00	LSR buffer management
	IKQBFC00	Shareoption 4 hold control
	IKQBFD	"Steal" a buffer from another buffer string
	IKQBLD	Build RDFs for all changes to a control interval
	IKQBRP	Build VSAM resource pool
	IKQCAS00	Split a control area or high-level index record
	IKQCIL	Special CI split processing for load mode
	IKQCIR	CNV space reclamation routine
	IKQCIS00	Split a control interval or get a new control interval
	IKQCIU	Special CI split processing for non-load mode
	IKQCLIF	Close interface routine for dynamic storage area
	IKQDDR	Duplicate data recovery in case of system failure during CI split
	IKQERH	Handle errors for record management modules
	IKQERX	Process error exits for record management modules
	IKQGCI	Get CI in key sequence
	IKQGNX00	Get next buffer and read records into buffers in anticipation of further user request processing
	IKQGPT	Handle GET or POINT user requests
	IKQINT	PLH initialization for LSR processing
	IKQIOA	Build channel programs for READs and WRITEs and process I/O for CKD devices
	IKQIOB	Analyze hardware errors encountered in IKQIOA00
	IKQIOC	Build channel programs and process I/O for FBA devices
	IKQIOD	Issue EXCP and/or WAIT
	IKQIXE00	Make index entries and create high-level indexes
	IKQIXF00	Balance section entries in index record
	IKQIXS00	Search the index for desired key
	IKQJRN	Journad Exit
	IKQKRD	Initialize for key-range requests
	IKQLCD	Locate a specific record by key or RBA
	IKQLCN	Locate next sequential (logical or physical) record
	IKQLCP	Get backwards function (locate previous)
	IKQLNA	Locate next sequential (logical or physical) record, and release and reacquire SHAREOPTIONS(4) exclusive control

Figure 4.2 **Component index (part 1 of 2)**

Component	Module Name	Module Function
	IKQMDY	Modify a control interval (insert, update, or delete)
	IKQNCA00	Construct a new control area
	IKQPBF00	Complete I/O processing already initiated preparatory to mounting another volume
	IKQPFO00	Format a data control area or index control interval and write SEOF
	IKQRCL00	Clean up record management requests before a VSAM CLOSE can be completed
	IKQRQA	Analyze record management requests
	IKQRQB	Complete analysis of record management requests
	IKQRQC	PLH assignment for AIX processing (LSR)
	IKQRRP	Relative record preformat
	IKQRTV	Retrieve a specific record for caller
	IKQSCN	Scan a control interval for a specific record
	IKQSFT	Shift data in a control interval
	IKQSPM00	Apportion data or index space within extents
	IKQSRG	Spanned record GET
	IKQSRT	Insert a record in a control interval
	IKQSRU	Spanned record update
	IKQUPD	Update a record in a control interval
	IKQUPG	Alternate index upgrade routine
	IKQVFY	Reestablish high-used and high-key RBAs (VERIFY function)
	IKQVSM	Perform initial processing for all record management requests and activate the modules that perform the operations
Service Aids	IKQCLEAN	DADSM utility
	IKQDUMP	Dump non-catalog control blocks
	IKQDUMPC	Dump catalog control blocks
	IKQVEDA	Enable and disable VSAM snap dump routine
	$$BCVS03	Load a phase
	$$BCVS04	I/O routine for IKQVEDA

Figure 4.2 Component index (part 2 of 2)

Module Directory

The module directory (Figure 4.3) is organized alphabetically by symbolic module name. It lists the descriptive name, the component to which that module belongs, the method of operation diagram and program structure figure numbers in which that module is referenced, and the external entry point(s).

Module name	Descriptive name	Component	Diag.#	Structure Figure 3.x	External entry points
IGG0CLC9*	Catalog first load	Catalog	-	5, 6, 9	IGG0CLC9
IKQAIX	AIX routine	Rec. Mgmt.	BC		IKQAIX
IKQBFA00*	Buffer manager	Rec. Mgmt.	BS, DA-DF DI	2-9	IKQBFA00
IKQBFB00*	LSR buffer manager	Rec. Mgmt,	FE-FI	4, 9	IKQBFB00
IKQBFC00	Track hold control	Rec. Mgmt.	DA, DG, DH, DJ, DK, DL, DM	9	IKQBFC00 IKQBFC10 IKQBFC20 IKQBFC30 IKQBFC40 IKQBFC50
IKQBFD	Buffer "steal" function	Rec. Mgmt.	9	-	IKQBFD10
IKQBLD	RDF-build, non-spanned records	Rec. Mgmt.	BR	4, 7	IKQBLD
IKQCAS00	Control area split	Rec. Mgmt.	CL	5, 6	IKQCAS00
IKQCIL	CI split for load mode	Rec. Mgmt.	CB	5	IKQCIL
IKQCIR	CNV space reclamation routine	Rec. Mgmt.	CG	5	IKQCIR
IKQCIS00*	Control interval split	Rec. Mgmt.	CA-CD	4, 5	IKQCIS00 IKQCIS10 IKQCIS20 IKQCIS30
IKQCIU	CI split for non-load	Rec. Mgmt.	CC	5	IKQCIU10
IKQCLCAT*	Close catalog interface function	O/C/EOV	-	6	IKQCLCAT
IKQCLEAN	VTOC maintenance utility	Serv. aids	-	-	IKQCLEAN
IKQCLIF	DSA CLOSE interface function	O/C/EOV	-	6	IKQCLIF
IKQCLNLK	Phase and include statement	VSAM incl.	-	-	IKQCLEAN
IKQDDR	CI split duplicate data recovery	Rec. Mgmt.	CE	9	IKQDDR
IKQDUMP	Block dump	Serv. aids	-	-	IKQDUMP IKQDUMPP
IKQDUMPC	Dump catalog control blocks	Serv. aids	-	-	IKQDUMPC
IKQEDX	EDB extend	O/C/EOV	FB	6, 9	IKQEDX00
IKQEOV	Mount volume	O/C/EOV	FA	9	IKQEOV00
IKQERH*	Error handler	Rec. Mgmt.	CQ	2-4, 7, 8	IKQERH
IKQERX*	VSAM error exit	Rec. Mgmt.	CR	2-4, 7, 8	IKQERX
IKQGCI	Get CI	Rec. Mgmt.	CF	-	IKQGCI
IKQGNX00	Get next buffer and read ahead	Rec. Mgmt.	BS	2-4, 7	IKQGNX00
IKQGPT	Get/Point	Rec. Mgmt.	BE-BF	2, 3	IKQGPT
IKQINT	PLH initialization	Rec. Mgmt.	-	4	
IKQIOA	CKD I/O manager	Rec. Mgmt.	EA-EG, EI, EK, EN-EP, ES, EU	9	IKQIOA00
IKQIOB	I/O manager, I/O error analysis	Rec. Mgmt.	EW	9	IKQIOB00
IKQIOC	FBA I/O manager	Rec. Mgmt.	EA-EG, EI, EK, EN, EQ ES, EU	9	IKQIOC
IKQIOD	I/O manager EXCP/WAIT	Rec. Mgmt.	EH-EM, EQ, ER, ET, EV	9	IKQIOD10 IKQIOD20

* Refer to *VSE/VSAM VSAM Logic, Volume 1* for additional documentation.

Figure 4.3 Module directory (part 1 of 2)

Module name	Descriptive name	Component	Diag.#	Structure Figure 3.x	External entry points
IKQIXE00	Index enter	Rec. Mgmt.	CK	5	IKQIXE00 IKQIXE20
IKQIXF00	Index format	Rec. Mgmt.	CI	5, 6	IKQIXF00
IKQIXS00	Index search	Rec. Mgmt.	BU	2-5, 8	IKQIXS00
IKQJIBSM*	Build and delete extent blocks (JIBs)	O/C/EOV	-	9	IKQJIBSM
IKQJRN	JRNAD exit	Rec. Mgmt.	FD	3-7	IKQJRN
IKQKRD	Key range determination routine (KRDR)	Rec. Mgmt.	BQ	4	IKQKRD
IKQLCD	Locate direct	Rec. Mgmt.	BP	2-4	IKQLCD
IKQLCN	Locate next	Rec. Mgmt.	BL	2-4, 7	IKQLCN
IKQLCP	Locate previous	Rec. Mgmt.	BM, BO	3	IKQLCP
IKQLNA	Locate next by argument	Rec. Mgmt.	BM	3	IKQLNA
IKQMDY	Modify	Rec. Mgmt.	BI, BQ	4, 7	IKQMDY
IKQNCA00	Get new control area	Rec. Mgmt.	CJ	5, 6	IKQNCA00
IKQNEX*	Get new extent	O/C/EOV	CP	6	IKQNEX00
IKQOCMSG*	Open/Close message routine	O/C/EOV	CQ	-	IKQOCMSG
IKQPBF00*	Purge buffer	Rec. Mgmt.	FC	9	IKQPBF00
IKQPFO00	Format data CA or index CNV	Rec. Mgmt.	CN	6	IKQPFO00
IKQRBA	Update catalog for sharing	O/C/EOV	GC, CO	5, 6	IKQRBA00
IKQRCL00*	Record management close	Rec. Mgmt.	CS	-	IKQRCL00
IKQRQA*	Request analyzer 1	Rec. Mgmt.	BB	2-4, 7, 8	IKQRQA
IKQRQB*	Request analyzer 2	Rec. Mgmt.	-	2-4, 7, 8	IKQRQB
IKQRQC*	Request analyzer 3	Rec. Mgmt.	-	4	IKQRQC
IKQRRP	Relative record preformat	Rec. Mgmt.	CH	5	IKQRRP IKQRRP20
IKQRTV	Retrieve	Rec. Mgmt.	BF, CT	3	IKQRTV
IKQSCN	Scan control interval	Rec. Mgmt.	BG, BP	2-4	IKQSCN
IKQSFT*	Shift	Rec. Mgmt.	CI, CL	4, 5-7	IKQSFT
IKQSGP**	Used for Space Mgmt. Feature	Rec. Mgmt.	-	-	IKQSGP
IKQSIN**	Used for Space Mgmt. Feature	Rec. Mgmt.	-	-	IKQSIN
IKQSLD**	Used for Space Mgmt. Feature	Rec. Mgmt.	-	-	IKQSLD
IKQSLN**	Used for Space Mgmt. Feature	Rec. Mgmt.	-	-	IKQSLN
IKQSLP**	Used for Space Mgmt. Feature	Rec. Mgmt.	-	-	IKQSLP
IKQSPM00	Manage space within extents	Rec. Mgmt.	CM	6	IKQSPM00
IKQSRG	Spanned record GET	Rec. Mgmt.	BJ	3	IKQSRG
IKQSRT	Insert	Rec. Mgmt.	BG	4	IKQSRT
IKQSRU	Spanned record UPDATE	Rec. Mgmt.	BK	4, 7	IKQSRU
IKQSUP**	Used for Space Mgmt. Feature	Rec. Mgmt.	-	-	IKQSUP
IKQUPD	Update	Rec. Mgmt.	BH, BI	4, 7	IKQUPD
IKQUPG	Alternate index upgrade routine	Rec. Mgmt.	BD	4	IKQUPG
IKQVEDA	Enable and disable VSAM diagnostic aids	Service Aids			IKQVEDA
IKQVFY	Verify	Rec. Mgmt.	BT	8	IKQVFY
IKQVSMLK	Phase and include statements	VSAM	-	-	-
IKQVSM	VSAM request driver	Rec. Mgmt.	BB	2-4, 7, 8	IKQVSM
$SVAVSAM	SVA module list	VSAM	-	-	
$$BODADE*	End of message interface	DADSM	-	9	$$BODADE
$$BOVSAM*	Open interface	O/C/EOV	-	9	$$BOVSAM
$$BOVS01*	Catalog/DADSM interface to mount volume	O/C/EOV	FA	9	$$BOVS01

*Refer to *VSE/VSAM VSAM Logic, Volume 1* for additional documentation.

**Refer to *VSE/VSAM Space Management for SAM Feature Logic* for additional documentation.

Figure 4.3 **Module directory (part 2 of 2)**

Routine Directory

Some of the VSAM modules contain several routines which are listed alphabetically by the entry points along with the appropriate module. Figure 4.4 contains record management modules.

Entry point	Procedure description
IKQBFA10	Buffer manager, GETBUFF
IKQBFA20	Buffer manager, FREEBUFF
IKQBFA30	Buffer manager, get a BCB
IKQBFA40	Buffer manager, free a buffer
IKQBFA50	Buffer manager, do I/O
IKQBFA60	Buffer manager, get a scratch BCB
IKQBFA70	Buffer manager, check RBA for exclusive control
IKQBFA80	Buffer manager, return a BCB
IKQBFB10	Buffer manager, defer writing buffer
IKQBFB20	Buffer manager, write deferred buffers
IKQBFB30	Buffer manager, get scratch buffer from resource pool
IKQBFB40	Buffer manager, return scratch buffer to resource pool
IKQBFB50	Buffer manager, search resource pool for requested RBA
IKQBFC10	Buffer manager, share option 4 hold
IKQBFC20	Buffer manager, share option 4 free
IKQBFC30	Buffer manager, share option 4 hold time-out
IKQBFC40	Buffer manager, issue LOCK.
IKQBFC50	Buffer manager, issue UNLOCK.
IKQBFD10	Buffer manager, "steal" a BCB
IKQCAS80	Control area split, count index entries
IKQCAS90	Control area split, decompress keys
IKQCIS10	CI split, assign a free CI for use
IKQCIS20	CI split, initialize a buffer for a new CI
IKQCIS30	CI split, update high-key RBA
IKQCIU10	CI split, re-enter non-load-mode CI split when index has to be split
IKQIOD0T	Special label on a test-under-mask/branch sequence that bypasses EXCPAD if the I/O has already completed. Used to ZAP the sequence for test purposes so that the bypass will not occur.
IKQIOD10	Issue EXCP.
IKQIOD20	Issue WAIT.

Figure 4.4 External entry points of record management modules

Control Block Directory

The control block directory (Figure 4.5) contains a short entry for each of the most important VSAM control blocks for the components documented in this volume, giving the length and the purpose of each block.

Data Area	Total size	Purpose
ACB	68 bytes	To describe a VSAM cluster
AMBL	112 bytes	To connect an ACB to the PLH and AMDSB(s)
AMDSB	200 bytes	To record data set status and statistics (not including buffer header and first EDB)
ARDB	48 + (2 x key length)	To record data about volumes and RBAs per key range
BCB	108 bytes	To point to a buffer
BHD	52 bytes	To contain information about buffers and buffer processing
BKPHD	64 bytes	To describe the storage allocation for the CCW build area
BSPH	72 bytes	To contain information about buffers in a subpool of the resource pool
CIW	404 bytes + (5 x keylength)	To describe a control interval split workarea
EDB	52 bytes	To contain the extent descriptions
EXLST	30 bytes (variable)	To contain addresses for user exit routines
FCDB	64 bytes	Describes the channel program block.
LPMB	24 bytes	To describe the logical and physical nature of device
PLH	ESDS 504 bytes KSDS 572 + RRDS keylength	To determine record or CNV position
RPHD	8 bytes	To contain information about the resource pool
RPL	52 bytes	To contain user request information and error feedback information
RSCB	16 bytes	To contain information needed for sharing resources
SHRW	60 bytes +key length	To contain information and serve as a work area for special handling related to file sharing (SHAREOPTIONS(4))
THB	60 bytes	To contain information needed to do SHAREOPTIONS(4) locking
USB	40 bytes (variable)	To maintain request information

Figure 4.5 Control block directory

Section 5. Data Areas

This section deals with the internal data areas of VSAM record management, describing their formats, functions, and interrelationships. It is assumed that you are familiar with the basic structure of VSAM, such as the types of data sets, the structure of indexes, the concept of the catalog, etc., as these are described in *Using VSE/VSAM Commands and Macros.*

The section contains a description and format of the VSAM record management control blocks, together with figures showing their interrelationships. VSAM control blocks that do not appear in this volume are documented in *VSE/VSAM VSAM Logic, Volume 1.*

VSAM Data Set

A VSAM data set is a collection of records grouped into control intervals. Control intervals are grouped into larger units called control areas. If the VSAM data set is key-sequenced, then the control interval(s) in which it resides are pointed to by entries in an associated index. The VSAM stored record, control interval, control area, and index are described in the topics that follow.

VSAM Record

Records are normally treated by VSAM as variable-length records. Records can be spanned across control intervals within a control area, and their maximum size is thus equal to the length of a control area. The only exception to this is a relative-record data set, whose records must have a fixed length.

Control Interval

A control interval is a continuous area of auxiliary storage that VSAM uses for storing records. The control interval is the unit of information that VSAM transfers between virtual and auxiliary storage.

The length of each control interval is an integral multiple of block size. The size of a control interval is determined by the system from the size of the records, user-specified minimum buffer size, device characteristics, and the user-specified percentage of free space. You can specify the size of the control interval, but it must be within limits acceptable to VSAM. Control interval length must be in the range of 512 to 32,768 bytes. If the length is between 512 and 8,192, the value must be a multiple of 512; if the length is between 8,193 and 32,768, the value must be a multiple of 2,048.

Data records are put in the low-address portion of the control interval. Control information about each data record is put in the high-address portion of the control interval. The combination of a data record and its control information, though they are not physically adjacent, is called a stored record. The control information in a control interval consists of a Control Interval Definition Field and one or more Record Definition Fields. Figure 5.1 shows the format of a control interval.

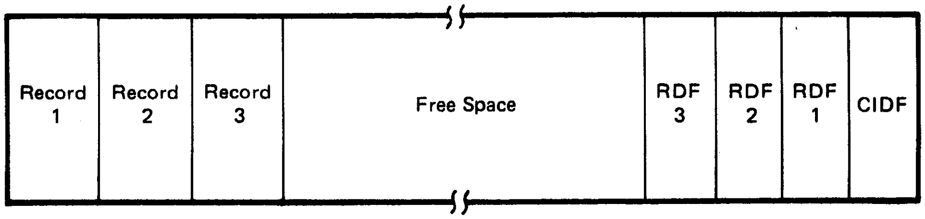


Figure 5.1 **Control interval format**

Control Interval Definition Field

The Control Interval Definition Field (CIDF) describes the control interval. Its format is shown in Figure 5.2.

Offset		Bytes and		
Dec	Hex	Bit Pattern	Field Name	Description
0	0	2	CIDFDD	Free space offset (binary) Displacement from the beginning of the control interval to the beginning of the free space[1]
2	2	2	CIDFLL	Free space length (binary) Length of the free space area within this control interval[1]
2	2	1...	CIDFDDP	The process of moving records from this control interval to another is not complete; duplicate records may exist.

[1] If the CIDF contains only 0s, *end-of-data-set* or *end-of-key-range* is indicated; either the end of the data set was detected or the end of a key range in a key-sequenced data set was detected when the data set was to be divided between volumes. Information in the volume group occurrence (see VOLFLG) in the data set's catalog record helps to differentiate between the end-of-data-set and end-of-key-range conditions.

Figure 5.2 **Control interval definition field format**

Record Definition Field

The Record Definition Fields (RDFs) describe the records in the control interval. They are inserted into the control interval from right to left, which means that the rightmost RDF describes the leftmost data record.

There is normally one RDF for each record, except in two special cases. These are:

- When two or more consecutive records in the control interval have the same length. In this case, two RDFs are used to describe the whole group of records. The first (right-hand) RDF describes the characteristics of the records, and the second (left-hand) RDF contains a count of the number of records.

  Note that this is true only for key-sequenced and entry-sequenced data sets. The slots or records in a relative record data set have a fixed length, but specific information is required for each one. The records cannot, therefore, be grouped, and one RDF is required for each record.

- When the record is spanned. In this case, only one segment of one record can be located in the control interval. Nevertheless, two RDFs are used. The first (right-hand) RDF describes the record segment, and the second (left-hand) RDF contains a 'level number', which is used for data integrity checking. This number is assigned and updated by VSAM whenever the spanned record is processed. The level number in all segments of a spanned record will always be the same, unless an error has occurred.

The format of an RDF is shown in Figure 5.3.

Offset		Bytes and		
Dec	Hex	Bit Pattern	Field Name	Description
0	0	1	RDFFLAG	Flag byte
			RDFEXT	RDF extension flag
		.0..		There is no RDF to the left of this RDF that contains additional information about record(s) described in this RDF.
		.1..		There is an RDF to the left of this RDF that contains additional information about the record(s) described in this RDF. Byte two and three of the RDF to the left contain the following information: • if there are more consecutive records than one of fixed length they contain the number of these records beginning with the record associated with the previous (to the right) RDF (see replication count flag) • in the case of spanned records they contain the level number
		..00		This is the only segment of a stored record.
		..11	RDFSRM	The RDF to the left contains information about spanned records (middle segment)
		..10	RDFSRL	The same as above but last segment
		..01	RDFSRF	The same as above but first segment
			RDFREPL	Replication count flag
		.... 0...		The second and the third bytes of this RDF contain the data records length
		.... 1...		This RDF contains additional information about the record(s) described in the RDF to the right.
		.... x...	RDFRESL	Empty slot indicator (for relative record processing where one RDF is associated with one slot in the control interval - no extended RDFs)
		.... .1..		The record in the corresponding slot is invalid (it has been deleted or not yet inserted)
		.... .0..		The record in the corresponding slot is valid
				Depending on the kind of record(s) described, byte two and three of an RDF contain one of the following values:

Figure 5.3 **Record definition field format (part 1 of 2)**

Offset Dec	Offset Hex	Bytes and Bit Pattern	Field Name	Description
1	1	2	RDFLL	Length field Always present in the rightmost (or only) RDF for a record or group of fixed-length records. These bytes contain the data record's length or the length of the segment of a spanned record. Byte 0, Bit 4 = 0, Bit 2, 3 = 0
1	1	2	RDFCOUNT	Count field These bytes contain the number of consecutive fixed-length records. It is a type of RDF that contains additional information about the records described in the RDF to the right. Byte 0 (of the RDF to the right) Bit 4 = 1, Bit 2, 3 = 0
1	1	2	RDFSRLVL	Level number These bytes contain the level number for spanned records. It is a type of RDF that contains additional information about the records described in the RDF to the right. Byte 0 (of the RDF to the right) Bit 4 = 1, Bit 2,3 = 11 or 10 or 01

Figure 5.3 **Record definition field format (part 2 of 2)**

Control Area

A control area consists of control intervals; the number of control intervals in a control area is determined by VSAM. The control area is the amount of space that VSAM preformats so that data integrity is ensured for records added to a data set.

Control areas are also used to simplify and localize the movement of records when records are inserted in a key-sequenced data set. If an insertion requires a free control interval and there isn't one, a control-area split results. VSAM establishes a new control area and moves the contents of approximately half of the full control area to free control intervals in the new control area. The new records, as their keys dictate, are then inserted into one of the two control areas. The control area has no specific control information.

Index

An index is created at the same time as a key-sequenced data set. The index structure exists in its own address space and consists of one or more levels. The lowest level or *sequence set* consists of one or more index records. There is an index record in the sequence set for each formatted control area. Within a sequence-set record there is either an index entry or a free data control interval pointer for each control interval in the control area. (Free data control interval pointers are discussed later in this section.) The key in each entry of a sequence set record is the same as the key of the last (highest) entry in the corresponding control interval. To save space, VSAM compresses the keys in the index.

The upper levels of the index are collectively called the index set, and contain index entries which point to the next lower level of the index. Figure 5.4 shows a simple index structure.

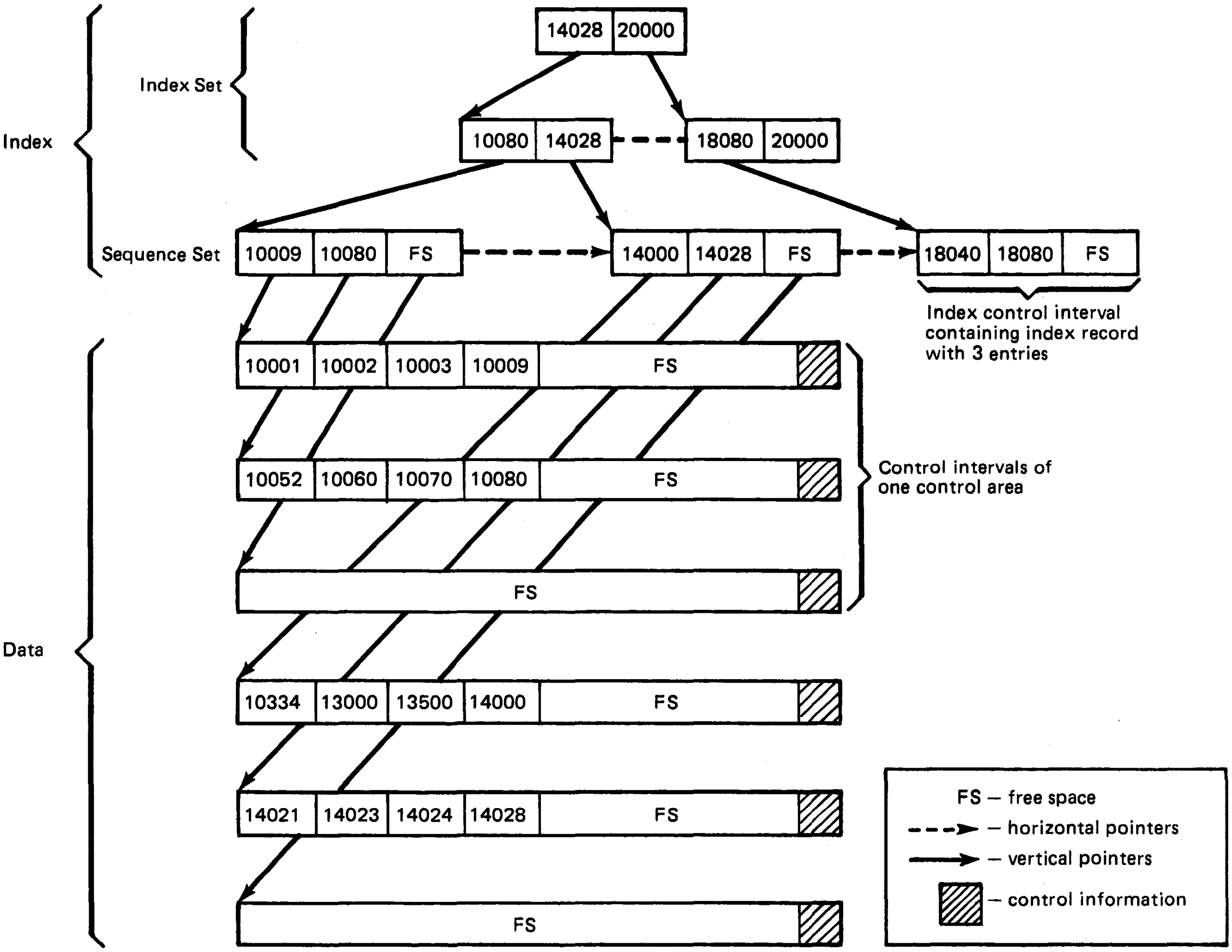


Figure 5.4 **Example of a simple VSAM index**

Index Record

The index records and control intervals are fully compatible with VSAM data records and control intervals, and are handled by record-management modules in the same way. The only differences between index records and data records are:

- There is only a single index record in an index control interval (and thus only one RDF).
- The internal format of an index record is fixed. This format is shown in the example of Figure 5.5, and its various parts are discussed.

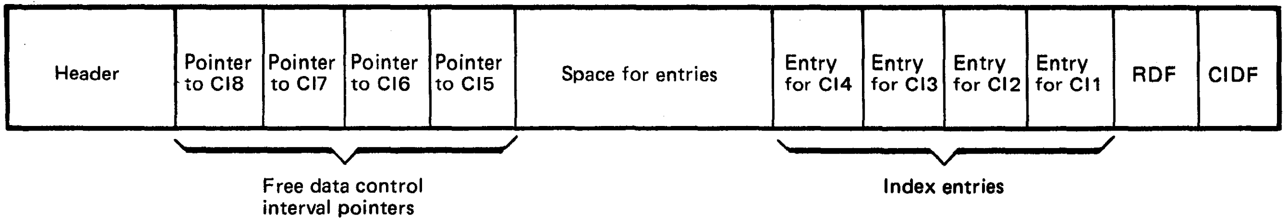


Figure 5.5 **Example of an index control interval**

Index Record Header

The index record header contains the information needed to insert index entries, to locate entries within the record, and to convert pointers into RBAs. The format of the index record header is shown in Figure 5.6.

Offset			Field	
Dec	Hex	Field Name	Size	Description
0	0	IXRL	2	Length in bytes, of the index record, including this field.
2	2	IXCINL	1	Length, in bytes, of the control information (the IXENTRYF, IXENTRYL, and IXENTRYP fields) in each index entry.
3	3	IXPMASK	1	Length of the pointers to free data control intervals in this index record[1]. This field is used as a mask for insert character (store character) under mask instructions that are used to access pointers. The value contained in this field specifies the length of these pointers, as follows: B'0001' 1-byte pointer B'0011' 2-byte pointer B'0111' 3-byte pointer
4	4	IXBASRBA	4	For a sequence-set index record, the RBA of a data control area that contains data to be referenced. This RBA and index-entry pointers are used together to calculate the 4-byte RBA of another index record or of a data control interval (0 for high-level indexes).
8	8	IXNXTIR	4	Pointer to the logically next index record in this index level. (Horizontal pointer)
12	C		4	Reserved (0).
16	10	IXLVLNO	1	Index level number. A sequence-set index is assigned a value of 1; the next higher-level index is assigned a value of 2; etc.
17	11		1	Reserved (0).
18	12	IXINSOS	2	Displacement from the beginning of this record to the space available for inserting index entries. For higher-level indexes, the entry space immediately follows the record header; for sequence-set indexes, the entry space follows the record header and free data-control-interval pointers.
20	14	IXLENTRY	2	Displacement from the beginning of this record to the last (high-key) entry in the index record.[2]
22	16	IXFSECTN	2	Displacement from the beginning of this record to the first (low key) section entry in the index record.[2]

[1] Pointers are allowed to vary in length to conserve index space. If, for example, the number of items to be referenced by an index record is less than 256, a one-byte pointer can be used; if the number is greater than 256 and less than 65,536, a two-byte pointer can be used; and if the number is greater than 65,536, a three-byte pointer can be used.

