LY24-5216-2
File No. S370/4300-50

Program Product

SQL/Data System Logic Volume 1

Introduction
Method of Operation
Program Organization
Directory

Program Number 5748-XXJ

Release 2

IBM

Third Edition (August 1983)

This edition, LY24-5216-2, is a revision of LY24-5216-1. This edition applies to the Structured Query Language/Data System, Release 2, until otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/370 and 4300 Processors Bibliography, GC20-0001, for the editions that are applicable and current.

Changes and additions to the SQL/Data System for Release 2 are described in the publication SQL/Data System Release 2 Guide, GH24-5042. Changes or additions to the text and illustrations are indicated by vertical line to the left of the change.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

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, Dept. G60, P.O. Box 6, Endicott, NY, U.S.A. 13760. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

PREFACE

This logic manual provides some measure of detail about the internals of SQL/Data System (SQL/DS). It is intended for persons involved in determining programming problems, especially to determine whether there is a user or IBM programming problem. Further, if it is determined that there is an IBM problem, this manual, along with Volumes 2 and 3, helps to provide (a) the information needed to determine where the problem lies or (b) sufficient information to assist IBM personnel in locating the problem and providing the necessary program modification.

ORGANIZATION OF THIS MANUAL

This manual is designed to give a semi-detailed view of SQL/DS. It contains four sections:

- "Section 1: Introduction" provides a high-level view of the SQL/DS major functions and how they interact.
- "Section 2: Method of Operation" provides graphic descriptions of the modules or groups of modules for the major functions and how these modules (or module groups) interact, giving the flow of control.
- "Section 3: Program Organization" gives textual descriptions of the SQL/DS modules in an alphameric sequence. Most of the module descriptions are very brief. A few contain extended descriptions. Generally, those in the latter group have multiple entry points, or are not described at all in Section 2, or need to have some complex algorithm explained.
- "Section 4: Directory" contains a Link Edit Book Directory and a Module Directory, which contains a list of all the SQL/DS modules, page references to sections 2 and/or 3, and the phase name(s) to which the modules belong. It is organized alphamerically according to module name.

Volume 2 (LY24-5217) contains Section 5 (Data Areas), Section 6 (Diagnostic Aids), and three appendixes.

USING THIS MANUAL

This manual is designed to be used with Volume 2 and, if you are a VSE user and interested in Extract, Volume 3. (Volume 3 is a stand-alone volume for Extract and contains all six sections for Extract.)

Read Section 1 to get a high-level view of SQL/DS. Go to Section 2 to examine the control flow for a suspected problem area, or to Section 3 for descriptions of the individual modules. Most of the module descriptions in Section 3 are very brief; however, there are a few extended descriptions for some of the modules needing additional explanation not given in Section 2.

Section 4 is the directory to modules, phases, and link books. You can use this section to locate pages on which a particular module is presented/described, to determine in which phase(s) a module is included, and to match up the component with the link book names and phases.

PREREQUISITE PUBLICATIONS

SQL/Data System General Information, GH24-5012
SQL/Data System Concepts and Facilities, GH24-5013
SQL/Data System Planning and Administration -- VSE, SH24-5014
SQL/Data System Planning and Administration -- VM/SP, SH24-5043
SQL/Data System Installation -- VSE, SH24-5015
SQL/Data System Installation -- VM/SP, SH24-5044
SQL/Data System Terminal User's Guide -- VSE, SH24-5016
SQL/Data System Terminal User's Guide -- VM/SP, SH24-5045
SQL/Data System Terminal User's Reference, SH24-5017
SQL/Data System Application Programming, SH24-5018
SQL/Data System Operation, SH24-5020
SQL/Data System Data Base Services Utility, SH24-5046

RELATED PUBLICATIONS

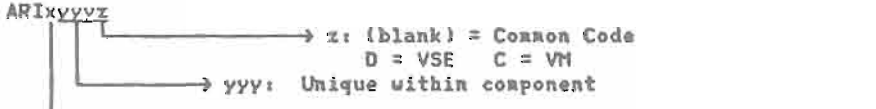
SQL/Data System Messages and Codes, SH24-5019

CONTENTS

SECTION 1: INTRODUCTION

This section consists of the high-level organization and flow of SQL/DS as related to its own components and, where necessary, the VSE or VM system. It will prepare the setting for Section 2, "Method of Operation", which will cover SQL/DS in a more detailed fashion, but not at a very low level of detail. Refer to Section 3 for textual descriptions of individual modules.

Generally, you can recognize the component to which a module belongs through the following module naming convention:



B = RDS Control Blocks/Macros
C = SQL/DS Control (DSC)
D = DBS Utility
I = ISQL
M = Modules Common Across SQL/DS, Messages, and Trace
P = PREP
Q = SQL/DS SQLCODE Descriptive Message Text
R = Resource Manager
S = SYSGEN/Installation
X = RDS
Y = DBSS

AXT.... Extract

Further, for the DBSS and RDS components, you can recognize the subcomponent to which a modules belongs by the fifth character of the module name (ARIYy... or ARIXy...):

Yy = YC DBSS Data Control
YD DBSS Data Manipulation
YE DBSS Trace/Miscellaneous Services
YI DBSS Storage and I/O
YK DBSS Lock
YL DBSS Log
YM DBSS Message Handling, Operator Services, and Miscellaneous
YS DBSS Sort
YT DBSS Work
YX DBSS Index
YZ DBSS Update Statistics/COUNTER Command

Xy = XA RDS Authorization
XC RDS Code Generator
XE RDS Executives
XF RDS Code Fragments
XI RDS Interpreter
XO RDS Optimizer
XP RDS Parser
XT RDS Trace

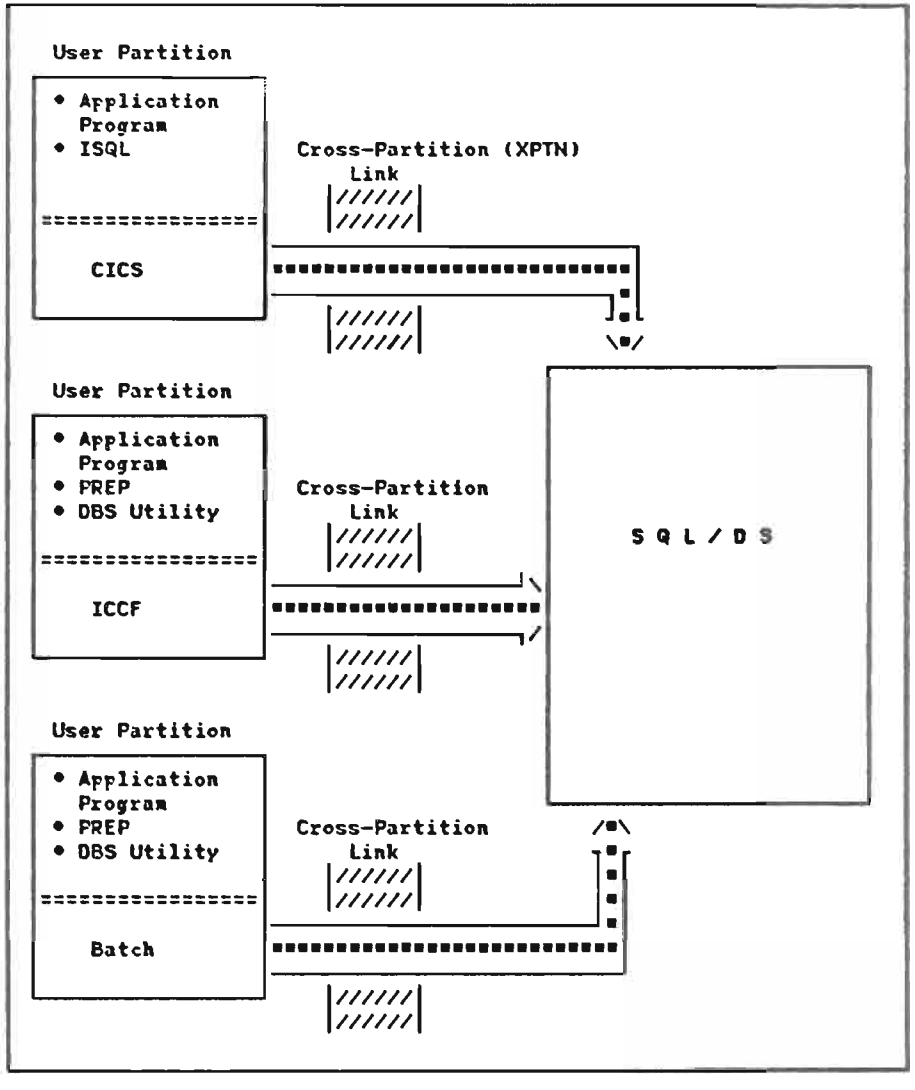


Figure 1. Multi-Partition Environment: Access to SQL/DS with CICS, ICCF, and Batch Support.

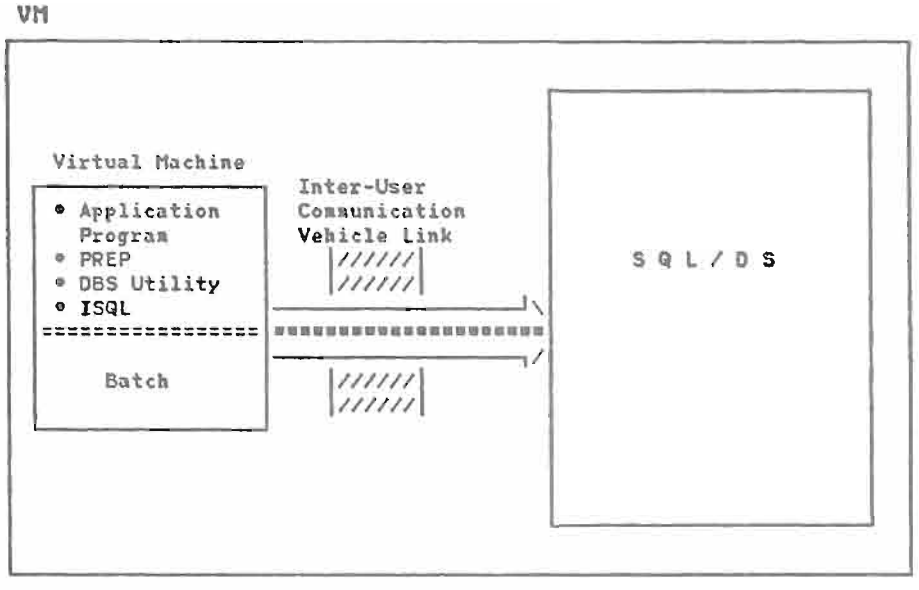


Figure 2. Multiple Virtual Machine Environment: Application Access to SQL/DS.

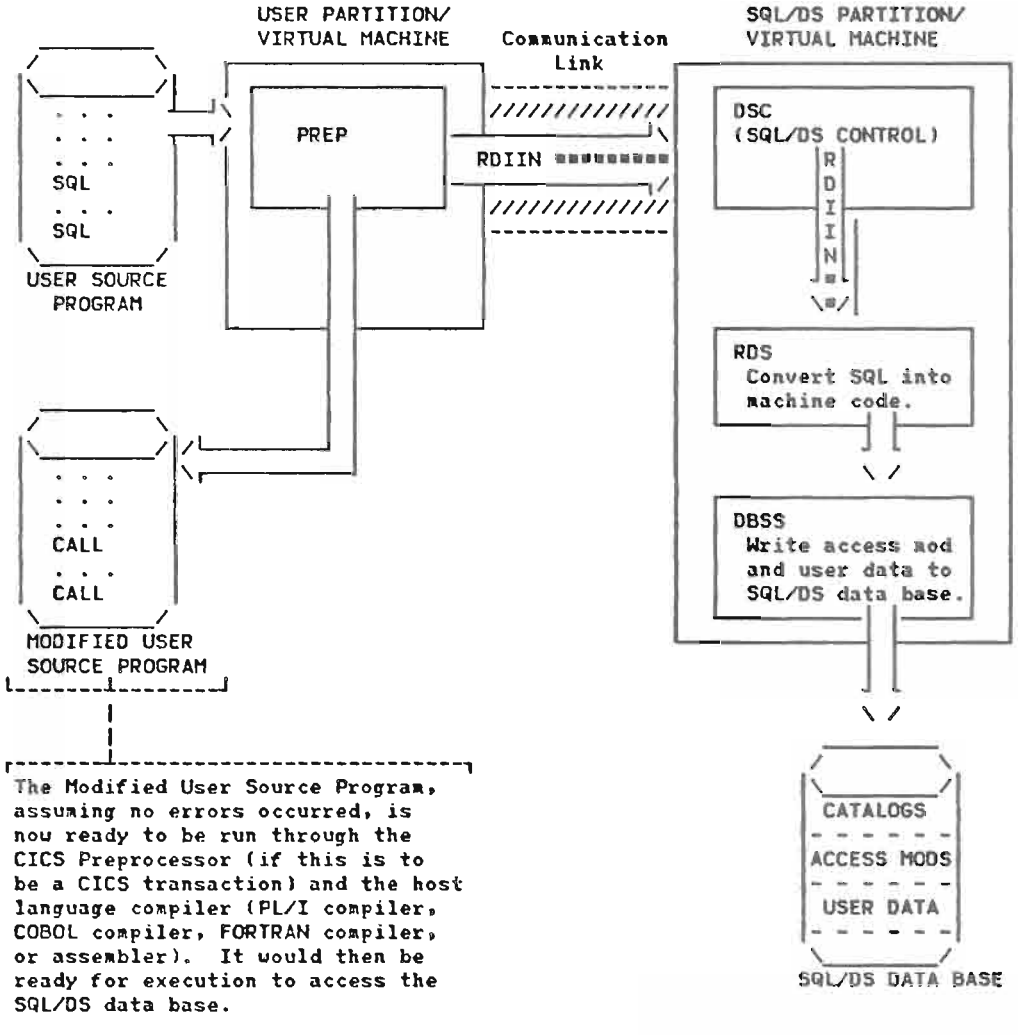


Figure 3. Preparing a SQL/DS Data Base Application Program for Execution (the PREP Process).

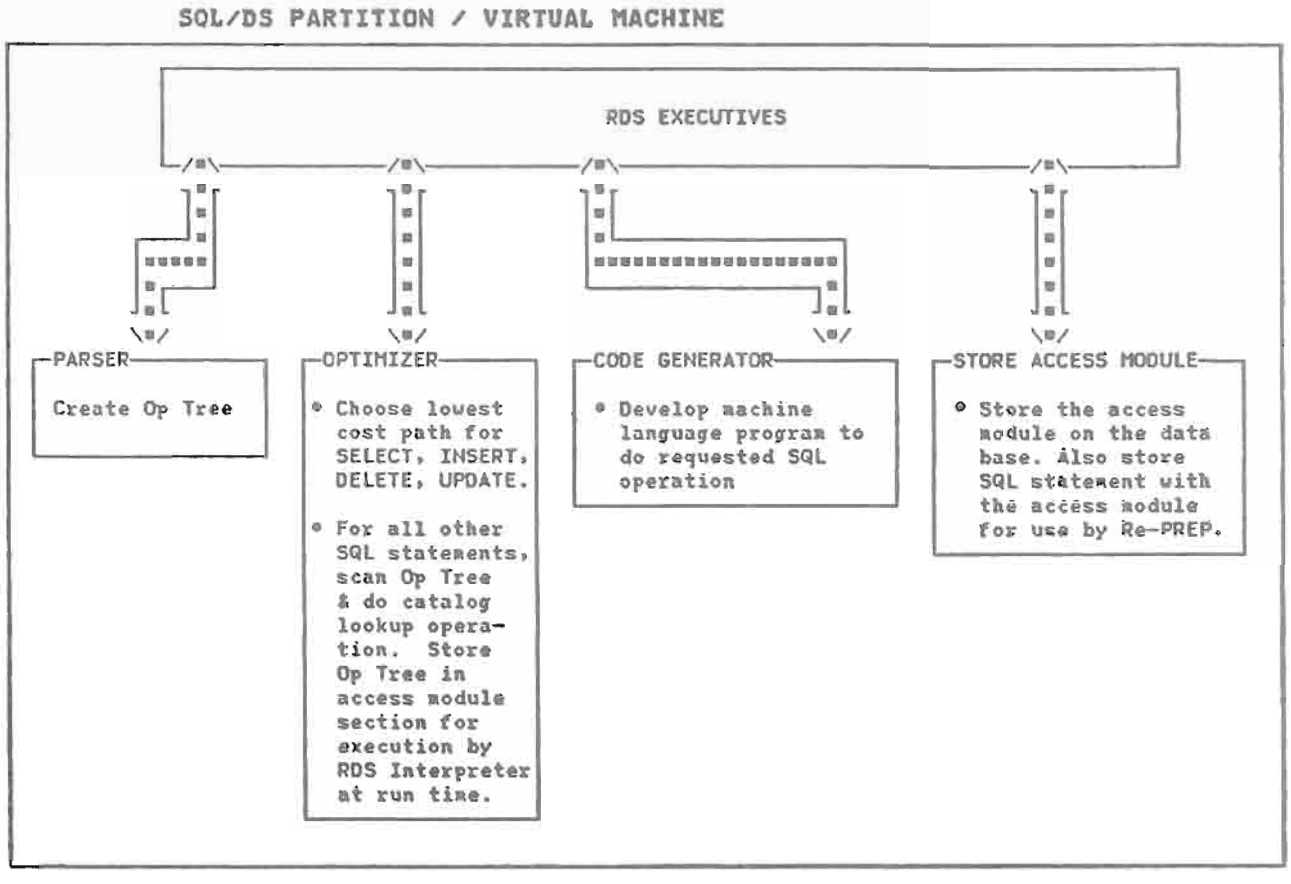
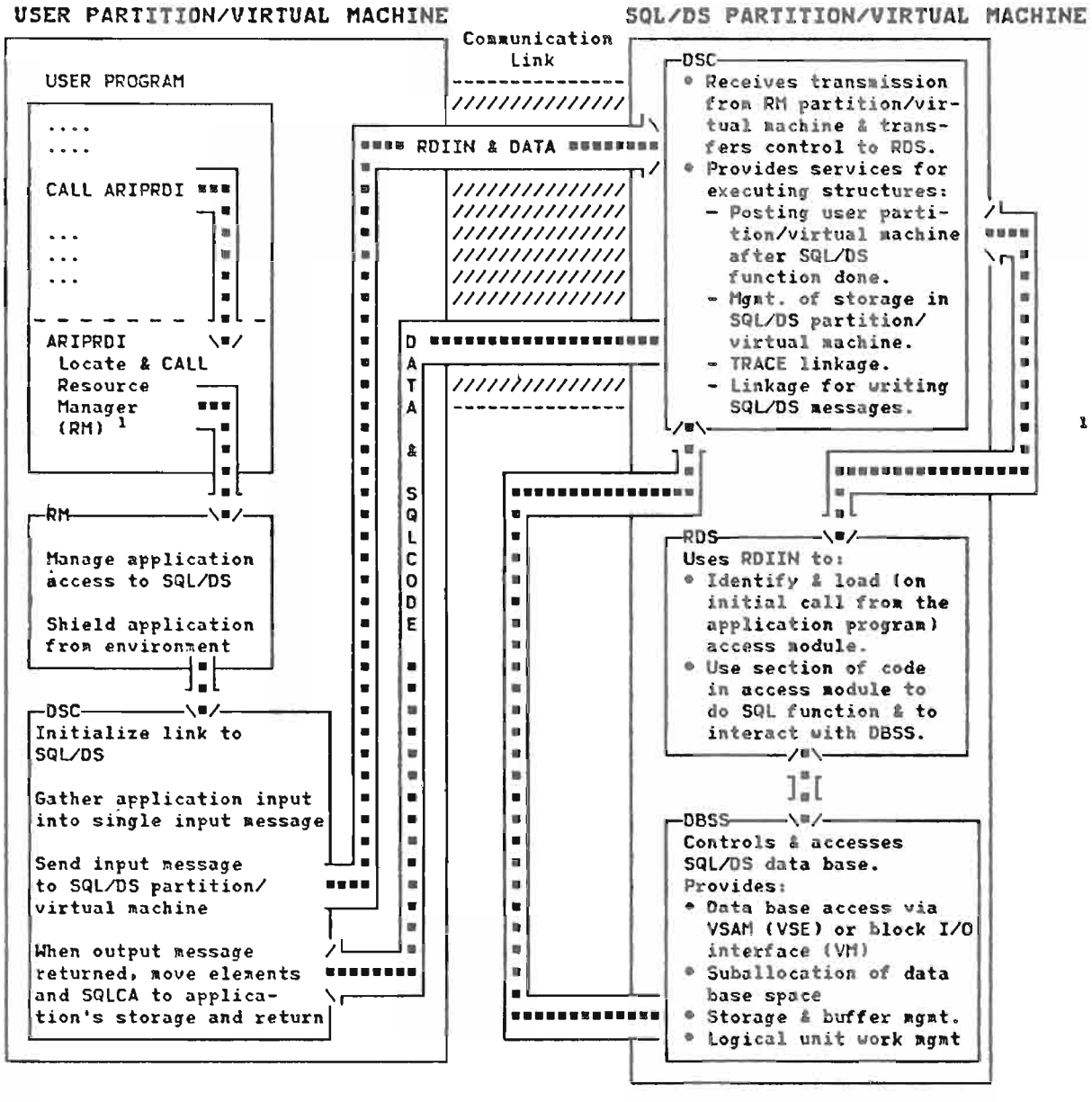
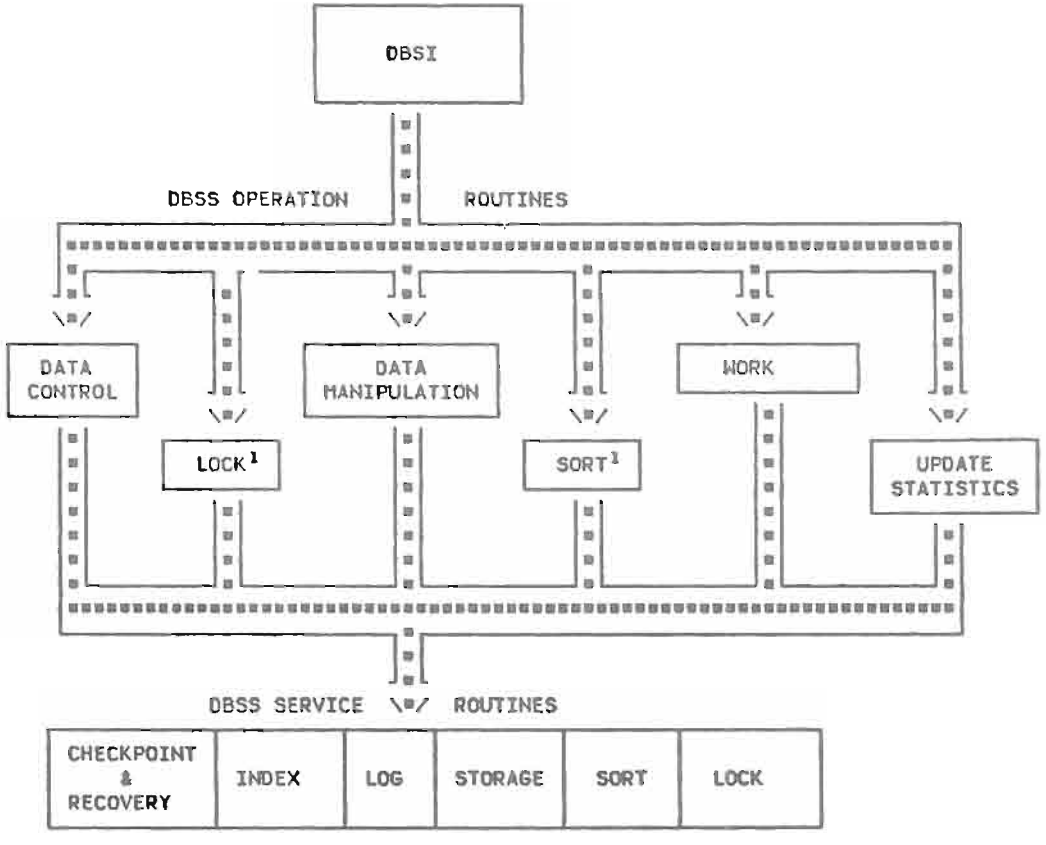


Figure 4. Preparation of an Application Program - RDS Functions.