[2] This displacement is to the F (front-key compression count) byte of the entry, not to the beginning of the entry.

Figure 5.6 Index record header format

Free Data-Control-Interval Pointers

Free data-control-interval pointers, which exist only in sequence-set index records, are used to calculate the RBAs of available data control intervals. The length of a pointer is specified in the record header.

When the index is first built, and before records have been loaded into the data set, the index records of the sequence set contain one free data control interval pointer for each data control interval.

VSAM always uses the rightmost free data-control-interval pointer when a data control interval is needed. The value of the pointer is set to 0 when the control interval is used. As pointers are set to 0, the displacement to space that is available for index entries (contained in the record header) is adjusted by the length of the free data-control-interval pointer. In this way, space used by free data-control-interval pointers is made available for index entries when the pointers are no longer required.

The example in Figure 5.5 shows a sequence set record for a control area with eight control intervals. Of these eight, the first four are now occupied by data, and the last four are still free.

Index Entries

The index entries are the link between the index and the data set. They contain the key, the pointer to the data control interval containing the data record, and information about key compression. The format of an index entry is shown in Figure 5.7.

Field Size (in bytes)	Field Name	Description
Variable	IXKEY	Key characters that determine the sequence of records in a key-sequenced data set.
1	IXENTRYF (F byte)	Front-key compression count, that is, the number of characters by which the beginning of the key has been compressed.
1	IXENTRYL (L byte)	Length of the IXKEY field.
1-3	IXENTRYP (P field)	Pointer to an index or data control interval. This value is the number of the CI within the CA (for example '4' for the fifth CI). To calculate the RBA of the CI, this value must be multiplied by the CI size and added to the contents of IXBASRBA.

Figure 5.7 **Index entry format**

Index Entries for Spanned Records

Since spanned records extend across two or more data control intervals, their index entries, sometimes called 'complex index entries', consist of a series of 'normal' entries (one for each data control interval). These entries, in turn, are basically standard index entries, but they have some special features:

- The key is contained only in the entry for the last segment of the spanned records, whose F byte contains the actual key compression count.
- The entries for all other segments contain no key, and their F byte contains a compression count equal to the key length, thus indicating a key length (in the entry) of zero.
- Each entry contains a pointer to its associated segment (or data control interval).

Index Entry Sections

To save time when searching index records for a given key, the index entries are grouped into sections. This allows a rapid search, scanning only the highest key in each section, to locate the correct section, which is then searched for the correct key.

A section is defined by a two-byte field to the left of the high-key entry in the section. This field contains the displacement from the F byte of the high-key entry in this section to the F byte of the high-key entry in the next section (to the left). The index record header contains a pointer to the F byte in the high-key entry in the first section.

For technical reasons, this division of the index entries into sections is not carried out until a control interval split is necessary in a control area. There will thus be no section definition fields in the index of a freshly loaded data set, and only some of the sequence set records in an 'older' data set will have such fields.

Alternate Index

The alternate index (AIX) provides an alternate means of access, using different keys, to the data records in the base cluster, which can be a key-sequenced or entry-sequenced data set, but not a relative-record data set. The alternate index itself is a key-sequenced data set. The index component of the AIX is identical in structure, format, and function to the index of any other key-sequenced data set. The basic structure of the data component of the AIX is also identical to that of a normal key-sequenced data set, as far as CIDFs and RDFs are concerned.

The only difference in format between the AIX and a normal key-sequenced data set concerns the records in the data component of the AIX, which have a fixed format, shown in Figure 5.8. These records form the logical connection between the AIX and the base cluster, and contain control information, the alternate key, and one or more pointers to the base cluster. If this base cluster is a key-sequenced data set, the pointers consist of the prime keys of the required data records, which are then located by means of the base cluster index. If the base cluster is an entry-sequenced data set, which has no index, the pointers are relative-byte addresses (RBAs) of the required records, which can then be located directly.

As it is possible to have more than one pointer in an AIX record, the length of such a record can vary. In extreme cases, it may be greater than the control interval length, and the record is treated as a spanned record.

Offset		Bytes and		
Dec	Hex	Bit Pattern	Field Name	Description
0	0	1	AIXFG	Flag byte
		xxxx xxx.		Reserved
		1	AIXPKP	Prime key pointers are used
		0		RBA pointers are used
1	1	1	AIXPL	Pointer length (in binary)
2	2	2	AIXPC	Number of pointers in this record (in binary)
4	4	1	AIXKL	Length of alternate key (in binary)
5	5	Note 1	AIXKY	Alternate key
Note 2		Note 3	AIXPT	First pointer to base cluster

Note 1: The length of this field is specified in AIXKL

Note 2: The displacement of this field is 5 + the length of AIXKY

Note 3: The length of this field is specified in AIXPL

Figure 5.8 **Alternate index record format**

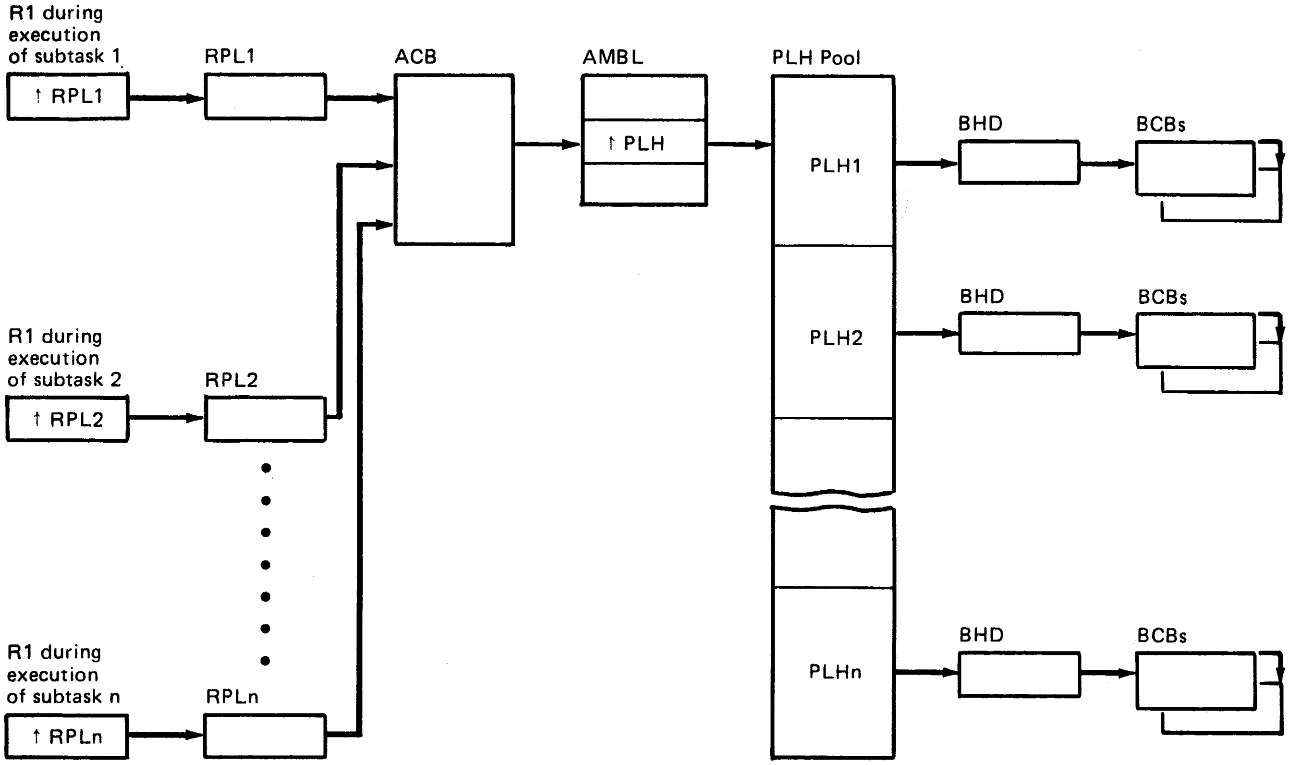

R1 during execution of subtask 1
↑ RPL1
RPL1
ACB
AMBL
↑ PLH
PLH Pool
PLH1
BHD
BCBs
BHD
BCBs
R1 during execution of subtask 2
↑ RPL2
RPL2
PLH2
R1 during execution of subtask n
↑ RPLn
RPLn
PLHn
BHD
BCBs

Figure 5.9 **Multiple string control block structure**

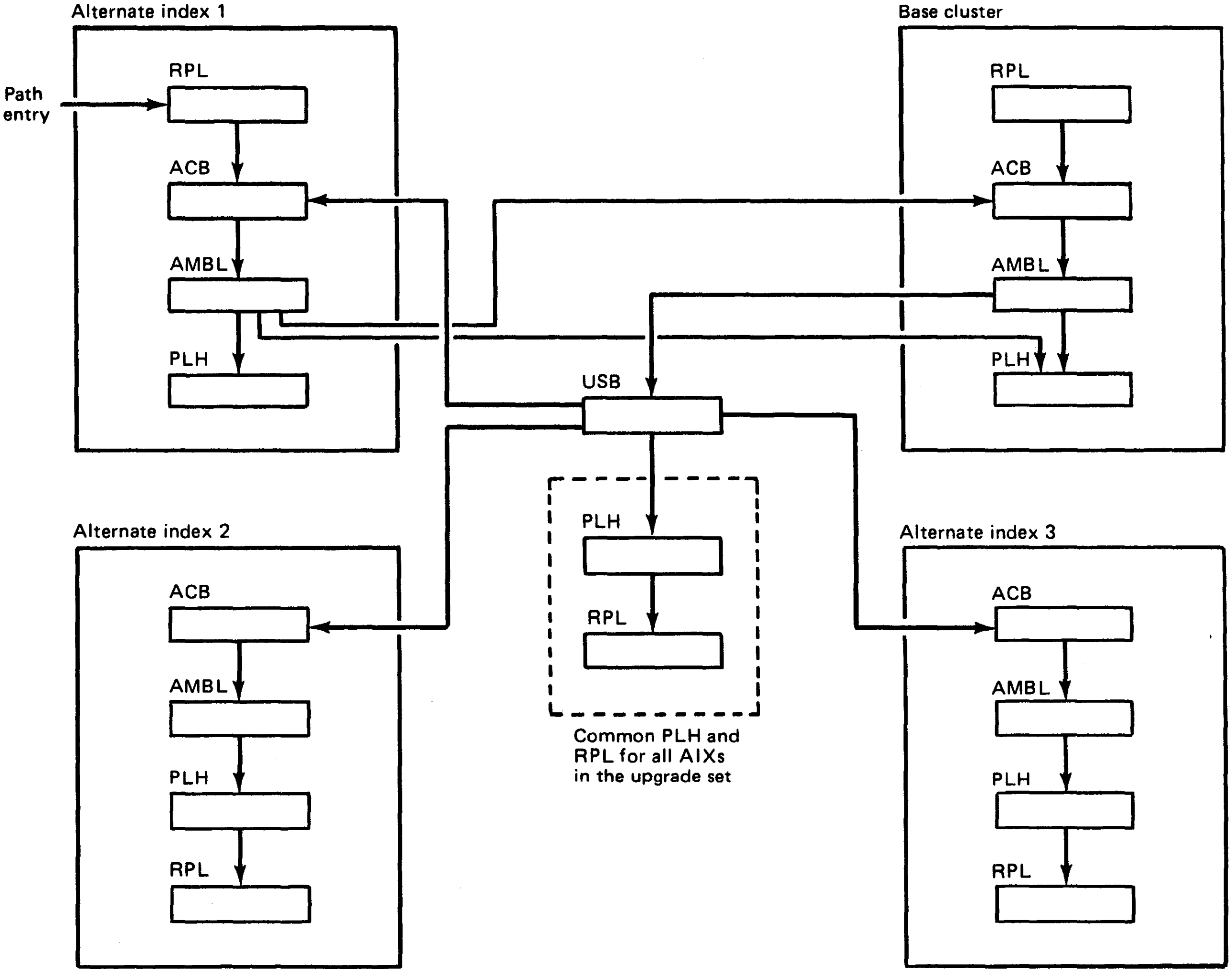


Figure 5.10 **Base cluster to alternate index control block structure**

Control Block Description and Format

Access Method Block List (AMBL)

The AMBL describes a VSAM cluster and points to the cluster's data set and index AMDSBs. When the cluster is opened, an AMBL is built to describe the cluster. AMDSBs are built to describe the cluster's data set and, if the cluster is key-sequenced, to describe the index. The AMBL is pointed to by the cluster's ACB (ACBAMBL).

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
0	0		AMBLST		Beginning of AMBL
0	0	1	AMBLID	X'11'	AMBL identifier
1	1	1	AMBLACT		AMBL active byte (X'FF'=AMBL is active)
2	2	2	AMBLLEN		Length of control block
4	4	4	AMBLDTA		Pointer to data AMDSB
8	8	4	AMBLIX		Pointer to index AMDSB
12	C	4	AMBLPLHF		Pointer to first PLH*
16	10	4	AMBCHAIN		Reserved
20	14	4	AMBLACB		Pointer to ACB
24	18	2	AMBLPLHL		Length of PLH*
26	1A	1	AMBLPLHN		Total number of strings*
27	1B	1	AMBLFLAG		Flag byte
			AMBLPOST	X'80'	POST must be issued
28	1C	4	AMBAMBUF		Size of buffer space
32	20	2	AMBMACRF		Flags (copy of flags in ACBMACR1 and ACBMACR2)
32	20	1	AMBMACR1		*First byte*:
			AMBKEY	X'80'	Access data via index or relative record number
			AMBADD	X'40'	Access via RBA
			AMBADR	X'40'	Access via RBA
			AMBCNV	X'20'	Control interval processing
			AMBSEQ	X'10'	Sequential processing
			AMBDIR	X'08'	Direct processing
			AMBIN	X'04'	GET, READ processing
			AMBOUT	X'02'	PUT, WRITE processing
			AMBUBF	X'01'	User buffers
33	21	1	AMBMACR2		*Second byte:*
			AMBLSR	X'80'	Local shared resources
			AMBDFR	X'40'	Defer writing of buffers
			AMBSKP	X'20'	Skip sequential accessing
			AMBRST	X'10'	Reusable Data Set
			AMBAIX	X'08'	AIX processing
			AMBJRACT	X'02'	JRNAD exit enabled
			AMBOPEN	X'01'	Open is in process
34	22	2	AMBLTLEN		Length of GETVIS for close work area
36	24	2	AMBDBUF		Number of data buffers
38	26	2	AMBIBUF		Number of index buffers
40	28	4	AMBLOPWA		Pointer to open work area

* When LSR is active, AMBLRPHD is not equal to 0, and AMBLSR in AMBMACR2 is set on. In this case, the fields indicated by the asterisk refer to the PLH pool.

Figure 5.11 **Access Method Block List (AMBL) description and format (part 1 of 3)**

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
				Split Control	
44	2C	4	AMBSECB		Split/pseudo-split ECB
			AMBSRCL	X'80'	IKQRCL00 set split lock
				X'40'	Reserved
				X'20'	Reserved
				X'10'	Reserved
				X'08'	Reserved
				X'04'	Reserved
				X'02'	Reserved
				X'01'	Reserved
45	2D	1			Reserved
46	2E	1	AMBSCOM		ECB post byte-split
			AMBSWAIT	X'80'	Wait bit-split
				X'40'	Reserved
				X'20'	Reserved
				X'10'	Reserved
				X'08'	Reserved
				X'04'	Reserved
				X'02'	Reserved
				X'01'	Reserved
47	2F	1	AMBSECBT		Test-and-set byte - split
48	30	4	AMBBECB		ECB for IKQRQC to ensure that only one string (at a time) extends the chain of base cluster RPLs during path processing in an LSR environment.
48	30	2			Reserved
50	32	1	AMBBCOM		ECB post byte-buffer
			AMBBWAIT	X'80'	Wait bit-buffer manager
				X'40'	Reserved
				X'20'	Reserved
				X'10'	Reserved
				X'08'	Reserved
				X'04'	Reserved
				X'02'	Reserved
				X'01'	Reserved
51	33	1	AMBBECBT		Test-and-set byte - buffer
52	34	8	AMBRBAS		RBAs for split locking
52	34	4	AMBLORBA		Low RBA of control area being split
56	38	4	AMBHIRBA		High RBA of control area being split
60	3C	1	AMBSTRID		ID of string which holds control area.

Figure 5.11 **Access Method Block List (AMBL) description and format (part 2 of 3)**

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
61	3D	1	AMBOCSW		OPEN/CLOSE switches
			AMBLAUTO	X'80'	A dynamic volume list was built for this ACB.
				X'40'	Reserved
				X'20'	Reserved
				X'10'	Reserved
				X'08'	Reserved
				X'04'	Reserved
				X'02'	Reserved
				X'01'	Reserved
62	3E	2			Reserved
64	49	4	AMBPLH		Address of PLH in control
				Pointers	
68	44	4	AMBALIST		Executive control list address
72	48	4	AMBLRPLS		Address of RPL causing split
76	4C	4	AMBLCLWA		Pointer to close work area
80	50	4	AMBLCIWA		Pointer to CI split work area
84	54	4	AMBLBC		Pointer to base cluster PLH pool
88	58	4	AMBLUSB		Pointer to USB
92	5C	4	AMBBCACB		Pointer to base cluster ACB
96	60	4	AMBPEACB		Pointer to path entry ACB
100	64	4	AMBLRPHD		Pointer to resource pool header
104	68	4	AMBDECB		ECB for duplicate data recovery
104	68	2			Reserved
106	6A	1	AMBDCOM		ECB post byte - duplicate data recovery
			AMBDWAIT	X'80'	Traffic bit
				X'40'	Reserved
				X'20'	Reserved
				X'10'	Reserved
				X'08'	Reserved
				X'04'	Reserved
				X'02'	Reserved
				X'01'	Reserved
107	6B	1	AMBDECBT		Test-and-set byte - duplicate data recovery
108	6C	0	AMBLEND		Pointer to file sharing work area

Figure 5.11 **Access Method Block List (AMBL) description and format (part 3 of 3)**

Access Method Control Block (ACB)

The VSAM ACB describes a VSAM cluster. It is built by the user's program. The ACB points to the Exit List (EXLST). After the cluster is opened, the ACB is pointed to by the RPL (RPLDACB) that describes the user's record processing request. The ACB also describes the processing options, password, and I/O buffer space applicable to the user's program.

Offset				Hex.	
Dec	**Hex**	**Bytes**	**Field Name**	**Digit**	**Description**
0	0	0	ACBST		
0	0	1	ACBID		ACB identifier = 'A0'
			ACBIDD	X'A0'	ACB equate
			ACBIDVAL	X'A0'	ACB equate
1	1	1	ACBSTYP		Release indicator
			ACBSDV1	X'00'	DOS/VS VSAM Release 1
			ACBSVSE1	X'10'	VSE/VSAM Release 1
				X'20'	VTAM
2	2	2	ACBLEN		Length of ACB in bytes
2	2	2	ACBLENG		Length of ACB in bytes[1]
4	4	4	ACBAMBL		Address of the AMBL
8	8	4	ACBAMO		Pointer to VSAM code
12	C	1	ACBACT		ACB active byte (X'FF'=ACB is active).
13	D	1	ACBINFLG		Catalog recovery flags
			ACBSCRA	X'80'	ACB is for a system-initiated OPEN of the CRA
			ACBUCRA	X'40'	ACB is for a user-initiated OPEN of the CRA
				X'20'	Reserved for CRA
				X'10'	Reserved for CRA
			ACBSTSKP	X'08'	Skip updating of statistics
				X'04'	Reserved for CMS
				X'02'	Reserved for CMS
				X'01'	Reserved for CMS
14	E	2	ACBDBUF		Number of data buffers
14	E	2	ACBBUFND		Number of data buffers
16	10	2	ACBIBUF		Number of index buffers
16	10	2	ACBBUFNI		Number of index buffers
18	12	2	ACBMACRF		MACRF
18	12	1	ACBMACR1		MACRF first byte
			ACBKEY	X'80'	Access data via index or relative record number
			ACBADD	X'40'	Access via RBA
			ACBADR	X'40'	Access via RBA
			ACBCNV	X'20'	Control interval processing
			ACBSEQ	X'10'	Sequential processing
			ACBDIR	X'08'	Direct processing
			ACBIN	X'04'	GET
			ACBOUT	X'02'	PUT
			ACBUBF	X'01'	User buffers

[1] If specified length is too small for a VSE/VSAM Release 1 ACB, a DOS/VS VSAM Release 1 ACB is built (X'00' in byte 1).

Figure 5.12　Access Method Control Block (ACB) description and format (part 1 of 6)

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
19	13	1	ACBMACR2		MACRF second byte
			ACBLSR	X'80'	Local shared resources
			ACBDFR	X'40'	Defer writing of buffers
			ACBSKP	X'20'	Skip sequential access
			ACBRST	X'10'	Reusable data set
			ACBAIX	X'08'	AIX processing only
			ACBJRACT	X'02'	JRNAD exit active
20	14	1	ACBDOSID		DOS DTF identifier
			ACBDTFID	X'28'	DTF type for VSAM
21	15	1	ACBOFLGS		Open/close flags
			ACBVOLMT	X'80'	Verify volume mounted
			ACBVMSG	X'40'	Message requested bit
			ACBEOV	X'20'	EOV detects completed
			ACBOPEN	X'10'	ACB is open
			ACBCAT	X'08'	ACB for VSAM catalog
			ACBEXFG	X'04'	User exit flag
			ACBSUB	X'02'	ACB is suballocated (is located in a control block allocation unit)
			ACBKEYOK	X'01'	Key processing all right for this ACB
22	16	1	ACBNST		Number of strings
22	16	1	ACBSTRNO		Number of strings
23	17	1	ACBERFLG		Error flags
				Open error return codes:	
			ACBOINCB	X'02'	Invalid control block structure
			ACBOALR	X'04'	This ACB is already open
			ACBOLLUB	X'0E'	The symbolic unit in the DLBL statement is invalid.
			ACBONJIB	X'0F'	No job information blocks (JIBs) are available from the label information cylinder.
			ACBOLIGN	X'11'	The address in the ASSGN statement for the logical unit was IGN (assignment ignored).
			ACBOLUNA	X'12'	The address in the ASSGN statement for the logical unit was UA (logical unit unassigned).
			ACBOAASF	X'13'	Unable to automatically assign a logical unit number
			ACBOIDSP	X'20'	The OPEN disposition specified for the file conflicts with other file characteristics
			ACBOCEXT	X'22'	The volume serial numbers specified in the EXTENT statement do not match those specified in the catalog entry.
			ACBONOAL	X'28'	No space available on any volume for primary allocation of a dynamic file
			ACBONANR	X'30'	An attempt was made to open a NOALLOCATION file which is not reusable

Figure 5.12 **Access Method Control Block (ACB) description and format (part 2 of 6)**

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
			ACBOCDLD	X'32'	Unable to load VSAM modules via a CDLOAD macro instruction.
			ACBONCIF	X'40'	An attempt was made to open a NOCIFORMAT file using VSAM (ACB) access
			ACBOSENS	X'41'	An attempt was made to open a SAM ESDS without the VSE/VSAM Space Management for SAM Feature installed
			ACBOIRCZ	X'42'	An attempt was made to open a DTF whose file characteristics do not match the file characteristics of the VSAM catalog
			ACBOUEXP	X'43'	An attempt was made to open an unexpired file for output using a DTF
			ACBODMOD	X'44'	The file to be opened has a name which begins with an invalid prefix
			ACBONSDS	X'45'	An attempt was made to open a nonSAM ESDS file using a DTF
			ACBOBNAM	X'46'	An invalid file-id was detected during implicit define or implicit delete.
			ACBORCSZ	X'47'	Allocation specifications for implicit define conflict with the file characteristics specified in the DTF and conversion to correct the conflict was unsuccessful
			ACBONALC	X'48'	The file-id specified in your DLBL statement was not found in the catalog and insufficient allocation information is specified for an implicit define.
			ACBOIDEL	X'4E'	A catalog management error was detected during implicit delete
			ACBOIDEF	X'4F'	A catalog management error was detected during implicit define.
			ACBONMNT	X'50'	Attempt to mount two volumes on the same drive when direct or keyed processing was specified. Or the operator failed to mount the volume.
			ACBONCRA	X'5C'	CRA volume not mounted
			ACBOIERR	X'60'	Unusable input data set
			ACBOUEMP	X'64'	Empty upgrade AIX
			ACBOTMST	X'68'	The time stamp of the volume on which a data set is stored doesn't match the system time stamp in the volume catalog entry.

Figure 5.12 **Access Method Control Block (ACB) description and format (part 3 of 6)**

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
			ACBOTIME	X'6C'	The system time stamps of a data set and its index do not match. This indicates that the data has been updated separately. This test is greater than or equal, i.e., no warning is given if the index time stamp is greater than the data time stamp.
			ACBOEMPT	X'6E'	Open empty data set for read only.
			ACBODSNC	X'74'	Data set was not closed the last time it was processed.
			ACBODEVT	X'75'	The symbolic unit specified in the EXTENT statements is not a valid VSAM device type.
			ACBONDLB	X'80'	The DLBL statement is missing or the filename in the DLBL doesn't match the ACB.
			ACBOIOER	X'84'	A permanent I/O error occurred while VSAM was reading label information from the label information cylinder.
			ACBONVRT	X'88'	Not enough virtual storage space is available in the partition for work areas, control blocks, or buffers.
			ACBOIOCA	X'90'	A permanent I/O error occurred while VSAM was reading or writing a catalog entry.
			ACBONCAT	X'94'	No entry was found in the catalog for this ACB.
			ACBOSECU	X'98'	Security verification failed; the password specified in the ACB for a specific level of access doesn't match the password in the catalog for that level of access.
			ACBOPARC	X'A0'	The operands specified in the ACB are inconsistent with each other or with the information in the catalog entry, for example, an open of an ESDS for keyed processing.
			ACBOKBUF	X'A1'	User-specified buffers with keyed access (user buffers can be specified only with CNV access).
			ACBOLIOE	X'A5'	A permanent I/O error was detected on the system lock file.
			ACBOLTEX	X'A6'	The system lock table is not large enough to accommodate the concurrent requests.
			ACBOLFEX	X'A7'	The system lock file is not large enough to accommodate the concurrent requests.
			ACBONAVA	X'A8'	The data set is not available because it is being updated by (under the exclusive control of) another ACB or has been exported by Access Method Services.

Figure 5.12 **Access Method Control Block (ACB) description and format (part 4 of 6)**

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
			ACBONOCT	X'B4'	The VSAM catalog is not connected to the system on logical unit SYSCAT, or insufficient virtual storage available for OPEN.
			ACBOACT	X'BC'	ACB was active
			ACBOOERR	X'C0'	Unusable output data set
			ACBOPEMP	X'C4'	Access via empty path
			ACBOLEMP	X'D4'	LSR is specified but the data set being opened is empty (which implies that it is to be loaded)
			ACBOLKEY	X'D8'	LSR is specified but the key length of the data set being opened is greater than the maximum key length specified for the resource pool.
			ACBOLBUF	X'DC'	LSR is specified but the CI size of the data set being opened is greater than the largest buffer size specified for the resource pool.
			ACBOLNRP	X'E4'	LSR is specified but there is no resource pool defined; may also be caused by problems while loading the resource.
			ACBONRST	X'E8'	Non-reusable file is not empty
			ACBOILAB	X'F8'	IKQLAB internal error
			ACBOLUNX	X'FE'	OPEN detected an unexpected return code from the lock manager.
			ACBOCTER	X'FF'	Unexpected return from catalog locate function.
				Close error return codes	
			ACBOINCB	X'02'	Invalid control block, or ACB address not in OAL
			ACBCALR	X'04'	ACB already closed
			ACBCDSFL	X'4C'	CLOSE disposition failed
			ACBCNVRT	X'88'	Insufficient space available in user's partition for work area.
			ACBCIOCA	X'90'	Permanent I/O error occurred while VSAM was reading or writing a catalog entry.
			ACBCNCAT	X'94'	No catalog entry found
			ACBCIOER	X'B8'	Permanent I/O error occurred while VSAM was completing outstanding I/O requests.
			ACBCBUSY	X'BC'	ACB busy.
			ACBCDTFA	X'FC'	Automatic close of the DTF for a managed-SAM file failed.

Figure 5.12 Access Method Control Block (ACB) description and format (part 5 of 6)

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
24	18	4	ACBAMBUF		Length of buffer pool
28	1C	8	ACBDDNM		DDname
36	24	4	ACBPRTCT		Pointer to password
40	28	4	ACBUAPTR		Pointer to user work area, or to CAXWA if ACB is for a catalog
44	2C	4	ACBBFPL		Pointer to first data buffer in buffer pool
48	30	4	ACBEXLST		User exit list pointer
52	34	4	ACBBPLB		BAM parameter list pointer
56	38	1			Reserved
57	39	1	ACBOFLG2		Second OPEN/CLOSE flag byte
				X'80'	Reserved
				X'40'	Reserved
				X'20'	Reserved
				X'10'	Reserved
			ACBKEEP	X'08'	Close disposition is KEEP
			ACBDELET	X'04'	Close disposition is DELETE
			ACBDATE	X'02'	Close disposition is controlled by the expiration date
				X'01'	Reserved
58	3A	2	ACBMSGLN		Message area length
60	3C	4	ACBMSGAR		Message area
64	40	4	ACBNMPTR		Pointer to 44 character name
68	44	0	ACBEND		End of ACB

Figure 5.12 **Access Method Control Block (ACB) description and format (part 6 of 6)**

Access Method Data Statistics Block (AMDSB)

The AMDSB contains statistical information about record processing in the data set. It also contains some of the data set's attributes and specifications. The AMDSB is built, using the data index or index catalog record's AMDSB group occurrence, when the cluster is opened. The AMBL (AMBLDTA/AMBLIX) points to the data and index AMDSBs.