1 Note: In SPM/SVMM the RM (Resource Manager) branches directly to RDS.

Figure 5. Execution of a SQL/DS Application (Batch, ICCF Program, or CICS Transaction (for VM - Batch only)).



[1] These are the same DBSS Service Routines as shown below.

Figure 6. Data Base Storage Service (DBSS).

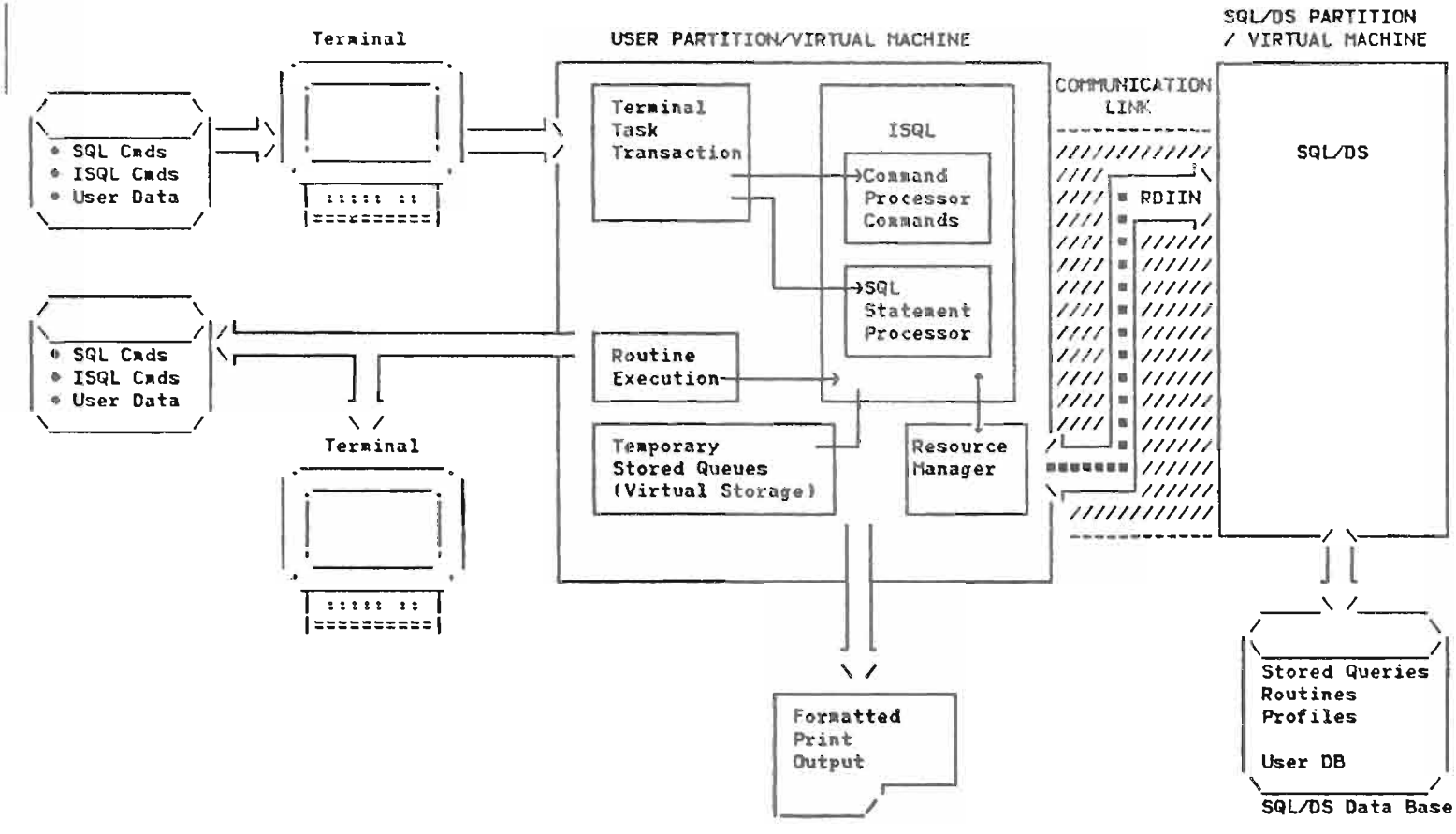


Figure 7. Interactive Structured Query Language (ISQL).

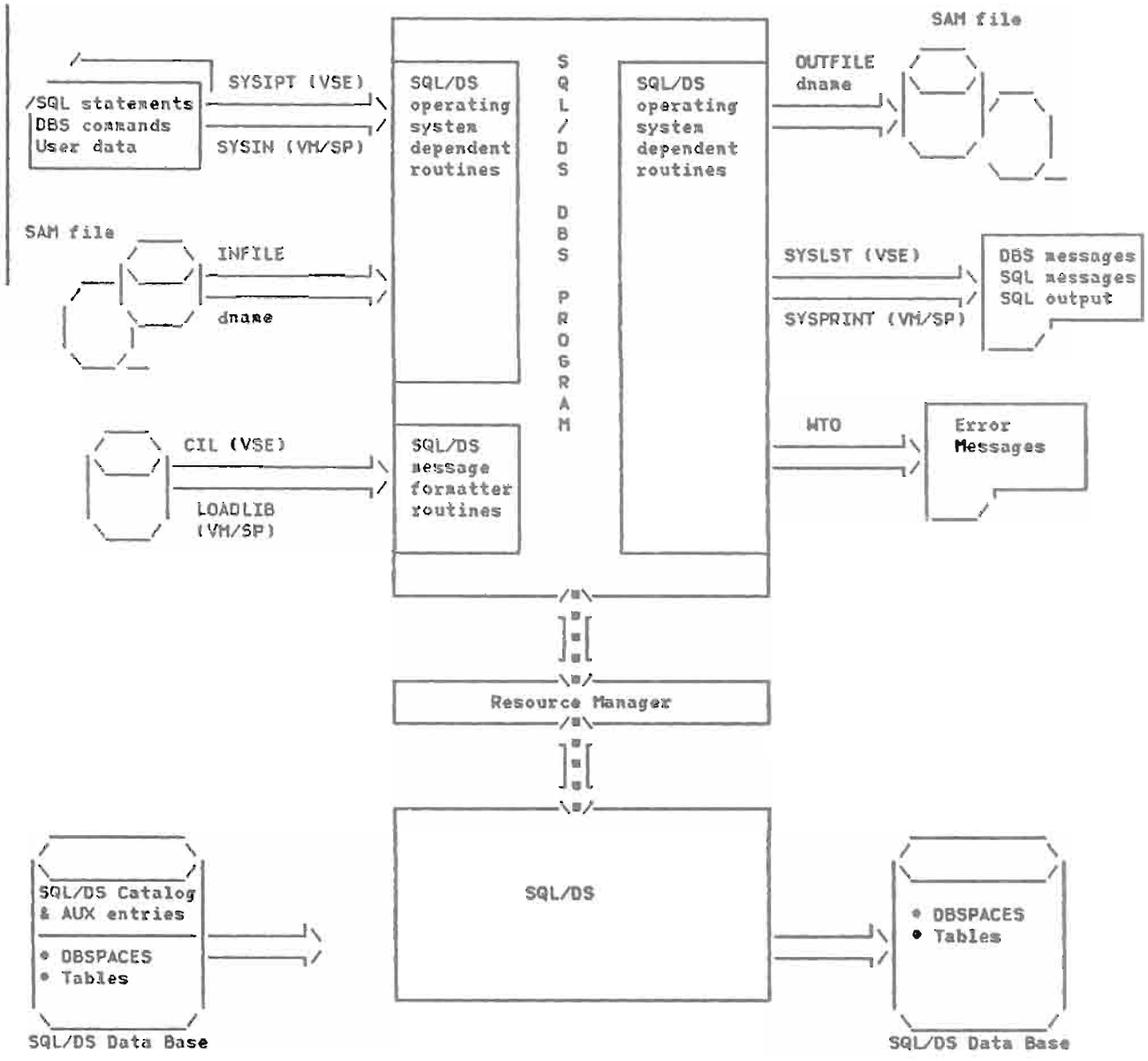


Figure 8. DBS (Data Base Services) Utility Program Overview.

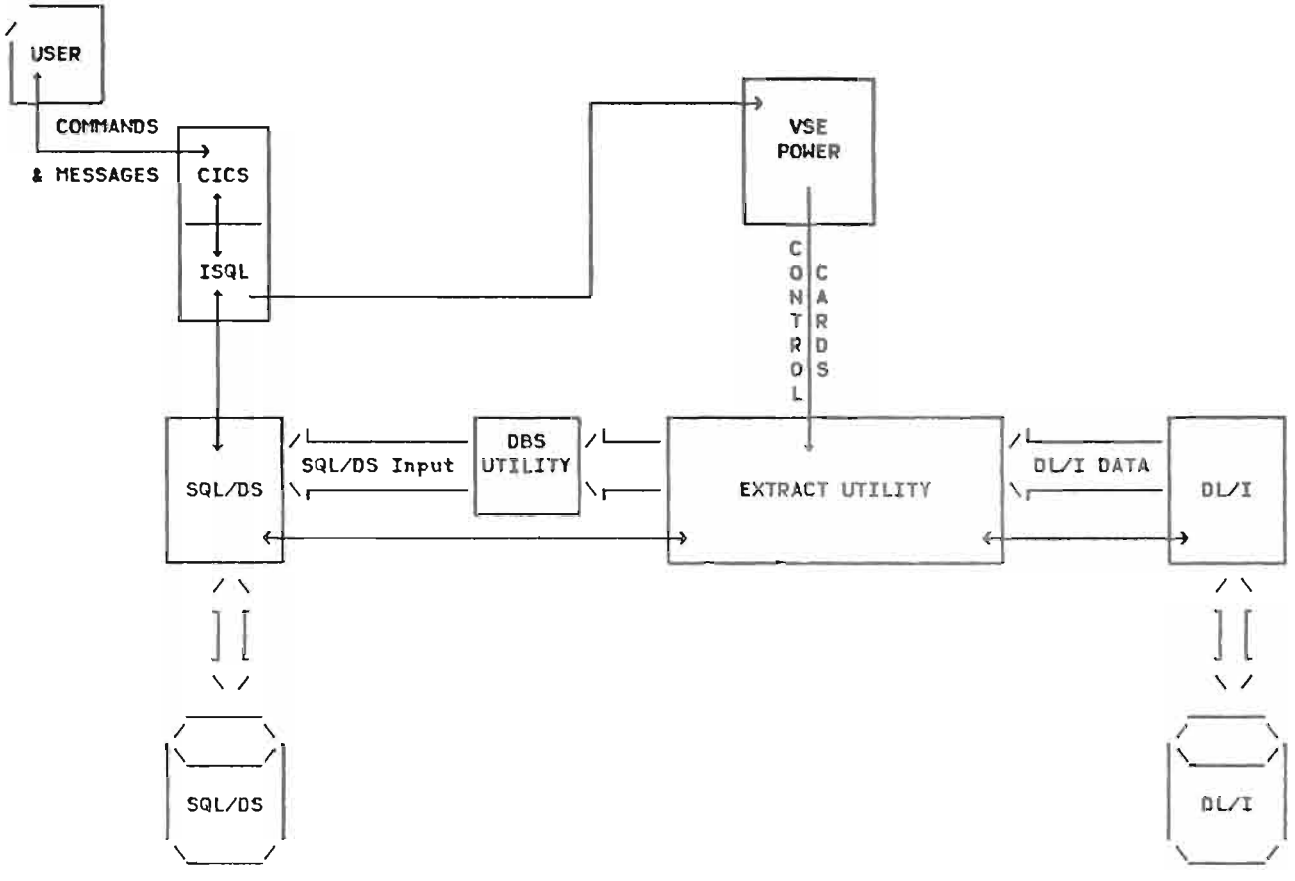


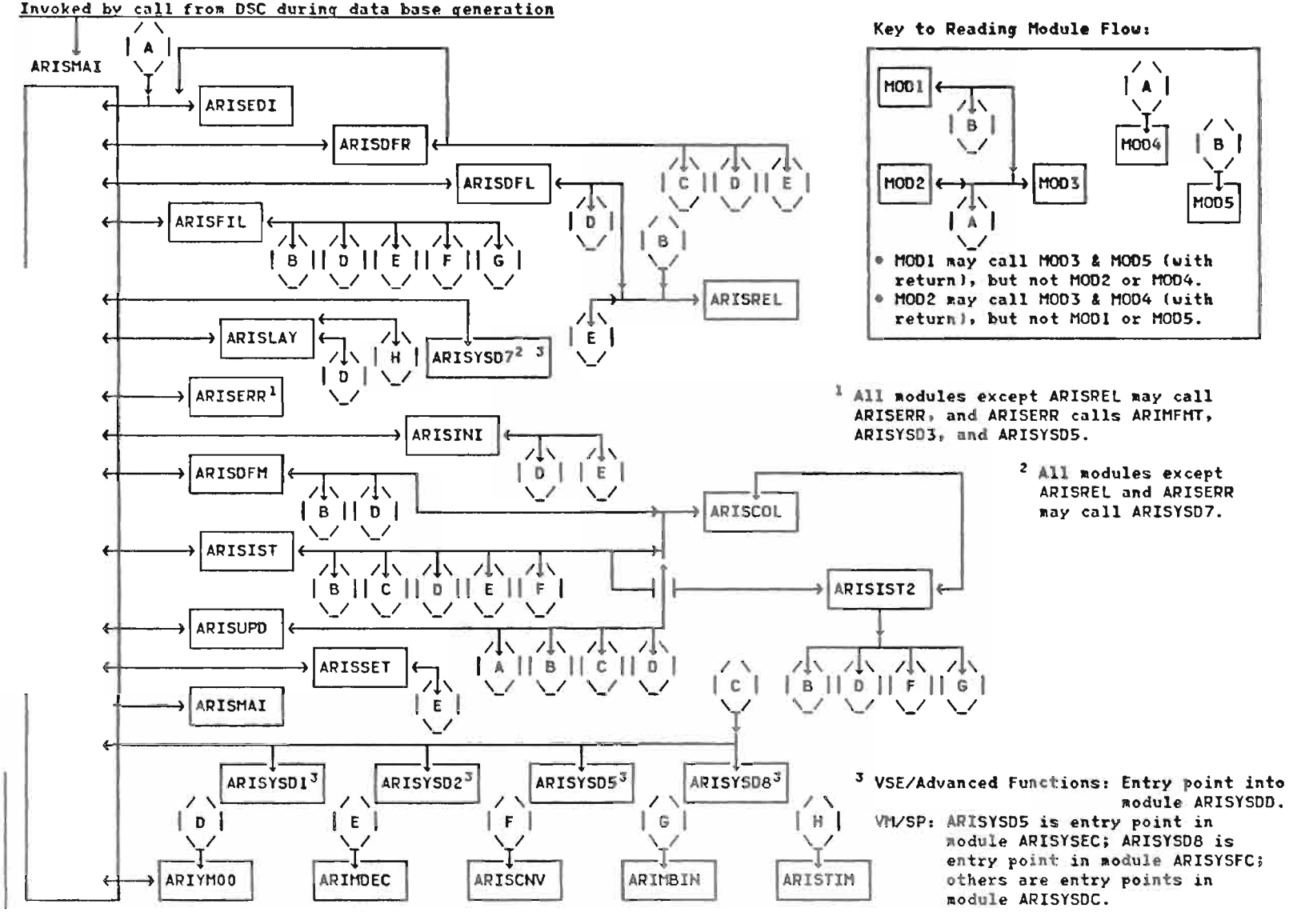
Figure 9. Extract Facility Overview (VSE only)

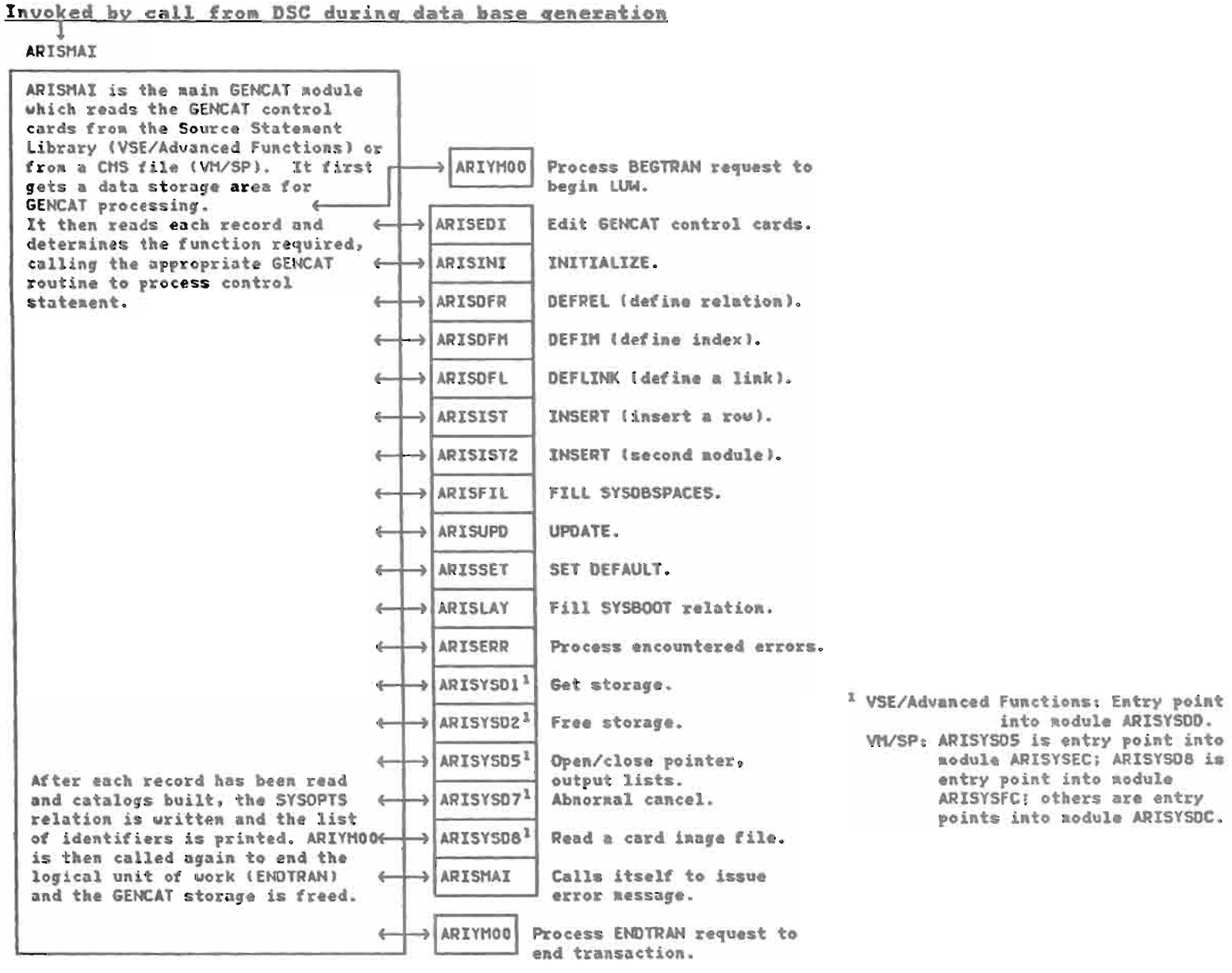
SECTION 2: METHOD OF OPERATION

Most of this section consists of control flow diagrams, both at a very high level and various levels of detail, depending on length and complexity of the functions and/or modules.

For some major functions and for some sub-functions, diagrams and text are included to provide some basic understanding of the concepts at the point at which they are introduced. Some of this material is introductory text. Other material includes example diagrams or format diagrams (for example, a broad overview of the associated data areas or tables) as is deemed appropriate.

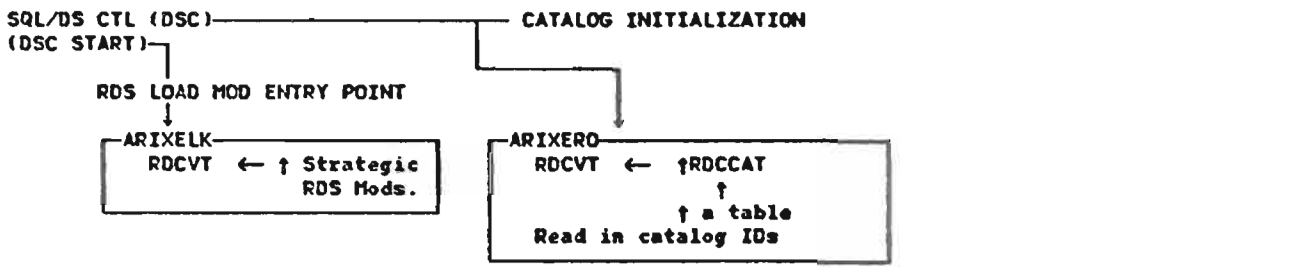
Overall GENCAT Control Flow

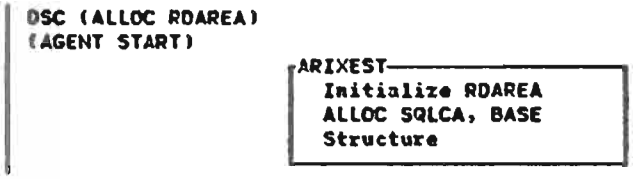


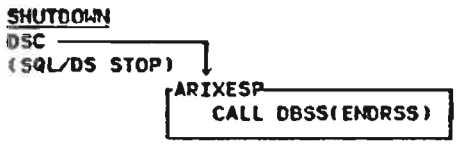


[1] VSE/Advanced Functions: Entry point into module ARISYSDD.
VM/SP: ARISYSD5 is entry point into module ARISYSEC; ARISYSD8 is entry point into module ARISYSFC; others are entry points into module ARISYSDC.