Offset				Hex.	
Dec	**Hex**	**Bytes**	**Field Name**	**Digit**	**Description**
				General	
0	0		AMDCOMM		Common part
0	0	1	AMDSBID	X'60'	AMDSB identifier
1	1	1	AMDATTR		Attributes of the data set
			AMDATTR1		Attributes (first byte):
			AMDDST	X'80'	Key-sequenced data set
			AMDWCK	X'40'	Check each record when it is written
			AMDSDT	X'20'	Sequence set is stored with the data
			AMDREPL	X'10'	Replication
			AMDORDER	X'08'	Use the volumes in the same order as in the volume list
			AMDRANGE	X'04'	The data set is divided into key ranges
			AMDRRDS	X'02'	Relative record data set
			AMDSPAN	X'01'	Spanned records
2	2	2	AMDLEN		Length of AMDSB in the catalog
4	4	2	AMDNEST		Number of entries in an index section (in all cases except AMDSB group occurrence in data record of AIX)[1]
4	4	2	AMDAXRKP		Relative key position in base record (only for AMDSB group occurrence in data record of a AIX)[1]
6	6	2	AMDRKP		Relative key position
8	8	2	AMDKEYLN		Key length
10	A	1	AMDPCTCA		Percentage of free control intervals in the control area
10	A	1	AMDRCFRM		SAM ESDS record format information
			AMDIMPDF	X'80'	File definition was by implicit define
			AMDNCIFT	X'20'	Non-control-interval format (processable by SAM only)
			AMDNCAFT	X'10'	Non-control-area format (This bit indicates the file is a SAM ESDS. If both this bit and AMDNCIFT are off, the file is a VSAM ESDS.)
			AMDSBLKD	X'04'	The SAM record format is blocked.
			AMDSVAR	X'02'	The SAM record format is variable length records.
			AMDSFIXD	X'01'	The SAM record format is fixed length records.

Figure 5.13 **Access Method Data Statistics Block (AMDSB) description and format (part 1 of 5)**

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
11	B	1	AMDPCTCI		Percentage of free bytes in the control interval
12	C	2	AMDCIPCA		Number of control intervals in a control area
14	E	2	AMDFSCA		Number of free control intervals in a control area
16	10	4	AMDFSCI		Number of free bytes in a control interval
20	14	4	AMDCINV		Control interval size
24	18	4	AMDLRECL		Maximum record size. For a SAM ESDS, this is the maximum SAM logical blocksize.
28	1C	4	AMDHLRBA		RBA of the high-level index record
28	1C	4	AMDNSLOT		Number of relative record slots
29	1C	4	AMDBLREC		SAM LRECL for a fixed-blocked SAM ESDS
32	20	4	AMDSSRBA		RBA of first sequence set record
32	20	4	AMDMAXRR		Max. relative record number
36	24	4	AMDPARDB		Pointer to first ARDB
40	28	1	AMDATTR3		Attributes
			AMDUNQ	X'80'	Non-unique keys in AIX
				X'00'	Unique keys in AIX
41	29	3			Reserved
44	2C	4			Reserved
				Statistics	
48	30		AMDSTAT		Statistics
48	30	8	AMDSTMST		System time stamp
48	30	8	AMDSTSP		System time stamp
56	38		AMDSTAT1		
56	38	2	AMDNIL		Number of index levels
58	3A	2	AMDNEDB		Number of EDBs
58	3A	2	AMDNEXT		Number of extents in the data set
60	3C	4	AMDNLR		Number of user-supplied (logical) records in the data set
64	40	4	AMDDELR		Number of deleted records
68	44	4	AMDIREC		Number of inserted records
72	48	4	AMDUPR		Number of updated records
76	4C	4	AMDRETR		Number of retrieved records
80	50	4	AMDASPA		Number of bytes of free space in the data set
84	54	4	AMDNCIS		Number of times a control interval was split
88	58	4	AMDNCAS		Number of times a control area was split
92	5C	4	AMDEXCP		Number of times EXCP was issued by VSAM I/O routines

Figure 5.13 **Access Method Data Statistics Block (AMDSB) description and format (part 2 of 5)**

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
				General Continue	
96	60	1	AMDSHOPT		Share option byte
			AMDSHR1	X'80'	Share option 1
			AMDSHR2	X'40'	Share option 2
			AMDSHR3	X'20'	Share option 3
			AMDSHR4	X'10'	Share option 4
97	61	4	AMDCDSN		Pointer to catalog ACB
101	65	3	AMDDSN		Catalog control interval number for data (index)
104	68	4	AMDHWRBA		High-water RBA for the data set
108	6C	1	AMDATTR2		Attributes (second byte):
			AMDREL	X'80'	Release unused space
			AMDLOAD	X'40'	Load mode
			AMDSPEED	X'20'	Speed option
			AMDINDX	X'10'	Index option
			AMDSHR	X'08'	Sharing
			AMDKR	X'04'	Key-range processing, duplicate of AMDRANGE
			AMDMXARC	X'02'	This component contains both fixed block and CKD files (set only when a mixed architecture index opens itself).
			AMDCAT	X'01'	AMDSB for catalog
109	6D	1	AMDACT		AMDSB test and set byte
110	6E	2	AMDFILT		User area (ISAM compatibility)
112	70	4	AMDPVOL		Pointer to volume list
116	74	1	AMDAMS		AMS flag byte
			AMDAIX	X'80'	Alternate index
			AMDPATH	X'40'	Access via path
			AMDBASE	X'20'	Access via base
117	75	1	AMDATTR4		Attributes (fourth byte):
			AMDARCH	X'80'	Data component: component resides on a fixed block device. Index component: high-level index is on a fixed block device.
			AMDARCHS	X'40'	Sequence set resides on a fixed block device (index component only).
			AMDRCHAN	X'20'	Relocating channel. IJBRCHAN (defined in the MAPSSID mapping macro of the supervisor) is on in the Subsystem Identification Block with the name SUPƀ. It is necessary to build a Fix List for I/O.
118	76	2	AMDAIRKP		Relative key position in base record (only in data AMDSB of AIX)[1]

[1] For more details of these fields, see the explanation of the AMDSB group occurrence.

Figure 5.13 **Access Method Data Statistics Block (AMDSB) description and format (part 3 of 5)**

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
				Local Statistics	
120	78		AMDLSTAT		Local statistics
120	78	2	AMDLNIL		Local number of index levels
122	7A	2	AMDLNEST		Local number of entries in the index section
124	7C	4	AMDLNLR		Local number of user-supplied (logical) records
128	80	4	AMDLDELR		Local number of deleted records
132	84	4	AMDLIREC		Local number of inserted records
136	88	4	AMDLUPR		Local number of updated records
140	8C	4	AMDLRETR		Local number of retrieved records
144	90	4	AMDLASPA		Local bytes of free space
148	94	4	AMDLNCIS		Local number of control interval splits
152	98	4	AMDLNCAS		Local number of control area splits
156	9C	4	AMDLEXCP		Local number of EXCPs issued by VSAM I/O routines
				Exception Exit	
160	A0	8	AMDEXEXT		Exception exit
				Buffer Management Information	
168	A8	2	AMDBCBNO		Number of buffers
170	AA	2	AMDBFREE		Number of unassigned buffers
172	AC	4	AMDFSBCB		Address of the first BCB (for LSR: address of the BSPH)
176	B0	4	AMDFFBCB		Address of the first free BCB
180	B4	4	AMDCCWA		Pointer to BKPHD, which is the first control block in the FCDB area. The rest of the FCDB area is divided into 64-byte FCDBs, which are suballocated as needed for CCB(s), CCW(s), FXL(s), and IOARG(s).
184	B8	4	AMDHERBA		High RBA of extent currently being processed for a SAM ESDS (Same value as EDBHIRBA)
188	BC	2	AMDCIMLT		CI multiplier, specifies the number of CIs that are to be considered as a CA in certain parts of Record Management processing. For a SAM ESDS, it has a value of one; otherwise, it has the same value as AMDCIPCA.

Figure 5.13 **Access Method Data Statistics Block (AMDSB) description and format (part 4 of 5)**

Offset					
Dec	**Hex**	**Bytes**	**Field Name**	**Hex. Digit**	**Description**
190	BE	1	AMDRCFM1		Same as AMDRCFRM; zero if not a SAM ESDS.
			AMDIMPD1	X'80'	Same as AMDIMPF.
			AMDNCIF1	X'20'	Same as AMDNCIFT
			AMDNCAF1	X'10'	Same as AMDNCAFT
			AMDSBLK1	X'04'	Same as AMDSBLKD
			AMDSVAR1	X'02'	Same as AMDSVAR
			AMDSFIX1	X'01'	Same as AMDSFIXD
191	BF	1			Reserved
				EDB Header	
192	C0	4	AMDFSEDB		Address of first EDB
196	C4	2			Reserved
198	C6	2	AMDLEDB		Length of EDB

Figure 5.13 **Access Method Data Statistics Block (AMDSB) description and format (part 5 of 5)**

Address Range Definition Block (ARDB)

The ARDB contains information about space allocated and space actually used by a data set. The block is built by the Open module from information in the data set's catalog record. The ARDB is updated by record management routines as additional space is used. The first ARDB in an ARDB chain is pointed to by the AMDSB (AMDPARDB).

Offset				Hex.	
Dec	**Hex**	**Bytes**	**Field Name**	**Digit**	**Description**
0	0	1	ARDID	X'AD'	Control block identifier
1	1	1	ARDTYPE		Identifies the type of space defined by the ARDB:
			ARDKR	X'80'	One key range of a key-range data set
			ARDHLI	X'40'	The total index of a key-sequenced data set that does not have the sequence set with the data or The non-sequence set levels of a key-sequenced data set's index, when the sequence set is stored with the data
			ARDSS	X'20'	The sequence set of a key-sequenced data set, when the sequence set is stored with data
			ARDUOVFL	X'10'	Use overflow volumes for this key
			ARDEOD	X'08'	End of data ARDB
			ARDLGCC	X'04'	Device contains more than 256 cylinders

Figure 5.14 **Address Range Definition Block (ARDB) description and format (part 1 of 2)**

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
2	2	2	ARDLEN		Length of the ARDB
4	4	1	ARDPRF		Address range definition preformat byte (this byte is a literal copy of the catalog record byte called ITYPEXT)
			ARDPRFMT	X'80'	Sequence set with data
				X'40'	No preformat done
5	5	3			Reserved
8	8	4	ARDNPTR		Address of the next ARDB in the ARDB chain
12	C	4	ARDHRBA		The RBA of the next free-space control interval at the end of the data set (RBA of VSAM SEOF)
16	10	4	ARDEDBA		Pointer to the active EDB
20	14	4	ARDPREL		Pointer to related ARDB (index)
24	18	4	ARDERBA		The RBA of the highest control interval allocated to the key range
28	1C	4	ARDPKEYS		Pointer to ARDKEYS
32	20	4	ARDHKRBA		The RBA of the data set control interval containing the key range's high-key value
36	24	2	ARDVOLNM		Number of volumes in list
The following ten-byte entry, called an ARDB volume group, repeats for each volume in this ARDB.					
38	26	10	ARDVOLGP		Volume serial (VOLSER) list
38	26	6	ARDVOLSR		The serial number of the volume containing the highest RBA allocated to the key range
44	2C	2	ARDRELRP		Catalog relative replication number
46	2E	2	ARDSYMU		Symbolic unit
46	2E	1	ARDSUCLS		Symbolic unit class
47	2F	1	ARDSUNUM		Symbolic unit number
Variable (after last volume group)		Variable	ARDKEYS		Space reserved for the key range's low and high key values. The length of this field equals twice the key length. Pointed to by ARDPKEYS.

Figure 5.14 **Address Range Definition Block (ARDB) description and format (part 2 of 2)**

Buffer Control Block (BCB)

The BCB consists of a buffer control entry that describes each buffer requested by the user and each buffer required for preformat processing. Each buffer control entry contains function codes, status indicators, and RBAs to describe the buffer. The buffer control entry also contains the address of the data buffer, the associated channel program built in the CCWAREA, and the next BCB in the chain. The buffer control entry is created by Open and released by Close. The BCB is the interface between the I/O Manager and the Buffer Manager modules. The BCB is pointed to by the PLH (PLHDBCB points to the data BCB and PHLXBCB points to the index BCB). IKQIODRB (see I/O Driver Block) maps the BUFRIODR and BUFWIODR fields.

Offset				Hex.	
Dec	**Hex**	**Bytes**	**Field Name**	**Digit**	**Description**
0	0	4	BUFNBCB		Address of the next BCB entry
			BUFCHAIN		Offset to chain pointer (equate value)
4	4	4	BUFCBAD		Buffer address
8	8	20	BUFRIODR		Read I/O driver block (see IODRB)
8	8	2	BUFCURRU		Read symbolic unit number
10	A	2	BUFBKSTR		Number of physical blocks to read
12	C	4	BUFRSTBB		Starting block of the CA to be read (fixed block device)
12	C	8	BUFRSEEK		Computed DASD address for read
12	C	1	BUFRM		M (Contains X'80' for an RPS device)
			BUFRDSK		Offset to read MBBCCHHR (equate value)
13	D	2	BUFRBB		BB
15	F	2	BUFRCC		CC
16	10	4	BUFRBBBB		Displacement to the first block in the CA to be read (fixed block device)
17	11	2	BUFRHH		HH
19	13	1	BUFRR		R
20	14	4	BUFCRRBA		RBA for the read
24	18	4	BUFRLPMB		Address of the read LPMB
28	1C	20	BUFWIODR		Write I/O driver block (see IODRB)
28	1C	2	BUFCURWU		Write symbolic unit number
30	1E	10	BUFCKIN		Write initialize area
30	1E	2	BUFBKSTW		Number of physical blocks to write
32	20	4	BUFWSTBB		Starting block of the CA to be written (fixed block device)

Figure 5.15 **Buffer Control Block (BCB) description and format (part 1 of 3)**

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
32	20	8	BUFWSEEK		Computed DASD address for write
32	20	1	BUFWM		M (Contains X'80' for an RPS device)
			BUFWTSK		Offset to write MBBCCHHR (equate value)
33	21	2	BUFWBB		BB
35	23	2	BUFWCC		CC
36	24	4	BUFWBBBB		Displacement to the first block in the CA to be written (fixed block device)
37	25	2	BUFWHH		HH
39	27	1	BUFWR		R
40	28	4	BUFCWRBA		RBA for the write
44	2C	4	BUFWLPMB		Address of the write LPMB
48	30	2	BUFFLAG		Flag bytes
48	30	1	BUFFLAG1		Flag byte 1
			BUFALLF1	X'FF'	BUFIOS + BUFRDAHD + BUFCVAL + BUFSSRCD + BUFRES1
			BUFIOS	X'E8'	BUFCMW + BUFCFMT + BUFCRRD + BUFPFMT
			BUFCMW	X'80'	Write indicator
			BUFCFMT	X'40'	Format writer indicator
			BUFCRRD	X'20'	Read indicator
			BUFWRIGN	X'10'	Write ignored; retry because either the wrong I/O manager was entered or the write operation was not performed due to an I/O error on another buffer. (BUFCMW temporarily turned off.)
			BUFPFMT	X'08'	Format remainder of control area
			BUFCVAL	X'04'	Buffer contents are valid
			BUFSSRCD	X'02'	Buffer is a sequence set record
			BUFRDIGN	X'01'	Read ignored; retry because either the wrong I/O manager was entered or the read operation was not performed due to an I/O error on another buffer. (BUFCRRD temporarily turned off.)
49	31	1	BUFFLAG2		Flag byte 2
			BUFALL2	X'FF'	BUFPURG1 + BUFPURG2 + BUFRIXRD + BUFWRINV + BUFFREP + BUFRDAHD + BUFRES2
			BUFPURG1	X'80'	Purge - must write or read
			BUFPURG2	X'40'	Purge - format
			BUFRIXRD	X'20'	Replicated index read
			BUFWRINV	X'10'	Control interval was written - another string
				X'08'	Reserved
			BUFRDAHD	X'04'	Read ahead request
			BUFRES2	X'03'	Reserved

Figure 5.15 **Buffer Control Block (BCB) description and format (part 2 of 3)**

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
50	32	10	BUFBKTWI		Write check initialize area
50	32	2	BUFBKTCK		Number of physical blocks to check
52	34	8	BUFWCKSK		Computed DASD address for check
52	34	1	BUFCM		M (Contains X'80' for an RPS device)
53	35	2	BUFCBB		BB
55	37	2	BUFCCC		CC
57	39	2	BUFCHH		HH
59	3B	1	BUFCR		R
60	3C	4	BUFVCCHH		CCHH for replicated index read
60	3C	4	BUFVCCB		CCB address
64	40	1	BUFERFLG		I/O error indicator
			BUFERALL	X'83'	BUFEIOER + BUFENTCM + BUFBDSK
			BUFEIOER	X'80'	I/O error on this buffer
				X'40'	Used by IKQIOB (see bit IOMERP2 in IOWKA)
			BUFENTCM	X'02'	Buffer operation not complete
			BUFBDSK	X'01'	2314 seek incorrect
65	41	1	BUFSTRID		String ID of this set of buffers
66	42	2	BUFCNOI		No. of blocks in control interval to process
68	44	4	BUFNABCB		Next BCB in AMDSB chain
			BCB Extension for Local Shared Resources		
72	48	4	BUFMDBTS		Modification mask (one bit per transaction - refer to BSPH)
76	4C	4	BUFUCHUP		Address of previous BCB in chain
80	50	4	BUFUCHDN		Address of next BCB in chain
84	54	4	BUFBSPH		Address of BSPH
88	58	2			Reserved
90	5A	1	BUFFLAG3		Reserved
91	5B	1	BUFUSE		Buffer use byte (X'FF'=in use)
92	5C	7	BUFHDSID		Catalog ACB address (4 bytes) and CI number (3 bytes) of the catalog record for this index or data component
99	63	1			Reserved
100	64	4	BUFAMDSB		Pointer to AMDSB
104	68	4	BUFACB		Pointer to the ACB

Figure 5.15 **Buffer Control (BCB) description and format (part 3 of 3)**

Block Pool Header (BKPHD)

The BKPHD is the first control block in the FCDB area. The rest of the FCDB area is divided into 64-byte FCDBs, which are suballocated as needed for CCW(s), CCB(s), FXL(s), and IOARG(s). The BKPHD points to the first unallocated FCDB and is pointed to by the AMDSB. It is built by IKQOPN and can be extended by IKQIOA/IKQIOC.

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
0	0	2	BKPLENG		Length of the pool of blocks
2	2	2			Available
4	4		BKPHDECB		Control allocation of blocks
4	4	2			Not used
6	6	1	BKPHDCOM		Communications byte
			BKPHWAIT	X'80'	Wait flag
7	7	1	BKPHDTS		Allocation test and set byte
8	8	32	BKPHRSAV		Space for saving registers during steal BCB
8	8	4	BKPHRS13		Save register 13, (original PLH)
12	C	4	BKPHRS14		Save register 14
16	10	4	BKPHRS15		Save register 15 during "buffer steal" (overlaid by R9)
20	14	4	BKPHRS00		Save register 0
24	18	4	BKPHRS01		Save register 1
28	1C	4			Save register 2
32	20	4			Save register 3
36	24	4			Save register 4
40	28	4	BKPHDBHD		Save data buffer header during steal BCB
44	2C	4	BKPHIBHD		Save index buffer header during steal BCB
48	30	4	BKPSPCHN		Address of next area of blocks
52	34	4	BKPERCCB		Address of error CCB (first error CCB in VSAM error queue)
56	38	4	BKPFSTBK		Address of first available block
60	3C	4	BKPSTECB		ECB - steal BCB, other string
60	3C	2			Available
62	3E	1	BKPSTCOM		Communications byte
			BKPSWAIT	X'80'	Wait flag
63	3F	1	BKPSTTS		Test and set byte
		Equate values			
			BKPSIZE	X'800'	Size of space to extend pool (2048)
			BKPBLKSZ	X'40'	Size of a block (64)
			BKPNOBKS	X'20'	Number of blocks in new space (32)

Figure 5.16 **Block Pool Header (BKPHD) description and format**

Buffer Header (BHD)

The BHD contains information about buffers and buffer processing. The PLH points to the data and index BHDs. The BHD is created by Open and released by Close.

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
0	0	2	BHDNO		Number of buffers.
2	2	2	BHDLEN		Length of control block.
4	4	2	BHDRMAX		Maximum number of buffers available.
6	6	2	BHDRMIN		Minimum number of buffers available.
8	8	2	BHDBRC		Read-ahead count.
10	A	1	BHDHFLAG		Buffer header flag 1:
			BHDRAHOK	X'80'	Read-ahead OK flag.
			BHDIXREP	X'40'	Replicated index read indicator.
			BHDNSKD	X'08'	I/O with wait for no-schedule queue (BCBNSKDQ).
			BHDSKD	X'04'	I/O with wait for schedule queue (BCBSKDQ).
			BHDSTL	X'02'	A "buffer steal" has been performed on this BHD.
11	B	1	BHDFLAG		Buffer header flag 2.
			BHDMVBCB	X'02'	"Free buffer" is really a move.
12	C	4			Reserved
16	10	4	BHD1STF		Address of chain of free buffers.
20	14	4	BHDSKDQ		Address of BCB chain with I/O scheduled.
24	18	4	BHDNSKDQ		Address of BCB chain with pending I/O.
18	1C	4	BHD1STW		Address of first BCB requiring I/O.
32	20	1	BHDID	X'77'	BHD identification.
33	21	1			Reserved.
34	22	2	BHDIOCNT		I/O count of no-schedule queue (BCBNSKDQ).
36	24	2	BHDWMIN		Write threshold.
38	26	2	BHDTRACT		Temporary read-ahead count.
40	28	2	BHDQNO		Number of BCBs on queues.
42	2A	2			Reserved.
44	2C	4	BHDCCHH		CCHH of last held control area.
48	30	4	BHDCCBCH		CCB chain pointer.

Figure 5.17 **Buffer Header (BHD) description and format**

Buffer Subpool Header (BSPH)

The Buffer Sub-Pool Header contains information concerning a buffer subpool. The BSPHs are chained together in a sequence of ascending buffer sizes. AMDFSBCB points to the BSPH associated with a particular data set component. The address of the first BSPH is stored in the VSRT.

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
0	0	1	BSPHID	X'72'	Control block identifier
1	1	1			Reserved
2	2	2	BSPHLEN		Length of BSPH
4	4	4	BSPHNM		Name 'BSPH'
8	8	4	BSPHNBSP		Pointer to next BSPH in pool
12	C	2	BSPHBFNO		Number of buffers in this subpool
14	E	2			Reserved
16	10	4	BSPHMDBN		Number of modified buffers in this subpool
20	14	4	BSPHFRBN		Number of free buffers in this subpool
24	18	4	BSPHBCB		Address of first BCB in the subpool
28	1C	4	BSPHMDBT		32-bit modification mask. Each bit corresponds to a transaction which has modified the buffer
32	20	4	BSPHBSZ		Length of each buffer in this subpool
36	24	4	BSPHCPLH		Address of the PLH currently in control of the BSPH
40	28	16			Reserved*
56	38	4	BSPHUTOP		Pointer to the top of the use chain
60	3C	4	BSPHUBTM		Pointer to the bottom of the use chain*
64	40	4	BSPH1ST		Address of the first BSPH in the buffer pool
68	44	2	BSPHECB		Control bytes for changing use chain
70	46	1	BSPHCOM BSPHWAIT	 X'80'	Communications byte Wait flag
71	47	1	BSPHTS		Mask byte for test and set

* The use chain is a chain of all BCBs in the subpool. The least recently used BCB is at the bottom of the chain and the most recently used BCB is at the top.

Figure 5.18 Buffer Subpool Header (BSPH) description and format

Channel Command Word (CCW)

Record management uses the CCW macro to map a CCW slot within an FCDB for building CCW strings.

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
0	0	8	CCWAREA		Length of one CCW
0	0	1	CCWOP		Operation code
1	1	3	CCWARG		Argument address
4	4	1	CCWFLAG		CCW flags
			CCWDCH	X'80'	Data chaining
			CCWCCH	X'40'	Command chaining
			CCWSLI	X'20'	Suppress incorrect length
			CCWSKIP	X'10'	Skip data transfer
5	5	1			Reserved
6	6	2	CCWCNT		Count field
8	8	1	CCWOP1		Next CCW operation code
			CCWSRHE	X'31'	Search ID equal
			CCWSSEC	X'23'	Set sector
			CCWWTCKD	X'1D'	Write count key data
			CCWSKHD	X'1B'	Seek head
			CCWRDC	X'12'	Read count
			CCWTIC	X'08'	TIC
			CCWSEEK	X'07'	Full seek
			CCWRD	X'06'	Read data
			CCWWT	X'05'	Write data
			CCWNOP	X'03'	NOP

Figure 5.19 Channel Command Word (CCW) description and format

CCW Skeleton DSECT (IKQCWS)

The I/O manager (IKQIOA/IKQIOC) uses the IKQCWS DSECT to map the CCW skeletons used for building CCW strings.

Offset				Hex.	
Dec	**Hex**	**Bytes**	**Field Name**	**Digit**	**Description**
0	0	1	CWSOFSET		Offset of CCWs together
			CWSFLAG		CCW skeleton flag byte 1
			CWSPLHAD	X'80'	BUFVCCHH in the BCB is used as the I/O area for the Read Count CCW.
			CWSIVLR	X'40'	The R field (WKAR) in the PLHWAREA is set to X'FF' (or X'00' for a Write Data 2 CCW) and forces a Search ID Equal - TIC CCW sequence to be placed in the CCW chain during the next call to the Build CCW (BDCCW) routine.
			CWSBFADC	X'20'	The address of the buffer is used as the I/O address argument for the CCW. The physical block size in the LPMB is used as the CCW count field. The number of blocks to process is decremented by one. BUFCNOI, in the BCB, controls assignment of the correct location in the buffer to be used as the CCW I/O address when the CI spans more than one physical block.
			CWSARGAD	X'10'	The I/O arguments in the I/O Data Block are used as CCW I/O address arguments.
			CWSASTER	X'08'	The address of the previous CCW is placed in the current CCW I/O address (used only by the TIC CCW). The TIC op code is set to X'07' and the I/O address is set to -8 in the CCW skeleton. Adding the address of the TIC CCW in the CCW block to this field causes the resulting address to be pointing to the previous CCW; the overflow converts the op code to the correct value of X'08'.
			CWSINCR	X'04'	The increment R (update the DASD address) subroutine is invoked to update the CCHHR field in the BCB. (It is used for checking if the next CCW to be built is for the next contiguous physical block in the data set.) The R field (WKAR) in the PLHWAREA is also incremented by one.

Figure 5.20 **CCW Skeleton DSECT (IKQCWS) description and format (part 1 of 2)**

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
			CWSDECR	X'02'	The Search ID Equal I/O argument (R byte of the CCHHR) for the Write Count CCW is decremented by one. This only occurs when the CCW preceding the Write Count CCW is a TIC to the Search ID Equal CCW. (This TIC may be followed by a TIC to the Write Count CCW whenever the Write Count CCW starts a new CCW block, but it is ignored, and the Search ID Equal I/O argument is still decremented.) Succeeding Write Count CCWs do not decrement the R until an intervening Search ID Equal - TIC sequence occurs.
			CWSNOOPT	X'01'	Space is allocated in the I/O Data Block for use as the I/O address argument field. MBBCCHHRKDD is initialized (K=0).
2	2	1	CWSFLAGC		CCW skeleton flag byte 2:
			CCWSRPS	X'80'	The current I/O arguments in the I/O Data Block are located and updated with the device type for the SECTVAL (SVC 75) in the Build Channel Program routine (BLDCP). The blocksize was previously initialized by the Seek CCW.
2	2	1	CWSOPCOD		Op code specified in the Locate parameter field for read, read replicated, and write operations (defined under IOARG write-up).
3	3	8	CWSCCW		CCW to build
			Equate Value		
			CWSLENTH	X'0B'	One CCW string argument length

Figure 5.20 **CCW Skeleton DSECT (IKQCWS) description and format (part 2 of 2)**

CCW Skeletons

These are the I/O Manager (IKQIOA) CCW skeletons that CWS maps. They are used for building CCW strings. Figure 5.21 shows the CCW Skeletons description and format for CKD devices. Figure 5.22 shows the CCW Skeletons description and format for fixed block devices.