RDS EXECUTIVES AND MODULE FLOW







Extended Dynamic Statements Facility (EDSF) Termination

→(See page 20)

SQL CALL (SQL/DS CTL = DSC)

→ (Next page)

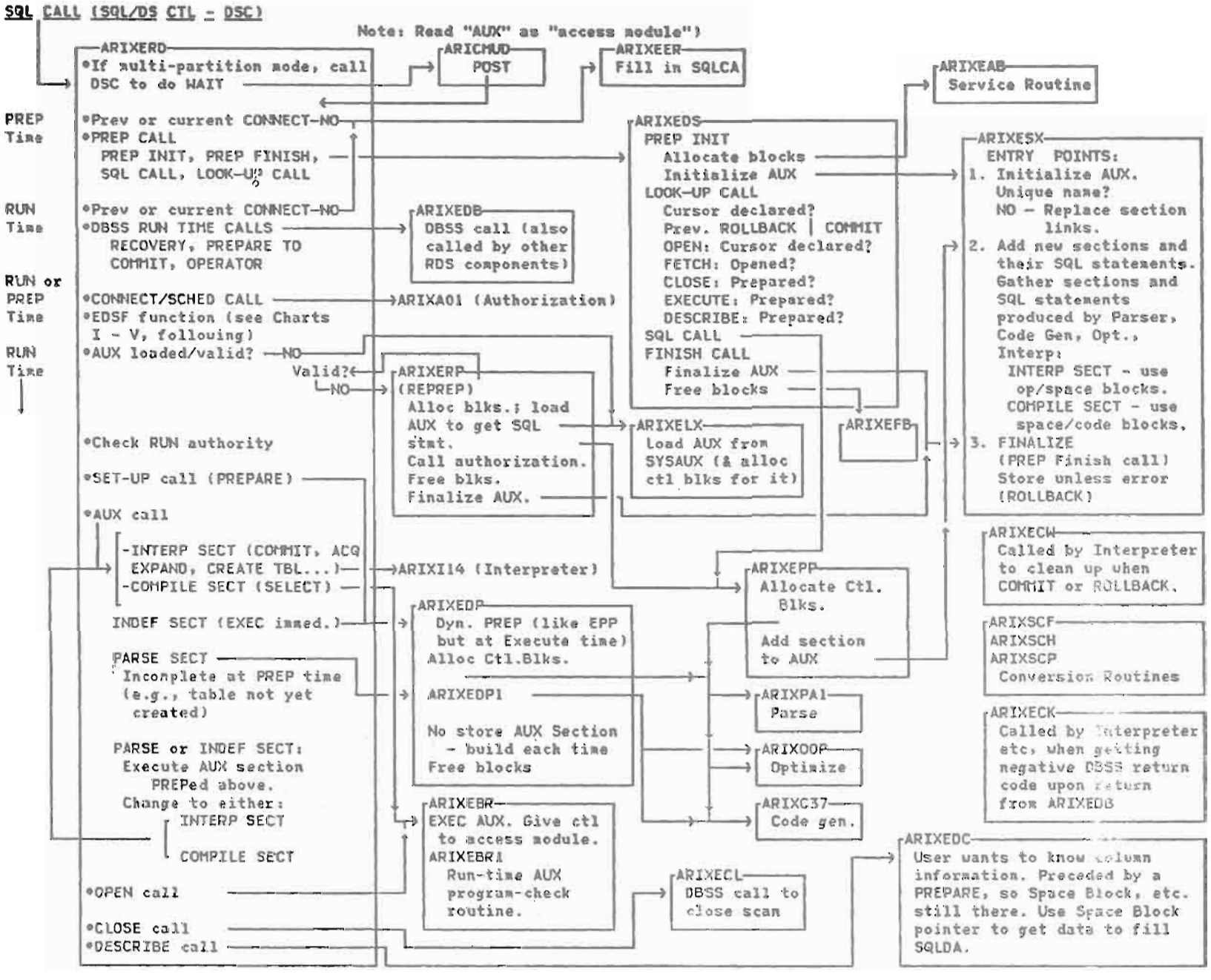

SQL CALL (SQL/DS CTL - DSC)
Note: Read "AUX" as "access module")
ARIXERD
•If multi-partition mode, call DSC to do WAIT
ARICMUD
POST
ARIXEER
Fill in SQLCA
ARIXEAB
Service Routine
PREP Time
•Prev or current CONNECT-NO
•PREP CALL
PREP INIT, PREP FINISH, SQL CALL, LOOK-UP CALL
ARIXEDS
PREP INIT
Allocate blocks
Initialize AUX
LOOK-UP CALL
Cursor declared?
Prev. ROLLBACK | COMMIT
OPEN: Cursor declared?
FETCH: Opened?
CLOSE: Prepared?
EXECUTE: Prepared?
DESCRIBE: Prepared?
SQL CALL
FINISH CALL
Finalize AUX
Free blocks
ARIXESX
ENTRY POINTS:
1. Initialize AUX.
Unique name?
NO - Replace section links.
2. Add new sections and their SQL statements.
Gather sections and SQL statements produced by Parser, Code Gen, Opt., Interp:
INTERP SECT - use op/space blocks.
COMPILE SECT - use space/code blocks.
3. FINALIZE
(PREP Finish call)
Store unless error
(ROLLBACK)
RUN Time
•Prev or current CONNECT-NO
•DBSS RUN TIME CALLS RECOVERY, PREPARE TO COMMIT, OPERATOR
ARIXEDB
DBSS call (also called by other RDS components)
RUN or PREP Time
•CONNECT/SCHED CALL
ARIXA01 (Authorization)
•EDSF function (see Charts I - V, following)
RUN Time
•AUX loaded/valid?
NO
Valid?
NO
ARIXERP
(REPREP)
Alloc blks.; load AUX to get SQL stmt.
Call authorization.
Free blks.
Finalize AUX.
ARIXELX
Load AUX from SYSAUX (& alloc ctl blks for it)
ARIXEFB
•Check RUN authority
•SET-UP call (PREPARE)
•AUX call
-INTERP SECT (COMMIT, ACQ EXPAND, CREATE TBL...)
-COMPILE SECT (SELECT)
ARIXI14 (Interpreter)
ARIXEPP
Allocate Ctl. Blks.
Add section to AUX
ARIXECW
Called by Interpreter to clean up when COMMIT or ROLLBACK.
INDEF SECT (EXEC immed.)
ARIXEDP
Dyn. PREP (like EPP but at Execute time)
Alloc Ctl.Blks.
ARIXSCF
ARIXSCH
ARIXSCP
Conversion Routines
PARSE SECT
Incomplete at PREP time (e.g., table not yet created)
ARIXEDP1
No store AUX Section - build each time
Free blocks
ARIXPA1
Parse
ARIXECK
Called by Interpreter etc, when getting negative DBSS return code upon return from ARIXEDB
ARIXOOP
Optimize
ARIXC37
Code gen.
PARSE or INDEF SECT:
Execute AUX section PREPed above.
Change to either:
INTERP SECT
COMPILE SECT
ARIXEBR
EXEC AUX. Give ctl to access module.
ARIXEBR1
Run-time AUX program-check routine.
ARIXECL
DBSS call to close scan
ARIXEDC
User wants to know column information. Preceded by a PREPARE, so Space Block, etc. still there. Use Space Block pointer to get data to fill SQLDA.
•OPEN call
•CLOSE call
•DESCRIBE call

Chart I - Creating an AUX which is NOT modifiable using the Extended Dynamic Statements Facility (EDSF)
(Example: How FORTRAN PREP creates and builds access module)

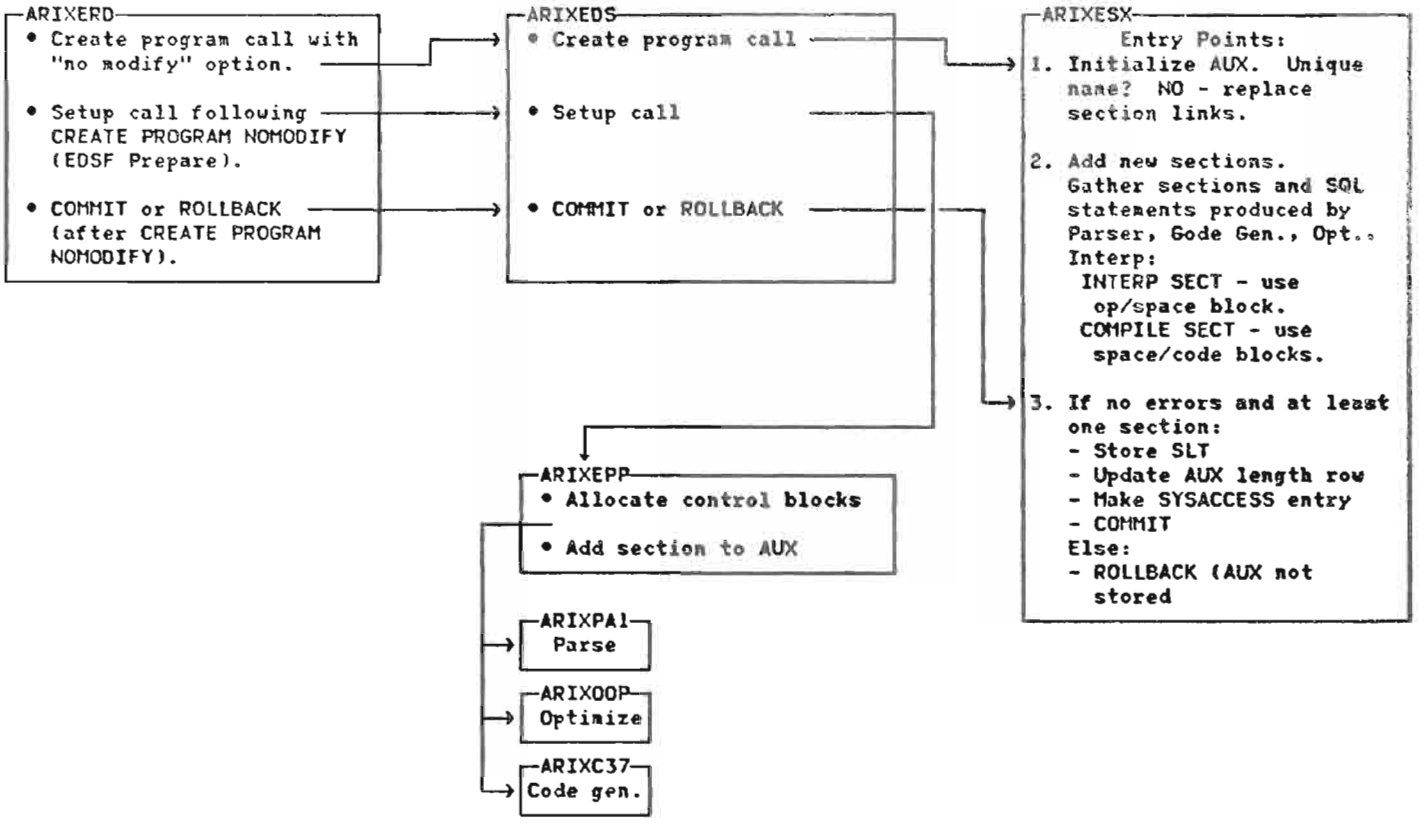


Chart II - Creating an AUX which is modifiable using Extended Dynamic Statements Facility (EDSF)

Run Time

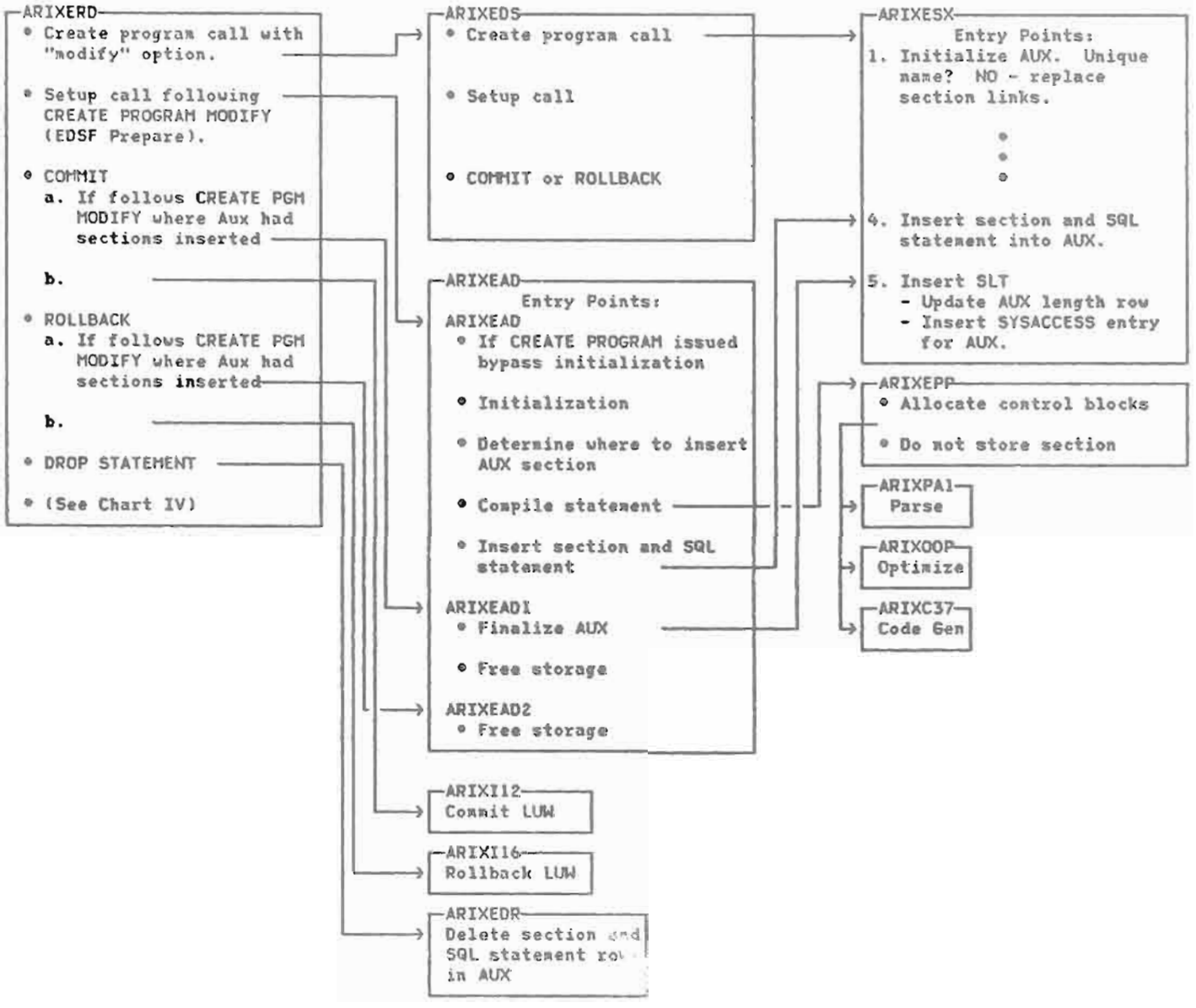


Chart III - Updating a Modifiable AUX using Extended Dynamic Statements Facility

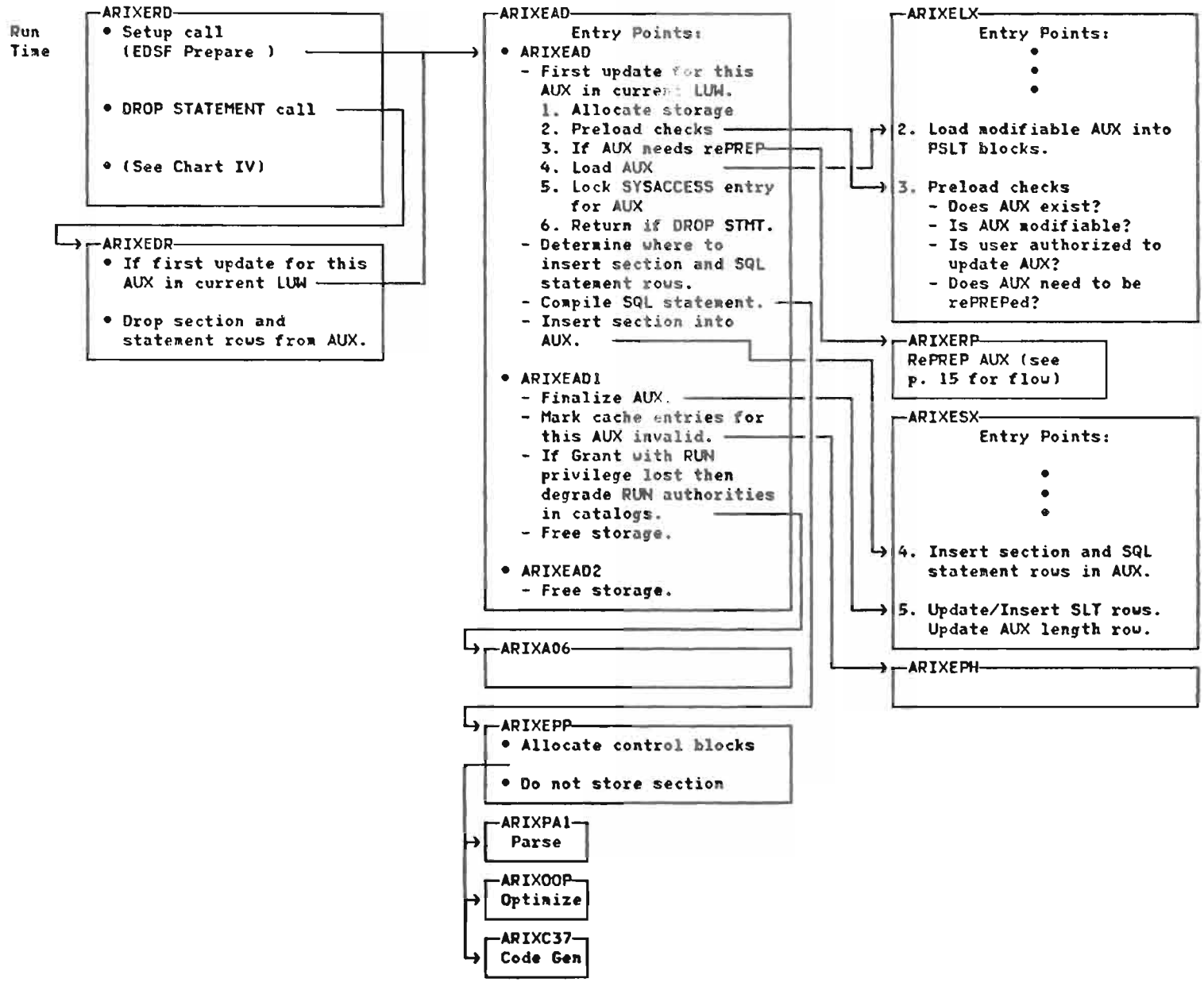


Chart IV - Executing Sections from Modifiable AUX which is being Updated Using Extended Dynamic Statements Facility

The Extended Dynamic Statements Facility (EDSF) allows a user to execute statements from an AUX which is being updated. Statements are not executed, however, from the AUX. Blocks containing the GEN code for each statement are kept only for the duration of the LUW. Each statement entry in the PSLT has a pointer to its blocks, thus permitting it to be executed.