Offset				Hex.	
Dec	**Hex**	**Bytes**	**Field Name**	**Digit**	**Description**
		Head Seek CCW			
0	0	1	SCCWSK	X'08'	Length of contiguous CCWs (FCBCINC)
1	1	1		X'51'	CCW build flags (CWSIVLR + CWSARGAD + CWSNOOPT)
2	2	1		X'00'	CCW build flags
3	3	1		X'1B'	CCW opcode (CCWSKHD)
4	4	3		X'01'	Offset into ARG address
7	7	1		X'40'	Command chain flag (CCWCCH)
8	8	1		X'00'	Reserved
9	9	2		X'0006'	Count field
11	B	1		X'00'	End of chain indicator
		Search CCW			
12	C	1	SCCWCRH	X'10'	Length of continuous CCWs (2 x FCBCINC)
13	D	1		X'10'	CCW build flags (CWSARGAD)
14	E	1		X'00'	CCW build flags
15	F	1		X'31'	CCW op code (CCWSRHE)
16	10	3		X'000003'	Offset into ARG address
19	13	1		X'40'	CC flag
20	14	1		X'00'	Reserved
21	15	2		X'0005'	Count field
23	17	1		X'08'	Length of contiguous CCWs (FCBCINC)
24	18	1		X'08'	CCW build flags (CWSASTER)
25	19	1		X'00'	CCW build flags
26	1A	1		X'07'	CCW op code minus carry (CCWTIC - 1)
27	1B	3		X'FFFFF8'	Offset in CCW chain from here (minus 8)
30	1E	4		4X'00'	Reserved
34	22	1		X'00'	End of chain

Figure 5.21 **CCW Skeletons (CKD devices) description and format (part 1 of 3)**

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
		Format Write CCW and Write CCW			
35	23	1	SCCWFMTW	X'08'	Length of contiguous CCWs (FCBCINC)
36	24	1		X'13'	CCW build flags (CWSARGAD + CWSDECR + CWSNOOPT)
37	25	1		X'00'	CCW build flags
38	26	1		X'1D'	CCW op code (CCWWTCKD)
39	27	3		X'000003'	Offset into ARG address
42	2A	1		X'80'	Data chain flag (CCWDCH)
43	2B	1		X'00'	Reserved
44	2C	2		X'0008'	Count field
46	2E	1		X'08'	Length of continuous CCWs (FCBCINC)
47	2F	1		X'24'	CCW build flags (CWSBFADC + CWSINCR)
48	30	1		X'00'	CCW build flags
49	31	1		X'05'	CCW op code (CCWWT)
50	32	3		X'000000'	Offset into buffer address
53	35	1		X'40'	Command chain flag (CCWCCH)
54	36	1		X'00'	Reserved
55	37	2		X'0000'	Count field
57	39	1		X'00'	End of chain
		Write CCW			
58	3A	1	SCCWWT	X'08'	Length of contiguous CCWs (FCBCINC)
59	3B	1		X'64'	CCW build flags (CWSBFADC + CWSINCR + CWSIVLR)
60	3C	1		X'00'	CCW build flags
61	3D	1		X'05'	CCW op code (CCWWT)
62	3E	3		X'000000'	Offset into buffer address
65	41	1		X'40'	Command chain flag (CCWCCH)
66	42	1		X'00'	Reserved
67	43	2		X'0000'	Count field
69	45	1		X'00'	End of chain
		Index Read CCW and Read CCW			
70	46	1	SCCWIXRD	X'08'	Length of contiguous CCWs (FCBCINC)
71	47	1		X'80'	CCW build flags (CWSPLHAD)
72	48	1		X'00'	CCW build flags
73	49	1		X'12'	CCW op code (CCWRDC)
74	4A	3		X'000000'	Offset into PLH address
77	4D	1		X'60'	SLI and command chain flag (CCWCCH + CCWSLI)
78	4E	1		X'00'	Reserved
79	4F	2		X'0004'	Count field
81	51	1	SCCWRD	X'08'	Length of contiguous CCWs (FCBCINC)
82	52	1		X'24'	CCW build flags (CWSBFADC + CWSINCR)
83	53	1		X'00'	CCW build flags
84	54	1		X'06'	CCW op code (CCWRD)

Figure 5.21 **CCW Skeletons (CKD devices) description and format (part 2 of 3)**

Offset				Hex.	
Dec	**Hex**	**Bytes**	**Field Name**	**Digit**	**Description**
85	55	3		X'000000'	Offset into buffer address
88	58	1		X'40'	Command chain flag (CCWCCH)
89	59	1		X'00'	Reserved
90	5A	2		X'0000'	Count field
92	5C	1		X'00'	End of chain
		Read Back Check CCW			
93	5D	1	SCCWRBCK	X'08'	Length of contiguous CCWs (FCBCINC)
94	5E	1		X'24'	CCW build flags (CWSBFADC + CWSINCR)
95	5F	1		X'00'	CCW build flags
96	60	1		X'06'	CCW op code (CCWRD)
97	61	3		X'000000'	Offset into buffer address
100	64	1		X'50'	Skip and command chain flags (CCWCCH + CCWSKIP)
101	65	1		X'00'	Reserved
102	66	2		X'0000'	Count field
104	68	1		X'00'	End of chain
		RPS Seek Head and Set Sector CCW			
105	69	1	SCCWRPS	X'08'	Length of contiguous CCWs (FCBCINC)
106	6A	1		X'51'	CCW build flags (CWSIVLR + CWSARGAD + CWSNOOPT)
107	6B	1		X'00'	CCW build flags
108	6C	1		X'1B'	CCW op code (CCWSKHD)
109	6D	3		X'000001'	Offset into ARG address
112	70	1		X'40'	Command chain flag (CCWCCH)
113	71	1		X'00'	Reserved
114	72	2		X'0006'	Count field
116	74	1		X'08'	Length of contiguous CCWs (FCBCINC)
117	75	1		X'00'	CCW build flag
118	76	1		X'80'	CCW build flag (CCWSRPS)
119	77	1		X'23'	CCW op code (CCWSSEC)
120	78	3		X'00000B'	Offset into ARG address
123	7B	1		X'40'	Command chain flag (CCWCCH)
124	7C	1		X'00'	Reserved
125	7D	2		X'0001'	Count field
127	7F	1		X'00'	End of chain

Figure 5.21 CCW Skeletons (CKD devices) description and format (part 3 of 3)

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
		Define Extent CCW			
0	0	1	SCCWDEX	X'08'	Length of contiguous CCWs (FCBCINC)
1	1	1		X'10'	CCW build flags (CWSARGAD)
2	2	1		X'00'	CCW build flags
3	3	1		X'63'	CCW op code (CCWDEX)
4	4	3		X'000000'	Offset into argument address
7	7	1		X'40'	Command chaining flag (CCWCCH)
8	8	1		X'00'	Reserved
9	9	2		X'0010'	Count field
11	B	1		X'00'	End of chain
		Locate CCW			
12	C	1	SCCWLOC	X'08'	Length of continguous CCWs (FCBCINC)
13	D	1		X'10'	CCW build flags (CWSARGAD)
14	E	1		X'00'	CCW build flags
15	F	1		X'43'	CCW op code (CCWLOC)
16	10	3		X'000000	Offset into argument address
19	13	1		X'40'	Command chain flag (CCWCCH)
20	14	1		X'00'	Reserved
21	15	2		X'0008'	Count field
23	17	1		X'00'	End of chain
		Write CCW			
24	18	1	SCCWWT	X'08'	Length of contiguous CCWs (FCBCINC)
25	19	1		X'20'	CCW build flags (CWSBFADC)
26	1A	1		X'01'	Locate parameters op code (IOAWRT)
27	1B	1		X'41'	CCW op code (CCWWRT)
28	1C	3		X'000000'	Offset into buffer address
31	1F	1		X'C0'	Command chain and data chain flags (CCWCCH + CCWDCH)
32	20	1		X'00'	Reserved
33	21	2		X'0000'	Count field
35	23	1		X'00'	End of chain
		Index Read CCW and Read CCW			
36	24	1	SCCWRD	X'08'	Length of contiguous CCWs (FCBCINC)
37	25	1		X'20'	CCW build flags (CWSBFADC)
38	26	1		X'06'	Locate parameters op code (IOARD)
39	27	1		X'42'	CCW op code (CCWREAD)
40	28	3		X'000000'	Offset into buffer address
43	2B	1		X'C0'	Command chain and data chain flags (CCWCCH + CCWDCH)
44	2C	1		X'00'	Reserved
45	2D	2		X'0000'	Count field
27	2F	1		X'00'	End of chain

Figure 5.22 **CCW Skeletons (FBA devices) description and format**

Control Interval Work Area (CIW)

The CIW describes a control interval split workarea. It contains workareas for all routines that are activated during a control interval pseudo split, control interval split or a control area split. It is created by IKQCIS00 whenever a split occurs. It points to the data buffer needed in case of a split and is pointed to by the AMBL (AMBLCIWA). The space is acquired as needed by GETVIS. At completion of CI-split processing, it is freed via FREEVIS, and AMBLCIWA is set to zeros.

Offset		Bytes and		
Dec	**Hex**	**Bit Pattern**	**Field Name**	**Description**
		Register Save Area for IKQCIS		
0	0	48	CIWAVE	Register save area (12 Reg.)
0	0	4	CIWAVR14	Register 14
4	4	4	CIWAVR15	Register 15
8	8	4	CIWAVR0	Register 0
12	C	4	CIWAVR1	Register 1, RDF shift count on entry
16	10	4	CIWAVR2	Register 2, RDF modification offset
20	14	4	CIWAVR3	Register 3, RDF data work area
24	18	24		Reserved
48	30	4	CIWLNGTH	Length of work area
		Register Save Area		
This save area is used when IKQCIS calls IKQCIL or IKQCIU.				
52	34	4	CIWCLU03	Register 3
56	38	4	CIWCLU14	Register 14
		Register Save Area		
This save area is used when IKQCIL or IKQCIU calls any of the common subroutines in IKQCIS.				
60	3C	4	CIWCIS03	Register 3
64	40	4	CIWCIS14	Register 14
		Space Manager Save Area		
60	3C	4	CIWSPA14	Register 14
64	40	4	CIWSPA15	Register 15
68	44	4	CIWSPA03	Register 3
		IKQPFO Work Area		
72	48	4	CIWPFO14	Register 14
76	4C	4	CIWPFO00	Register 0
80	50	4	CIWPFO01	Register 1
84	54	4	CIWPFO02	Register 2
88	58	4	CIWPFO03	Register 3
92	5C	4	CIWPFO04	Register 4
96	60	4	CIWACB	ACB pointer for TCLOSE call
100	64	2	CIWSVC	SVC2 in TCLOSE call list
102	66	2		Unused

Figure 5.23 **Control Interval Work Area (CIW) description and format (part 1 of 5)**

Offset Dec	 Hex	Bytes and Bit Pattern	 Field Name	 Description
		IKQRRP Work Area		
The work area for IKQRRP overlays the work area for IKQPFO				
72	48	4	CIWRRP14	Register 14
76	4C	4	CIWRRPOO	Register 0
80	50	4	CIWRRP01	Register 1
84	54	4	CIWRRP02	Register 2
88	58	4	CIWRRP03	Register 3
92	5C	4	CIWRRBA	Beginning of RBA in extent
96	60	4	CIWRRPLN	Preformat length
100	64	2	CIWRSEOF	SEOF indicator
102	66	2		Unused
		IKQNCA Work Area		
104	68	4	CIWNEW14	Register 14
108	6C	4	CIWNEW01	Register 1
112	70	4	CIWNEW03	Register 3
116	74	4	CIWCARBA	Low RBA of data control area (new control area)
120	78	4	CIWCIRBA	Index RBA of old sequence set record
124	7C	4	CIWNXRBA	Index RBA of new sequence set record
128	80	4	CIWDARDB	Data ARDB
		IKQCAS Work Area		
132	84	4	CIWCAS14	Register 14
136	88	4	CIWCAS03	Register 3
140	8C	4	CIWHINEW	High section of new control area
144	90	4	CIWSPTR	Pointer save section
148	94	4	CIWHIOLD	High section of old control area
152	98	4	CIWEPTR	Entry pointer
156	9C	4	CIWAKEY	Address of key save area
160	100	2	CIWEINC	Entry increment bytes
162	102	2	CIWSRR	Offset of last section from the high section of the new control area
164	104	4	CIWXBUFA	Address of new index buffer
		IKQCIR Work Area **Control Interval Space Reclamation Work Area**		
The work area for IKQCIR overlays the work areas for IKQNCA and IKQCAS				
104	68	4	CIWCIR14	Register 14
108	6C	4	CIWCIR09	Register 9
112	70	4	CIWCIR03	Register 3
116	74	4	CIWSAVP	Free data of pointer save for control interval
120	78	1	CIWCIRSW	Switch byte
			CIWNEXT	X'80' Position to next entry index
			CIWSPAN	X'40' Spanned entry index
			CIWRECL	X'20' Space reclamation index
			CIWNOSPL	X'10' No control area split indicator
			CIWXWRT	X'08' Write index indicator

Figure 5.23 Control Interval Work Area (CIW) description and format (part 2 of 5)

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
121	79	3			Reserved
124	7C	0	CIWLASMD		IKQLASMD parameter list
124	7C	1	CIWLID		Request type
			CIWLTST	X'04'	Test request
125	7D	0	CIWLDSID		Data set identification
125	7D	3	CIWLDSCI		Control interval number
128	80	4	CIWLACB		Pointer to catalog ACB
132	84	1	CIWLSOPT		Share option
133	85	1	CIWLFLG		Flag byte
			CIWLIN	X'80'	Input indicator
134	86	2	CIWLOUT		Output count
136	88	8	CIWRES		Resource name field
144	90	24			Unused
		IKQCIS Work Area			
168	A8	32	CIWCIWA		Copy of PLH work area
200	C8	4	CIWRCDCT		Record count save for move
204	CC	4	CIWSPLPT		Pointer to modification point
208	D0	4	CIWFPTR		Pointer to next record to be moved
212	D4	4	CIWFRDF		Pointer to RDF of the next record
216	D8	4	CIWCLNUP		RBA of control interval requires an update
220	DC	4	CIWDCRDB		Save of current ARDB pointer
224	E0	4	CIWNIRBA		RBA of new sequence set
228	E4	2	CIWOLDCT		Save of RDF count
230	E6	1	CIWFLAGS		Flags
			CIWNCAS	X'40'	Control area split needed to continue
			CIWCASDN	X'20'	Control area split has been executed
			CIWUHKR	X'10'	ARDHKRBA requires update
			CIWCLN	X'08'	Control intervals written require clearing
			CIWCIR	X'04'	Space reclamation executed
231	E7	1			Unused
		IKQIXE Entry Stack			
232	E8	0	CIWENTRY		Index entry data stack
232	E8	12	CIWENT1		First stack position
232	E8	4	CIWRBA1		RBA to be put in entry
236	EC	4	CIWKADD1		Address of key
240	F0	2	CIWKL1		Length of key
242	F2	1	CIWFLG1		Flag byte
			CIWENTOK	X'81'	These two bits are used to indicate that this entry is valid
			CIWINC	X'40'	Index record in core
			CIWSPLIT	X'20'	Split entry to be done
			CIWNOIO	X'10'	No execution of input/output yet (I/O is required)

Figure 5.23 Control Interval Work Area (CIW) description and format (part 3 of 5)

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
243	F3	1	CIWIXLV1		Index level
244	F4	12	CIWENT2		Second stack position (Used to hold contents of stack 1 when stack 1 is needed for further processing.)
244	F4	4	CIWRBA2		RBA
248	F8	4	CIWKADD2		Key pointer
252	FC	2	CIWKL2		Key length
254	FE	1	CIWFLG2		Same as CIWFLG1
255	FF	1	CIWIXLV2		Index level
256	100	0	CIWSTKND		End of stack
256	100	4	CIWEKEYA		Address of index enter key
		Scratch Buffer Parameter List			
260	104	20	CIWDCNV		Scratch CI descriptor
260	104	4	CIWDRBA		Scratch control interval RBA
264	108	8	CIWDBUF		Buffer parameter list
264	108	4	CIWDBCB		Address of control block
268	10C	4	CIWDBAD		Address of buffers
272	110	4	CIWDCIDF		CIDF descriptor
272	110	2	CIWDFSO		Free space offset
274	112	2	CIWDFSL		Free space length
276	114	1	CIWDSW		Switch byte
277	115	1			Reserved
278	116	2	CIWDCSZ		Length of buffer - 10
		IKQIXE Work Area			
280	118	4	CIWIXEBA		Caller base save
284	11C	4	CIWIXERT		Return register save
288	120	4	CIWIXER0		Save GETVIS length
292	124	4	CIWIXER1		Save GETVIS address
		Work Area for Linkage from IKQCIS to IKQCAS			
296	128	4	CIWCILST		CI list for multi-string CA split
300	12C	4	CIWCISR8		Register save for linkage return
		AMDSB Save Area for Updates to AMDSB Control Fields			
304	130	4	CSXHLRBA		AMDHLRBA index
308	134	2	CSXNIL		AMDNIL index
310	136	2			Unused
		IXFORMAT Work Space			
312	138	4	CIWIXFBA		Save callers base
316	13C	4	CIWIXFRT		Save return register
320	140	4	CIWLSEP		Entry pointer for last section
324	144	4	CIWANLSE		Entry address for last section
328	148	4	CIWANLE		Last entry address

Figure 5.23 **Control Interval Work Area (CIW) description and format (part 4 of 5)**

Offset Dec	 Hex	Bytes	Field Name	Hex. Digit	Description
332	14C	2	CIWKEYL		Length of current key
334	14E	2	CIWNLSEL		Length of last section key
336	150	2	CIWNLEL		Length of last entry key
338	152	2			Unused
340	154	4	CIWXNSA		Address of next section
344	158	4	CIWXSOP		Offset pointer of last section
348	15C	2	CIWFCNT		Format count
350	15E	2	CIWCINL		Control entry length
352	160	44	CIWAREA		Work area for RDF build
396	18C	8	CIWLKSP		Parameters describing the DTL (Define-The-Lock) for the Space Allocation Lock used in share option 4
396	18C	4	CIWLKSPL		Length of DTL
400	190	4	CIWLKSPA		Address of DTL
404	194	*	CIWKEY		Index key work area

* Length = 5 x keylength

Figure 5.23 Control Interval Work Area (CIW) description and format (part 5 of 5)

Duplicate Data Recovery Work Area (DDRW)

If a system failure occurs during control interval split processing, duplicate data records may exist for a file. IKQDDRW maps the DDRW to use as a save area, while determining if duplicate records exist and correcting the situation, if necessary.

Offset Dec	 Hex	Bytes	Field Name	Hex. Digit	Description
0	0	60	DDRWPHS		Register save area for IKQBFA00.
60	3C	48	DDRWIXS		Save area for IKQIXS00 save area.
108	6C	4	DDRW14S		Save area for register 14 for IKQIXS00.
112	70	4	DDRWXSOP		Save area for last section entry offset pointer (PLHXSOP).
116	74	2	DDRWXSEO		Save area for next section entry offset (PLHXSEO).
118	76	2	DDRWXEO		Save area for index entry offset (PLHXEO).
120	78	2	DDRWXEOP		Save area for previous entry offset (PLHXEOP).
122	7A	2	DDRWXLEV		Save area for previous index level (PLHXLEVP).
124	7C	2	DDRWXSEP		Save area for previous section entry offset (PLHXSEOP).
126	7E	2	DDRWFL		Save area for F and L bytes of index entry:
126	7E	1	DDRWF		F byte.
127	7F	1	DDRWL		L byte.
128	80	2	DDRWRC		Record count work field for RDFs describing multiple records of the same length.

Figure 5.24 Duplicate Data Recovery Work Area (DDRW) description and format

Extent Definition Block (EDB)

The EDB describes all extents of the space allocated to the cluster's data set. The EDB is built by the Open module from information in the data set's catalog record. The AMDSB contains the length of the EDB (AMDLEDB), the number of EDB entries (AMDNEDB) that follow the header, and the address of the first EDB (AMDFSEDB). Each EDB entry describes an extent, and contains the address of the associated LPMB.

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
0	0	4	EDBNEDB		Address of next EDB
4	4	2	EDBSYMU		Symbolic unit (for CCB)
4	4	1	EDBSUCLS		Symbolic unit class
5	5	1	EDBSUNUM		Symbolic unit number
6	6	2	EDBNUMTR		Number of tracks of extent
8	8	1	EDBFLGS		Flags
			EDBDWSS	X'80'	Data RBA with sequence set
			EDBSSWD	X'40'	Sequence set RBA with data
			EDBIXREP	X'20'	Index replication
			EDBMNT	X'10'	Volume mount flag
			EDBLGCC	X'08'	Device contains more than 256 cylinders
			EDBRPS	X'04'	Indicator for RPS device
			EDBARCH	X'02'	Extent is on a fixed block device
9	9	3	EDBMBB		Extent (M) and bin (BB) number
9	9	1	EDBM		Extent (M) number
10	A	2	EDBBB		Bin (BB) number
12	C	4	EDBLBBBB		Starting block of a fixed block extent
12	C	8	EDBXTNT		Combined name for low and high CCHH
12	C	4	EDBLCCHH		Low cylinder and head numbers
12	C	2	EDBLCC		Lowest cylinder
14	E	2	EDBLHH		Lowest head
16	10	4	EDBHBBBB		Ending block of a fixed block extent
16	10	4	EDBHCCHH		High cylinder and head numbers

Figure 5.25 Extent Definition Block (EDB) description and format (part 1 of 2)

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
16	10	2	EDBHCC		Highest cylinder
18	12	2	EDBHHH		Highest head
20	14	4	EDBLPMBA		Address of associated LPMB
24	18	4	EDBPARDB		Address of ARDB
28	1C	2	EDBVLSQ		Index to the VOLSER list
30	1E	2	EDBSTTRK		Relative track address of the start of the extent (zero for fixed block devices)
32	20	8	EDBRBAS		Combined name for low and high RBAs
32	20	4	EDBLORBA		Low RBA limit
36	24	4	EDBHIRBA		High RBA limit
40	28	4	EDBTLBCA		Total number of data blocks and sequence set blocks per CA, minus 1 (fixed block devices).
44	2C	4	EDBCASZ		Number of bytes per control area
48	30	4	EDBTRKCA		Number of tracks per control area
48	30	4	EDBTPBCA		Total number of data blocks, sequence set blocks, and wasted blocks per CA (fixed block devices).

Figure 5.25 Extent Definition Block (EDB) description and format (part 2 of 2)

EXCPAD Parameter List

EXP is a mapping macro that maps the following parameter list when an EXCPAD exit is taken.

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
0	0	4	EXPST		Start of EXCPAD list
0	0	256	EXPAREA		Length of parameter list
0	0	56	EXPSAVE		Register 2-15 save area
0	0	4	EXPSAV02		Register 2 save area
4	4	4	EXPSAV03		Register 3 save area
8	8	4	EXPSAV04		Register 4 save area
12	C	4	EXPSAV05		Register 5 save area
16	10	4	EXPSAV06		Register 6 save area
20	14	4	EXPSAV07		Register 7 save area
24	18	4	EXPSAV08		Register 8 save area
28	1C	4	EXPSAV09		Register 9 save area
32	20	4	EXPSAV10		Register 10 save area
36	24	4	EXPSAV11		Register 11 save area
40	28	4	EXPSAV12		Register 12 save area
44	2C	4	EXPSAV13		Register 13 save area
48	30	4	EXPSAV14		Register 14 save area
52	34	4	EXPSAV15		Register 15 save area
56	38	3			Reserved
59	3B	1	EXPPECBT		Test and set byte
60	3C	68			Reserved
128	80	128	EXPARML		User's parameter list
128	80	4	EXPRPLC		Address of calling RPL
132	84	4	EXPECB		Address of ECB
136	88	4	EXPRPLS		Address of splitting RPL (zero indicates no split)
149	8C	116			Rest of user's parameter list (available to user)

Figure 5.26 EXCPAD Parameter List description and format

Exit List (EXLST)

The EXLST contains addresses for the user-exit processing routines EODAD, SYNAD, LERAD, EXCPAD, and JRNAD. The address of the EXLST is in the ACB (ACBEXLST).

Offset				Hex.	
Dec	**Hex**	**Bytes**	**Field Name**	**Digit**	**Description**
0	0	1	EXLID		Control block identifier = X'81'
0	0	0	EXLIDD	X'81'	EXLST identifier equate
1	1	1	EXLSTYP		Release indicator
			EXLSDV1	X'00'	DOS/VS VSAM Release 1
			EXLSVSE1	X'10'	VSE/VSAM Release 1
				X'20'	VTAM
2	2	2	EXLLEN		Length of EXLST
4	4	1	EXACT		Active byte
5	5	0	EXLEOD		EODAD entry
5	5	1	EXLEODF		Entry description bits
6	6	4	EXLEODP		Address of the EODAD exit routine
10	A	0	EXLSYN		SYNAD entry
10	A	1	EXLSYNF		Entry description bits
11	B	4	EXLSYNP		Entry of the SYNAD exit routine
15	F	0	EXLLER		LERAD entry
15	F	1	EXLLERF		Entry description bits
16	10	4	EXLLERP		Address of the LERAD exit routine
20	14	0	EXLIOEX		EXCPAD entry
20	14	1	EXLIOEXF		Entry description bits
21	15	4	EXLIOEXP		Address of the EXCPAD exit routine
25	19	0	EXLJRN		JRNAD entry
25	19	1	EXLJRNF		Entry description bits
26	1A	4	EXLJRNP		JRNAD pointer
Bits used in individual exit flags in bytes shown as entry description:					
0	0	1	EXENFL		Flag byte
			EXENEXB	X'80'	Entry present bit
			EXENACTB	X'40'	Entry active bit
			EXENLEB	X'20'	Load bit
1	1	4	EXENADDR		Exit address
Minimum length EXLST for specified entry:					
				Dec.	
				Digit	
			EXLEODL	10	Minimum length if EODAD
			EXLSYNL	15	Minimum length if SYNAD
			EXLLERL	20	Minimum length if LERAD
			EXLIOEXL	25	Minimum length if EXCPAD
			EXLJRNL	30	Minimum length if JRNAD
Minimum and maximum size of EXLST:					
			EXLMINL	10	Minimum length of EXLST
			EXLMAXL	30	Maximum length of EXLST
32	20	0	EXLSTEND		End of EXLST

Figure 5.27 **Exit List (EXLST) description and format**

Field Control and Data Block (FCDB)

Many FCDBs comprise the FCDB area. These FCDBs are suballocated as needed for CCW(s), CCB(s), FXL(s), IOARG(s), and one BKPHD. This BKPHD points to the first unallocated FCDB, which is used by IKQIOA/IKQIOC to construct the channel program.

Offset				Hex.	
Dec	**Hex**	**Bytes**	**Field Name**	**Digit**	**Description**
0	0	64	FCB		Maps the module FCB
0	0	56			Space for use in the block
56	38	1	FCBTIC		Reserved for a TIC operation code
57	39	3	FCBCHAIN		Pointer to next block
60	3C	1	FCBCFL		Reserved for chaining flag
61	3D	1	FCBALI		Allocation indicator
			FCBPRVA	X'04'	Previous request allocated
			FCBPRVSV	X'08'	Previous request save
62	3E	2	FCBOFSET		Offset pointer in block
		Equate values			
			FCBCMAX	X'38'	Maximum CCW offset (CKD) when full
			FCBFMAX	X'38'	Maximum data offset (fixed block) when full
			FCBDMAX	X'30'	Maximum data offset when full
			FCBDINC	X'0C'	Increment for data in space
			FCBCINC	X'08'	Increment for CCW(s) (CKD) in space
			FCBFINC	X'08'	Increment for data (fixed block) in space
			FCBCP2	X'03'	CCW increment in powers of 2

Figure 5.28 **Field Control and Data Block (FCDB) description and format**

File Sharing Work Area (SHRW) for SHAREOPTIONS (4)

The SHRW contains information used only in SHAREOPTIONS (4) processing, and is used as a work area in sequential processing of SHAREOPTIONS (4) files. The AMBL points to the SHRW. The SHRW is created by Open and released by Close.

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
0	0	1	SHRWID	X'87'	Control block identification
1	1	1			Reserved
2	2	2	SHRWLEN		Length of this work area
4	4	4	SHRWECB		Event Control Block for serializing use of this work area
6	6	1	SHRWECOM		ECB post byte
			SHRWETRA	X'80'	Traffic bit
7	7	1	SHRWECBT		Test-and-set byte
8	8	4	SHRWSV14		Save area for register 14
12	C	4	SHRWSV0		Save area for register 0
16	10	4	SHRWSV3		Save area for register 3
20	14	4	SHRWSV4		Save area for register 4
24	18	4	SHRWSV5		Save area for register 5
28	1C	4	SHRWSV6		Save area for register 6
32	20	4	SHRWSV7		Save area for register 7
36	24	4	SHRWSV8		Save area for register 8
40	28	4	SHRWSV9		Save area for register 9
44	2C	2	SHRWCINL		Length of the control information (F+L+P fields) in each index entry.
46	2E	12	SHRWLCKN		Basic lock name for use with the LOCK macro when doing lock requests for this data set.
46	2E	1			Lock name prefix - "V" indicates VSAM
47	2F	6			Volume ID of the catalog volume that owns the file.
53	35	3			CI number of the catalog record that defines the file.
56	38	2	SHRWLCKS		Lock name suffix.
58	3A	2			Reserved
60	3C		SHRWARG		Variable length argument field - actual length is 4, or for keyed access to a KSDS the key length (specified by AMDKEYLN).

Figure 5.29 **File Sharing Work Area (SHRW) for SHAREOPTIONS (4) description and format**

Fix List (FXL)

IKQFXL maps a 64-byte FCDB into a Fix List during the channel program build process. It contains seven 8-byte entries, each consisting of an entry address, a virtual starting address, and a virtual ending address. The I/O Manager reformats each 2-byte entry to a 4-byte address before issuing an EXCP.

Offset				Hex.	
Dec	**Hex**	**Bytes**	**Field Name**	**Digit**	**Description**
FXLCP, FXLSA, and FXLEA are repeated seven times.					
0	0	4	FXLCP		Pointer to a chain of virtual addresses (in low-to-high order). If zero, the associated address entry pairs are the highest in the chain.
4	4	2	FXLSA		Starting virtual page number (rounded down to 2K boundary).
6	6	2	FXLEA		Ending virtual page number (rounded up to 2K boundary).
8	8	8			Second entry.
16	10	8			Third entry.
24	18	8			Fourth entry.
32	20	8			Fifth entry.
40	28	8			Sixth entry.
48	30	8			Seventh entry.
The following field normally appears at offset 56 (X'38'), but it may appear after any Fix List entry to indicate the end of a Fix List chain.					
56	38	4	FXLNFXL		Flag byte and chain pointer to next Fix List.
56	38	1	FXLNFLG		Flag byte:
			FXLEOC	X'80'	End of Fix List chain.
			FXLNL	X'01'	More Fix List entries exist in the next block.
57	39	3			Chain pointer to next Fix List.
60	3C	2			Reserved.
62	3E	2	FXLOFST		Offset to next available entry (maximum 56 - FXLMAX).

Figure 5.30 **Fix List (FXL) description and format**

I/O Arguments (IOARG)

IKQIOARG maps the I/O data areas in the FCDB.

Offset				Hex.	
Dec	**Hex**	**Bytes**	**Field Name**	**Digit**	**Description**
0	0	8	IOASEEK		Computed DASD address
0	0	1	IOAFLAG		Flag byte
			IOARPS	X'80'	RPS device
1	1	2	IOABB		BB
3	3	4	IOACCHH		CCHH
3	3	2	IOACC		CC
5	5	2	IOAHH		HH
7	7	1	IOAR		R
8	8	1	IOAKEY		Key size
9	9	1	IOADATA		Data size
11	B	1	IOASEC		RPS sector size
		Mapping for Define Extent Parameters			
0	0	16	IOADE		Define extent argument.
0	0	4	IOAMASK		Define extent mask.
0	0	1	IOAFMSK		File mask bits
			IOAINHW	X'40'	Inhibit all writes
1	1	3			Reserved (Zero)
4	4	4	IOASTBB		Starting block number of extent (or control area for record mgt.) to be processed.
8	8	4	IOASTDS		Starting block displacement (zero for record mgt.)
12	C	4	IOAENDS		Ending block displacement (or number of blocks in control area, excluding waste, for record mgt.).
		Mapping for Locate Parameters			
0	0	8	IOALOC		Locate argument.
0	0	1	IOAOPCOD		Operation.
			IOAWRT	X'01'	Write operation.
			IOARDREP	X'02'	Read-replicated operation.
			IOAWCK	X'05'	Write and write check operation.
			IOARD	X'06'	Read operation.
1	1	1	IOAREPCT		Replication count.
2	2	2	IOABLKCT		Number of blocks to process.
4	4	4	IOABBBB		Address of first block in the block range (or control area for record management) to be processed.
		Fixed Block Save Area			
0	0	8	IOCFBASV		Fixed block save area.
0	0	4	IOCSTBB		Starting block number.
4	4	2	IOCDATA		Number of FCDBs used.

Figure 5.31 **I/O Arguments (IOARG) description and format**

I/O Driver Block (IODRB)

IKQIODRB maps the BUFRIODR and BUFWIODR fields of the BCB.

Offset				Hex.	
Dec	**Hex**	**Bytes**	**Field Name**	**Digit**	**Description**
0	0	20	IODLEN		
0	0	2	IODCURU		Symbolic unit number
2	2	2	IODBKSTI		Number of blocks to I/O
4	4	$	IODSTBB		Starting block number of control area (fixed block devices)
4	4	8	IODSEEK		Compiled DASD address
4	4	1	IODFLAG		Flag byte
			IODRPS	X'80'	RPS device indicator
5	5	2	IODBB		BB
7	7	4	IODCCHH		CCHH
7	7	2	IODCC		CC
8	8	8	IODBBBB		Displacement to the first block in the CA for read or write operation (fixed block devices)
9	9	2	IODHH		HH
11	B	1	IODR		R
12	C	4	IODRBA		RBA for I/O
16	10	4	IODLPMB		Address of associated LPMB

Figure 5.32 **I/O Driver Block (IODRB) description and format**

I/O Request Block (IORB)

Record management uses its own CCB macro to map a FDCB into a CCB.

Offset Dec	Offset Hex	Bytes	Field Name	Hex. Digit	Description
0	0		CCBST		
0	0		CCBD		
0	0	16	CCBLEN		Map of CCB.
0	0	2	CCBCNT		Residual count.
2	2	4	CCBERMAP		Error codes.
2	2	1	CCBCOM1		Communication byte 1:
			CCBWAIT	X'80'	Traffic switch (set at channel end).
			CCBEOF	X'40'	End of file.
			CCBIOERR	X'20'	Unrecoverable I/O error.
			CCBERROK	X'10'	Accept unrecoverable I/O error.
			CCBRDC	X'08'	Return data checks.
			CCBPDE	X'04'	Post at device end.
			CCBDCV	X'02'	Return data check read/check.
			CCBUERR	X'01'	User error routine.
3	3	1	CCBCOM2		Communication byte 2:
			CCBDCCNT	X'80'	Data check in count field.
			CCBTRKOV	X'40'	Track overrun.
			CCBEOC	X'20'	End of cylinder.
			CCBDC	X'10'	Data check.
			CCBNOREC	X'08'	No record found.
			CCBRETRY	X'04'	Retry no record found.
			CCBVER	X'02'	Verify error.
			CCBCC	X'01'	Command chain (retry).
4	4	1	CCBCSW1		CSW status byte 1:
			CCBATTN	X'80'	Attention.
			CCBSTMOD	X'40'	Status modifier.
			CCBCUE	X'20'	Control unit end.
			CCBBUSY	X'10'	Busy.
			CCBCE	X'08'	Channel end.
			CCBDE	X'04'	Device end.
			CCBUC	X'02'	Unit check.
			CCBUE	X'01'	Unit exception.
5	5	1	CCBCSW2		CSW status byte 2:
			CCBPCI	X'80'	Program-controlled interrupt.
			CCBILEN	X'40'	Incorrect length.
			CCBPROGM	X'20'	Program check.
			CCBPROT	X'10'	Protection check.
			CCBCHAND	X'08'	Channel data check.
			CCBCHANC	X'04'	Channel control check.
			CCBICTRL	X'02'	Interface control check.
			CCBCHAIN	X'01'	Chaining check.

Figure 5.33 I/O Request Block (IORB) description and format (part 1 of 2)

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
6	6	2	CCBSYMU		Symbolic unit.
6	6	1	CCBSUCLS		U - LUB class.
7	7	1	CCBSUNUM		N - LUB number within class.
8	8	1	CCBLIOCS		Reserved for LIOCS.
9	9	3	CCBCCW		Address of channel program.
12	C	1	CCBCOM3		Communication byte 3:
			CCBAPEND	X'40'	Appendage end at interrupt.
13	D	3	CCBCSW		Address of CSW in appendage routine.
16	10	4	CCBHFXL		Address of the VSAM-supplied Fix List.
16	10	1	CCBFIX		EXCP should not reformat the Fix List during retry on I/O errors. The supervisor uses this bit to indicate the Fix List is compressed (contains no redundancies in its entries).
20	14	2	CCBIORID		Reserved for IORB ID (field must be zero).
22	16	1	CCBSVOP		Save area for op code of the CCW following the last Write CCW in the chain ("piggy-back" operations).
23	17	1	CCBUFLGS		I/O manager CCB flags:
			CCBUEAIC	X'80'	Error analysis in control.
			CCBUEAC	X'40'	Error analysis complete.
			CCBURDCW	X'20'	Read CCW active.
			CCBRPS	X'10'	RPS channel program candidate.
24	18	4	CCBFXLEN		Address of chain of Fix List entries sorted by virtual address. The first entry represents the lowest virtual address space to be fixed.
28	1C	4	CCBDATB		Address of last data block.
32	20	4	CCBLCCWB		Address of last CCW block.
36	24	4	CCBRDCCW		Save area for first read CCW.
40	28	4	CCBWTCCW		Save area for first write CCW.
44	2C	4	CCBLWCCW		Save area for last write CCW.
48	30	4	CCBSVLOC		Save area for first four bytes of second CCW (temporary TIC during error recovery of "piggy-back" operation).
52	34	4			Reserved.
56	38	4	CCBNCCB		Address of next CCB block.
60	3C	4			Reserved.

Figure 5.33 **I/O Request Block (IORB) description and format (part 2 of 2)**

I/O Work Area (IOWKA)

IKQIOWKA maps the PLH Workarea (PLHWAREA, displacement X'D4').

Offset		Bytes	Field Name	Hex.	Description
Dec	Hex			Digit	
212	D4	44	WKAREA		Beginning of IOWKA.
212	D4	4	WKASTBB		Starting block number of the CA (fixed block devices).
212	D4	8	WKASEEK		Work area DASD address:
212	D4	1	WKAM		M
213	D5	2	WKABB		BB
215	D7	4	WKACCHH		CCHH
215	D7	2	WKACC		CC
216	D8	4	WKABBBB		Displacement to the first block in the CA to be processed (fixed block devices).
217	D9	2	WKAHH		HH
219	DB	1	WKAR		R
220	DC	4	WKABDC14		BLDCP000 save area.
224	E0	12	WKATEMP		Temporary area.
224	E0	12	WKAGETVS		ALLBK save area.
236	EC	20	WKAFXLSV		Save area for Fix List routine (R15 - R3).
256	100	2	WKARTNCD		Return code.
258	102	2	WKABLKS		Number of blocks on active CCB.
260	104	1	WKAERRSW		Hold error indicator in IKQIOB:
			IOMERP2	X'40'	The CCW in error has been passed over in the scan down the CCW chain for setting bits in the BCB.
261	105	1	WKAIOMSW		I/O manager flags:
			IOMPROCR	X'80	Process reads.
			IOMPROCW	X'40'	Process writes.
			IOMPROCK	X'20'	Process write checks.
			IOM1RD	X'08'	At least one read operation must be performed; if off, bypass I/O manager read loop.
			IOM1WRT	X'04'	At least one write operation must be performed; if off, bypass I/O manager write/write check loop.
			IOMWRPS	X'02'	This is a write pass through the BCBs for RBA conversion.
262	106	2			Reserved.
264	108	4	WKADBHD		Address of working BHD.
268	10C	4	WKAREGSV		ALLOC register save area.
272	110	4	WKASVCCB		Save address of previous CCB.

Figure 5.34 **I/O Work Area (IOWKA) description and format**

Logical-to-Physical Mapping Block (LPMB)

The LPMB contains information about the direct-access device that contains the user's data set. The LPMB is built by the Open module, using information in the data set's catalog record. The EDB (EDBLPMBA) contains the address of the LPMB.

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
0	0	1	LPMID	X'FF'	Control block identifier.
1	1	1	LPMBDTF		DTF device type indicator.
2	2	2	LPMLEN		Length of the LPMB.
4	4	4	LPMBPTRK		Number of bytes per track.
4	4	2	LPMNREP		Number of replicated sequence set CIs (fixed block devices).
6	6	2	LPMTLBCA		Total number of data blocks and sequence set blocks per CA, minus 1 (fixed block devices).
8	8	4	LPMCASZ		Number of bytes per control area.
12	C	4	LPMBLKSZ		Physical block size (512 for fixed block devices).
16	10	2	LPMTRKCA		Number of tracks per control area.
16	10	2	LPMTPBCA		Total number of data blocks, sequence set blocks, and wasted blocks per CA (fixed block devices).
18	12	2	LPMTIBCA		Number of sequence set blocks in the CA, or zero if there is no imbedded sequence set (fixed block devices).
18	12	2	LPMTPC		Number of tracks per cylinder.
20	14	2	LPMBNQBK		Number of physical records per track, or total blocks occupied by replicated index CIs.
22	16	2	LPMBPBCI		Physical blocks per control interval.

Figure 5.35 Logical-to-Physical Mapping Block (LPMB) description and format

Placeholder (PLH)

The PLH contains current information about a request. This information includes positioning information, request options, and buffer location and status. The PLH is built by the Open module and is pointed to by the AMBL (AMBLPLH). When a record management module is processing a PLH, the PLH's address is in register 13.

Offset				Hex.	
Dec	**Hex**	**Bytes**	**Field Name**	**Digit**	**Description**
			Standard Save Area		
0	0	0	PLHSAREA		Register save area
0	0	4			Reserved
4	4	4	PLHSADDR		Address of user's save area
8	8	4			Reserved
			Buffer Manager Save Area		
12	C	60	PLHSAVE		Save area for 15 registers
			I/O Manager Save Area and JRNAD Extended Save Area		
72	48	56	PLHBSAVE		I/O manager save area (R9-R6)
128	80	48	PLHIXSSV		Index search and get next save area
176	B0	16	PLHJRNSV		JRNAD save area
			Return Register Stacks		
192	C0	0	PLHSTCK		Fixed return register stack
192	C0	4	PLHSTCK1		Return register from level 1
196	C4	4	PLHSTCK2		Return register from level 2
			RPL Pointers		
200	C8	4	PLHHRPL		Pointer to header RPL
204	CC	4	PLHCRPL		Pointer to current RPL
			PLH ECB (see I/O Work Area - IOWKA)		
208	D0	4	PLHECB		Event control block
208	D0	1			Reserved
209	D1	1	PLHAUSE		Request active on PLH
210	D2	1	PLHECOM		Communications byte
			PLHEWAIT	X'80'	Wait flag on ECB
211	D3	1	PLHECBT		Test and set byte for ECB
			PLH Work Area (see I/O Work Area - IOWKA)		
212	D4	64	PLHWAREA		PLH work area
			PLH Identification Byte		
276	114	1	PLHID	X'55'	PLH identification byte

Figure 5.36 Placeholder (PLH) description and format (part 1 of 9)

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
			PLH Use Gate		
277	115	1	PLHUSE		PLH use gate If ON (X'FF'), this PLH is available only to an RPL whose string identifier (RPLSTRID) is equal to the string identifier (PLHSTRID) of this PLH.
			PLH Condition Flags		
278	116	1	PLHFLAG		PLH condition flags
			PLHST	X'80'	PLH status flag (bit 0) 1 - PLH set 0 - PLH invalid
			PLHPOS	X'40'	PLH position flag (bit 1) 1 - Next record 0 - previous record
			PLHEOD	X'20'	PLH end-of-data-condition flag (bit 2) 1 - EOD reached 0 - Not EOD
			PLHWAIT	X'10'	PLH wait flag (bit 3) 1 - I/O pending 0 - No I/O pending
			PLHSKIP	X'08'	PLH skip flag (bit 4) 1 - Skip control interval 0 - Don't skip control interval
			PLHRST	X'04'	PLH restart flag (bit 5) 1 - Restart 0 - No restart
			PLHFST	X'02'	PLH first-time flag (bit 6) 1 - First time 0 - Not first time
			PLHRREAD	X'01'	PLH exclusive control reread flag (bit 7) 1 - Need reread 0 - Reread not needed
279	117	1	PLHFLG		PLH spare condition flag
			PLH Communication Switches		
280	118	1	PLHSWTCH		PLH communication switches
			PLHLOAD	X'80'	PLH load or resume load indicator
			PLHKRCH	X'40'	PLH key range change indicator
			PLHMSRT	X'20'	Mass insert indicator
			PLHFSR	X'10'	First request for data set indicator
				X'08'	Reserved
			PLHSTBCB	X'04'	Demand a BCB from STEAL000 (IKQBFD)
			PLHEC	X'02'	Exclusive control needed
			PLH2AROW	X'01'	Two inserts in a row (consecutive records) in skip sequential processing
			Previous Request Characteristics		
281	119	3	PLHPREQ		Previous request information
281	119	1	PLHRTC		Previous request-type code
282	11A	2	PLHOPT		Previous request option bytes
282	11A	1	PLHOPT1		First option byte

Figure 5.36 **Placeholder (PLH) description and format (part 2 of 9)**

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
283	11B	1	PLHOPT2		Second option byte
284	11C	4	PLHPKEYA		Pointer to save area for keyed access
			Internal request characteristics for LSR		
288	120	4	PLHILTM		Test mask for WRTBFR
292	124	4	PLHILRM		Reset mask for WRTBFR
296	128	1	PLHPERC		Percentage value for number of least recently used buffers
297	129	1	PLHIOPT1		Option byte for WRTBFR
			PLHIDSC	X'80'	Data set processing for forced close
			PLHIDS	X'40'	Write buffers for a data set
			PLHILRU	X'20'	Write least recently used buffers
			PLHIALL	X'10'	Write all buffers
			PLHITRN	X'08'	Write buffers for specified transaction identifier
			PLHIBCB	X'04'	Subpool contains at least one modified buffer
			PLHIFIO	X'02'	Force I/O for buffer
298	12A	2			Reserved
300	12C	4	PLHDBSPH		Address of data buffer subpool
304	130	4	PLHIBSPH		Address of index buffer subpool
308	134	4	PLHACB		Pointer to data set's ACB
312	138	4	PLHEACB		Address of ACB of buffer with error
316	13C	4	PLHLSRA		Address of LSR save area
			Multiple String Support		
320	140	1	PLHSTRID		PLH string ID (1-255)
321	141	1	PLHENDRQ		ENDREQ request gate byte
322	142	1	PLHINDS		Indicator byte
			PLHCLOSE	X'80'	Close-type ENDREQ request
323	143	1			Reserved
			EXCPAD Parameter List Pointer		
324	144	4	PLHPARML		EXCPAD parameter list pointer
			JRNAD Parameter List Pointer		
328	148	4	PLHAJRN		JRNAD parameter list pointer
			I/O Manager Entry Point		
332	14C	4	PLHIOMGR		I/O Manager (IKQIOA00) entry point
			Key Range Support Fields		
336	150	4	PLHDCRDB		Address current ARDB
340	154	4	PLHDTRDB		Address target ARDB
			Pointers to Buffer Headers (BHDs)		
344	158	4	PLHDBHD		Address of data BHD
348	15C	4	PLHIBHD		Address of index BHD
352	160	4	PLHBRPL		Save header RPL
356	164	4	PLHTHB		Address of THB (share option 4)
360	168	4			Reserved

Figure 5.36 Placeholder (PLH) description and format (part 3 of 9)

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
			Data PLH		
364	16C	36	PLHDATA		Data PLH
364	16C	20	PLHDCNV		The buffer manager parameter list for the data control interval currently being used to satisfy GET or PUT request - the primary data buffer
364	16C	4	PLHDRBA		Data CI RBA
368	170	8	PLHDBUF		Data buffer description
368	170	4	PLHDBCB		Address of data BCB
372	174	4	PLHDBAD		Address of data buffer
376	178	4	PLHDCIDF		Data CI CIDF
376	178	2	PLHDFSO		Data CI free space offset
378	17A	2	PLHDFSL		Data CI free space length
380	17C	1	PLHDSW		Data CI buffer manager interface switches
			PLHHOLD	X'80'	Track hold indication
			PLHHELD	X'40'	Track free indication
			PLHNORDD	X'20'	No read from the data set. No buffer is returned to the buffer manager requestor.
			PLHNOINV	X'10'	No invalidation. If this bit is on, no invalidation takes place, even for SHAREOPTIONS (4). If this bit is off, then the meaning is dependent on PLHNORDD as follows:
				..00....	PLH and PLHNOIVN both off (normal GETBUFF) - Only if SHAREOPTIONS (4), then any existing buffer for the RBA will be invalidated before reading from the data set. If not SHAREOPTIONS (4), no invalidation takes place.
				..10....	PLHNORDD on and PLHNOINV off (explicit invalidation) - Scan the buffer queues of this PLH and invalidate the buffer with the the specified RBA if it exists. Do not read from the data set (No buffer is returned to the buffer)
			PLHNORD	..11....	No GETBUFF activity - no scan of the buffer queues and no read from the data set
			PLHLOG	X'08'	Logical GETBUFF request
			PLHRAHD	X'04'	Read-ahead request
			PLHCATH	X'02'	CA split SHAREOPTIONS (4) hold - used to request an extra SHAREOPTIONS (4) hold on the new CA during a CA split. This CA split hold will be released whenever the normal SHAREOPTIONS (4) hold is released

Figure 5.36 **Placeholder (PLH) description and format (part 4 of 9)**

Offset Dec	 Hex	Bytes	Field Name	Hex. Digit	Description
381	17D	1	PLHDSW1		Buffer request control switch
			PLHEHOLD	X'80'	Exclusive control desired
			PLHEHELD	X'40'	Exclusive control held
			PLHEACTV	X'20'	Exclusive control active
			PLHKGEHD	X'10'	Key-greater-than-or-equal hold - indicates that during a GET KGE, the SHAREOPTIONS (4) hold on the CA in which the search began is to be specially retained while the search proceeds to the next CA.
			PLHBRKHD	X'08'	Break the SHAREOPTIONS (4) hold - do not allow it to continue uninterrupted even though the hold request may be for the same CA as is currently held.
			PLHCATH	X'10'	CA split track hold
			PLHCATF	X'08'	CA split track free
			PLHDIRQ	X'04'	Indication to the buffer manager that this direct write request is to be deferred (set and reset by IKQMDY)
			PLHNODDR	X'02'	No duplicate data recovery
382	17E	2	PLHDCSZ		Data CI size minus 10 (offset to rightmost RDF)
			Data Record Description		
384	180	16	PLHDRCD		Data record description
384	180	2	PLHDRO		Data record offset
386	182	2	PLHDRDF		Data record RDF-offset
388	184	2	PLHDRIX		Data record RDF-index
390	186	2			Spare
392	188	4	PLHDRRBA		Data record RBA
396	18C	4	PLHDRL		Data record length

Figure 5.36 Placeholder (PLH) description and format (part 5 of 9)

Offset				Hex.	
Dec	**Hex**	**Bytes**	**Field Name**	**Digit**	**Description**
			Read-Ahead Data PLH		
400	190	24	PLHBDATA		Data read-ahead PLH
400	190	4	PLHBRBA		RBA of next CNV to read ahead
			Read-Ahead Data CI Description		
404	194	20	PLHBDCNV		Read-ahead data buffer manager parameter list
404	194	4	PLHBDRBA		Data CI RBA
408	198	8	PLHBDBUF		Data buffer description
408	198	4	PLHBDBCB		Address of data BCB
412	19C	4	PLHBDBAD		Address of data buffer
416	1A0	4	PLHBDCDF		Data CI CIDF
416	1A0	2	PLHBDFSO		Data CI free space offset
418	1A2	2	PLHBDFSL		Data CI free space length
420	1A4	1	PLHBDSW		Data CI switches (same as PLHDSW)
421	1A5	1	PLHBDSW1		Buffer request control switch (same as PLHDSW1)
422	1A6	2	PLHBDCSZ		Data CI size minus 10
			Alternate Index Record Information		
424	1A8	16	PLHAIX		AIX record information
424	1A8	4	PLHAIXPT		Address of base cluster pointer
428	1AC	4	PLHBCPLH		Address of base cluster's PLH
432	1B0	4	PLHAIXWL		Reserved
436	1B4	2	PLHAIXPN		Number of base cluster pointers still to be processed in this AIX record
438	1B6	2	PLHAIXOP		RPL Option bytes
440	1B8	12	PLHUPG		Upgrade set information
440	1B8	4	PLHUPGP1		Current USB entry address
444	1BC	4	PLHUPGP2		Last USB entry address
448	1C0	4	PLHUPGAD		Address of prime key (KSDS) or RBA (ESDS) of base cluster record
452	1C4	24	PLHAIXSV		AIX save area
			Spanned Record Flag Byte		
476	1DC	1	PLHSWT2		Spanned record switch byte
			PLHSPAN	X'80'	Spanned record indicator
			PLHSRU	X'40'	Called from IKQSRU
			PLHSRUF	X'20'	First call from IKQSRU
			PLHSRUL	X'10'	Last call from IKQSRU
			PLHSRCAS	X'08'	CA-split necessary
				X'04'	Reserved
			PLHSREC	X'02'	Exclusive control indicator
				X'01'	Reserved

Figure 5.36 **Placeholder (PLH) description and format (part 6 of 9)**

Offset				**Hex.**	
Dec	**Hex**	**Bytes**	**Field Name**	**Digit**	**Description**
			JRNAD Flag Byte		
477	1DD	1	PLHJRN		JRNAD flag byte
			PLHJRVSM	X'40'	JRNAD called from IKQVSM
			PLHJRMDY	X'20'	JRNAD called from IKQMDY
			PLHJRCIS	X'10'	IRNAD called from IKQCIS
			PLHJRCA1	X'08'	JRNAD first call from IKQCAS
			PLHJRCA2	X'04'	JRNAD second call from IKQCAS
			PLHJRSRG	X'02'	JRNAD called from IKQSRG
			PLHJRSRU	X'01'	JRNAD called from IKQSRU
			Spanned Record Information		
478	1DE	2	PLHSRCNT		Number of segments
480	1E0	22	PLHSPREC		Spanned record information
480	1E0	8	PLHRCD		Spanned record description
480	1E0	4	PLHAREA		Pointer to user area
484	1E4	4	PLHRLEN		Length of spanned record
488	1E8	4	PLHSRRBA		RBA of record
492	1EC	2	PLHX1EO		Index entry offset of first part
494	1EE	2	PLHXPTR		Pointer number
496	1F0	6	PLHSRRDF		Double RDF for spanned record
496	1F0	1	PLHSRR2		R byte of 2nd (leftmost) RDF
497	1F1	2	PLHSRLVL		Level number
499	1F3	1	PLHSSR1		R byte of 1st (rightmost) RDF
500	1F4	2	PLHSRLL		Length of segment
			Additional PLH Switches		
502	1F6	1	PLHSWT1		PLH communication switch control
			PLHRSI	X'10'	RPL area switch indicator (indicates that the user RPL areas of an AIX and the base cluster have been switched)
			PLHUPRES	X'08'	AIX upgrade reset switch
			PLHPCI	X'04'	Previous index control interval required
			PLHBWD	X'02'	Backward processing
			PLHLRD	X'01'	Last record processing
503	1F7	1	PLHFLG1		Flag byte continuation
			PLHDUKEY	X'08'	Duplicate key in AIX record
			PLHAIXRP	X'04'	AIX repositioning flag (Previous AIX record must be read)
			Index PLH		
504	1F8	40	PLHINDEX		Index PLH
			PLHESDS		Length of PLH for ESDS (equate value)

Figure 5.36 **Placeholder (PLH) description and format (part 7 of 9)**

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
			Index CI Description		
504	1F8	20	PLHXCNV		The buffer manager parameter list for the index control interval currently being processed - the primary index buffer
516	204	4	PLHXCIDF		The CI CIDF
516	204	4	PLHXCIDF		Index CI CIDF
516	204	2	PLHXFSO		Index CI free space offset
518	206	2	PLHXFSL		Index CI free space length
520	208	1	PLHXSW		Index CI switches (same as PLHDSW)
521	209	1	PLHXSW1		Buffer request control (same as PLHDSW1)
522	20A	2	PLHXCSZ		Index CI size minus 10
			Index Entry Description		
524	20C	20	PLHXETRY		Index entry description
524	20C	2	PLHXEO		Index entry offset
526	20E	2	PLHXSEO		Next section entry offset
528	210	4	PLHXSOP		Last section entry offset pointer
532	214	2	PLHXLVL		Present index level in process
534	216	2	PLHXLEVP		Previous level index
536	218	4	PLHXPTRP		Previous entry's P field
536	218	2	PLHXEOP		Previous entry offset
538	21A	2	PLHXSEOP		Previous section entry offset
540	21C	4	PLHXRBAP		Previous index record RBA
			Read-Ahead Index PLH		
544	220	28	PLHBINDX		Read-ahead index PLH
			Read-Ahead Index CI Description		
544	220	20	PLHBXCNV		Read-ahead index buffer manager parameter list
544	220	4	PLHBXRBA		Index CI RBA
548	224	8	PLHBXBUF		Index buffer description
548	224	4	PLHBXBCB		Address of index BCB
552	228	4	PLHBXBAD		Address of index buffer
556	22C	4	PLHBXCDF		Index CI CIDF
556	22C	2	PLHBXFSO		Index CI free space offset
558	22E	2	PLHBXFSL		Index CI free space length
560	230	1	PLHBXSW		Index CI switches (same as PLHDSW)
561	231	1	PLHBXSW1		Buffer request control switch (same as PLHDSW1)
562	232	2	PLHBXCSZ		Index CI size minus 10

Figure 5.36 **Placeholder (PLH) description and format (part 8 of 9)**

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
			Read-ahead Index Entry Description		
564	234	2	PLHBXEO		Index entry offset
566	236	2	PLHBXSEO		Next section entry offset
568	238	4	PLHBXSOP		Last section entry offset pointer
			Previous Record Key Information		
572	23C	*	PLHPKEY		Key of previous record
572	23C	0	PLHEND		End of PLH
			PLHLKSDS		Length of PLH for KSDS (equate value)

* Variable, equal to key length.

Figure 5.36 Placeholder (PLH) description and format (part 9 of 9)

Request Parameter List (RPL)

The RPL contains user-request information and error feedback information. It also maintains information required by GET and PUT. The RPL is created by the user with the RPL macro instruction.