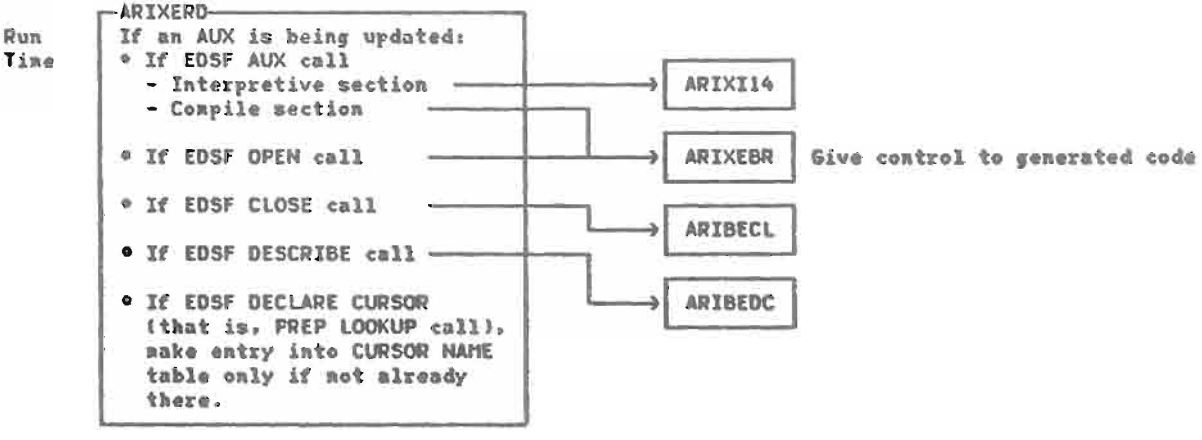
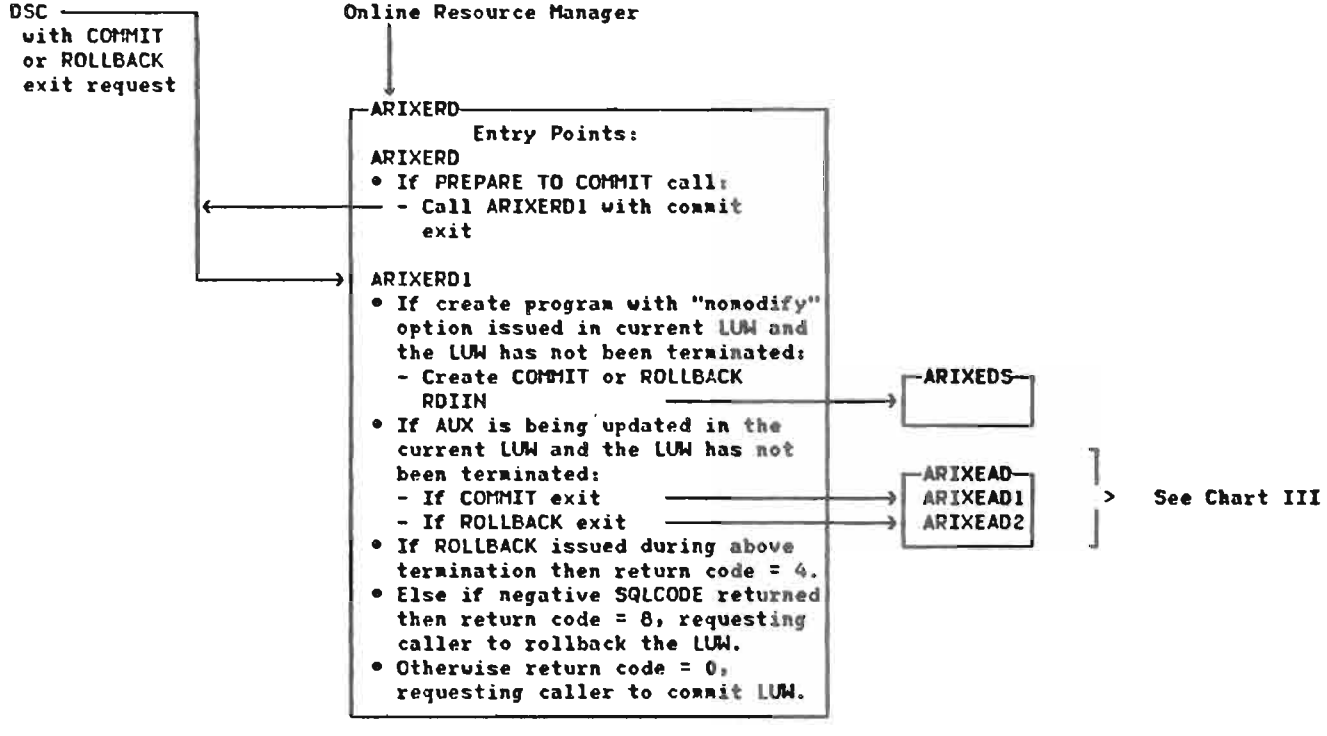


Chart V - Termination Support for Extended Dynamic Statements Facility (EDSF)

If an EDSF application ended normally or abnormally without issuing a COMMIT or ROLLBACK, then RDS is given the opportunity to clean up the EDSF environment which was established to process the user's EDSF application requests.

ARIXERD1 entry point is called to clean up the EDSF environment.

It should be noted that the resulting AUX finalization in ARIXESX3 (see Chart II) will exclude the COMMIT. The COMMIT is left to the caller (DSC or Online Resource Manager).



See Chart III

How AUX TIDs are used by Extended Dynamic Statements Facility to add and delete AUX section and SQL statement

A. Criteria used in ARIXEAD to determine where to insert a section:

An attempt is made to use the first available section which is reusable (i.e., PSLTREUS = ON as a result of the section having previously been deleted). If there are not reusable sections an additional entry is added to the PSLT (or SLT, Section Location Table) for the new section.

Once a PSLT entry is found or created, the insert TIDs (Tuple or Row IDs) and their qualifiers are determined for insertion of the first section row and first SQL statement row. The following table describes the criteria used in choosing TIDs and qualifiers. Note that a null AUX has only the AUX length row at the start of the update activity against the AUX. An AUX may not have any sections but may have SLT rows. Such an AUX is not considered null.

See Section 5, "Data Areas", for STQLDSTR mapping of PSLT.

DESCRIPTION OF SECTION BEING INSERTED	SECTION INSERT	SQL STATEMENT INSERT
Add section to null AUX	After TID of AUX row 1: STLD1TID	After TID of last inserted section row: TBTID of insert
First section of SLT is reusable	1. If there were no SLT rows loaded: After TID of first row in AUX: STLD1TID 2. If SLT was loaded: After TID of last SLT row: STLDSTID	1. If this is only section or all other sections are reusable: As null AUX case 2. Otherwise: After TID of last inserted SQL stmt row: STLDSCSQI
No SLT sections are reusable and at least 1 section. Section which is reusable after all used sections.	Before TID of the first SQL statement row in the AUX: PSLTOVAR(1)	After TID of last inserted SQL statement row: STLDCSQI
Section is reusable but not in above categories (there are used sections before and after).	Before TID of the first section row of next section: PSLTIVRP of reusable section	Before TID of the first SQL statement row of next section: PSLTOVRP of reusable section

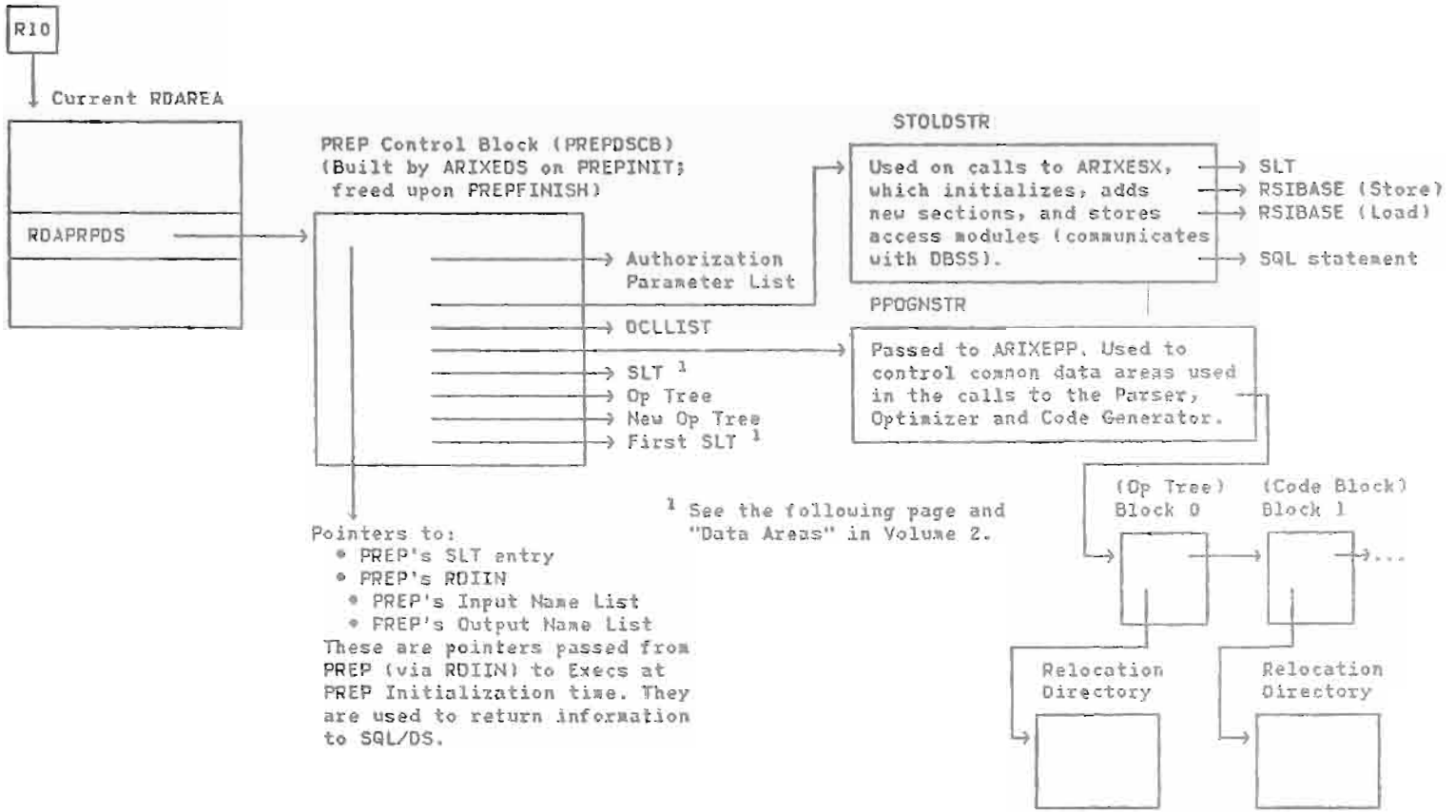
B. Criteria used in ARIXEDR to determine the AUX row which ends section and SQL statement row deletion.

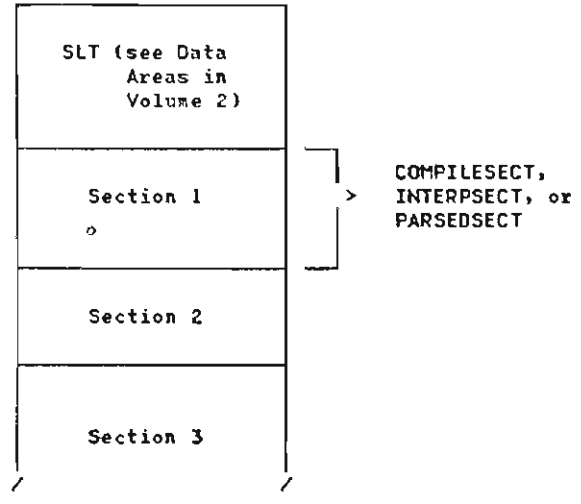
PSLTIVRP and PSLTOVRP of the target section are the TIDs of the first section and SQL statement rows, respectively, of the section to be deleted. These TIDs are used as the starting point for the delete. The TIDs used to terminate the delete are chosen as described in the table below.

See Section 5, "Data Areas", for STOLDSTR and PSLT mappings.

DESCRIPTION OF SECTION BEING DELETED	TIDS USED TO END SECTION AND STATEMENT ROW DELETION
Section is the only used section remaining in the AUX	Section termination TID: TID of the first SQL statement row to be deleted: PSLTOVRP of target section SQL statement termination TID: end of file on scan
Section is last used section of the group of used sections remaining in the AUX	Section termination TID: TID of the first SQL statement row in the AUX: placed in ENDSETID during scan of PSLT. SQL Statement termination TID: end of file on scan
Section is sandwiched between used sections whose PSLT entries are not necessarily adjacent (to PSLT entry of section being deleted)	Section termination TID: TID of the first section row of next used section: PSLTOVRP of next PSLT entry. SQL Statement termination TID: TID of first SQL statement row of next used section: PSLTOVRP of next PSLT entry.

The following overview diagram is presented here as an introduction to the data areas directly associated with the RDS Executives when servicing requests from an SQL/DS preprocessor (see previous page) in particular (although other functions use them). See "Data Areas" in SQL/DS Logic, Volume 2, for the individual data areas in detail.





1. COMPILESECT - Block 0 (first code block) as produced by the Code Generator. Optionally followed by Block 1, Block 2, etc (chained together). Block 10 is a Block Directory (always present).

   a. Code Block 0 - Entry to generated machine code. (This is the physical replacement for the original Block 0 where the Op Tree was located.) Block 0 is chained to the Space Block.

   b. Space Block - Filled by the Optimizer. Contains DBSS linkage structures used in the generated code for calls to DBSS. Space Block is chained to Block 10.

   c. Block 10 - Code block directory (for the case where there are more than one Code Blocks). Block 10 is chained to Code Block 2 (if present). Block 2 is chained to Block 3, etc, if applicable.

   Each block has an associated Relocation Directory, which contains the index locations of pointers within the block that must be changed by the Executive routines when the access module is loaded. In addition, each code block has a dynamic save area for use when calling other blocks or run-time routines.

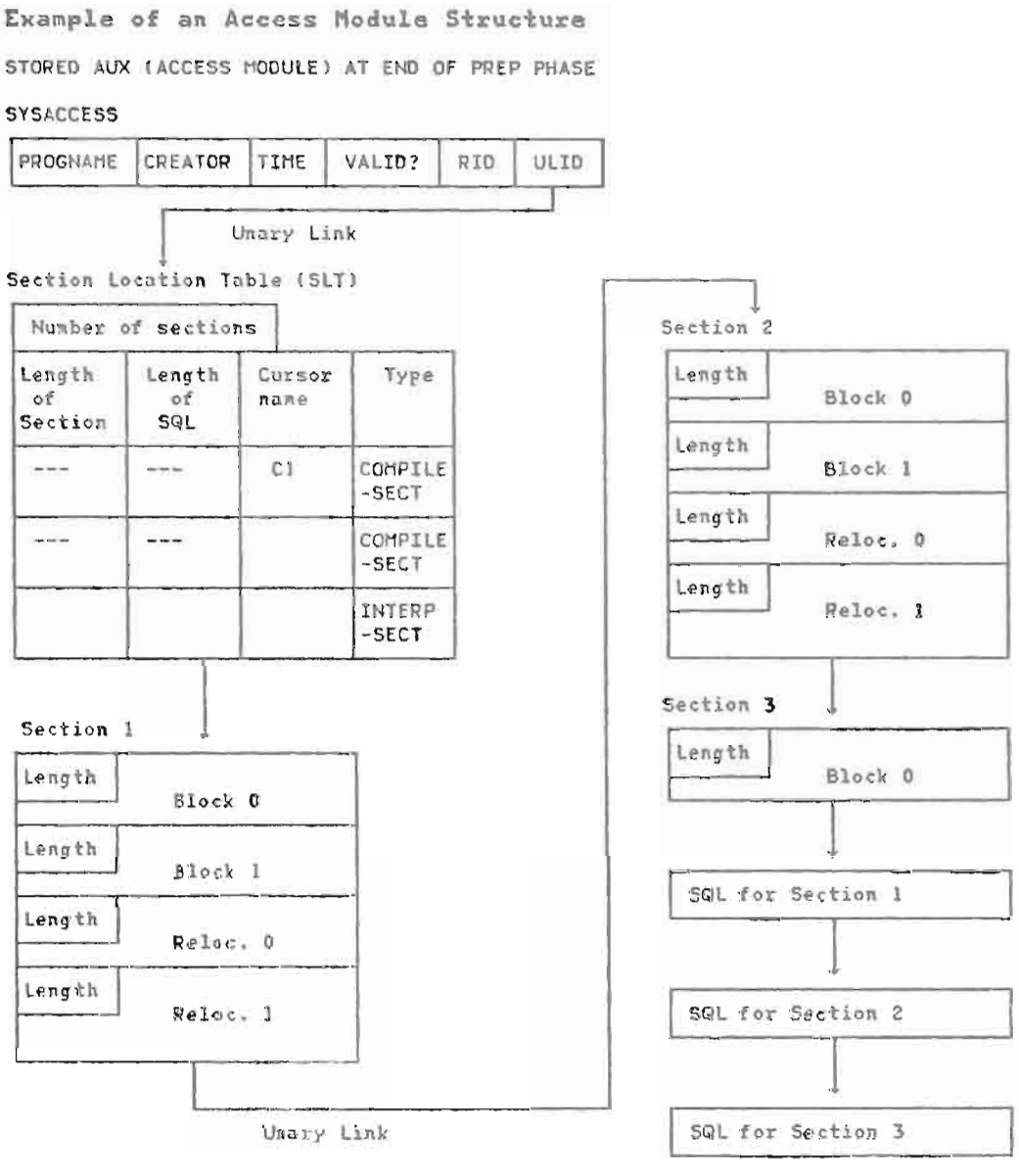
2. INTERPSECT - Contains Block 0 (Op Tree) as built by the Parser. This is the input to the Interpreter at run-time.

3. PARSEDSECT - Contains the same thing as the INTERPSECT, but it is created for a statement that references a table that was not created when the statement was preprocessed. This is input to the Optimizer and Code Generator at run-time.

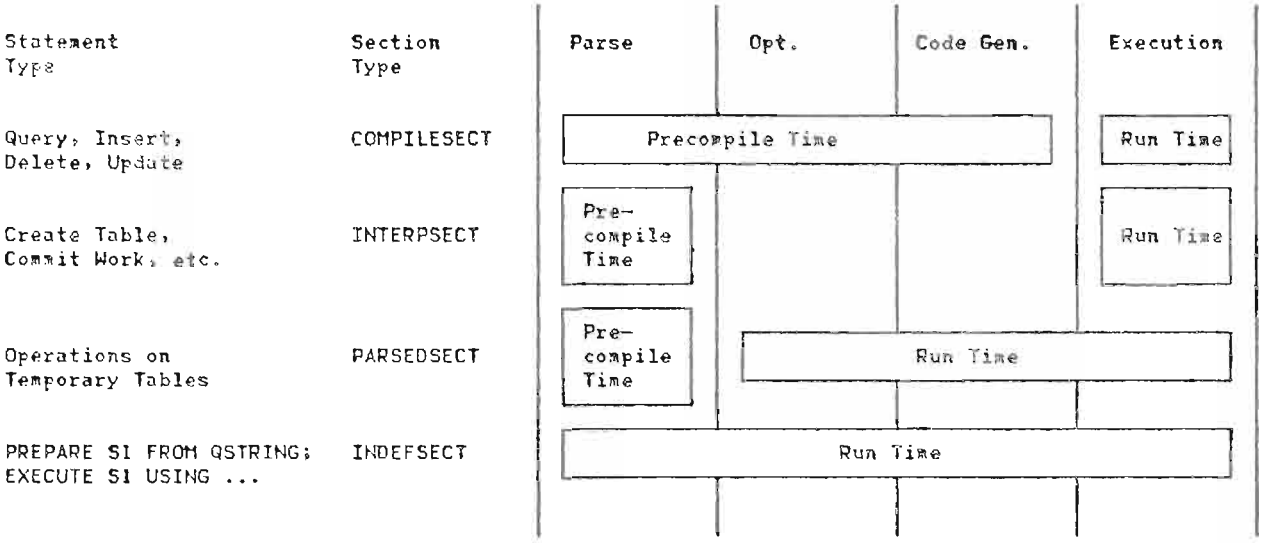
4. INTERPSECT - Has no data associated with it. It is represented only by an entry in the SLT, indicating the section type is "INDEFSECT". This results from PREPARE/EXECUTE, EXECUTE IMMEDIATE, or DESCRIBE statement.

Also stored in the access module for COMPILESECTs, PARSESECTs, and INTERPSECTs is the original SQL statement that caused the section to be built. This is used in case a re-PREP is required.

Example of an Access Module Structure



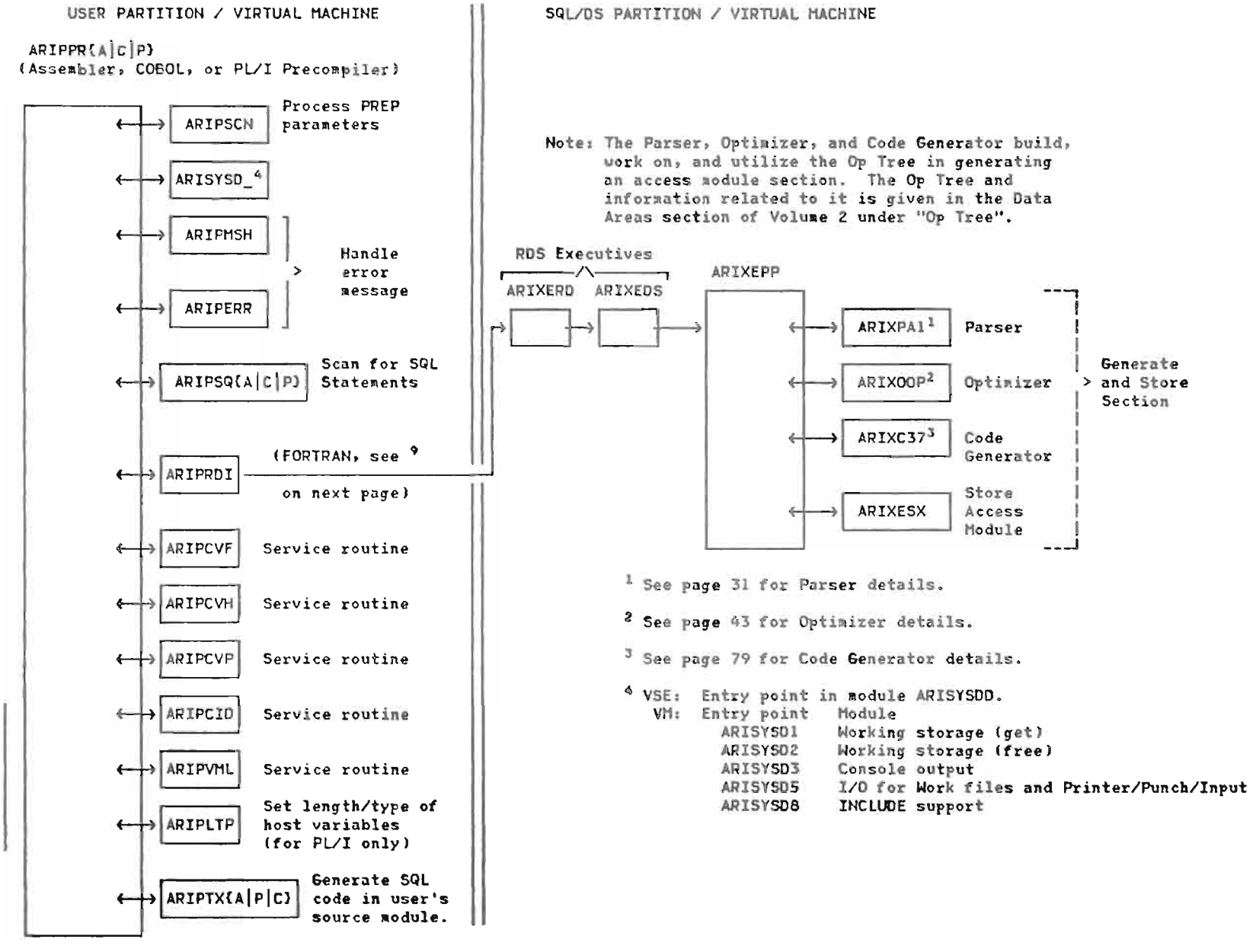
SPECTRUM OF BINDING TIMES IN SQL/DS



SECTION TYPES AND CALL TYPES

	COMPILESECT	INTERPSECT	PARSEDSECT	INDEFSECT
AUXCALL	Execute the machine code in the section.	Execute a standard routine controlled by the content of the section.	Invoke the Optimizer and Code Generator to convert the section into a COMPILESECT or INTERPSECT then execute it.	Invoke the Parser, Optimizer, and Code Generator to convert a new SQL statement into a COMPILESECT or INTERPSECT.
OPENCALL	Execute the machine code in the section, with OPCODE=OPEN.	(Not used)	(Not used)	(Not used)
CLOSECALL	Execute the machine code in the section, with OPCODE=CLOSE.	(Not used)	(Not used)	(Not used)
SETUPCALL	Throw away the current content of the section and invoke the Parser, Optimizer, and Code Generator to build a new COMPILESECT or INTERPSECT from a new SQL statement.		(Not used)	(Not used)
DESCRIBECALL	Return a description of the answer set.	(Not used)	(Not used)	(Not used)

THE PREP PROCESS FOR ASSEMBLER, PL/I, AND COBOL



1 See page 31 for Parser details.

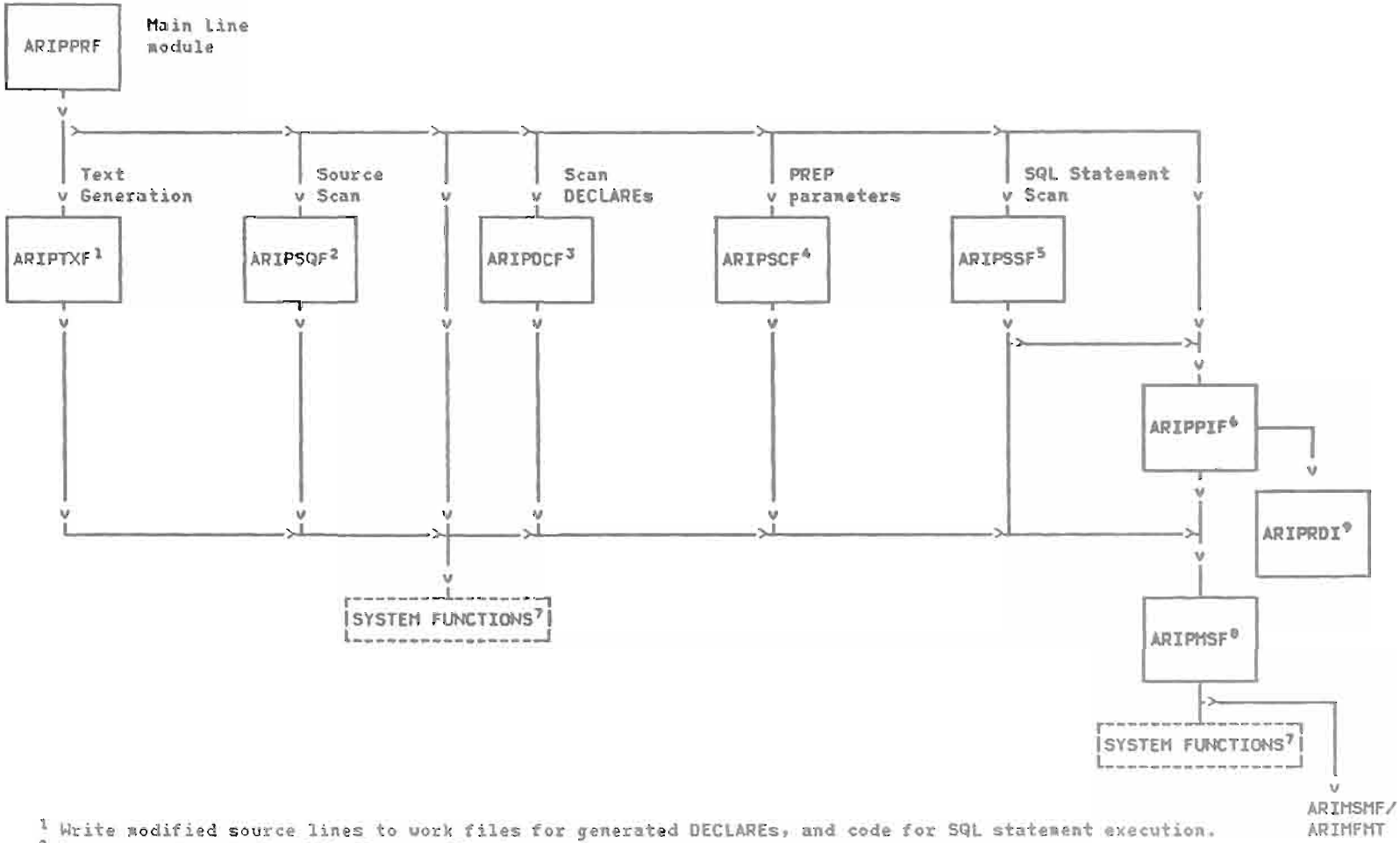
2 See page 43 for Optimizer details.

3 See page 79 for Code Generator details.

4 VSE: Entry point in module ARISYSDD.
VM: Entry point Module
ARISYSD1 Working storage (get)
ARISYSD2 Working storage (free)
ARISYSD3 Console output
ARISYSD5 I/O for Work files and Printer/Punch/Input
ARISYSD8 INCLUDE support

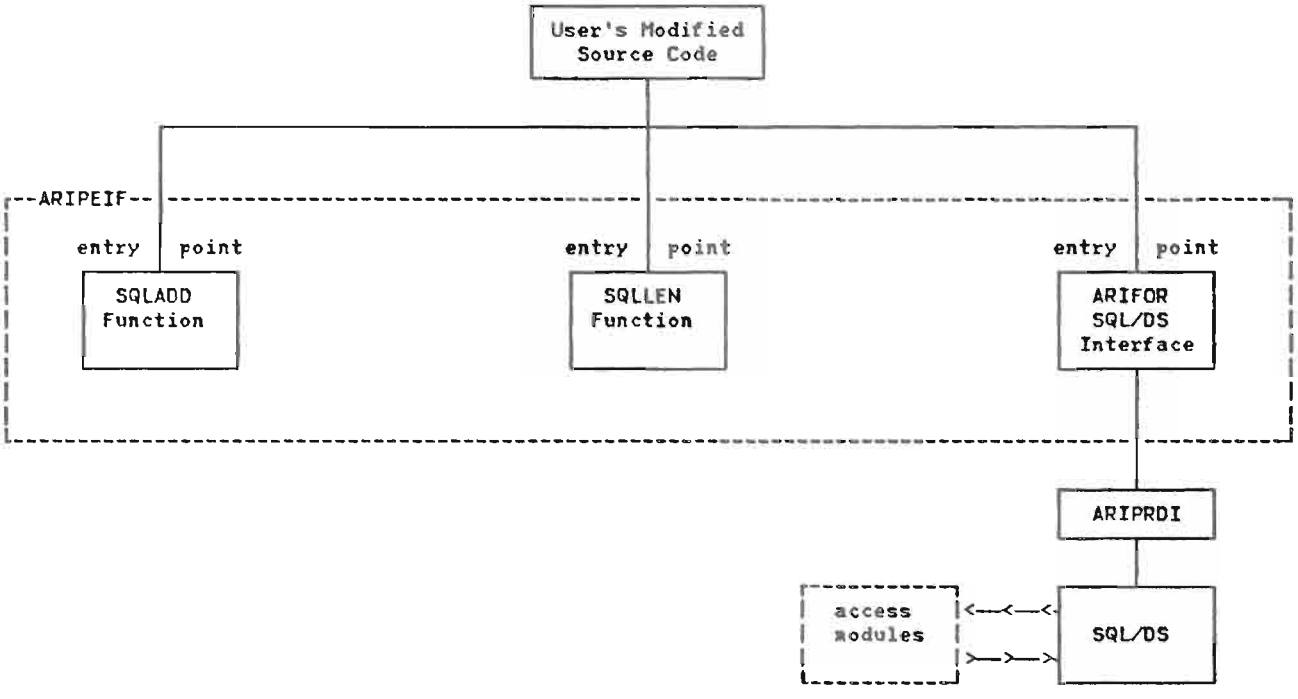
PREP TIME MODULE FLOW DIAGRAM FOR FORTRAN

Flow is from left to right and downward. All called modules return to the calling modules (first statement after the CALL)



[1] Write modified source lines to work files for generated DECLAREs, and code for SQL statement execution.
[2] Locate host variable DECLAREs and SQL statements.
[3] Identify valid host variable declarations and specified type and length. Place in DCLLIST.
[4] Identify invocation parameters specified, and return parameter values.
[5] Identify SQL statement type and locate host variables. Replace variables with "?".
[6] Assembler SQL module to do CREATE PROG, PREPAREs, etc., (functions of API interface).
[7] Performs system-dependent I/O and get/free storage requirements. Write to LST, SYS001, SYS002, and PCH. Read from SYSIPT and SYS001, SYS002.
[8] Handle PREP and SQLCODE messages.
[9] Normal SQL functions.

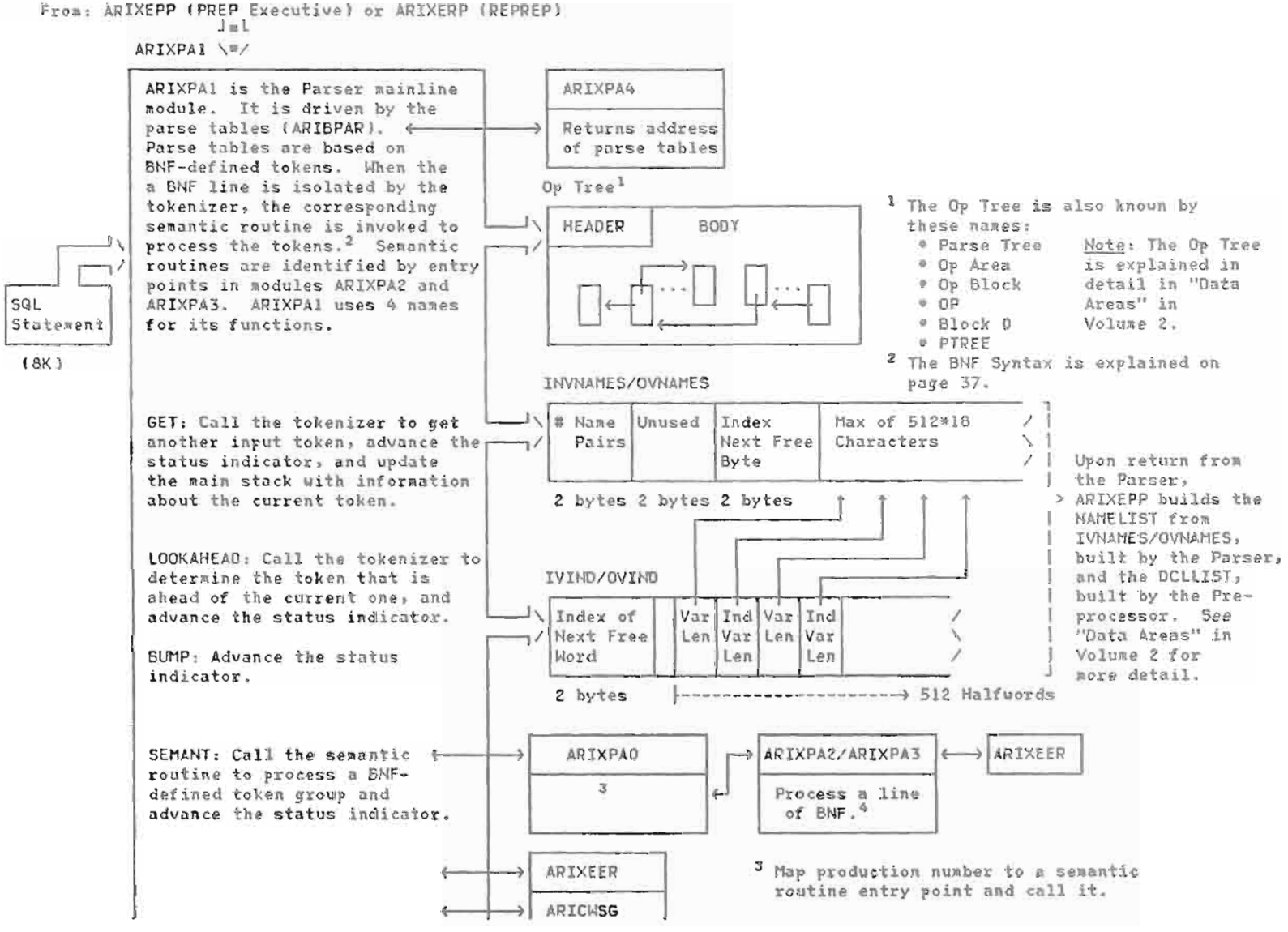
EXECUTION TIME MODULE FLOW DIAGRAM



PARSER

The Parser section contains an overview of Parser functions, an explanation of the stack addressing used by the Parser, and BNF (Backus Normal Form) syntax as an aid in understanding the Parser and other RDS functions as well.

If you do not have a basic understanding of the Op Tree and/or BNF, you should review this section and the subsection "Op Tree" in "Data Areas" of Volume 2.



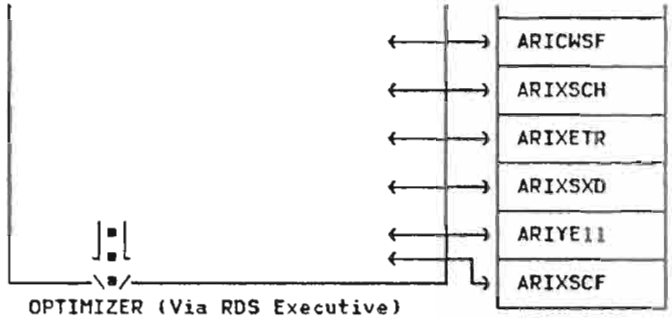
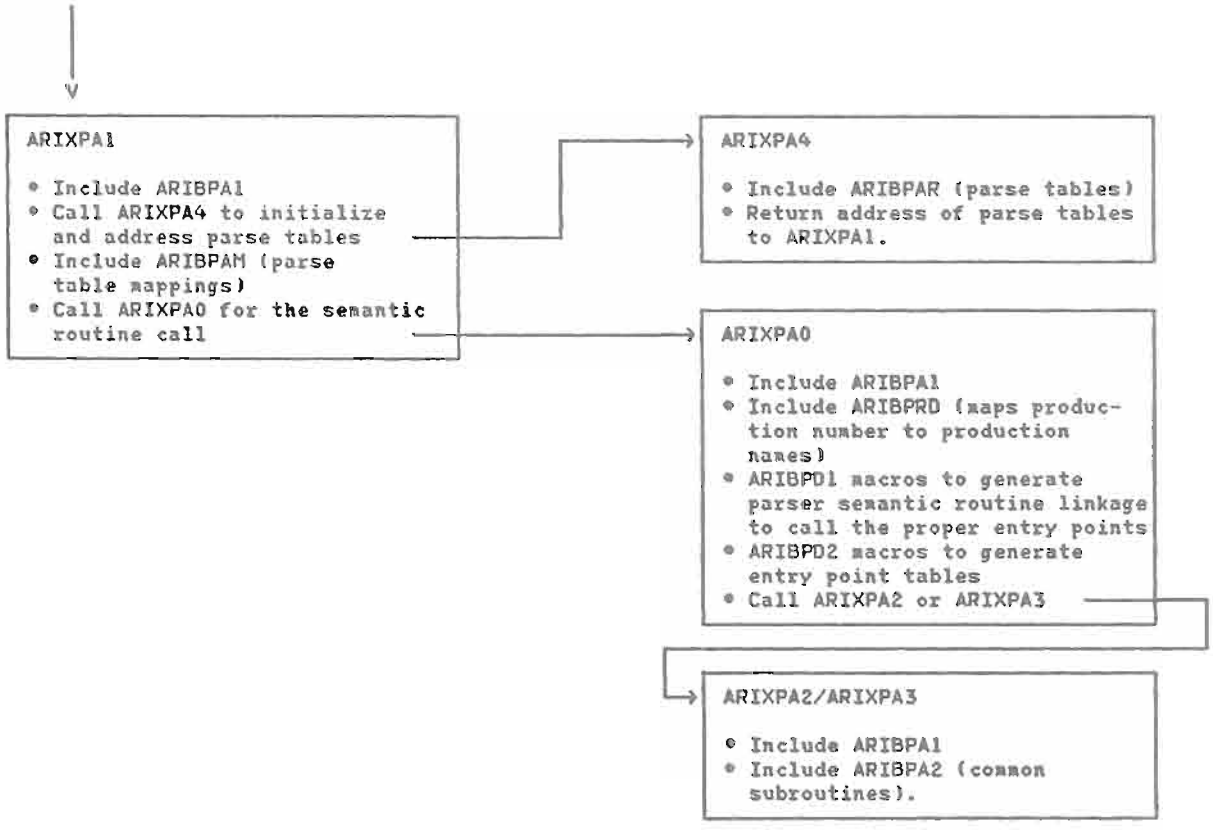
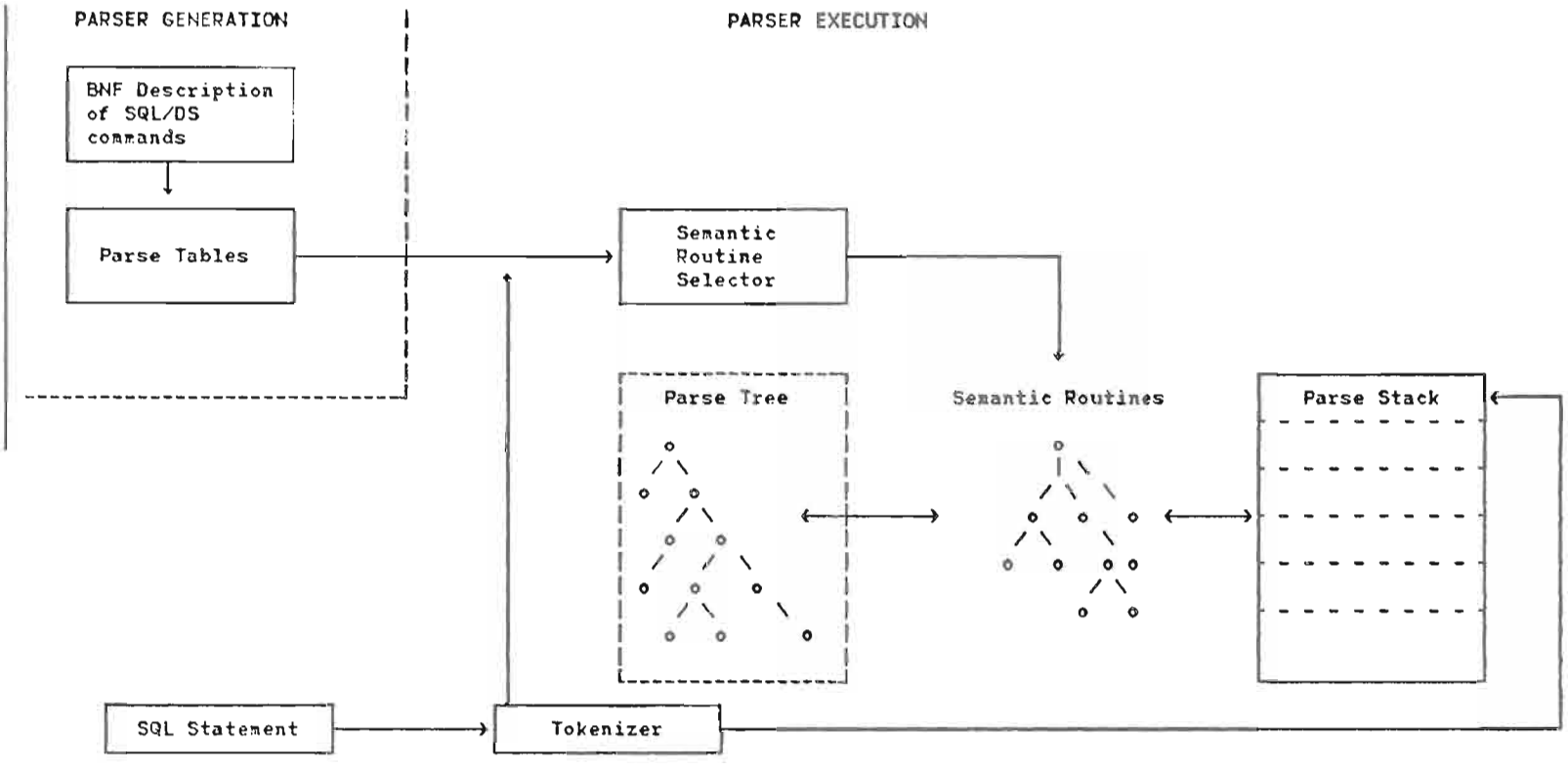


Figure 10. RDS Parser Module Flow

[4] The BNF Syntax is given on page 37.

PARSER MODULE/MACRO STRUCTURE



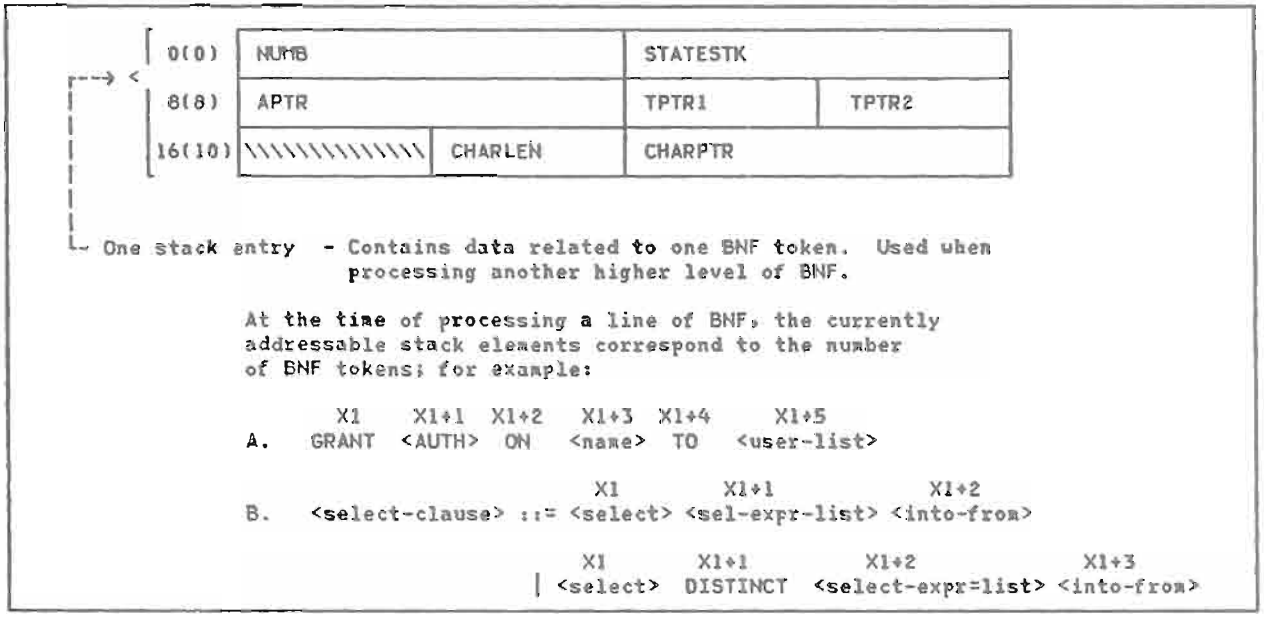


PARSER MAIN STACK ADDRESSING

The following description concerns the semantic routines. Lower level semantic routines save data in main stack entries for higher level semantic routines.