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
0	0	1	RPLID		Control block identifier = X'00'
0	0	1	RPLIDD	X'00'	RPL equate
1	1	1	RPLSTYP		Release indicator
			RPLSDV1	X'00'	DOS/VS VSAM Release 1
			RPLSVSE1	X'10'	VSE/VSAM Release 1
				X'20'	VTAM
2	2	2	RPLLEN		Length of RPL
4	4	4	RPLRBA		RBA of last record processed
4	4	4	RPLDDDD		DD field
8	8	4	RPLARG		Pointer to search argument
12	C	8	RPLRCD		Record description
12	C	4	RPLAREA		Address of the caller's work area
16	10	4	RPLRLEN		Length of record
20	14	4	RPLBUFL		User buffer size
24	18	4	RPLACB		Address of the caller's ACB
24	18	4	RPLDACB		Catalog compatibility
28	1C	1	RPLSTRID		RPL string identifier
29	1D	1	RPLREQ		Request type*
			RPLPOINT	X'00'	POINT request
			RPLGET	X'04'	GET request
			RPLERASE	X'08'	ERASE request
			RPLPUT	X'0C'	PUT request
			RPLUPDTE	X'0C'	Update request
			RPLINSRT	X'10'	Insert request
			RPLCHECK	X'14'	Check request
			RPLRCLSE	X'18'	RCLOSE request
			RPLENDRQ	X'1C'	ENDREQ request
			RPLFRCIO	X'1C'	FORCIO request
			RPLVERFY	X'20'	VERIFY request
			RPLPUTL	X'24'	PUT locate request
			RPLWRBFR	X'2C'	Write buffer request
30	1E	2	RPLKEYL		Key length
32	20	2	RPLOPTCD		Option codes

* This value may be altered internally by VSAM, for example, X'24' from application program is changed to X'0C' by IKQRQA

Figure 5.37 **Request Parameter List (RPL) description and format (part 1 of 4)**

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
32	20	1	RPLOPT1		First byte of options
			RPLKEY	X'80'	Keyed access
			RPLADR	X'40'	Addressed access
			RPLSEQ	X'20'	Sequential
			RPLDIR	X'10'	Direct processing
			RPLASY	X'08'	Asynchronous
			RPLSKP	X'04'	Skip sequential access
			RPLCNV	X'02'	CNV access (RBA)
			RPLUPD	X'01'	Update
33	21	1	RPLOPT2		Second byte of options
			RPLKGE	X'80'	Search key greater than or equal
			RPLGEN	X'40'	Generic key request
			RPLNSP	X'20'	Note string position
			RPLNUP	X'10'	No update
			RPLLOC	X'08'	Locate mode
			RPLUBF	X'04'	User buffers
			RPLBWD	X'02'	Backward processing
			RPLLRD	X'01'	Last record processing
34	22	1	RPLHLD2	X'FF'	Second test and set byte (RPL not available)
				X'00'	RPL available
35	23	1	RPLHLD	X'FF'	Test and set byte (RPL held - request not completed)
				X'00'	Request completed
36	24	1	RPLFLAG		Flag byte
			RPLECBPR	X'80'	CMS ECB indicator
37	25	3	RPLFDBK		Error feedback area
37	25	1	RPLFDB1		Error class (return) code
37	25	1	RPLRTNCD		Error class code
Error class codes (stored from Register 15)					
			RPLNOERR	X'00'	No error detected
			RPLNORPL	X'04'	RPL held by another request
			RPLLOGER	X'08'	Logical error
			RPLPHYER	X'0C'	Physical error
			RPLVABND	X'3C'	TP I/O prohibited
38	26	1	RPLFDB2		Function type code
38	26	1	RPLFTNCD		Function type code
			RPLFUPG	X'04'	Upgrade processing
			RPLFAIX	X'02'	AIX processing
			RPLFINC	X'01'	Upgrade set is incorrect
39	27	1	RPLFDB3		Error type code
39	27	1	RPLERRCD		Error type code
39	27	1	RPLERCD		Error type code
39	27	1	RPLFDBKC		Error type code
The following equates are for the various feedback returns that may be set for offset 39 (27). They fall into the three categories shown.					
Returns that are not errors (Register 15 = X'00')					
			RPLEOV	X'04'	EOV called during request
			RPLDPKEY	X'08'	Duplicate key (in AIX record)
			RPLNEWCA	X'10'	Index full - CA split required.
			RPLCIWNG	X'1C'	Possible duplicate records in this CI (address processing of KSDS)

Figure 5.37 Request Parameter List (RPL) description and format (part 2 of 4)

Offset Dec	 Hex	 Bytes	 Field Name	Hex. Digit	 Description
Logical errors (register 15 = X'08')					
			RPLEOFDS	X'04'	End of data set encountered
			RPLEODER	X'04'	End of data set encountered
			RPLDUPRC	X'08'	Duplicate record
			RPLDUP	X'08'	Duplicate record
			RPLSEQCK	X'0C'	Sequence error
			RPLNRFND	X'10'	No record found
			RPLNOREC	X'10'	No record found
			RPLEXCTL	X'14'	Data already in exclusive control
			RPLNVOLM	X'18'	Volume or extent unavailable
			RPLNRSPA	X'1C'	No DASD space available
			RPLNOEXT	X'1C'	No DASD space available
			RPLSPACE	X'1C'	No DASD space available
			RPLINRBA	X'20'	Invalid RBA specified
			RPLNKEYR	X'24'	No key range for new record
			RPLNOVIR	X'28'	Insufficient virtual storage
			RPLWRKAS	X'2C'	User's work area not large enough
			RPLCDLOD	X'30'	CDLOAD failure
			RPLVLERR	X'34'	Internal VSAM logic error
			RPLNOPLH	X'40'	PLH in use (no string available)
			RPLNOPEN	X'44'	Access type not requested at Open
			RPLKEYES	X'48'	Keyed request for ESDS
			RPLADRKS	X'4C'	ADR or CNV insert for KSDS
			RPLINERS	X'50'	Illegal ERASE request
			RPLINLOC	X'54'	Illegal locate mode specification
			RPLNOPOS	X'58'	Positioning error
			RPLNGUPD	X'5C	No valid GET UPD issued
			RPLUPDKC	X'60'	Key change during update
			RPLLENCN	X'64'	Length change for addressed update
			RPLCONOP	X'68'	Improper or conflicting RPL options
			RPLIMRCL	X'6C'	Improper RECLEN specified
			RPLIMGKL	X'70'	Improper generic key length specified
			RPLINLD	X'74'	Illegal request during data set load
			RPLCATLG	X'80'	Internal catalog call failure
			RPLSRLOC	X'84'	Illegal locate mode
			RPLSRADR	X'88'	Illegal request for spanned record
			RPLINCSR	X'8C'	Inconsistent spanned record
			RPLNOBAS	X'90'	No base record
			RPLMAXPT	X'94'	Maximum of pointers exceeded
			RPLNOBUF	X'98'	No buffers available (LSR only)
			RPLINCNV	X'9C'	Invalid CI, possibly duplicate data addressed using address mode for update
			RPLINVRR	X'C0'	Invalid relative record number
			RPLRRADR	X'C4'	Illegal address requested (RRDS)
			RPLIPATH	X'C8'	Illegal path access
			RPLINBWD	X'CC'	Illegal backward mode requested

Figure 5.37 **Request Parameter List (RPL) description and format (part 3 of 4)**

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
Physical errors (register 15 = X'0C')					
			RPLRDERD	X'04'	Data read error
			RPLRDERI	X'08'	Index read error
			RPLRDERS	X'0C'	Sequence set read error
			RPLWTERD	X'10'	Data write error
			RPLWTERI	X'14'	Index write error
			RPLWTERS	X'18'	Sequence set write error
40	28	4	RPLCHAIN		Pointer to next RPL
44	2C	1	RPLAIXID		AIX information byte
			RPLAXPKP	X'01'	Prime key pointers are used (base cluster is a KSDS)
45	2D	1			Reserved
46	2E	2	RPLAIXPC		Number of base cluster pointers in the AIX record
48	30	1	RPLXID		Transaction ID
49	31	3			Reserved
52	34	0	RPLEND		End of RPL

Figure 5.37 **Request Parameter List (RPL) description and format (part 4 of 4)**

Resource Pool Header (RPHD)

The VSAM Resource Pool Header contains general information concerning the VSAM Resource Pool. Its address is stored in the AMBL.

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
0	0	2	RPHDASTR		Number of active strings
2	2	2	RPHDMSTR		Maximum number of strings ever simultaneously active
4	4	2			Reserved
6	6	2	RPHDSTNO		Total number of PLHs in the resource pool

Figure 5.38 **Resource Pool Header (RPHD) description and format**

Resource Sharing Control Block (RSCB)

The Resource Sharing Control Block provides exclusive control facilities for the VSAM shared resources. The VSRT points to the RSCB.

Offset Dec	 Hex	Bytes	Field Name	Hex. Digit	Description
0	0	4	RSCBRBA		RBA to be held in exclusive control
4	4	7	RSCBDSID		Data set identifier
4	4	4	RSCBCAT		Catalog ACB pointer
8	8	3	RSCBDSCI		CI number of data set component
11	b	1	RSCBCSID		ID of string using field RSCBGATE (for a control area split)
12	C	4	RSCBECB		ECB used by IKQBFA and IKQCAS
12	C	1			Reserved
13	D	1	RSCBSTID		ID of string (set by IKQBFA)
14	E	1	RSCBCOM		Communication byte
			RSCBWAIT	X'80'	Wait flag
15	F	1	RSCBGATE		Exclusive control byte: X'00' = ECB is free X'FF' = ECB is in use

Figure 5.39 **Resource Sharing Control Block (RSCB) description and format**

The Hold Block (THB) for SHAREOPTIONS (4)

The THB contains information necessary to lock a control area of a SHAREOPTIONS (4) file during updates and inserts. The PLH points to the THB. The THB is created by Open and released by Close and by Delete VSAM Resource Pool (DLVRP).

Offset				Hex.	
Dec	**Hex**	**Bytes**	**Field Name**	**Digit**	**Description**
0	0	1	THBID	X'88'	Control block identification
1	1	1	THBFLGBY		Flag byte
			THBACTV	X'80'	This THB is active for a SHAREOPTIONS (4) hold on a control area.
			THBPSUDO	X'40'	The hold represented by this THB is a "pseudo hold" of the same control area that is also held by another THB. The other THB has the THBREAL bit on.
			THBREAL	X'20'	This THB really holds the control area (the supervisor LOCK macro has been issued).
			THBNOTPR	X'10'	Not processing a LOCK request for this THB. This bit is intended as a potential problem determination aid. If it is off, processing for a LOCK macro is being done for this THB, and the THB is probably in LOCK wait.
2	2	2	THBLEN		Length of THB
4	4	4	THBNTHB		Address of the next THB (the CA-split THB)
8	8	16	THBLK		Lock description and status information for special locks (locks other than the SHAREOPTIONS (4) control area locks)
8	8	1	THBLSTAT		Status flags for special locks
			THBLKUSB	X'80'	Lock is set for USB serialization
			THBNOTUS	X'40'	Same function as THBNOTPR, but pertaining to the USB serialization lock. If this bit is off, processing is being done for a LOCK macro to set the lock for USB serialization, and the request is probably in LOCK wait.
			THBLKSP	X'20'	Lock is set for Record Management allocation of space within a file - for allocation of a new control area
			THBNOTSP	X'10'	Same function as THBNOTPR, but pertaining to the Record Management space lock. If this bit is off, processing is being done for a LOCK macro to set the Record Management space lock, and the request is probably in LOCK wait.
9	9	3			Reserved

Figure 5.40 **The Hold Block (THB) for SHAREOPTIONS (4) description and format (part 1 of 2)**

Offset				Hex.	
Dec	Hex	Bytes	Field Name	Digit	Description
12	C	12	THBNM		Lock name passed to the Supervisor LOCK service. This field is used for both the SHAREOPTIONS (4) control area locks and for the special locks.
12	C	10	THBNMBAS		LOCK name - basic part
12	C	1			Lock name prefix ('V') that identifies VSAM
13	D	6			Volume identifier of the catalog volume that owns the file
19	13	3			CI number of the catalog record that defines the file.
22	16	2	THBNMCA		The CA number within the data component or CI number within the index component that is being locked, with 1024 added to it.
24	18	8	THBSTAMP		Time stamp when Supervisor LOCK request issued - in time-of-day clock format
24	18	4	THBSTAM0		High order word of time stamp
28	1C	4	THBSTAM4		Low order word of time stamp
32	20	4	THBRBA		RBA of CA locked
36	24		THBDTL		Define-The-Lock, for Supervisor LOCK service

Figure 5.40 **The Hold Block (THB) for SHAREOPTIONS (4) description and format (part 2 of 2)**

Upgrade Set Block (USB)

The USB declaration maintains information required by PUT and ERASE requests to the base cluster. The USB is created by OPEN (IKQOPNUS).

Offset				Hex.	
Dec	**Hex**	**Bytes**	**Field Name**	**Digit**	**Description**
0	0	1	USBID USBIDD	 X'E0'	USB identifier USB equate
1	1	1	USBACT		Active byte, test and set
2	2	2	USBLEN		Length of this block
4	4	2	USBMAXDB		Max. data buffer in upgrade set
6	6	2	USBMAXIB		Max. index buffer in upgrade set
8	8	4	USBWAPTR		Pointer to work area pool
12	C	2	USBMIN		Min. required record length
14	E	2	USBWALEN		Work area length
16	10	4	USBPLH		Pointer to common USB PLH
20	14	4	USBRPL		Pointer to RPL
24	18	4	USBDBHD		Pointer to data buffer header
28	1C	4	USBIBHD		Pointer to index buffer header
32	20	4	USBDTL		Pointer to DTL (Define-The-Lock) for serializing use of the USB.
			Begin of First/only Index Entry		
36	24		USBAIX		
		4	USBACB USBLAST	 X'80'	Pointer to ACB Last entry indicator
40	28	2	USBRKP		Relative key position
42	2A	2	USBKL		Key length
			Further Alternate Index entries		
44	2C	variable			

Figure 5.41 **Upgrade Set Block (USB) description and format**

Section 6. Diagnostic Aids

This chapter provides several aids that can be useful when trying to diagnose difficulties with VSAM modules. These aids include:

- A list of VSAM lock resource names (Figure 6.1) and their associated use by VSAM.
- A chart (Figure 6.2) showing the lock option/control for locking various types of files
- A list of macro instructions (Figure 6.3) issued by VSAM users, modules or other macros and their use.
- Cross reference tables (Figure 6.4) showing the VSAM modules and the macros they issue.
- A list of error codes (Figure 6.5) set in the RPL which indicate record management errors. The list shows also the relationship between internal and external error codes.
- A list of error codes (Figure 6.6) showing record management modules and the error code(s) they might issue.
- A list of error codes (Figure 6.7) showing record management modules and the error code(s) they might issue when manipulating control blocks.
- A description of service aid phases and how to use them.

Additional Aids

Further aids can be found in other parts of the book and in the program listings. These include:

- Register contents on entry to a module, which are under *Input* in the module prologues.
- Use of registers and equated names for registers, which can be found under *Notes* in the module prologues.
- Error codes, which are under *Exit-Error* in the module prologues.
- A list, which is in the *Directory*, of modules, their component, their entry points, and their associated method of operation and program structure diagrams.
- A cross-reference list, which is in the *Directory*, of catalog external entry points and their associated modules.

VSAM Use of Locks

Figure 6.1 is a list of the lock resource names used by VSAM and their associated functions.

Resource Name	Function
V.addr.CAX.X‘0000’	Serialize access on the C/M CAXWA chain during delete, update, or search operations on the chain.
V.OAL.X‘0000000000000000’	Maintains integrity of OAL by serializing access through OPEN/CLOSE.
V.SYSMCO.X‘0000000000’	Serialize master catalog define and open.
V.SYSOPEN.X‘00000000	Serialize OPEN, CLOSE, DELETE, and DEFINE access to the catalog (e.g. OPEN indicator) and synchronize the catalog with share options locks.
V.volser.ci#.X‘0000’[1]	File lock - Used to enforce SHAREOPTIONS protection for components of a file. The name volser.ci# uniquely identifies a component being protected. The volser is the serial number of the volume containing the catalog describing the component and ci# is the number of the control interval in the catalog where the component is being described.
V.volser.ci#.X‘0001’	Outcount lock - Maintains a count of output users of the file denoted by volser.ci#. This lock is maintained for SHAREOPTIONS(3) and SHAREOPTIONS(4) files.
V.volser.ci#.X‘0002’	Keyed access lock - Represents keyed access for output to a SHAREOPTIONS(4) file. It is used together with the address access lock to prevent concurrent keyed access and address access for output to a SHAREOPTIONS(4) file.
V.volser.ci#.X‘0003’	Address access lock - Represents address or CNV access for output to a SHAREOPTIONS(4) file.
V.volser.ci#.X‘0004’	Used by Record Management to serialize use of the Upgrade Set Block (USB) when the ACB has been opened with multiple strings.
V.volser.ci#.X‘0005’	Used by Record Management to serialize allocation of control areas within an extent of a SHAREOPTIONS(4) file.
V.volser.X‘0000000006’	Volume mount serialization - Used to synchronize mount requests for a given volume.
V.volser.ci#.X‘nnnn’	Used for Record Management basic SHAREOPTIONS(4) locks on control areas, where nnnn is the CA number (CI number for index component) plus 1024.
V.volser.UPL.X‘0000’	Serialize master and user catalog update and locate functions.

(Note: The period (.) as used in this list of lock resource names, represents concatenation only and is not part of the lock resource name.)

[1] The file lock is maintained by open/close using OWNER=PARTITION so that an ACB may be closed by a different task than the opening task. Figure 6.2 shows which lock option/control is used for locking various types of files.

Figure 6.1 **VSAM Use of Locks**

File being opened for:	File defined share option			
	1	2	3	4

INPUT	LOCKOPTION=1
	CONTROL=SHARED
		LOCKOPTION=2
		CONTROL=SHARED
			LOCKOPTION=3
			CONTROL=SHARED
				LOCKOPTION=4
				CONTROL=SHARED

OUTPUT	LOCKOPTION=1
	CONTROL=EXCLUSIVE
		LOCKOPTION=2
		CONTROL=EXCLUSIVE
			LOCKOPTION=3
			CONTROL=EXCLUSIVE
				LOCKOPTION=4
				CONTROL=EXCLUSIVE

Figure 6.2 Lock option/control for locking various types of files

Macro-to-Module Relationships

The following list in Figure 6.3 contains the macro instructions issued by VSAM users, modules, or other macros. Their types are identified as follows:

G	–	generating macro
SA	–	VSE action macro
M	–	mapping macro
I	–	internal (called by another macro)
A	–	VSAM action macro
S	–	copy source book macro

Macro	Type	SVC	Use
ACB	G		Generate an ACB
ASYSCOM	SA		Get address of systems communications region
AVRLIST	M		With DCTENTRY, map device characteristics
CANCEL	SA		Cancel a task
CATLG	A		Load address of catalog parameter list (CTGPL) into R1 and invoke catalog management
CCB	G		Build a CCB
CDLOAD	SA	65	Load module(s)
CLOSE	SA	2	Disconnect a user's program from a VSAM data set
COMRG	SA	33	Get communication region address
CVTOC	SA		CVH close VTOC
DCTENTRY	M		With AVRLIST, map device characteristics
DEQB	SA		Free B-transient
DTFCN	SA		SYSLOG DTF
ENDREQ	SA		Free a PLH and terminate processing on associated string
ENQB	SA		Hold B-transient
EOJ	SA		End of job
ERASE	SA		Delete a record
EXCP	SA	0	Execute channel program
EXLST	G		Generate EXLST
EXTRACT	SA	98	Get control block information from supervisor
FREEVIS	SA	62	Free virtual storage
GENCB	A		Generate a control block
GENDTL	SA		Generate a DTL (Define-the-Lock parameter list for the LOCK and UNLOCK macros)
GET	SA		Retrieve a record
GETFLD	SA	107	Get specified field value
GETVCE	SA	99	Get device characteristics
GETVIS	SA	61	Get virtual storage
IDCDF60	M		Map Access Method Services catalog communication table (ACC), catalog CI number to CRA CI number translation table (CTT), and volume timestamp table (VTT).
IGGCAXWA	M		Map catalog auxiliary work area (CAXWA)
IGGCCA	M		Map catalog communications area (CCA)
IGGMCDCL	M		Issue the following macros to define the commonly used declarations for VSAM catalog management modules: IGGCAXWA, IGGCCA, IGGMCTRC, IKQACB, IKQAMCBS, IKQCOMRG, IKQCTGFL, IKQCTGFV, IKQCTGPL, IKQVRGN
IGGMCMDM	M		Map the VSAM catalog management commonly used record structures
IGGMCMWA	M		Map the VSAM catalog management services work area
IGGMCTRC	M		Map catalog return codes
IGGMDLWA	M		Delete work area layout
IGGMDRWA	M		Map the VSAM catalog VTOC label read-in work area
IGGMDVCH	M		Map VSAM catalog management device characteristics
IGGMEND	G		Generate exit code at the end of catalog management modules
IGGMFDNM	M		VSAM catalog dictionary information for external field names
IGGMGVO	M		Map the volume information group occurrence

Figure 6.3 **Macro types and uses (part 1 of 4)**

Macro	Type	SVC	Use
IGGMNAME	I		Generate catalog module name for error and reason codes
IGGMODUL	G		Generate header code for catalog modules
IGGMPROC	G		Generate header code for catalog internal procedures
IGGMSAWA	M		Map the VSAM catalog management suballocate work area
IGGMUPDE	M		Issue IGGMVEDC, IGGMCDCL, IGGMCMDM, IKQAMDSB, and IGGMSAWA to define the commonly used declarations for VSAM catalog management Update-Extend modules
IGGMVEDC	M		Map the volume catalog record
IIPAMDTF	G/M		Generate/map AMDTF table
IIPDTF	G/M		Generate/map DTF table
IIPPRAT	G/M		Generate/map address list
IJBLBRC	M		Map label area record
IJJHCPL	M		Map CVH Parameter List
IJJHDLST	M		Map CVH Volume Descriptor List
IJJHFMT1	M		Map format-1 VTOC label
IJJHFMT3	M		Map format-3 VTOC label
IJJHFMT4	M		Map format-4 VTOC label
IKQACB	M		Map ACB
IKQACBG	I		Generate ACB (called by IKQACB1)
IKQACB1	I		Generate ACB (called by ACB)
IKQAIR	M		Map alternate index record
IKQAMBL	M		Map AMBL
IKQAMCBS	M		Map AMCBS
IKQAMDSB	M		Map AMDSB
IKQARDB	M		Map ARDB
IKQAREX	M		Map EXLST argument entry
IKQARGH	M		Map argument header
IKQASGN	A		Invoke automatic assign function
IKQBHD	M		Map buffer header
IKQBKPHD	M		Map header for CCW area
IKQBLARD	G		Build an ARDB
IKQBUFE	M		Map BCB
IKQCBMTB	G		Define table of constants for control block generation modules
IKQCB1	I		Transform operands for control block manipulation macro instructions GENCB, TESTCB, MODCB, SHOWCB, IKQCB2, and IKQERMAC
IKQCB2	I		Scan keywords and generate code for control block manipulation macro instructions GENCB, TESTCB, MODCB, SHOWCB, IKQCB1, and IKQERMAC
IKQCCB	M		Map IORB
IKQCCBCW	M		Map IORB
IKQCCW	M		Map CCW
IKQCGETC	S		Obtain storage in which to copy old ARDB
IKQCIW	M		Map control interval split work area
IKQCLCOR	S		Get address of space in which to build EDB(s)
IKQCLNUP	S		Disconnect ACB and AMBL
IKQCLRLS	S		Free storage obtained by Open and/or EOV
IKQCLWA	M		Close work area

Figure 6.3 **Macro types and uses (part 2 of 4)**

Macro	Type	SVC	Use
IKQCOMB	G		Generate a combination name entry for the VSAM catalog dictionary
IKQCOMRG	M		Map communication region
IKQCTGFL	M		Map field parameter list (CTGFL)
IKQCTGFV	M		Map catalog field vector table (CTGFV)
IKQCTGPL	M		Map catalog parameter list (CTGPL)
IKQCWS	M		Map CCW skeletons
IKQDDR	M		Map duplicate data recovery work area
IKQDEVT	A	65	Read label area and/or determine the device type for the file-ID (IKQDEVT uses CDLOAD)
IKQECB	M		Map Event Control Block
IKQEDB	M		Map EDB
IKQEDBLD	G		Build EDB
IKQEQU	M		Map register equates
IKQERC	M		Internal error codes equate
IKQERMAC	I		Issue M-notes (assembler macro error messages) for control block manipulation macro instructions GENCB, TESTCB, MODCB, SHOWCB, IKQCB1, and IKQCB2
IKQEXLG	I		Generate EXLST (called by IKQEXL1)
IKQEXLST	M		Map EXLST
IKQEXL1	I		Generate EXLST (called by EXLST)
IKQEXP	M		Description of EXPAD parameter list
IKQFCDB	M		Map CCW blocks in CCW pool
IKQFNDLB	S		Find LUB (logical unit block) for symbolic unit
IKQFXL	M		Map Fix List (used with IORB)
IKQGCB	I		Generate a control block (called by GENCB)
IKQDTL	G		Generate a DTL for use by the lock manager
IKQIOARG	M		Map DASD address
IKQIODRB	M		Map I/O driver block
IKQIORQU	M		Map register equates
IKQIOWKA	M		Map I/O work area in PLH
IKQIXHDR	M		Map index record header
IKQJIB	M		Map JIB (job information block)
IKQJRNDS	M		Parameter list for journalling
IKQKWTB	G		Define table of constants for control block manipulation modules
IKQLOCK	A	63	Lock a system resource or file by means of the LOCK macro
IKQLPMB	M		Map LPMB
IKQLUB	M		Map logical unit block
IKQMCB	I		Modify a control block (called by MODCB)
IKQMDADS	M		Map DADSM parameter list (interface block to DADSM)
IKQMSGPL	M		OPEN/CLOSE message primary list
IKQOAL	M		Open ACB list
IKQOCFSP	I		Free space for DSA
IKQOCGSP	I		Get space for DSA
IKQOCPRC	S		Connect dynamic storage area
IKQOPCLR	M		Register equates
IKQOPCLW	M		Map of common section of work area
IKQOPLCT	M		Map fields located by catalog
IKQOPNWA	M		Map Open Work Area
IKQPARM	M		Map Buffer Manager Parameter List

Figure 6.3 **Macro types and uses (part 3 of 4)**

Macro	Type	SVC	Use
IKQPLH	M		Map PLH
IKQPUB	M		Map physical unit block
IKQRDF	M		Map RDF and CIDF fields
IKQRLSE	M	64	Dequeue a system resource by means of a RELEASE macro
IKQRPL	M		Map RPL
IKQRPLG	I		Generate RPL (called by IKQRPL1)
IKQRPL1	I		Generate RPL (called by RPL)
IKQRQM	G		Generate modules IKQRQA and IKQRQB
IKQSCB	I		Display a control block (called by SHOWCB)
IKQSHRW	M		Map SHRW (File Sharing Work Area)
IKQTCB	I		Test a control block (called by TESTCB)
IKQTHB	M		Map THB
IKQUSB	M		Upgrade set block
IKQUSE	A	63	Enqueue a system resource by means of a USE macro
IKQUNLK	M		Unlock a system resource of file by means of the UNLOCK macro
IKQVLST	M		Map list of volume unit, symbolic unit, and volume time stamp
IKQVOL1	M		Map volume-1 label
IKQVRGN	M		Map anchor table
IKQVRPPL	M		Map parameter list for BLDVRP function (IKQBRP)
IKQVSMDP	A		Map VSAM dump
LABEL	SA		Interface macro to call symbolic label access
LPLDCT	M		Map of label parameter list
LOAD	A		Load a phase
LOCK	SA	110	Serialize on a named resource
MAPBDY	M		Map for partition boundaries
MAPCOMR	M		Map partition COMREG layout
MAPPIB	M		Map program information block
MODCB	A		Modify a control block
MODDTL	SA		Modify a DTL
MODFLD	SA		Modify a specified field value
OPEN	SA	2	Connect a user's program to a VSAM data set
OVTOC	SA		CVH open VTOC
POINT	SA		Position VSAM at a record
POST	SA		Post an ECB
PUT	SA		Store a new or updated record
PVTOC	SA		CVH process VTOC
RPL	G		Generate an RPL
SHOWCB	A		Display a control block
SYSCOM	M		Map system communication region layout
TCLOSE	SA		Purge buffer and update catalog (no disconnect)
TESTCB	A		Test a control block
UNLOCK	SA	110	Release serialization on a named resource
VERIFY	A		Build calling sequence for VSAM function VERIFY
WAIT	SA	7	Wait on a CCB for I/O to complete

Figure 6.3 **Macro types and uses (part 4 of 4)**

Macro / Module	ACB	CANCEL	CATLG	CCB	CDLOAD	COMRG	EXCP	FREEVIS	GENDTL	GETVIS	IKQACB	IKQAIR	IKQAMBL	IKQAMDSB	IKQARDB	IKQBHD	IKQBKPHD	IKQBLARD	IKQBSPH	IKQBUFE
IKQAIX											X	X	X	X						
IKQBFA													X	X		X			X	X
IKQBFB											X		X	X		X	X		X	X
IKQBFC00													X	X						
IKQBFD											X		X	X		X	X			X
IKQBLD																				
IKQBRP					X			X		X						X	X		X	X
IKQCAS					X			X	X				X	X	X					X
IKQCIL								X	X				X	X	X					X
IKQCIR					X								X	X	X					X
IKQCIS					X			X		X			X	X	X	X				X
IKQCIU													X	X	X					X
IKQDDR										X			X	X						X
IKQEDX			X		X	X		X		X			X	X	X			X		
IKQEOV		X			X	X		X		X			X	X	X					
IKQERH					X						X		X	X						
IKQERX					X						X		X							
OKQGCI													X	X						X
IKQGNX00													X	X	X	X				X
IKQGPT													X	X	X					X
IKQINT													X	X						
IKQIOA					X					X	X		X	X		X	X		X	X
IKQIOB											X			X		X	X			X
IKQIOC					X					X	X		X	X		X	X		X	X
IKQIOD							X			X	X		X	X		X	X		X	X
IKQIXE00			X		X			X		X	X		X	X						X
IKQIXF00													X	X						
IKQIXS00														X						X
IKQJRN					X					X	X		X	X						
IKQKRD														X	X					X
IKQLCD													X	X	X					X
IKQLCN														X						
IKQLNA													X	X						
IKQLCP													X	X	X	X				X
IKQMDY													X	X						X
IKQNCA00					X			X		X	X		X	X	X	X				X