Each stack entry is made up of a set of named storage slots for saving data of different types. There are a couple of slots for halfword data (TPTR1 and TPTR2), one for character strings (CHARLEN (length) and CHARPTR (pointer to the string)) etc.

The following diagram shows the format of the main stack entry:



The main stack entries are addressed via the index 'X1'. The current topmost entry that applies to a particular semantic routine is addressed with the X1 index. The next stack entry is addressed with X1+1, etc. For each semantic routine, the pertinent stack entries correspond to each BNF token. For example, take the two BNF lines:

```
1.   <sel-clause> ::= <select> <sel-expr-list> <into-from>
2.         | <select> DISTINCT <sel-expr-list> <into-from>
```

The semantic routine for the first line uses the following indexes for the specified BNF tokens:

```
X1   - <select>
X1+1 - <sel-expr-list>
X1+2 - <into-from>
```

The second line of BNF uses the following indexes:

```
X1   - <select>
X1+1 - DISTINCT
X1+2 - <sel-expr-list>
X1+3 - <into-from>
```

Therefore, the semantic routine for the second line, for example, would be able to address the results of the lower-level semantic routine, <into-from>, by addressing the stack entry using X1+3 as an index. Let's say, for example, that the semantic routine for <into-from> stored an Op Tree node index in TPTR1 of its topmost stack entry (TPTR1(X1)). This same Op Tree node index would be available to the semantic routine associated with the second line of BNF in

our example above by using the addressing: TPTR1(X1+3). A semantic routine should pass results and data to higher-level semantic routines (those higher up in the BNF tree) by storing them in the current topmost stack entry. For example, if TIPTR1 is the stack field used, then TPTR1(X1) is used (the stack entries associated with TPTR1(X1+1), TPTR1(X1+2), etc, are lost at the completion of the current semantic routine, and only the stack entry associated with X1 is preserved).

The following general explanation has to do with BNF lines that describe a list of items. For example, for table names in the "from" list, the BNF takes the following general form:

```
<item-list> ::= <item>
              | <item-list> , <item>
```

This pair of BNF lines indicates that there is a list called <item-list> and that it may have one or more entries, each defined by the name <item>. Since there are two BNF lines for describing a list, there are also two semantic routines for handling it. The first routine is executed only for the first (or only) element in the list. The second semantic routine is executed for each element in the list after the first one. Typically, the first semantic routine will initialize processing, and the second one will continue it. For example, the first might move "1" to a counter (count the number of items) and the second might add "1" to the counter.

BNF (BACKUS NORMAL FORM) SYNTAX

On the following pages is a simplified form of BNF syntax. It is used heavily in the Parser by the semantic routines ARIXPA2 and ARIXPA3. See the processing description on page 31.

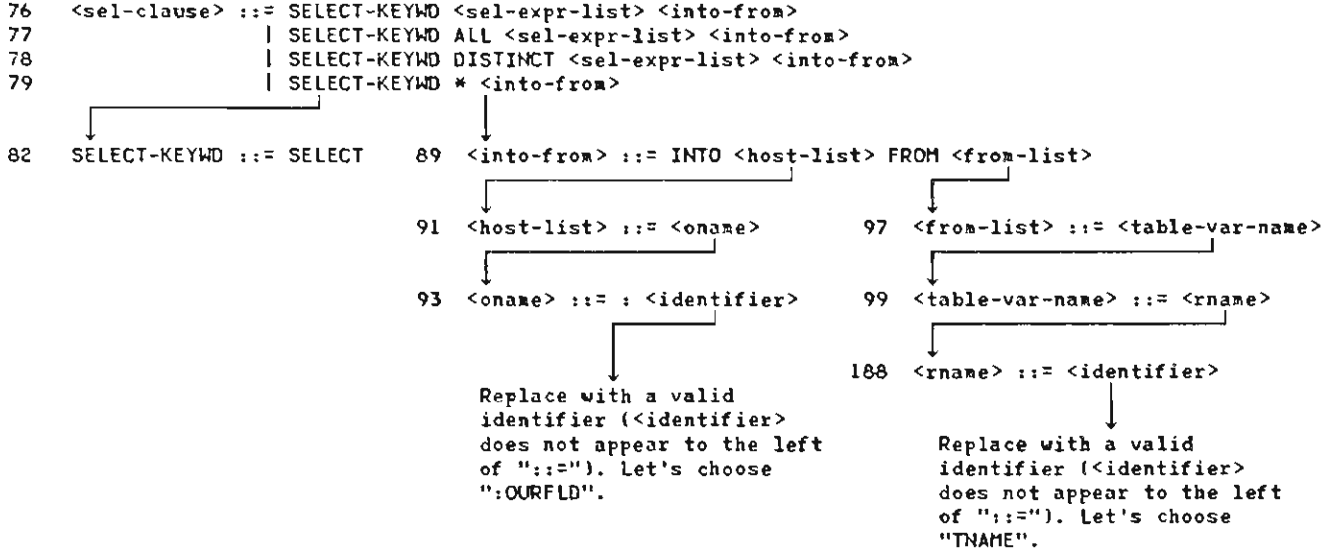
The following description is an explanation and an example of how to read the syntax:

The items enclosed in the pairs of characters "<" and ">" are variables. Items in capital letters, numbers, or punctuation (including parentheses) are exact replacements.

You will always find a variable on the left side of "::=". The "::=" has the meaning "is to be replaced by". For each variable on the right side of "::=", look further in the list on the left side of "::=" for that same variable. There it may give exact replacements, or it may contain still other variables. Continue in this fashion until all replacements have been made. Follow this next sequence as an example (the line numbers refer to the BNF numbers at the left of the page): Line 1 says that the "goal" is to be replaced by a "statement". "<statement>" starts at line 2, which says that "statement" is to be replaced by either the variable "query", or "INSERT INTO ...", or "DELETE FROM ...", or UPDATE ...", etc (the replacements continue down to line 34). Let's choose "<query>" (we will do a simple query).

Go to line 45. Line 45 though line 48 show the replacements for "query". Let's choose "<query-expr>", which appears on the left of "::=" at line 65. There we see "<query-expr> ::= <query-block>" (another replacement follows on the next line, but let's use "<query-block>").

Lines 68 through 75 show the possible replacements for "query-block"; let's choose "<sel-clause>", and go to lines 76 though 81 to see its replacements. Now that we've gotten this far, let's look at the rest graphically, taking only the indicated replacement (last one of the list shown):



Now taking all the replacements from beginning to end, we finish with the statement:

```
SELECT * INTO :OURFLD FROM :TNAME
```

Following is the current BNF for the RDS Parser

```
ENTRY POINTS
(ARIXPA2/ARIXPA3)        BNF

GOAL010     1   <goal> ::= <statement> ;
STMNT01     2   <statement> ::= EXPLAIN <explain-spec> FOR <access-stmt>
STMNT02     3          | EXPLAIN <explain-spec> SET <identifier> = <integer-spec> FOR <access-stmt>
STMNT03     4          | <access-stmt>
STMNT04     5          | ACQUIRE <dbs-type> DBSPACE NAMED <name>
STMNT05     6          | ACQUIRE <dbs-type> DBSPACE NAMED <name> ( <dbs-parms> )
STMNT06     7          | CREATE TABLE <name> ( <field-defn-lst> )
STMNT07     8          | CREATE TABLE <name> ( <field-defn-lst> ) IN <name>
STMNT08     9          | ALTER TABLE <name> ADD <field-defn>
STMNT09    10          | ALTER DBSPACE <name> ( <change-s-lst> )
STMNT10    11          | CREATE INDEX <indef-clause>
STMNT11    12          | CREATE UNIQUE INDEX <indef-clause>
STMNT12    13          | CREATE VIEW <receiver> AS <query-expr>
STMNT13    14          | DROP <system-entity> <pname>
STMNT14    15          | COMMENT ON <system-entity> <name> IS <quot-string>
STMNT15    16          | COMMENT ON COLUMN <identifier> . <identifier> IS <quot-string>
STMNT16    17          | COMMENT ON COLUMN <identifier> . <identifier> . <identifier> IS <quot-string>
STMNT17    18          | CREATE SYNONYM <identifier> FOR <identifier> . <identifier>
STMNT18    19          | GRANT <auth> ON <name> TO <userorpw-list>
STMNT19    20          | GRANT <auth> ON <name> TO <userorpw-list> WITH GRANT OPTION
STMNT20    21          | GRANT <special-priv> TO <userorpw-list>
STMNT21    22          | GRANT <special-priv> TO <userorpw-list> IDENTIFIED BY <userorpw-list>
STMNT22    23          | REVOKE <auth> ON <name> FROM <userorpw-list>
STMNT23    24          | REVOKE <special-priv> FROM <userorpw-list>
STMNT24    25          | COMMIT WORK
STMNT25    26          | COMMIT WORK RELEASE
STMNT26    27          | ROLLBACK WORK
STMNT27    28          | ROLLBACK WORK RELEASE
STMNT28    29          | LOCK TABLE <name> IN <mode-setting> MODE
STMNT29    30          | LOCK DBSPACE <name> IN <mode-setting> MODE
STMNT30    31          | UPDATE STATISTICS FOR TABLE <name>
STMNT31    32          | UPDATE ALL STATISTICS FOR TABLE <name>
STMNT32    33          | UPDATE STATISTICS FOR DBSPACE <name>
STMNT33    34          | UPDATE ALL STATISTICS FOR DBSPACE <name>
EXIT01     35   <access-stmt> ::= <query>
ACSTM02    36          | INSERT INTO <receiver> <insert-spec>
ACSTM03    37          | DELETE FROM <table-var-name>
ACSTM04    38          | DELETE FROM <table-var-name> <where-clause>
ACSTM05    39          | UPDATE <table-var-name> SET <set-clause-lst>
ACSTM06    40          | UPDATE <table-var-name> SET <set-clause-lst> <where-clause>
EXIT01     41   <explain-spec> ::= <explain-list>
EXPSPC2    42          | ALL
EXPLST1    43   <explain-list> ::= <identifier>
EXPLST2    44          | EXPLAIN-LIST , <identifier>
QUERY010   45   <query> ::= <query-expr>
QUERY020   46          | <query-expr> <order-by> <ord-spec-list>
QUERY030   47          | <query-expr> FOR UPDATE OF <field-name-lst>
QUERY040   48          | <query-expr> order-by <ord-spec-list> FOR UPDATE OF <field-name-lst>
ORDBY01    49   <order-by> ::= ORDER BY
```

```
EXIT01          50    <receiver> ::= <rname>
EXIT01          51         | <subtab>
SUBTB01         52    <subtab> ::= <rname> ( <field-name-lst> )
FLDNLT01        53    <field-name-lst> ::= <identifier>
FLDNLT02        54                       | <field-name-lst> , <identifier>
INSPEC01        55    <insert-spec> ::= <query-block>
INSPEC03        56                      | <values> ( <entry-list> )
INSPEC03        57                      | <values> ( <entry> )
WHERCL01        58    <uhere-clause> ::= WHERE <boolean>
WHERCL02        59                     | WHERE CURRENT OF <cname>
SCLLST01        60    <set-clause-lst> ::= <set-clause>
SCLLST02        61                       | <set-clause-lst> , <set-clause>
SETCLS01        62    <set-clause> ::= <identifier> = <expr>
SETCLS02        63                   | <identifier> = ( <query-block> )
SETCLS03        64                   | <identifier> = NULL
EXIT01          65    <query-expr> ::= <query-block>
QUEEXP02        66                     | <query-expr> UNION <query-block>
QUEEXP03        67                     | ( <query-expr> )
QUEBLK01        68    <query-block> ::= <sel-clause>
QUEBLK02        69                      | <sel-clause> WHERE <boolean>
QUEBLK03        70                      | <sel-clause> GROUP BY <field-spec-lst>
QUEBLK04        71                      | <sel-clause> <having-keyud> <boolean>
QUEBLK05        72                      | <sel-clause> GROUP BY <field-spec-lst> <having-keyud> <boolean>
QUEBLK06        73                      | <sel-clause> WHERE <boolean> GROUP BY <field-spec-lst>
QUEBLK07        74                      | <sel-clause> WHERE <boolean> <having-keyud> <boolean>
QUEBLK08        75                      | <sel-clause> WHERE <boolean> GROUP BY <field-spec-lst> <having-keyud> <boolean>
SELCLS01        76    <sel-clause> ::= <select-keyud> <sel-expr-lst> <into-from>
SELCLS02        77                     | <select-keyud> ALL <sel-expr-lst> <into-from>
SELCLS03        78                     | <select-keyud> DISTINCT <sel-expr-lst> <into-from>
SELCLS04        79                     | <select-keyud> * <into-from>
SELCLS05        80                     | <select-keyud> ALL * <into-from>
SELCLS06        81                     | <select-keyud> DISTINCT * <into-from>
SELECT01        82    <select-keyud> ::= SELECT
HAVING01        83    <having-keyud> ::= HAVING
SLEPRL01        84    <sel-expr-lst> ::= <sel-expr>
SLEPRL02        85                       | <sel-expr-lst> , <sel-expr>
SELEXP01        86    <sel-expr> ::= <expr>
SELEXP02        87                   | <identifier> . *
SELEXP03        88                   | <identifier> . <identifier> . *
INTFRM01        89    <into-from> ::= INTO <host-list> FROM <from-list>
INTFRM02        90                    | FROM <from-list>
HSTLST01        91    <host-list> ::= <oname>
HSTLST02        92                    | <host-list> , <oname>
ONAME010        93    <oname> ::= : <identifier>
ONAME020        94                  | : <identifier> : <identifier>
HNAME010        95    <hname> ::= : <identifier>
HNAME020        96                | : <identifier> . : <identifier>
FRMLST01        97    <from-list> ::= <table-var-name>
FRMLST02        98                    | <from-list> , <table-var-name>
TABVNM01        99    <table-var-name> ::= <rname>
TABVNM02       100                       | <rname> <identifier>
FLDSPL01       101    <field-spec-lst> ::= <field-spec>
FLDSPL02       102                       | <field-spec-lst> , <field-spec>
COLOSL01       103    <col-ord-s-lst> ::= <col-ord-spec>
```

```
COLOSL02        104                 | <col-ord-s-lst> , <col-ord-spec>
ORDSL01         105     <ord-spec-list> ::= <ord-spec>
ORDSL02         106                 | <ord-spec-list> , <ord-spec>
EXIT01          107     <boolean> ::= <boolean-term>
BOOL02          108                 | <boolean> OR <boolean-term>
EXIT01          109     <boolean-term> ::= <boolean-factor>
BOLTM02         110                 | <boolean-term> AND <boolean-factor>
EXIT01          111     <boolean-factor> ::= <boolean-primry>
BOOLFC02        112                 | NOT <boolean-primry>
EXIT01          113     <boolean-primry> ::= <predicate>
BOOLPR02        114                 | ( <boolean> )
PREDCT01        115     <predicate> ::= <expr> <comparison> <expr>
PREDCT02        116                 | <expr> BETWEEN <expr> AND <expr>
PREDCT03        117                 | <expr> NOT BETWEEN <expr> AND <expr>
PREDCT04        118                 | <primary> IS NULL
PREDCT05        119                 | <primary> IS NOT NULL
PREDCT06        120                 | <expr> LIKE <constant>
PREDCT07        121                 | <expr> NOT LIKE <constant>
PREDCT08        122                 | <expr> <comparison> ( <entry-list> )
PREDCT09        123                 | <expr> <comparison> ( <query-expr> )
PREDCT10        124                 | EXISTS ( <query-expr> )
EXIT01          125     <expr> ::= <arith-term>
EXPR020         126                 | <expr> <add-op> <arith-term>
EXIT01          127     <arith-term> ::= <arith-factor>
ARTRM02         128                 | <arith-term> <mult-op> <arith-factor>
EXIT01          129     <arith-factor> ::= <primary>
ARFACT02        130                 | <add-op> <primary>
PRIMAR01        131     <primary> ::= <field-spec>
EXIT01          132                 | <set-fn>
EXIT01          133                 | <constant>
PRIMAR04        134                 | ( <expr> )
FLDSPC01        135     <field-spec> ::= <identifier>
FLDSPC02        136                 | <identifier> . <identifier>
FLDSPC03        137                 | <identifier> . <identifier> . <identifier>
COLOSP02        138     <col-ord-spec> ::= <field-spec>
COLOSP02        139                 | <field-spec> ASC
COLOSP03        140                 | <field-spec> DESC
ORDSPC02        141     <ord-spec> ::= <integer-spec>
ORDSPC02        142                 | <integer-spec> ASC
ORDSPC03        143                 | <integer-spec> DESC
ORDSPC04        144                 | <col-ord-spec>
EXIT01          145     <comparison> ::= <comp>
COMPAR02        146                 | <comp> ALL
COMPAR03        147                 | <comp> ANY
COMPAR04        148                 | IN
COMPAR05        149                 | NOT IN
COMP010         150     <comp> ::= =
COMP020         151                 | ¬ =
COMP030         152                 | >
COMP040         153                 | > =
COMP050         154                 | <
COMP060         155                 | < =
ADDOP01         156     <add-op> ::= +
ADDOP02         157                 | -
```

```
MULTOP01        158   <mult-op> ::= *
MULTOP02        159                 | /
SETFN01         160   <set-fn> ::= AVG ( <arg> )
SETFN02         161                 | MAX ( <arg> )
SETFN03         162                 | MIN ( <arg> )
SETFN04         163                 | SUM ( <arg> )
SETFN05         164                 | COUNT ( <arg> )
SETFN06         165                 | COUNT ( * )
SETFN07         166                 | <identifier> ( <arg> )
ARG010          167   <arg> ::= <expr>
ARG020          168                 | DISTINCT <expr>
ARG030          169                 | ALL <expr>
ENTLST02        170   <entry-list> ::= <entry> , <entry>
ENTLST02        171                 | <entry-list> , <entry>
EXIT01          172   <entry> ::= <constant>
ENTRY020        173                 | <add-op> <number>
ENTRY030        174                 | NULL
CNSTNT02        175   <constant> ::= <quot-string>
CNSTNT02        176                 | <number>
CNSTNT03        177                 | : <identifier>
CNSTNT04        178                 | : <identifier> : <identifier>
CNSTNT05        179                 | USER
CNSTNT06        180                 | ?
CNSTNT02        181                 | <graph-string>
CNSTNT02        182                 | <hex-string>
PNAME010        183   <pname> ::= <hname>
EXIT01          184                 | <name>
NAME010         185   <name> ::= <identifier>
NAME030         186                 | <identifier> . <identifier>
NAME030         187                 | PUBLIC . <identifier>
RNAME010        188   <rname> ::= <identifier>
RNAME030        189                 | <identifier> . <identifier>
RNAME030        190                 | PUBLIC . <identifier>
CNAME010        191   <cname> ::= <identifier>
INTEGR01        192   <integer-spec> ::= <number>
DBSTYP01        193   <dbs-type> ::= PUBLIC
DBSTYP02        194                 | PRIVATE
EXIT01          195   <dbs-parms> ::= <dbs-spec>
EXIT01          196                 | <dbs-parms> , <dbs-spec>
DBSSPC01        197   <dbs-spec> ::= NHEADER = <integer-spec>
DBSSPC02        198                 | PAGES = <integer-spec>
DBSSPC03        199                 | PCTINDEX = <integer-spec>
DBSSPC04        200                 | PCTFREE = <integer-spec>
DBSSPC05        201                 | LOCK = <lock-spec>
DBSSPC06        202                 | STORPOOL = <integer-spec>
LOCK01          203   <lock-spec> ::= DBSPACE
LOCK02          204                 | PAGE
LOCK03          205                 | ROW
FDFLST01        206   <field-defn-lst> ::= <field-defn>
FDFLST02        207                 | <field-defn-lst> , <field-defn>
FLDDFN01        208   <field-defn> ::= <identifier> <type>
FLDDFN02        209                 | <identifier> <type> NOT NULL
TYPE010         210   <type> ::= char ( <integer-spec> )
TYPE020         211                 | VARCHAR ( <integer-spec> )
```

```
TYPE030          212                 | LONG VARCHAR
TYPE040          213                 | INTEGER
TYPE050          214                 | SMALLINT
TYPE060          215                 | DECIMAL
TYPE070          216                 | DECIMAL ( <integer-spec> )
TYPE080          217                 | DECIMAL ( <integer-spec> , <integer-spec> )
TYPE090          218                 | FLOAT
TYPE100          219                 | GRAPHIC ( <integer-spec> )
TYPE110          220                 | VARGRAPHIC ( <integer-spec> )
TYPE120          221                 | LONG VARGRAPHIC
EXIT01           222    <change-s-lst> ::= <change-spec>
EXIT01           223                 | <change-s-lst> , <change-spec>
CHSPEC01         224    <change-spec> ::= PCTFREE = <integer-spec>
CHSPEC02         225                 | LOCK = <lock-spec>
INDFCL01         226    <indef-clause> ::= <name> ON <name> ( <col-ord-s-lst> )
INDFCL02         227                 | <name> ON <name> ( <col-ord-s-lst> ) PCTFREE = <integer-spec>
SYSENT01         228    <system-entity> ::= TABLE
SYSENT02         229                 | VIEW
SYSENT03         230                 | PROGRAM
SYSENT04         231                 | INDEX
SYSENT05         232                 | DBSPACE
SYSENT06         233                 | SYNONYM
EXIT01           234    <auth> ::= <operation-list>
AUTH030          235                 | ALL PRIVILEGES
AUTH030          236                 | ALL
USRPLS02         237    <userorpw-list> ::= PUBLIC
USRPLS02         238                 | <identifier>
USRPLS03         239                 | <userorpw-list> , <identifier>
OPLIST01         240    <operation-list> ::= <operation>
OPLIST02         241                 | <operation-list> , <operation>
OPERAT01         242    <operation> ::= SELECT
OPERAT02         243                 | INSERT
OPERAT03         244                 | DELETE
OPERAT04         245                 | UPDATE
OPERAT05         246                 | UPDATE ( <field-name-lst> )
OPERAT06         247                 | ALTER
OPERAT07         248                 | INDEX
OPERAT08         249                 | RUN
SPPRIV01         250    <special-priv> ::= DBA
SPPRIV02         251                 | RESOURCE
SPPRIV03         252                 | CONNECT
SPPRIV04         253                 | SCHEDULE
MODE01           254    <mode-setting> ::= SHARE
MODE02           255                 | EXCLUSIVE
```

OPTIMIZER

Essentially, the Optimizer performs the following:

1. Accumulates the names of tables and columns in internal tables.

2. Looks up these names in the system catalogs and retrieves information about them.

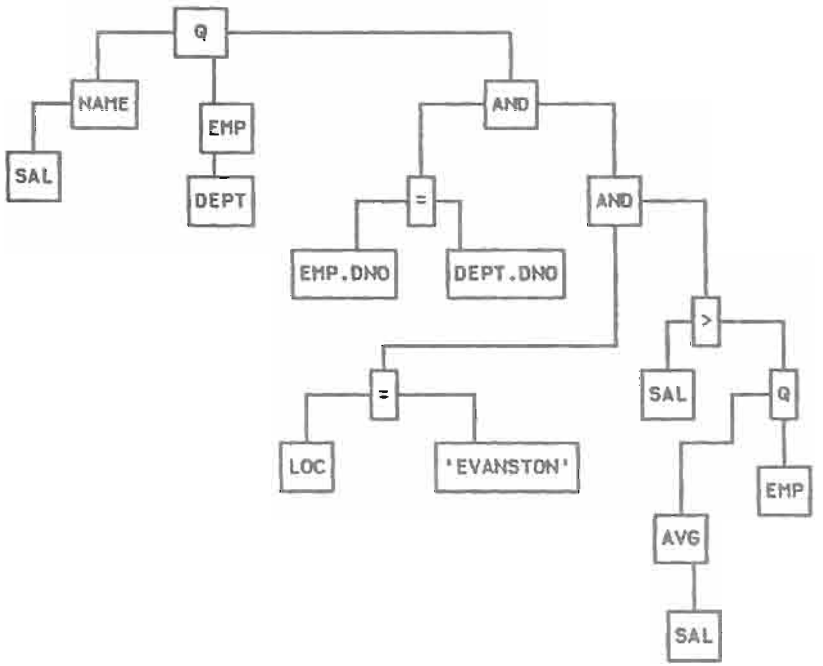
3. Does authorization and records dependencies.