Figure 6.4 **Macro-to-module relationships for record management and EOV modules (part 1 of 6)**

Macro Module	IKQCCB	IKQCCW	IKQCGETC	IKQCIW	IKQCLCOR	IKQCOMRG	IKQCTGFL	IKQCTGPL	IKQCWS	IKQDDRW	IKQECB	IKQEDB	IKQEDBLD	IKQEQU	IKQERC	IKQEXLST	IKQEXP	IKQFCDB	IKQFNDLB	IKQFXL	IKQIOARG	IKQIODRB	IKQIOEQU	IKQIOWKA	IKQIXHDR	IKQJRNDS	IKQLPMB	IKQLUB
IKQAIX															X													
IKQBFA															X								X	X				
IKQBFB															X								X					
IKQBFC00																							X		X			
IKQBFD											X												X	X				
IKQBLD																												
IKQBRP																												
IKQCAS				X											X										X			
IKQCIL				X																					X			
IKQCIR				X																					X			
IKQCIS				X											X										X			
IKQCIU				X																					X			
IKQDDR										X					X										X			
IKQEDX			X		X	X	X	X				X	X						X								X	X
IKQEOV						X						X							X									X
IKQERH															X													
IKQERX																X												
IKQGCI															X													
IKQGNX00												X			X								X		X			
IKQGPT															X													
IKQINT																												
IKQIOA	X	X							X		X	X			X			X		X	X	X	X	X			X	
IKQIOB	X	X													X			X					X	X				
IKQIOC	X	X							X		X	X			X			X		X	X	X	X	X			X	
IKQIOD	X	X									X				X	X	X	X		X			X	X				
IKQIXE00				X			X	X							X										X			
IKQIXF00				X								X													X			
IKQIXS00																									X			
IKQJRN				X											X	X										X		
IKQKRD												X			X													
IKQLCD																									X			
IKQLCN															X													
IKQLNA																							X		X			
IKQLCP												X			X								X		X			
IKQMDY															X													
IKQNCA00				X								X			X										X			

Figure 6.4 Macro-to-module relationships for record management and EOV modules (part 2 of 6)

Macro / Module	IKQPARM	IKQOPCLR	IKQOPCLW	IKQOPNWA	IKQPLH	IKQRDF	IKQRPHD	IKQRPL	IKQRQM	IKQRSCB	IKQSHRW	IKQTHB	IKQUSB	IKQVLST	IKQRPPL	IKQVSMDP	IKQVSRT	LOCK	MODDTL	POST	UNLOCK	WAIT
IKQAIX					X			X														
IKQBFA	X				X	X		X		X		X				X				X		X
IKQBFB	X				X		X	X					X							X		X
IKQBFC00	X				X	X		X				X						X	X	X	X	X
IKQBFD					X			X												X		X
IKQBLD					X			X														
IKQBRP					X		X			X					X		X					
IKQCAS					X	X		X			X											
IKQCIL					X	X		X			X											
IKQCIR					X			X														
IKQCIS					X	X		X												X		X
IKQCIU					X	X		X														
IKQDDR	X				X	X		X												X		X
IKQEDX		X	X											X								
IKQEOV		X	X											X								
IKQERH					X			X								X						
IKQERX					X			X														
IKQGCI					X	X		X														
IKQGNX00	X				X	X		X														
IKQGPT					X			X														
IKQINT					X		X	X														
IKQIOA					X			X		X										X		X
IKQIOB					X			X								X						
IKQIOC	X				X			X		X										X		X
IKQIOD					X			X		X						X				X		X
IKQIXE00					X	X			X													
IKQIXF00				X	X	X																
IKQIXS00					X				X													
IKQJRN					X	X			X													
IKQKRD					X				X													
IKQLCD					X				X													
IKQLCN					X	X			X													
IKQLCP	X				X	X			X													
IKQLNA					X			X										X		X		X
IKQMDY					X				X												X	X
IKQNCA00					X	X			X													

Figure 6.4 **Macro-to-module relationships for record management and EOV modules (part 3 of 6)**

Macro Module	ACB	CANCEL	CATLG	CCB	CDLOAD	COMRG	EXCP	FREEVIS	GENDTL	GETVIS	IKQACB	IKQAIR	IKQAMBL	IKQAMDSB	IKQARDB	IKQBHD	IKQBKPHD	IKQBLARD	IKQBSPH	IKQBUFE
IKQNEX			X		X	X		X		X	X		X	X	X			X		
IKQOCMSG			X	X	X		X				X									
IKQPBF00													X	X		X				X
IKQPFO00					X			X		X			X	X	X					X
IKQRBA			X		X			X		X			X	X	X					
IKQRCL00								X		X			X	X	X					X
IKQRQA											X		X	X	X					
IKQRQB											X		X	X	X					
IKQRQC	X				X								X	X						
IKQRRP					X			X	X				X	X	X					X
IKQRTV													X	X						
IKQSCN														X						
IKQSFT																				
IKQSPM00					X								X	X	X					
IKQSRG													X	X						
IKQSRT													X	X	X					
IKQSRU													X	X	X					X
IKQUPD													X	X						X
IKQUPG								X		X	X	X	X	X	X					X
IKQVFY											X		X	X	X					
IKQVSM								X		X	X		X	X						

Figure 6.4 Macro-to-module relationships for record management and EOV modules (part 4 of 6)

Macro Module	IKQCCB	IKQCCW	IKQCGETC	IKQCIW	IKQCLCOR	IKQCOMRG	IKQCTGFL	IKQCTGPL	IKQCWS	IKQDDRW	IKQECB	IKQEDB	IKQEDBLD	IKQEQU	IKQERC	IKQEXLST	IKQEXP	IKQFCDB	IKQFNDLB	IKQFXL	IKQIOARG	IKQIODRB	IKQIOEQU	IKQIOWKA	IKQIXHDR	IKQJRNDS	IKQLPMB	IKQLUB
IKQNEX			X		X	X	X	X				X	X						X								X	X
IKQOCMSG						X	X	X																				
IKQPBF00												X		X														
IKQPFO00				X								X			X													
IKQRBA							X	X				X																
IKQRCL00				X								X			X											X		
IKQRQA															X										X			
IKQRQB															X										X			
IKQRQC															X													
IKQRRP				X								X			X													
IKQRTV															X													
IKQSCN																												
IKQSFT																												
IKQSPM00				X								X			X													
IKQSRG															X										X			
IKQSRT															X										X			
IKQSRU																									X			
IKQUPD															X													
IKQUPG															X												X	
IKQVFY																									X			
IKQVSM	X														X													

Figure 6.4 Macro-to-module relationships for record management and EOV modules (part 5 of 6)

Macro Module	IKQOPCLR	IKQOPCLW	IKQOPNWA	IKQPARM	IKQPLH	IKQRDF	IKQRPHD	IKQRPL	IKQRQM	IKQRSCB	IKQSHRW	IKQTHB	IKQUSB	IKQVLST	IKQRPPL	IKQVSMDP	IKQVSRT	LOCK	MODDTL	POST	UNLOCK	WAIT
IKQNEX	X	X												X								
IKQOCMSG	X	X																				
IKQPBF00					X																	
IKQPFO00					X	X		X														
IKQRBA	X																					
IKQRCL00				X	X			X														
IKQRQA					X			X	X				X									
IKQRQB					X			X	X				X									
IKQRQC					X		X	X					X							X		X
IKQRRP					X	X		X			X											
IKQRTV					X			X														
IKQSCN					X			X														
IKQSFT					X																	
IKQSPM00					X			X														
IKQSRG					X			X												X		X
IKQSRT					X			X														
IKQSRU					X	X		X														
IKQUPD					X			X														
IKQUPG					X	X		X					X									
IKQVFY					X			X														
IKQVSM					X			X												X		X

Figure 6.4 Macro-to-module relationships for record management and EOV modules (part 6 of 6)

Record Management Error Code-to-Module Relationship

There are internal and external error codes. Internal error codes are set in register 15 by record management modules and passed to IKQERH for handling. Three classes of internal error codes exist:

- Specification errors (with a value from X'01' to X'1F')
- Processing errors (with a value from X'20' to X'3F')
- I/O errors (with a value from X'40' to X'5F')

External error codes are set in the RPL (see section *Data Areas*) and register 15 by IKQERH, according to the internal error codes, and passed back to the user. Figure 6.5 shows the Record Management internal - external error code relationship.

Internal error codes (IKQERC macro)		**External error codes (IKQRPL macro)**			**Meaning**
Symbolic code	**R15**	**R15**	**Symbolic name**	**RPL**	
E01	X'01'	X'04'	-	-	RPL held by another request
E02	X'02'	X'02'	-	-	Reserved
E03	X'03'	X'08'	RPLNOPLH	X'40'	No PLH available
E04	X'04'	X'08'	RPLNOPEN	X'44'	CNV access not requested at open, or ADR access not requested at open
E05	X'05'	X'08'	RPLNOPEN	X'44'	Keyed access not requested
E06	X'06'	X'08'	RPLNOPEN	X'44'	Output not requested
E07	X'07'	X'08'	RPLRRADR	X'C4'	Invalid address requested
E08	X'08'	X'08'	RPLKEYES	X'48'	Keyed access requested for ESDS
E09	X'09'	X'08'	RPLADRKS	X'4C'	ADR or CNV insert for KSDS
E10	X'0A'	X'08'	RPLINERS	X'50'	Illegal ERASE request
E11	X'0B'	X'08'	RPLINLOC	X'54'	Illegal Locate mode specification
E12	X'0C'	X'08'	RPLINLD	X'74'	Illegal request during data set load
E13	X'0D'	X'08'	RPLNOPOS	X'58'	No keyed positioning done
E14	X'0E'	X'08'	RPLNOPOS	X'58'	No sequential positioning done
E15	X'0F'	X'08'	RPLNGUPD	X'5C'	No valid GET UPD issued
E16	X'10'	X'08'	RPLUPDKC	X'60'	Key change during update
E17	X'11'	X'08'	RPLLENCN	X'64'	Length change for addressed update
E18	X'12'				Reserved
E19	X'13'	X'08'	RPLCONOP	X'68'	Improper RPL-option (BWD)
E20	X'14'	X'08'	RPLCONOP	X'68'	Improper or conflicting RPL options, invalid transaction ID or LRU percentage value, or WRTBFR without LSR/DFR, or ARG parameter not specified when required.
E21	X'15'	X'08'	RPLIMGKL	X'70'	Improper generic key length
E22	X'16'	X'08'	RPLIMRCL	X'6C'	Improper RECLEN
E23	X'17'	X'08'	RPLINERS	X'50'	Invalid ERASE request (AIX)
E24	X'18'				Reserved
E25	X'19'	X'08'	RPLNOPOS	X'58'	Invalid switching FWD-BWD
E26	X'1A'				Reserved

Figure 6.5 Record Management internal/external error code relationship (part 1 of 3)

Internal error codes (IKQERC macro)		External error codes (IKQRPL macro)			Meaning
Symbolic code	R15	R15	Symbolic name	RPL	
E27	X'1B'	X'08'	RPLINVRR	X'C0'	Invalid RR number (RRDS)
E28	X'1C'	X'08'	RPLIPATH	X'C8'	Invalid path access (AIX)
E29	X'1D'	X'08'	RPLINBWD	X'CC'	Illegal PUT in BWD-mode
E30	X'1E'				Reserved
E32	X'20'	X'08'	RPLINRBA	X'20'	Invalid RBA
E33	X'21'	X'08'	RPLNRFND RPLNOREC	X'10'	No record found
E34	X'22'	X'08'	RPLEOFDS RPLEODER	X'04'	End of data set encountered
E35	X'23'	X'08'	RPLWRKAS	X'2C'	User's work area not large enough
E36	X'24'	X'08'	RPLSEQCK	X'0C'	Sequence error
E37	X'25'	X'08'	RPLDUPRC RPLDUP	X'08'	Duplicate record
E38	X'26'	X'08'	RPLNKEYR	X'24'	No key range for new record
E39	X'27'	X'08'	RPLNOVIR	X'28'	Insufficient virtual storage
E40	X'28'	X'08'	RPLNRSPA RPLNOEXT RPLSPACE	X'1C'	No DASD space available
E41	X'29'	X'08'	RPLNVOLM	X'18'	Volume or extent unavailable
E42	X'2A'	X'08'	RPLCDLOD	X'30'	CDLOAD failure
E43	X'2B'	X'08'	RPLVLERR	X'34'	No BCB available or invalid attempt to insert RRDS after preformat
E44	X'2C'	X'08'	RPLEXCTL	X'14'	Exclusive control failure
E45	X'2D'	X'08'	RPLCATLG	X'80'	Internal catalog call failure
E46	X'2E'	X'08'	RPLSRLOC	X'84'	Illegal GET in LOC-mode
E47	X'2F'	X'08'	RPLINCSR	X'8C'	Inconsistent spanned record
E48	X'30'	X'08'	RPLSRADR	X'88'	Illegal addr. retrieval for spanned records KSDS
E49	X'31'	X'08'	RPLNOBAS	X'90'	No base record for associated AIX pointer
E50	X'32'				Reserved
E51	X'33'	X'08'	RPLMAXPT	X'94'	Max. no of AIX pointers exceeded
E52	X'34'	X'08'	RPLNOBUF	X'98'	No buffers available (LSR)
E53	X'35'	X'08'	RPLINCNV	X'9C'	Invalid CI, possibly duplicate data accessed using address mode for update

Figure 6.5 Record Management internal/external error code relationship (part 2 of 3)

Internal error codes (IKQERC macro)		External error codes (IKQRPL macro)			Meaning
Symbolic code	R15	R15	Symbolic name	RPL	
E54	X'36'	X'08'	RPLNASN	X'38'	No programmer logical I/O units available for a dynamic assignment
E55	X'37'	X'08'	RPLEXCL1	X'D0'	LOCK (most often the share option 4 lock on a CA) previously requested under the same VSE task, but not under the same string-set.
E56	X'38'	X'08'	RPLNOLKS	X'D4'	VSAM received a return code from the LOCK macro indicating that no space is available in the lock table.
E57	X'39'	X'08'	RPLCAOV	X'D8'	For a share option 4 file, an RBA exceeded 64511 times the CA-size (64511 times the CI-size for an index component).
E64	X'40'	X'0C'	RPLRDERD	X'04'	Data read error
E65	X'41'	X'0C'	RPLRDERI	X'08'	Index read error
E66	X'42'	X'0C'	RPLRDERS	X'0C'	Sequence set read error
E72	X'48'	X'0C'	RPLWTERD	X'10'	Data write error
E73	X'49'	X'0C'	RPLWTERI	X'14'	Index write error
E74	X'4A'	X'0C'	RPLWTERS	X'18'	Sequence set write error

Figure 6.5 **Record Management internal/external error code relationship (part 3 of 3)**

Internal error codes	Module
-1	IKQBFC00, IKQCIS00, IKQGNX00, IKQLNA
-5	IKQIXE00, IKQIXF00
E01	IKQERH, IKQVSM
E03	IKQRQC, IKQVSM
E04	IKQRQB
E05	IKQRQB
E06	IKQRQB
E07	IKQRQB
E08	IKQRQB
E09	IKQRQB
E10	IKQRQB
E11	IKQRQB
E12	IKQRQB
E13	IKQRQB
E14	IKQRQB
E15	IKQRQB
E16	IKQRQB
E17	IKQRQB
E19	IKQRQB
E20	IKQGPT, IKQRQB
E21	IKQRQB
E22	IKQRQB, IKQUPG
E23	IKQRQB
E25	IKQRQB
E27	IKQRQB
E28	IKQRQB
E29	IKQRQB
E32	IKQAIX, IKQGPT, IKQIOA, IKQIOC, IKQRQB
E33	IKQAIX, IKQERH, IKQGPT, IKQUPG
E34	IKQAIX, IKQERH, IKQGCI, IKQGPT, IKQUPD, IKQUPG
E35	IKQRQB, IKQRTV, IKQSRG, IKQUPG, IKQVSM
E36	IKQGPT, IKQSRT
E37	IKQSRT, IKQUPG
E38	IKQKRD
E39	IKQCAS00, IKQCIL, IKQCIS00, IKQDDR, IKQEDX, IKQEOV, IKQIXE00, IKQJRN, IKQNCA00, IKQNEX, IKQPFO00, IKQRBA, IKQRCL00, IKQRQB, IKQRQC, IKQRRP, IKQUPG, IKQVSM
E40	IKQNEX
E41	IKQEDX, IKQIOA, IKQIOC
E42	IKQCAS00, IKQCIR, IKQCIS00, IKQEDX, IKQIOA, IOQIOC, IKQIXE00, IKQJRN, IKQNCA00, IKQNEX, IKQPFO00, IKQRQB, IKQRQC, IKQRRP, IKQSPM00, IKQVSM
E43	IKQBFA00, IKQBFC00, IKQIOB, IKQMDY
E44	IKQBFA00, IKQBFC00, IKQCIR, IKQSRG
E45	IKQEDX, IKQIXE00, IKQNCA00, IKQNEX, IKQPFO00, IKQRBA, IKQRRP

Figure 6.6 **Record Management internal error code-to-module relationship (part 1 of 2)**

Internal error codes	Module
E46	IKQJRN, IKQSRG
E47	IKQSRG
E48	IKQGPT
E49	IKQAIX
E51	IKQUPG
E52	IKQBFA00, IKQRQB, IKQRQC
E53	IKQDDR
E54	IKQEOV
E55	IKQBFC00
E56	IKQBFC00
E57	IKQGCI, IKQBFC00, IKQSPM00
E64	IKQBNX00, IKQIOB
E65	IKQIOA, IKQIOB
E66	IKQIOA, IKQIOB, IKQIOD
E72	IKQIOB
E73	IKQIOB
E74	IKQIOB

Figure 6.6 **Record Management internal error code-to-module relationship (part 2 of 2)**

Service Aids

Service aid phases are available for:

- Enabling and disabling snap dumps within the VSAM component.
- Obtaining snap dumps of control blocks.
- Using UPSI to obtain diagnostic information for the VSAM catalog.
- Maintaining DSCBs in the VTOC and VOL1 labels on DASD.
- Loading a VSAM phase or a program you have written.

The service aid phases IKQVDUMP and $$BCVS03 are included in the link-edit of VSAM. The other three phases, IKQVEDA, IKQVDU, and $$BCVS04 can be placed in the core image library by executing the following job.

```
// JOB          JOBNAME
// OPTION       CATAL
   INCLUDE      IKQCLNLK
/*
// EXEC         LNKEDT,REAL
/&
```

Enabling and Disabling Snap Dumps

The following snap points are available in VSAM. Each snap ID, if enabled with IKQVEDA, will produce the result indicated. If VSAM is running in the SVA, it must be reloaded from the core image library after the snap dump has been enabled in order to activate the snap, except for SNAP=0010 which takes effect immediately.

Snap number	**Result of Enabling this Snap**
0001	This snap allows Catalog Management diagnostic information to be obtained. (See section "Using UPSI to obtain Diagnostic Information for the VSAM Catalog" for details.) As snap 0001 uses the UPSI byte, it cannot be run when the user program in the partition also uses the UPSI byte.
0002	This snap enables the Buffer Manager trace, which provides the current usage of VSAM buffering.
0003	This snap enables the CLOSE control block dump at the beginning of CLOSE processing.
0004	This snap enables the VSAM I/O trace facility.
0005	This snap enables the I/O error trace.
0006	This snap enables the OPEN control block dump facility when open processing is complete.
0007	This snap enables the OPEN error trace. Control blocks are printed if an error occurs during open processing.
0008	This snap enables the Catalog Management I/O trace. All I/O operations done by catalog management are printed on SYSLST.
0009	This snap enables the VSAM Record Management error handler trace, allowing display of control blocks for any error detected by VSAM record management.

0010 This snap enables automatic close. VSAM is shipped with this snap enabled. To disable automatic close, disable this snap.

0011 This support enables the managed-SAM control block trace. Refer to "VSE/VSAM Space Management for SAM Feature Logic" for further information.

IKQVEDA is called by:

```
// EXEC IKQVEDA
```

The routine will print on SYSLOG:

```
ENTER FUNCTION ENABLE|DISABLE|END
```

You must enter either:

```
ENABLE SNAP=xxxx
               (where xxxx is one of the snap numbers)
```

or

```
DISABLE SNAP=xxxx
```

or

```
END
```

(to terminate processing).

The program will look for a private core image library and print:

```
NO PRIVATE CORE IMAGE LIBRARY ASSIGNED
```

if it cannot be found and will then look in the core image library for the VSAM phase needed.

If the phase needed cannot be found in a library the program will inform you with the following message:

```
phase NOT FOUND IN THE SYSTEM PRIVATE
CORE IMAGE LIBRARY
```

(where phase is the actual phase name)

Any error in input will result in the INVALID REPLY message and the ENTER FUNCTION message is reissued.

Entering ENABLE SNAP=0011 in a system without the VSE/VSAM Space Management for SAM Feature installed results in an INVALID REPLY message.

The program can only be ended by the END reply as noted earlier.

The following examples illustrate the use of IKQVEDA to enable and disable SNAP 0001:

```
// EXEC IKQVEDA
   ENTER FUNCTION ENABLE|DISABLE|END
   ENABLE SNAP=0001
   NO PRIVATE CORE IMAGE LIBRARY ASSIGNED
   SNAP 0001 ENABLED
   ENTER FUNCTION ENABLE|DISABLE|END
   DISABLE SNAP=0001
   NO PRIVATE CORE IMAGE LIBRARY ASSIGNED
   SNAP 0001 DISABLED
   ENTER FUNCTION ENABLE|DISABLE|END
   END
```

Obtaining Snap Dumps of Control Blocks

IKQVDUMP enables you to print out snap dumps of record management and catalog control blocks. Code is provided at certain points in VSAM modules which is nonoperational so far as normal execution of the modules is concerned. Refer to "Enabling and Disabling Snap Dumps," above.

IKQVDUMP is called by the following sequence of instructions (see also

"Loading a VSAM phase or a Program You Have Written"):

```
          LA    1,PARMLIST
          SVC   2
          .
          .
          .
PARMLIST DC    CL8'$$BCVS03'     B transient
         DC     CL8'IKQVDUMP      phase that provides dump
                                   of control blocks
```

When the program has completed processing, $$BCVS03 returns the program to the instruction immediately following the SVC instruction.

Note that IKQVDUMP requires SYSLST to be assigned to a printer; assigment to disk or tape will result in an error.

Figure 6.7 shows the description and format of the parameter list that follows the two phase names in the above calling sequence.

Offset		Bytes and		
Dec	Hex	Bit Pattern	Field Name	Description
0	0	1	PARMSW1	First byte of parameter list
		1...	PARMAMBL	Dump the AMBL
		.1..	PARMACB	Dump the ACB
		..1.	PARMAMDS	Dump the AMDSB
		...1	PARMARDB	Dump the ARDB
		.... 1...	PARMBCB	Dump the BCB
		.... .1..	PARMBUFE	Dump the buffer
		.... ..1.	PARMEDB	Dump the EDB
		1	PARMLPMB	Dump the LPMB
1	1	1	PARMSW2	Second byte of parameter list
		1...	PARMCCW	Dump the CCW
		.1..	PARMPLH	Dump the PLH
		..1.	PARMBHD	Dump the BHD
		...1	PARMRPL	Dump the RPL
		.... 1...	PARMEXCP	Dump the EXCPAD work area
		.... .1..	PARMCAT	Dump the catalog blocks
		.... ..1.	PARMDATA	Dump the non-catalog blocks
		1	PARMTHB	Dump the THB
2	2	1	PARMSW3	Third byte of parameter list
		1...	PARMOPEN	Dump the open work area
		.1..	PARMCLOS	Dump the close work area
		..1.	PARMCIW	Dump the control interval split area
		...1	PARMVLST	Dump the volume list
		.... 1...	PARMREGS	Dump the registers
		.... .1..	PARMCECL	Dump the control interval exclusive control list
		.... ..1.	PARMODLB	Dump the open DLBL
		1	PARMREQR	Dump the requester's registers
3	3	1	PARMSW4	Fourth byte of parameter list
		1...	PARMPAMB	1=Pointer to start dump is in parameter list (PARMAMBA) 0=Pointer to start dump is in register 11
		..1.	PARMCCAA	1=Pointer to CCA 0=Pointer to AMBL
		...1	PARMRTNA	Call the test routine
		.... 1...	PARMHDID	Dump the header ID
		.x.. .xxx		Available
4	4	4	PARMAMBA	Pointer to start dump
8	8	4	PARMID	Pointer to header
8	8	1	PARMIDLN	Length of the header
9	9	3	PARMIDAD	Address of the ID
12	C	1	PARMSW5	Fifth byte of parameter list
		1...	PARMCCA	Dump the CCA
		.1..	PARMCADL	Dump the CCA DLBL
		..1.	PARMCADP	Dump the CCA DADSM parameter list
		...1	PARMCARA	Dump the CCA record areas
		.... 1...	PARMCPL	Dump the catalog parameter list (CTGPL)
		.... .1..	PARMPLDN	Dump the CTGPL data set name
		.... ..1.	PARMPLNN	Dump the CTGPL new name
		1	PARMPLPW	Dump the CTGPL password

Figure 6.7 **IKQVDUMP parameter list description and format (part 1 of 2)**

Offset Dec	 Hex	Bytes and Bit Pattern	Field Name	Description
13	D	1	PARMSW6	Sixth byte of parameter list
		1...	PARMPLCN	Dump the CTGPL catalog name
		.1..	PARMPLCI	Dump the CTGPL control interval number
		..1.	PARMPLDL	Dump the CTGPL file CTGDDNM field
		...1	PARMPLWA	Dump the CTGPL work area
		.... 1...	PARMCFL	Dump the catalog field parameter list (CTGFL)
		.... .1..	PARMFLFD	Dump the CTGFL fields
		.... ..1.	PARMFLFN	Dump the CTGFL field name
		x		Available
14	D	1	PARMSW7	Seventh byte of the parameter list
		1...	PARMCFV	Dump the catalog field vector table (CTGFV)
		.1..	PARMFVDL	Dump the CTGFV file name
		..1.	PARMFVEN	Dump the CTGFV entry name
		...1	PARMFVKR	Dump the CTGFV key range list
		.... 1...	PARMFVVL	Dump the CTGFV volume serial list
		.... .1..	PARMDPDL	Dump the DADSM parameter list DLBL
		.... ..1.	PARMDPIO	Dump the DADSM parameter list I/O area
		1	PARMDPWA	Dump the DADSM parameter list work area
15	F	1	PARMSW8	Eighth byte of parameter list
		1...	PARMDPSV	Dump the DADSM parameter list save I/O area
		.1..	PARMCBS	Dump the AMCBS
		..1.	PARMCAXW	Dump the CAXWA
		...1	PARMCXRL	Dump the CAXWA RPL
		.... 1...	PARMCXDR	Dump the CAXWA VTOC label read-in work area (DRWA)
		.... .1..	PARMCMSW	Dump the CMS work area
		.... ..1.	PARMMSAM	Dump the managed SAM control blocks
		x		Available
16	10	8	PARMRTNN	Name of test routine

Figure 6.7 **IKQVDUMP parameter list description and format (part 2 of 2)**

Testing if a Dump is Required

IKQVDUMP allows a phase to be called before a dump is taken to see if a dump is desired. (The name of the test routine must be inserted into the parameter list at field name PARMRTNN.) The phase can use any logic to determine whether a dump is needed, and this logic will override a call for a dump if it is not needed. If a 0 is returned in register 15, the dump will be taken; if register 15 holds a nonzero return, the dump will not be taken.

The registers on entry to the test routine have the following contents:

R2 = Pointer to the parameter list
R11 = Caller's register 11
R13 = Pointer to 18-word save area
R14 = Return address of calling phase
R15 = Address of entry point

Using UPSI to Obtain Diagnostic Information for the VSAM Catalog

Manipulation of the UPSI job control statement enables you to screen catalog return codes and obtain a snap dump, cancel a job (which causes a full dump to be taken), or simply continue processing. You must first use IKQVEDA to enable Snap = 0001. Otherwise the UPSI statement will be inoperative. As snap 0001 uses the UPSI byte, it cannot be run when the user program in the partition also uses the UPSI byte.

The purpose of this service aid is to diagnose catalog errors that occur while running any program that causes the VSAM catalog to execute. Typically this would be an Access Method Services module or a record management program you have written.

The // UPSI nnnnnnnn job control statement must precede the // EXEC [progname] statement. If no UPSI statement is included, the default is // UPSI 00000000 (see type 3 request below).

On exit from catalog management after processing, a message will be printed out depending on the type of UPSI bit setting you have selected. Some messages require a reply from the operator. The return codes in the message are obtained from register 15. The format is:

```
** NNN,MN,RRR,FFFF,CCCCCCCCCCCCCCCC
```

where

NNN is the return code in decimal

MN are the last two characters of the module name which issued the error. This is blank in case of error code 0.

RRR is the reason code in decimal

FFFF is one of the following catalog management functions that had been processed:

DEFC	(define catalog)
DEFA	(define non-VSAM data set)
DEFS	(define space)
DEF	(define VSAM data set)
ALT	(alter)
DELC	(delete catalog)
DELS	(delete space)
DEL	(delete VSAM or non-VSAM data set)
LSTC	(list catalog)
UPD	(update or update-extend)
LOC	(locate)

C...C is either the control interval number in decimal or the first 16 characters of the data set name or volume serial number in EBCDIC.

If a reply is required from the system operator for certain types of requests, the operator must enter one of the following replies from the system console:

- Type in SNAP to get a snap dump by means of IKQVDUMP (see *IKQVEDA for enabling snap dumps*). The message will then be repeated and the operator should press the END key to continue processing.
- Type in CANCEL to cancel the job and obtain a full partition dump.
- Press the END key to resume processing.