4. Composes any views, if present, into the Op Tree.

5. Does type distribution and checking, analyzes predicates, and accumulates more information in internal tables.

6. Uses the information accumulated in steps 1-5 to make a plan for accessing each table and satisfying the predicates. It modifies the Op Tree to express this plan to the Code Generator as ASL (Access Specification Language) (see Section 5 in Volume 2, "Op Tree Node Encodings" and "Descriptor Records" for the Op Tree and Descriptors). It also creates all DBSS calling structures necessary for the Code Generator to use in DBSS calls.

Part of the input to the Optimizer is an Op Tree, such as in the following figure:



EXAMPLES OF ASL (ACCESS SPECIFICATION LANGUAGE) REPRESENTATIONS AS OUTPUT FROM THE OPTIMIZER

Implied in generating an access plan is a knowledge of what DBSS operations need to be performed to fulfill the user's SQL request.

As part of the output of the Optimizer, the ASL generator builds all the DBSS linkage (parametric) structures in Block1, the Space Block, that will be used in the generated code when issuing calls to DBSS. The typical call to DBSS consists of an operation code and its necessary DBSS parametric or linkage structure.

The majority of DBSS operations are those necessary to fulfill SQL Query, Update, Insert, and Delete requests. The DBSS operators used for these requests are known as "Operators on Rows and Scans". Two of the more common operators are 'Open Scan' and 'Next Row'. These will be referred to in the following SQL statement ASL representations as the boxes labelled 'OPEN' and 'NEXT'. They are pointed to out of the control block associated with a table, the control block being the Optimizer linkage to the Code Generator for DBSS operations against a table.

The DBSS parametric or linkage structure used with operators on rows and scans is referred to in DBSS documentation as "BASE". The BASE is a fixed-length structure. Contained within that structure are three pointers of significance to our ASL representations:

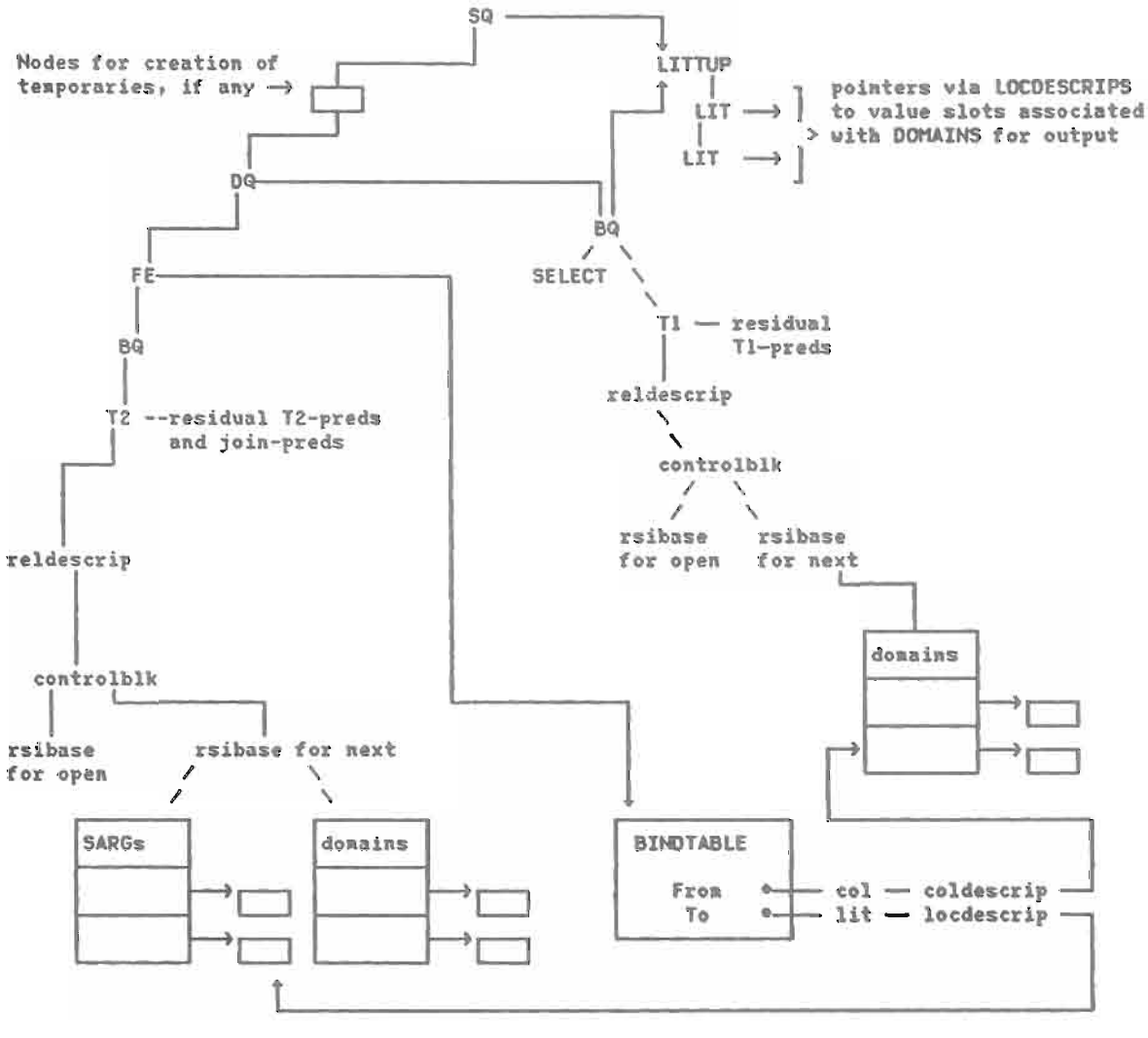
1. A pointer to a 'DOMAINS' structure (DOMS), which describes the requested or submitted fields. Each entry in the DOMAINS structure contains a pointer to the I/O areas used to retrieve/submit field values. These areas also reside in the Space Block.

2. An optional pointer value to a 'KDOMAINS' (KDOMS) structure, which describes a submitted key for an index identified by other fields in BASE. A KDOMAINS entry contains a pointer to a data area in the Space Block. This data area contains the actual key value.

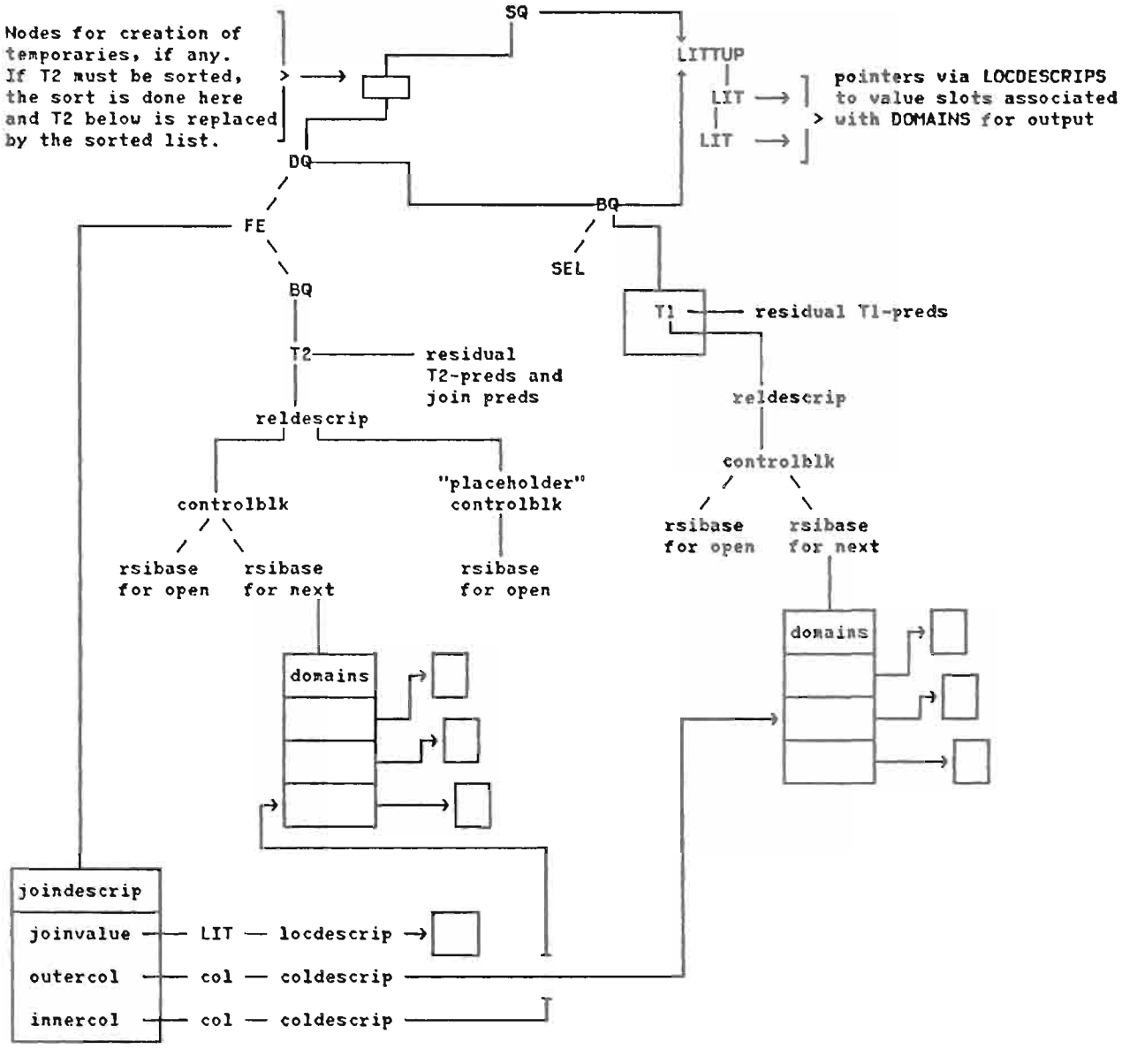
3. An optional pointer value to a 'SARGS' structure, which is used to provide search arguments used in OPEN and NEXT DBSS operations. Each search argument entry describes a row-column number, a comparison operator, and a pointer to an input area (in the Space Block) for the actual column value.

A more detailed description of DBSS operators and parametric structures can be found in Section 6, Diagnostic Aids, under "DBSS Op Codes".

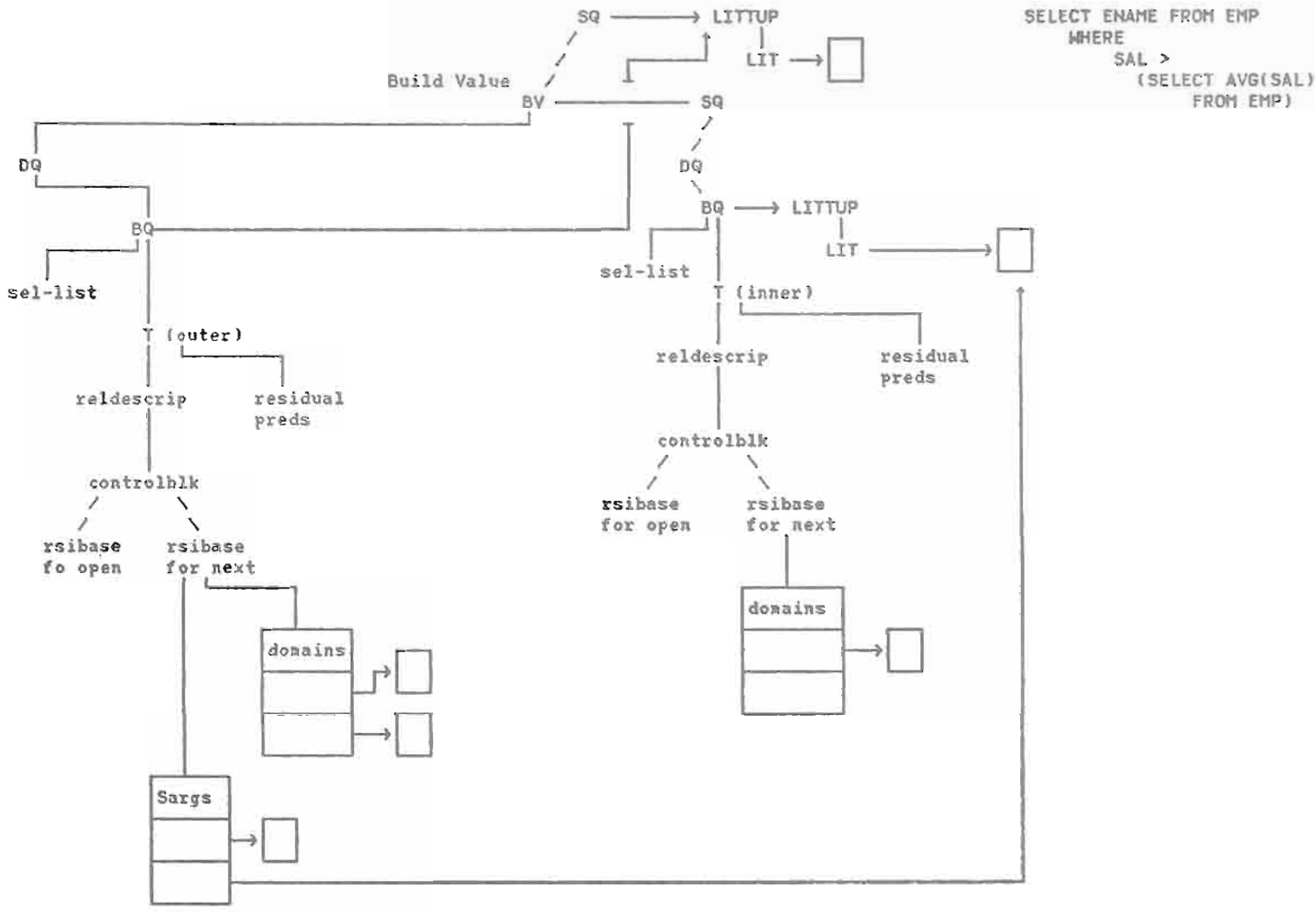
1. JOIN BY METHOD 1 (Nested Loops)