The following paragraphs describe the four types of UPSI settings you can use to elicit a message and/or to determine the degree of return code screening done:

Type 1 UPSI Setting. If you want to obtain an operator message for all VSAM catalog return codes (including 0), you must include one of the following statements:

`// UPSI 11000000`	No reply is required from the operator
`// UPSI 01100000`	A reply is required from the operator

Type 2 UPSI Setting. An operator message is issued only if the return code is not 0 for the following statements:

`// UPSI 10000000` No reply is required from the operator

`// UPSI 01000000` A reply is required from the operator

Type 3 UPSI Setting. An operator message is not issued if one of the following conditions exists:

1. the Access Method Services command being processed was a LISTCAT and the return code is 8, or
2. the return code is 0, 40, 68, or 160

(these codes occur during normal processing and are, therefore, excluded).

If neither of these conditions exists, an operator message is issued for the following statements:

`// UPSI 00000000` No reply is required from the operator

`// UPSI 01110000` No reply is required from the operator

Type 4 UPSI Setting. If you want an operator message on a specific return code, you must include the following statements:

`// UPSI 00nnnnnn` nnnnnn is set to the value, in binary, of the code divided by 4. A reply is required from the operator

Maintaining VTOC and VOL1 Labels on DASD

A VSAM DADSM service aid has been provided to assist the programmer and operator in maintaining the VTOC and VOL1 labels on DASD devices.

The following procedures should be followed to use IKQVDU at the system console for such maintenance. The key difference in the three procedures is the presence, or absence, of a // UPSI job control statement. Steps of the procedure in lower case letters are typed in at the console; steps in upper case letters are printed out.

Procedure 1	**Explanation**
`// assgn sys000,x'cuu'` (press END key)	*cuu* points at the volume you want to use.
`// upsi 1` (press END key)	This job control statement is optional. If it is included, the following events take place on the volume that was assigned to SYS000: • The VSAM volume ownership bit and CRA pointer in the F4 VTOC label are reset. • The entire VTOC is scratched, that is, empty VTOC labels are written over existing F1, F2, and F3 labels, with the exception of labels that have names starting with the characters "DOS." or "PAGE". • An operator authorization prompt is issued if the VTOC label to be scratched is security protected.

`// exec ikqvdu,size=auto` (press END key)	Start execution of the IKQVDU phase

Procedure 2	**Explanation**
`// assgn sys000,x'cuu'` (press END key)	*cuu* points at the volume you want to use.
`// upsi 11` (press END key)	This job control statement is optional. If it is included, the following events take place on the volume that was assigned to SYS000: • The VSAM volume ownership bit and CRA pointer in the F4 label are reset. • The entire VTOC is scratched, that is, F0 labels are written over existing F1, F2, and F3 labels, with the exception of labels that have names starting with the characters "DOS." or "PAGE"
`// exec ikqvdu,size=auto` (press END key)	Start execution of the IKQVDU phase.

Procedure 3	**Explanation**
`// assgn sys000,x'cuu'` (press END key)	*cuu* points at the volume you want to use.
`// exec ikqvdu,size=30k` (press END key)	Start execution of the IKQVDU phase.

`SPECIFY FUNCTION OR REPLY`
`'?' FOR OPTIONS READY`
`?`
(press END key)
	The character ? causes a list of the various functions that IKQVDU performs to be printed out at the system console.

```
TO SET THE VOLUME OWNERSHIP FLAG REPLY 'SET OWNERSHIP'
TO SET THE CRA POINTER REPLY 'SET OWNERSHIP'
TO RESET THE VOLUME OWNERSHIP FLAG AND CRA POINTER REPLY
    'RESET OWNERSHIP' OR 'RESET CRA'
TO SET THE SECURITY FLAG IN A F1 LABEL REPLY 'SET
     SECURITY'
TO RESET THE SECURITY FLAG IN A F1 LABEL REPLY 'RESET
     SECURITY'
TO REMOVE A LABEL FROM THE VTOC REPLY 'SCRATCH'
TO RENAME A LABEL REPLY 'RENAME'
TO ALLOCATE A LABEL REPLY 'ALLOCATE'
TO REINITIATE PROCESSING REPLY 'RESTART'
TO ALTER OR DISPLAY A DASD VOL1 LABEL
REPLY 'CLIP LABEL=SER=N..N' OR 'CLIP LABEL=DISPLAY'
TO TERMINATE PROCESSING REPLY 'END'
READY
```

You can avoid printing out this list of functions simply by specifying the function you wish as follows:

Procedure	Explanation
`set ownership` (press END key)	Causes the VSAM ownership bit to be set in the F4 VTOC label and optionally allows the user to set the CRA pointer.

`reset CRA or`
`reset ownership`
	Causes the VSAM ownership bit and CRA pointer to be reset in the F4 VTOC label.

`set security` (press END key)	Causes the security bit to be set in the F1 VTOC label. When the console responds with ENTER DSN, reply with the data set name of the VTOC label to be modified.
`reset security` (press END key)	Causes the security bit in the F1 label to be reset. When the console responds with ENTER DSN, reply with the data set name of the VTOC label to be modified.
`scratch dsn=dsname` (press END key)	Causes the VTOC label with the specified data set name to be scratched.
`scratch vtoc` (press END key)	Causes the entire VTOC to be scratched with the exception of data set names starting with the characters "DOS." and "PAGE". In addition, an operator-authorization prompt will be issued if the VTOC label is security-protected or describes a catalog.
`rename` (press END key)	Causes the DSNAME portion of the F1 VTOC label to be changed. When the console responds with ENTER OLD DSN, reply with the data set name of the VTOC label to be changed. Be sure to enter the correct OLD DSN. No error checking is performed in case an invalid name is specified When the console responds with ENTER NEW DSN, reply with the new data set name.

`allocate`
(press END key)

Causes a new label to be created and written in the VTOC. In order to utilize this function, a DLBL/EXTENT job control statement must be provided.

When the console responds with ENTER FILENAME, reply with the same filename as that in the DLBL statement referred to above.

When the console responds with ENTER NEW DSN, reply with the data set name of the data set to be created.

When the console responds with DO YOU WISH TO SECURITY PROTECT THIS DATA SET? reply YES or NO. A reply of YES causes the data security bit to be set in the F1 VTOC label. A reply of NO causes the data security bit to be reset.

`restart`
(press END key)

Causes processing to be reinitiated with a READY prompt. This keyword can be used as a response to any operator prompt.

`clip label=display`
(press END key)

Causes the volume serial number to be displayed on the system console.

`clip label=ser=n..n`
(press END key)

Causes the existing volume serial number to be changed to the one specified as n..n.

`end`
(press END key)

Causes processing to terminate.

If an error occurs during execution of IKQVDU,

```
**ERROR** DADSM RETURN CODE IS nnn
```

prints out on the system console. The following shows the message code (nnn), the associated message, and the action required to correct the condition.

Example:

```
** ERROR** DADSM RETURN CODE IS 020 VTOC FULL
```

004 I/O ERROR WHILE READING VOLUME LABEL
Action: if the problem was not caused by a hardware error, restore the volume.

008 VOLUME NOT MOUNTED
Action: Mount the correct volume.

012 I/O ERROR ON VTOC
Action: If the problem was not caused by a hardware error, restore the volume.

016 DUPLICATE NAME ON VOLUME
Action: Choose another filename or scratch the original file from the volume. If duplication is due to key ranges, ensure each UNIQUE key range is on a separate volume.

020 VTOC FULL
Action: Delete any non-VSAM files or VSAM data spaces no longer needed from the volume to make additional Format 1 labels available, or reinitialize the volume with a larger VTOC.

024 EXTENT OVERLAPS EXPIRED FILE
Action: Examine the VTOC listing to determine where the overlap occurred. Correct the EXTENT statement causing the error. To delete the expired file, open a DTF using the same file-ID as that of the expired file, and instruct the operator to reply DELETE to message 4n33A when it is issued.

028 EXTENT OVERLAPS UNEXPIRED FILE
Action: Compare the high and low extent limits on the EXTENT statement or LSERV output with the file or data space limits on the VTOC display. If the extents overlap, correct the EXTENT statement in error.

032 EXTENT OVERLAPS PROTECTED UNEXPIRED FILE
Action: Examine the VTOC to determine where the overlap occurred. Correct the EXTENT statement causing the error. If necessary, use another volume.

036 EXTENT OVERLAPS VTOC
Action: Execute LVTOC. The Format 4 label (the first label in the VTOC display) contains the extent limits of the VTOC. If the program being executed uses a temporary label set and overlaps the VTOC, correct the EXTENT statements that overlap. If the job uses standard or partition standard labels, use the LSERV output to correct the extents of the overlapping file, VSAM data space, or UNIQUE VSAM file. Then rebuild the appropriate label tracks.

040 REQUIRED EXTENTS MISSING
Action: If temporary labels were used, match the extents on the incoming EXTENT card with the extents in the LVTOC output. If standard (permanent) labels were used, match the extents in the LSERV output with those in the LVTOC output.

044 LABEL NOT FOUND
Action: Use the LVTOC output to check for all file labels used in OPEN macros. If the file has been destroyed, it was probably due to deletion of overlapping extents on an unexpired file, and the file must be rebuilt.

048 INVALID LABEL ADDRESS
Action: Examine the VTOC for a label having an invalid forward chain pointer, and delete it. If no invalid labels are found, just rerun the job.

056 EXTENT OVERLAPS PROTECTED EXPIRED FILE
Action: Examine the VTOC listing to determine where the overlap occurred. Correct the EXTENT statement causing the error. If it is not necessary to save the expired file, open a DTF using the same file-ID as that of the expired file, and instruct the operator to reply DELETE to message 4n33A when it is issued.

064 GETVIS FAILURE ENCOUNTERED
Action: Allocate GETVIS area. If VSAM is running in the SVA, re-IPL and specify a new value for SET SVA. If VSAM is running in a partition, rerun the job in a larger partition.

072 CDLOAD FAILURE ENCOUNTERED
Action: Either the CDLOAD directory or the GETVIS area is full. Allocate more space.

080 OVERLAP AMONG NEW EXTENTS
Action: If DLBL and EXTENT statements are included in the program, determine the conflicting extents and correct them. If a standard label set is being used, use the LSERV output to locate and correct the conflicting file extents, and rebuild the standard label tracks.

088 FORMAT 4 LABEL NOT FOUND
Action: Reinitialize the VTOC to create a format-4 label.

092 VOL1 LABEL NOT FOUND
Action: Reinitialize the volume to create a VOL1 label.

096 JIB PROCESSING FAILURE
Action: Rerun the job when more JIBs are available.

Loading a VSAM Phase or a Program You Have Written

If you want to load and transfer control to and from a selected VSAM phase or a program you have written, you can use B-transient $$BCVS03 without destroying any registers in the following calling sequence:

```
          LA    1,PARMLIST
          SVC   2
           .
           .
           .
PARMLIST  DC    CL8'$$BCVS03'     B transient
RTNNAME   DC    CL8'XXXXXXXX'     Name of phase or program
                                    you have written
USERLIST  DC                       Parameter list for phase
           .                        'XXXXXXXX'
           .
           .
```

When control is received by 'XXXXXXXX', the registers have the following contents:

R0	=	Address of a work area (the size of the work area is specified by a halfword at offset 4 of 'XXXXXXXX' phase)
R1	=	Pointer to user's parameter list (USERLIST)
R2-13	=	Remain the same as they were when SVC 2 was issued
R14	=	Return address of calling module
R15	=	Address of entry point in 'XXXXXXXX

Control is returned from 'XXXXXXXX' by a BR 14 instruction.

Definitions of Terms Used In This Book

Access Method Services. A multifunction service program that defines VSAM data sets and allocates space for them, converts indexed sequential data sets to key-sequenced data sets with indexes, modified data-set attributes in the catalog, recognizes data sets, facilitates data portability between operating systems, creates backup copies of data sets and indexes, helps make inaccessible data sets accessible, and lists data-set records and catalog entries.

address direct access. The retrieval or storage of a data record identified by its RBA (relative byte address) independent of the record's location relative to the previously retrieved or stored record. (See also keyed direct access, addressed sequential access, and keyed sequential access.)

addressed sequential access. The retrieval or storage of a data record in its entry (RBA) sequence relative to the previously retrieved or stored record. (See also keyed sequential access, addressed direct access, and keyed direct access.)

allocation chain (AC). All allocation units containing control blocks for the same ACB.

allocation unit (AU). One or more pages of virtual storage containing control blocks referenced by record management.

alternate index. A collection of index entries, related to a give base cluster and organized by a key other than the prime key of the associated base data records. Its function is to provide an alternate means of locating records in the data portion of the base cluster.

alternate index upgrade. The process of reflecting changes made to a base cluster in its associated alternate index(es). (See also upgrade set.)

alternate key. A key, other than the prime key, used to form an alternate index.

application. As used in this publication, the use to which an access method is put or the end result that it serves; contrasted to the internal operation of the access method.

backward processing. A variation of sequential processing, whereby the previous, rather than the next, record in the entry, key, or relative-record sequence is retrieved.

base catalog record. The first catalog record (control interval) that describes the VSAM object. This record contains the object's data set name, cluster name, or volume serial number in the ENTNAME field. This record also contains the header fields required for the object. The base catalog record can contain group occurrence pointers that point to group occurrences in the base catalog record, or that point to group occurrences in extension records (vertical extension). The base catalog record's extension pointer can point to a control interval that continues the information (group occurrence pointers) contained in the base catalog record (horizontal extension).

base cluster. A key-sequenced or entry-sequenced data set over which one or more alternate indexes are built.

buffer steal. The removal of a buffer from one string (PLH) so it can be used by another string.

candidate volume. A direct-access storage volume that has been defined in a VSAM catalog as a VSAM volume; VSAM can automatically allocate space on this volume, as needed. (See also overflow volume.)

catalog. (See VSAM catalog.)

cluster. A combination of related VSAM data sets, identified by one name in the VSAM catalog, that is, a key-sequenced data set and its index or an entry-sequenced data set alone.

collating sequence. An ordering assigned to a set of items, such that any two sets in that assigned order can be collated. As used in this publication, the order defined by the System/370 8-bit code for alphabetic, numeric, and special characters.

component. As used in this manual, a group of modules that perform a function, such as Open.

compression. (See key compression.)

control area. A group of control intervals used as a unit for formatting a data set before adding records to it. Also, in a key-sequenced data set, the set of control intervals pointed to by a sequence-set index record; used by VSAM for distributing free space and for placing a sequence-set index record adjacent to its data.

control-area split. The movement of the contents of some of the control intervals in a control area to a newly created control area, to facilitate the insertion or lengthening of a data record when there are no remaining free control intervals in the original control area.

control interval. A fixed-length area of direct-access storage in which VSAM stores records and distributes free space. It is the unit of information transmitted to or from direct-access storage by VSAM, independent of blocksize.

control interval access. The retrieval or storage of the contents of a control interval.

control-interval split. The movement of some of the stored records in a control interval to a free control interval, to facilitate the insertion or lengthening of a record that won't fit in the original control interval.

CRA. Catalog recovery area. An entry-sequenced data set which exists on each volume owned by a recoverable catalog, including the catalog volume itself. The CRA contains records which describe the volume and the data sets on the volume.

data integrity. Preservation of data or programs for their intended purpose. As used in this publication, the safety of data from inadvertent destruction or alteration.

data record. A collection of items of information from the standpoint of its use in an application and not from the standpoint of the manner in which it is stored (see also stored record).

data security. Prevention of access to or use of data or programs without authorization. As used in this publication, the safety of data from unauthorized use, theft, or purposeful destruction.

data set. The major unit of data storage and retrieval in the operating system, consisting of data in a prescribed arrangement and described by control information to which the system has access. As used in this publication, a collection of fixed- or variable-length records in direct-access storage, arranged by VSAM in key sequence or in entry sequence. (See also key-sequenced data set and entry-sequenced data set.)

data space. A storage area defined in the volume table of contents of a direct-access volume for the exclusive use of VSAM to store data sets, indexes, and catalogs.

direct access. The retrieval or storage of data by a reference to its location in a data set rather than relative to the previously retrieved or stored data. Direct access is equivalent to ISAM random access. (See also addressed direct access and keyed direct access.)

distributed free space. Space reserved within the control intervals of a key-sequenced data set for inserting new records into the data set in key sequence; also, whole control intervals reserved in a control area for the same purpose.

dynamic storage area (DSA). A block of storage set aside on entry to open/close which may be suballocated to provide for temporary storage requirements of individual modules.

entry-sequence. The order in which data records are physically arranged in direct-access storage, without respect to their contents. (Contrast to key sequence.)

entry-sequenced data set. A data set whose records are loaded without respect to their contents, and whose relative byte addresses cannot change. Records are retrieved and stored by addressed access, and new records are added at the end of the data set.

exclusive control. (See hold.)

extension record. The continuation of a catalog record that contains group occurrence pointers and their group occurrences. Group occurrence pointers in an extension record always point to group occurrences within the extension record. The extension record's extension pointer can point to a control interval that contains part of a group occurrence too large to fit in the extension record (horizontal extension).

extent. A continuous space allocated on a direct-access storage volume, reserved for a particular data space or data set.

external procedure. A procedure that can be called by any other VSAM procedure; a procedure whose name is in the module's (assembler listing) "external symbol dictionary."

field. In a record or a control block, a specified area used for a particular category of data or control information.

file. (See data set.)

fixed block architecture (FBA). A direct access storage device that supports a fixed, 512-byte physical record size. The counterpart, count-key-data (CKD) device, permits variable record sizes.

free space. (See distributed free space.)

free space percentage. (See packing factor.)

generic key. A high-order portion of a key, containing characters that identify those records that are significant for a certain application. For example, it might be desirable to retrieve all records whose keys begin with the generic key AB, regardless of the full key values.

group code. A code that identifies the type of group occurrence. (See Field Name Dictionary for a list of group codes.)

group occurrence. Related fields of information in catalog records. See "Group Occurrences in Catalog Records" in the "Data Areas" section for further details.

group occurrence pointer. A field used to identify and locate a group occurrence by its displacement from the beginning of the record's group occurrences (the group occurrence is in the same control interval as the group occurrence pointer) or by a control interval number (the group occurrence point is in the base catalog record or its extension and the group occurrence is in an extension record). Group occurrence pointers are grouped by type code and are in ascending sequence by sequence number.

high-used RBA. The next byte past the end of the last control interval containing significant data, for ESDA; otherwise, the RBA at which the last SEOF is written.

high-water RBA. The high-used RBA of a data set.

hold. Exclusive control exercised over data or index during an update, erase, or insert operation to prevent another request from making interim changes between initiation and completion of the original request.

horizontal extension. An extension record pointed to by a catalog record's extension field. (See also vertical extension.)

horizontal pointer. A pointer in a sequence set index record that gives the location of the next index record in the same sequence set; used for keyed sequential access.

index. As used in this publication, an ordered collection of pairs, each consisting of a key and a pointer, used by VSAM to sequence and locate the records of a key-sequenced data set; organized in levels of index records. (See also index level, index set and sequence set.)

index entry. A key and a pointer paired together, where the key is the highest key (in compressed form) entered in an index record in the next lower level or contained in a data record in a control interval, and the pointer gives the location of that index record or control interval.

index level. A set of index records that order and give the location of records in the next lower level or of control intervals in the data set that it controls.

index record. A collection of index entries that are retrieved and stored as a group. (Contrast to data record.)

index replication. The use of an entire track of direct-access storage to contain as many copies of a single index record as possible; reduces rotational delay.

index set. The set of index levels above the sequence set. The index set and the sequence set together comprise the index.

integrity. (See data integrity.)

internal procedure. A procedure that can be called only by another procedure within the module. (See also external procedure.)

I/O threshold. The maximum number of buffers that can be filled with data before I/O will be started.

ISAM interface. A set of routines that allow a processing program coded to use ISAM (Indexed Sequential Access Method) to gain access to a VSAM key-sequenced data set.

job catalog. A catalog made available for a job by means of the filename IJSYSUC in the corresponding DLBL job statement.

key. One or more characters within an item of data that are used to identify it or control its use. As used in this publication, one or more consecutive characters taken from a data record, used to identify the record and establish its order with respect to other records. (See also key field and generic key.)

key compression. The elimination of characters from the front and the back of a key that VSAM does not need to distinguish the key from the preceding or following key in an index record; reduces storage space for an index.

key-field. A field located in the same position in each record of a data set, whose contents are used for the key of the record.

key-sequence. The collating sequence of data records, determined by the value of the key field in each of the data records. May be the same as, or different from, the entry sequence of the records.

key-sequenced data set. A data set whose records are loaded in key sequence and controlled by an index. Records are retrieved and stored by keyed access or by addressed access, and new records are inserted in the data set in key sequence by means of distributed free space. Relative byte addresses of records can change.

keyed direct access. The retrieval or storage of a data record by use of either an index that related the record's key to its relative location in the data set, or a relative-record number, independent of the record's location relative to the previously retrieved or stored record. (See also addressed direct access, keyed sequential access, and addressed sequential access.)

keyed sequential access. The retrieval or storage of a data record in its key or relative-record sequence relative to the previously retrieved or stored record, as defined by the sequence set of an index. (See also addressed sequential access, keyed direct access, and addressed direct access.)

LOCK wait. A wait for the release of a named resource that is locked by another VSE task, using the LOCK macro.

mass sequential insertion. A technique VSAM uses for keyed sequential insertion of two or more records in sequence into a collating position in a data set; more efficient than inserting each record directly.

master catalog. The main VSAM catalog, that contains extensive data set and volume information required by VSAM to be able to locate data sets to allocate and deallocate storage space, to verify the authorization of a program or operator to gain access to a data set, and to accumulate usage statistics. (See also job catalog, user catalog.)

max-CA. A unit of allocation equivalent to the maximum control area size on a count-key-data or fixed block device. On a CKD device, the max-CA is equal to one cylinder.

min-CA. A unit of allocation equivalent to the minimum control area size on a count-key-data or fixed block device. On a CKD device, the min-CA is equal to one track.

module. As used in this manual, a program unit that is identifiable by means of a symbolic name starting with IGG0 or IKQ.

nonunique. Space for a nonunique data set or index must be a suballocation from existing data spaces.

object. As used in this manual, a cluster, a data set, an index, a catalog, or a data space.

overflow volume. When space on a candidate volume is allocated by VSAM, that volume is then termed an overflow volume. (See also candidate volume.)

overlapped operation. An operation in which processing continues without waiting for completion of input or output which had been initiated.

packing factor. Percentage of the data object's space allocation to be reserved during its initial loading and during subsequent reorganization. (See also distributed free space.)

password. A unique string of characters stored in a catalog that a program, a computer operator, or a terminal user must supply to meet security requirements before a program gains access to a data set.

physical record. The smallest readable or writable unit of data that is stored on a direct-access storage device. Records are separated from each other by interrecord gaps.

piggy-back I/O. One buffer is used for both writing and reading the same channel program. The buffer contents are written out, and then data is read into the buffer in the same I/O operation.

pointer. An address or other indication of location. For example, an RBA is a pointer that gives the relative location of a data record or a control interval in the data set to which it belongs. (See also horizontal pointer and vertical pointer.)

portability. The ability to use VSAM data sets with different operating systems. Volumes whose data sets are cataloged in a user catalog can be demounted from storage devices of one system, moved to another system, and mounted on storage devices of that system. Individual data sets can be transported between operating systems using Access Method Services.

prime index. The index of a key-sequenced data set which is a base cluster, and thus has one or more alternate indexes. (See also index, alternate index.)

prime key. The key which is used to form the prime index. (See also key, alternate key.)

procedure. A functional unit of VSAM code that is entered only at one entry point and exits at the end of the procedure (the last line of the procedure's code). The procedure can call (transfer control, with a return to the procedure expected) other procedures within the module (internal calls) and can call other procedures in other VSAM modules (external calls). (See also internal procedure and external procedure.)

pseudo hold. For SHAREOPTIONS(4), a second hold on the same control area by a single VSE task, under a single ACB (or both hold requests under the same Local Shared Resource pool). The control area is treated as held by the second request, as well as the first. If the second request encounters an actual conflict with the first request, then the second request will receive an "exclusive control conflict" return code (X'08' in register 15, X'14' in the RPL feedback).

random access. (See direct access.)

RBA. Relative byte address. The displacement of a data rcord or a control interval from the beginning of the data set to which it belongs; independent of the manner in which the data set is stored.

record. (See index record, data record, physical record, stored record.)

recoverable catalog. A catalog defined with the recoverable attribute. Duplicate catalog entries are stored in CRAs, and can be used to recover data in the event of a catalog failure. (See also CRA.)

recovery mode. A user option that causes the data object's initial allocation of space to be written throughout with special records, the last of which is set to 0 and is termed the SEOF (software end of file) record. This must be done if VERIFY is to be used. (See also speed mode.)

relative byte address. (See RBA.)

relative-record data set. A data set whose records are loaded into fixed-length slots and numbered by the relative numbers of the slots they occupy.

relative-record number. A number that identifies not only the slot, or data space, in a relative-record data set but also the record occupying the slot. Used as a key for keyed access to a relative-record data set.

relative repetition number. An integer representing the position of a particular field in a group of repeating fields. For example, in EOV, the relative repetition number (RELREPNO) of a particular volume in the catalog data record of a particular cluster is that number of its occurrence in the volume repeating group. EOV uses the RELREPNO to obtain information about a particular volume in order to update the ARDB and EDB.

replication. (See index replication.)

reusable data set. A VSAM data set which can be reused as a work data set, regardless of its old contents.

routine. As used in this manual, an ordered set of instructions that may have frequent use, generally internal usage within a module.

scratch (adj.). Used to describe the contents of a buffer that are no longer valid.

scratch (v.). In buffer management, used to indicate that a buffer contains nothing of significance; in DADSM, to remove a DSCB.

section. A subdivision of an index record used to expedite location of the place in an index record where an entry-by-entry key search can begin.

security. (See data security.)

SEOF. (See software end of file.)

sequence set. The lowest level of the index of a key-sequenced data set; it gives the locations of the control intervals in the data set and orders them by the key sequence of the data records they contain. The sequence set and the index set together comprise the index.

sequential access. The retrieval or storage of a data record in either its entry sequence, its key sequence, or its relative-record number sequence relative to the previously retrieved or stored record. (See also addressed sequential access and keyed sequential access.)

skip sequential access. Keyed sequential retrieval or storage of records in ascending, non-consecutive sequence (with skips); VSAM scans the sequence set to find a record or a collating position.

slot. A fixed-length, numbered space in a relative-record data set which accepts one data record. (See also relative-record data set, relative-record number.)

software end of file. A control interval with a CIDF of 0 that marks the end of preformatted records in a data object's initial allocation of space when the user specifies recovery mode of processing. (See also recovery mode.)

spanned record. A logical record whose length exceeds the control interval size and thus crosses, or spans, one or more control interval boundaries within a single control area.

stored record. A data record, together with its control information, as stored in a direct-access storage device.

string. A string is a single record or a sequentially ordered set of records in a data set. The maximum number of strings (STRNO) to be processed concurrently in a data set is established when a data set is opened. The number of active RPLs determines the number of concurrent strings being processed at any point in time.

string-set. Set of strings that are in communication with each other. For normal processing, this is the set of active RPLs referring to the same ACB. For Local Shared Resources, this is the set of all active RPLs using the Local Shared Resource pool.

unique. (1) A unique data space is occuplied by only one VSAM data set, and cannot be shared with other data sets. (2) A unique alternate key is one which occurs in only one data record in the base cluster. The alternate index record containing this key thus has only one pointer to the base cluster.

upgrade set. All the alternate indexes that VSAM has been instructed to update whenever there is a change to the data of the related base cluster.

user catalog. An optional catalog used in the same way as the master catalog and pointed to by the master catalog. It lessens the contention for the master catalog and facilitates volume portability.

vertical extension. An extension record pointed to by a group occurrence pointer in the object's base catalog record or its horizontal extension. (See also base catalog record and horizontal extension.)

vertical pointer. A pointer in an index record of a given level that gives the location of an index record in the next lower level or the location of a control interval in the data set controlled by the index.

VSAM catalog. A key-sequenced data set containing extensive data-set and volume information that VSAM requires to locate data sets, to allocate and deallocate storage space, to verify authorization of a program or operator to gain access to a data set, and to accumulate usage statistics for data sets. (See also master catalog, job catalog, user catalog.)

Index

This index lists the modules by their descriptive names, followed by their symbolic names. The symbolic names, together with further information about the modules, can be found in the module directory in the *Directory* chapter.

VSE/VSAM
VSAM Logic, Volume 2
LY24-5192-1

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

	Yes	*No*
• Does the publication meet your needs?	□	□
• Did you find the material:		
Easy to read and understand?	□	□
Organized for convenient use?	□	□
Complete?	□	□
Well illustrated?	□	□
Written for your technical level?	□	□

- What is your occupation? ____________________

- How do you use this publication:

As an introduction to the subject?	□	As an instructor in class?	□
For advanced knowledge of the subject?	□	As a student in class?	□
To learn about operating procedures?	□	As a reference manual?	□

Your comments:

If you would like a reply, please supply your name and address on the reverse side of this form.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

Reader's Comment Form

Fold and Tape Please Do Not Staple Fold and Tape

Cut or Fold Along Line

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department G60
P. O. Box 6
Endicott, New York 13760

Fold Fold

If you would like a reply, *please print:*

Your Name ____________________

Company Name ____________________ *Department* __________

Street Address ____________________

City ____________________

State ____________________ *Zip Code* __________

IBM Branch Office serving you ____________________

IBM®

Licensed Material — Property of IBM
LY24-5192-1

VSE/VSAM VSAM Logic, Volume 2 (File No. S370-30) Printed in U.S.A. LY24-5192-1

IBM®