2. JOIN BY METHOD 2 (Sort-Merge)

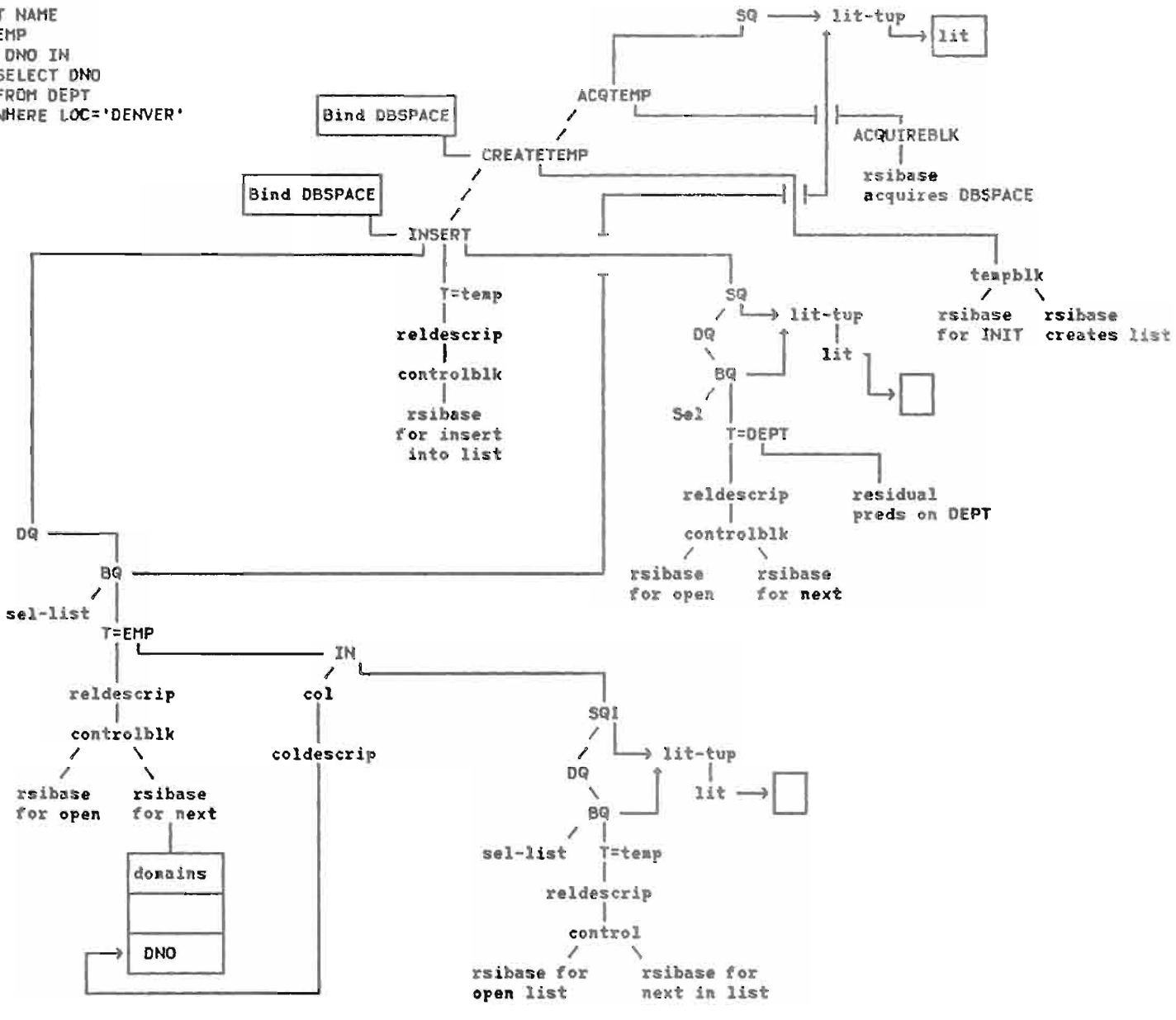


3. UNCORRELATED SUBQUERY RETURNING SCALAR

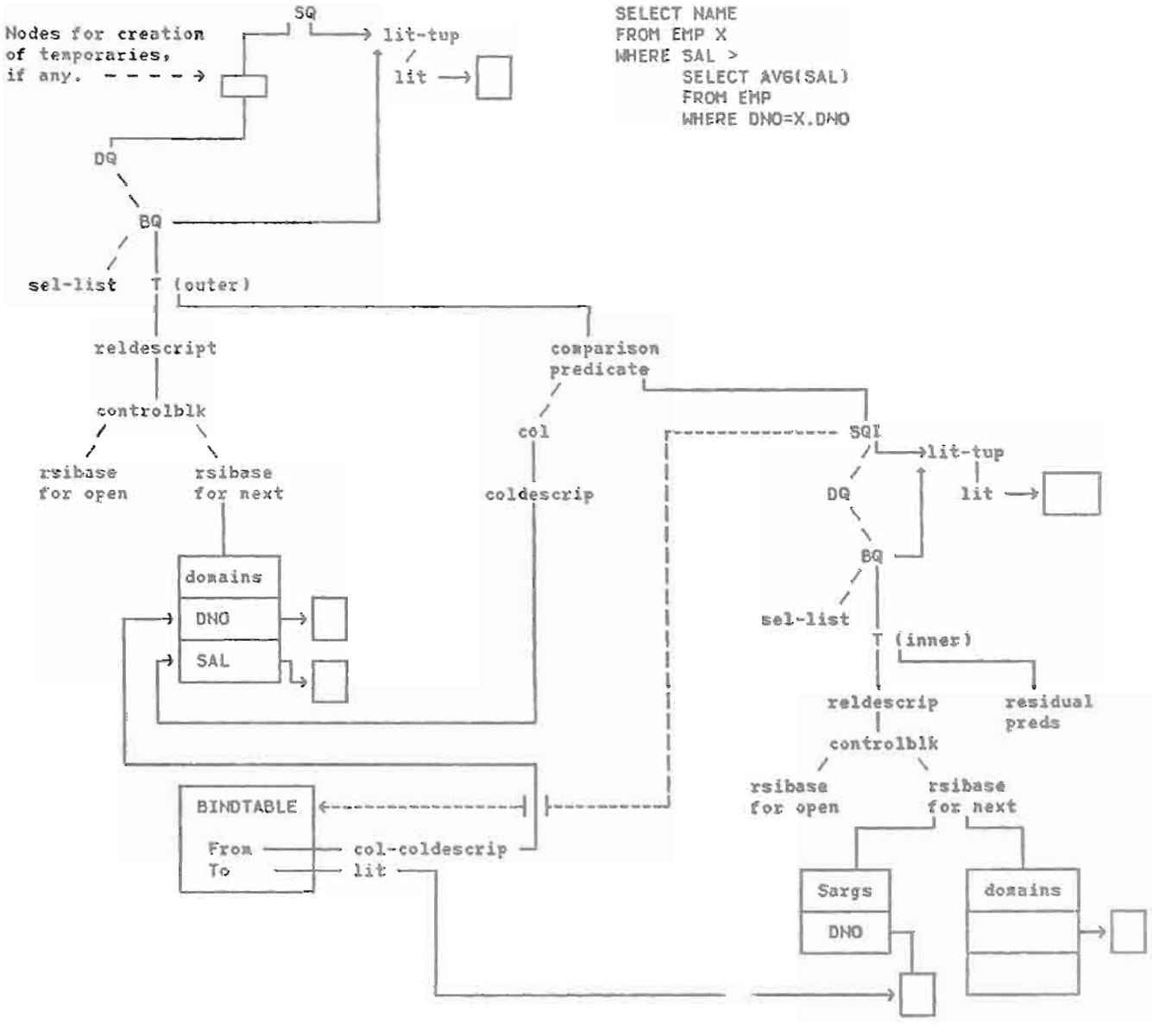


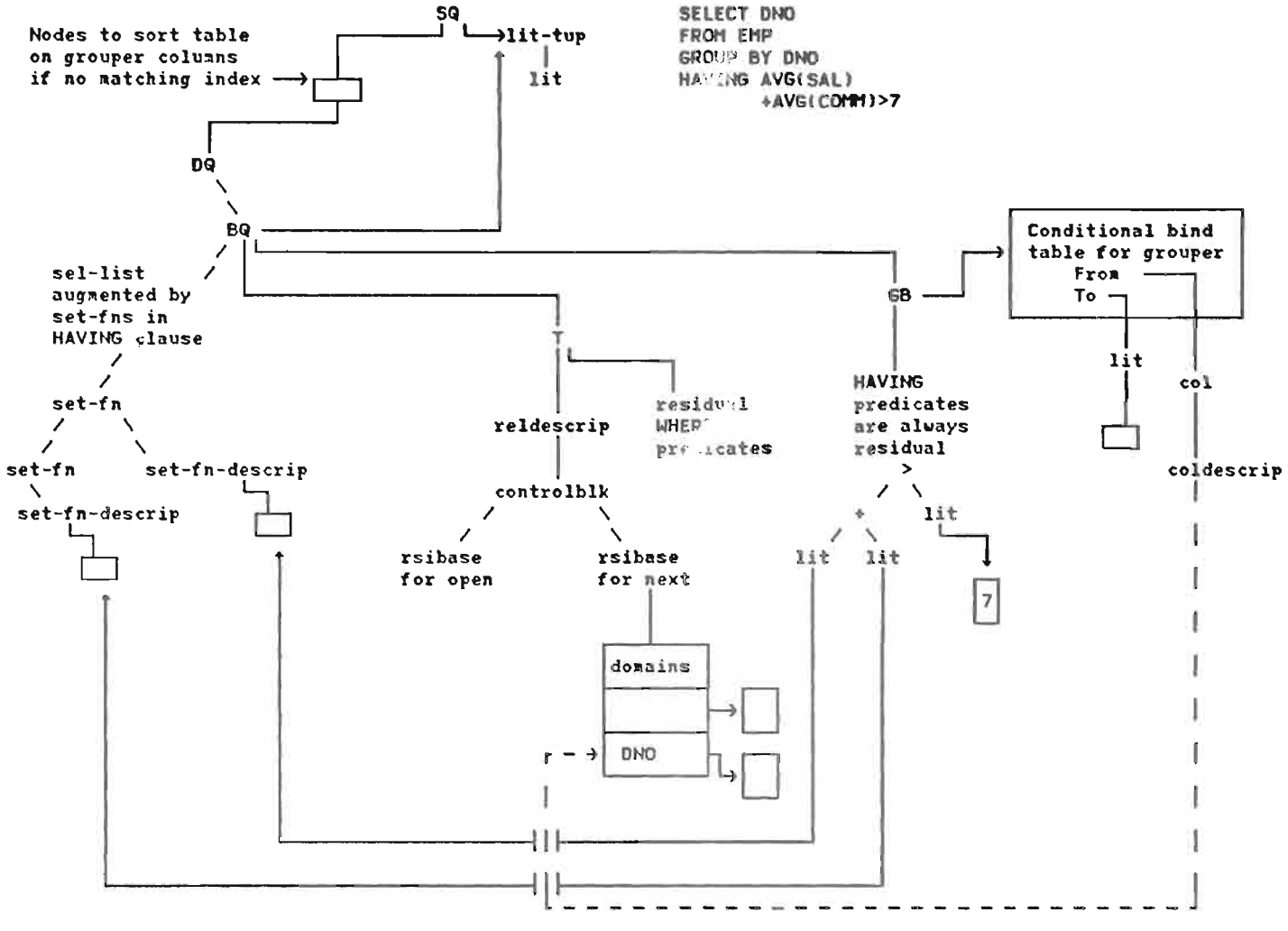
4. UNCORRELATED SUBQUERY RETURNING SET

```
SELECT NAME
FROM EMP
WHERE DNO IN
      SELECT DNO
      FROM DEPT
      WHERE LOC='DENVER'
```



5. CORRELATED SUBQUERY



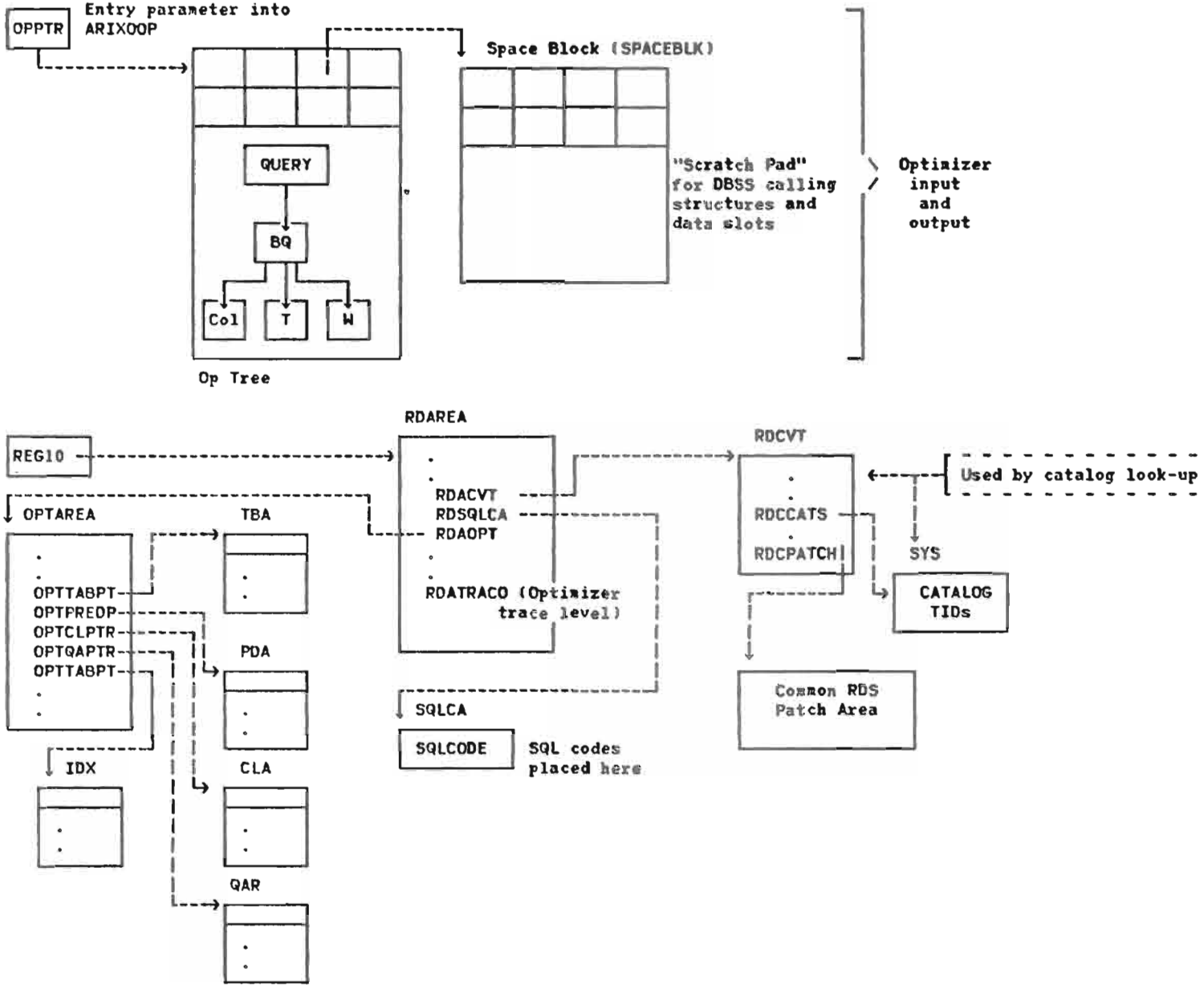


There are five tables (arrays) of significant importance to the Optimizer. (These are the tables mentioned on page 43 as "internal tables".) Pertinent information is accumulated and stored in these tables during Pass 1 and Pass 2 scans of the Op Tree nodes by the Optimizer. These tables are used in determining path selection and generating ASL (Access Specification Language) in the later stages of optimization. Each table has a pointer to it, along with its current element index, in the OPTAREA. The table dimensions are determined by macro variables whose values are assigned in the included code ARIBNUM. The tables and their associated macro dimension variables are:

Table Array	'TBA'	MAXQTAB (10)
Column Array	'CLA'	MAXQCOL (255)
Predicate Array	'PDA'	MAXPRED (25*MAXQ)
Index Array	'IDX'	MAXIMG (10*MAXQTAB)
Query Array	'QAR'	MAXQ (8)

An overview of the major data areas and control blocks follows (for the data areas and control blocks, see the data areas section in Volume 2).

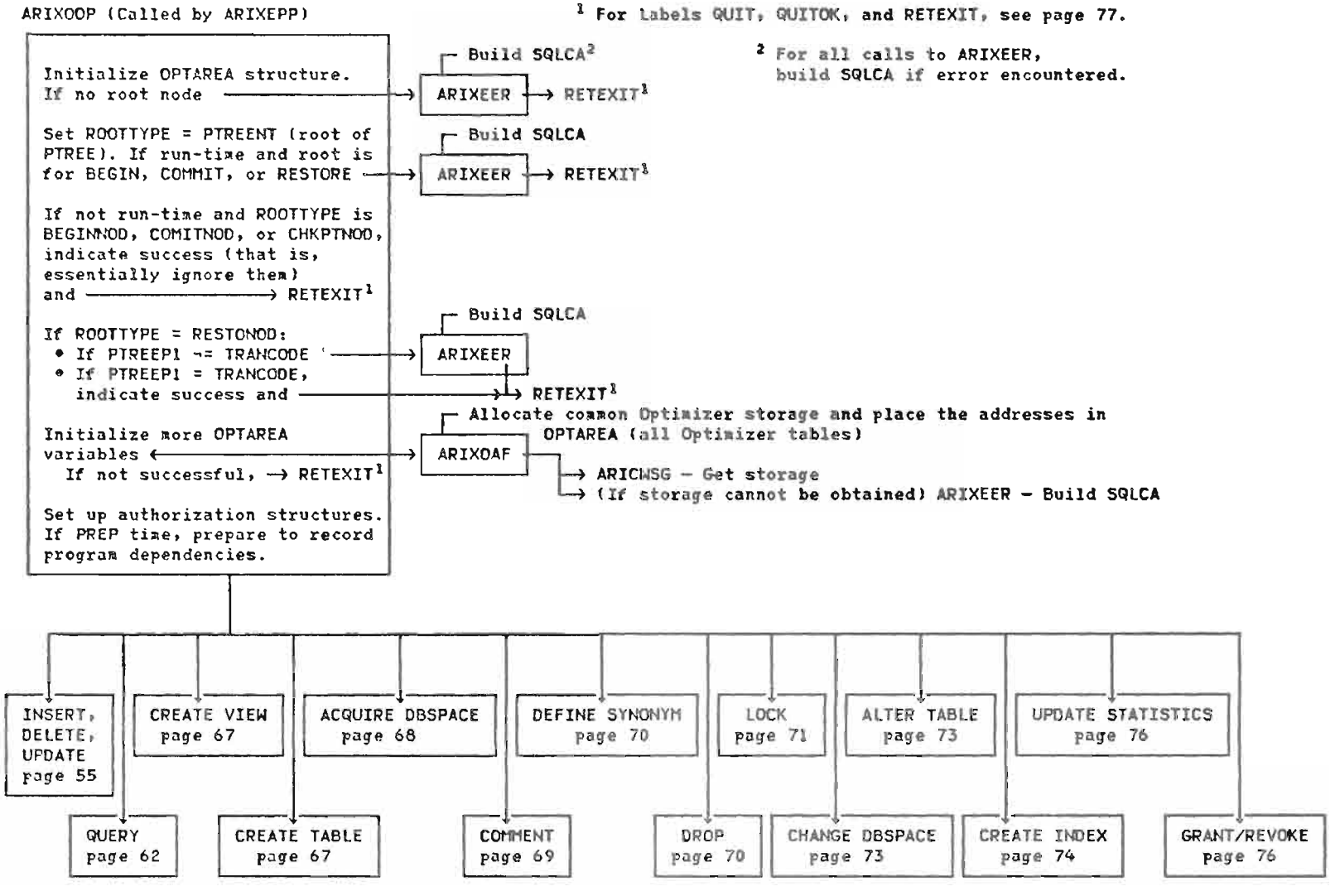
MAJOR DATA AREAS AND CONTROL BLOCKS USED BY THE OPTIMIZER



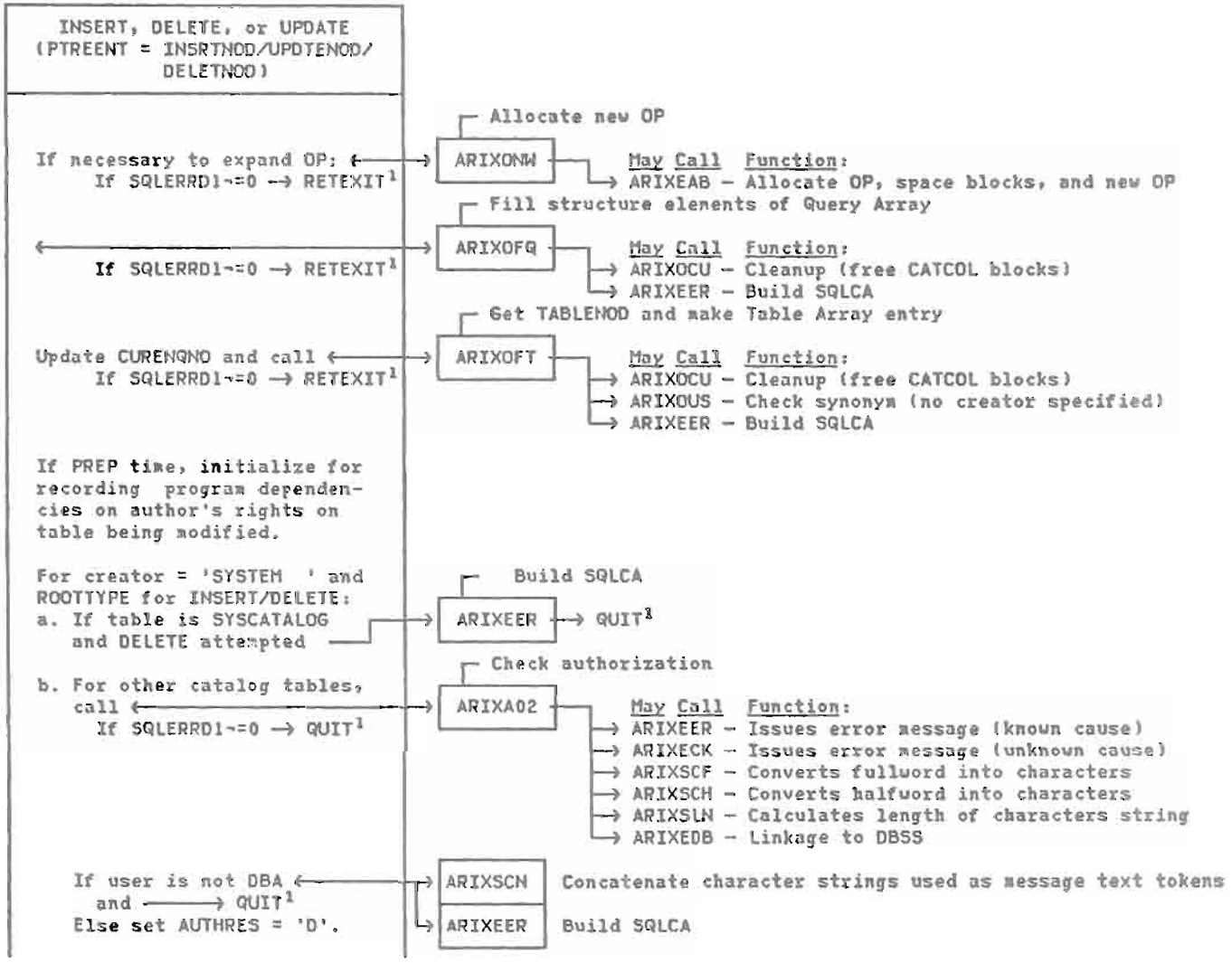
OPTIMIZER RETURN CODES TO EXECUTIVES

```
 111  -  Successful return code for updatable cursor query
 112  -  Successful return code
 113  -  Successful return code for deletable cursor query
-204  -  Indicates table not found. During PREP, this causes a
         PARSESECT to be created by the executives.  During run
         time, it is considered an error.

-551  >- These indicate that the necessary authorization does
-552  |  not exist. During PREP, this causes a PARSESECT to be
     ]  created by the executives.  During run time, it is
        considered to be an error.
Other negative SQLCODES - Are considered errors.
```

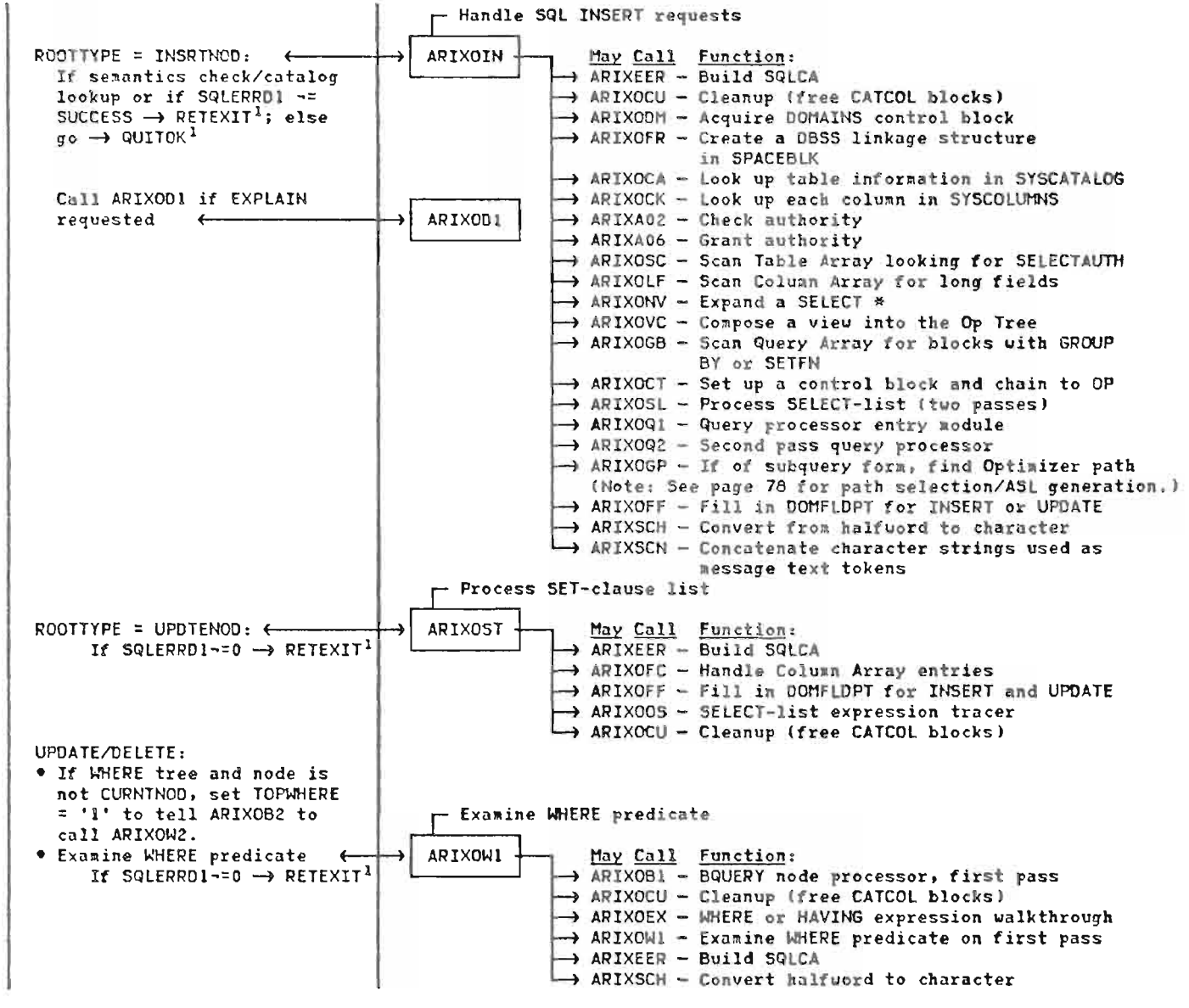
Note: Any negative SQLCODE returned during the PREP phase will cause the CHKMODE bit to be set on, which means that for subsequent SQL commands in the program, the Optimizer will process the statement only as far as catalog look-up (that is, **no** path selection will be performed).



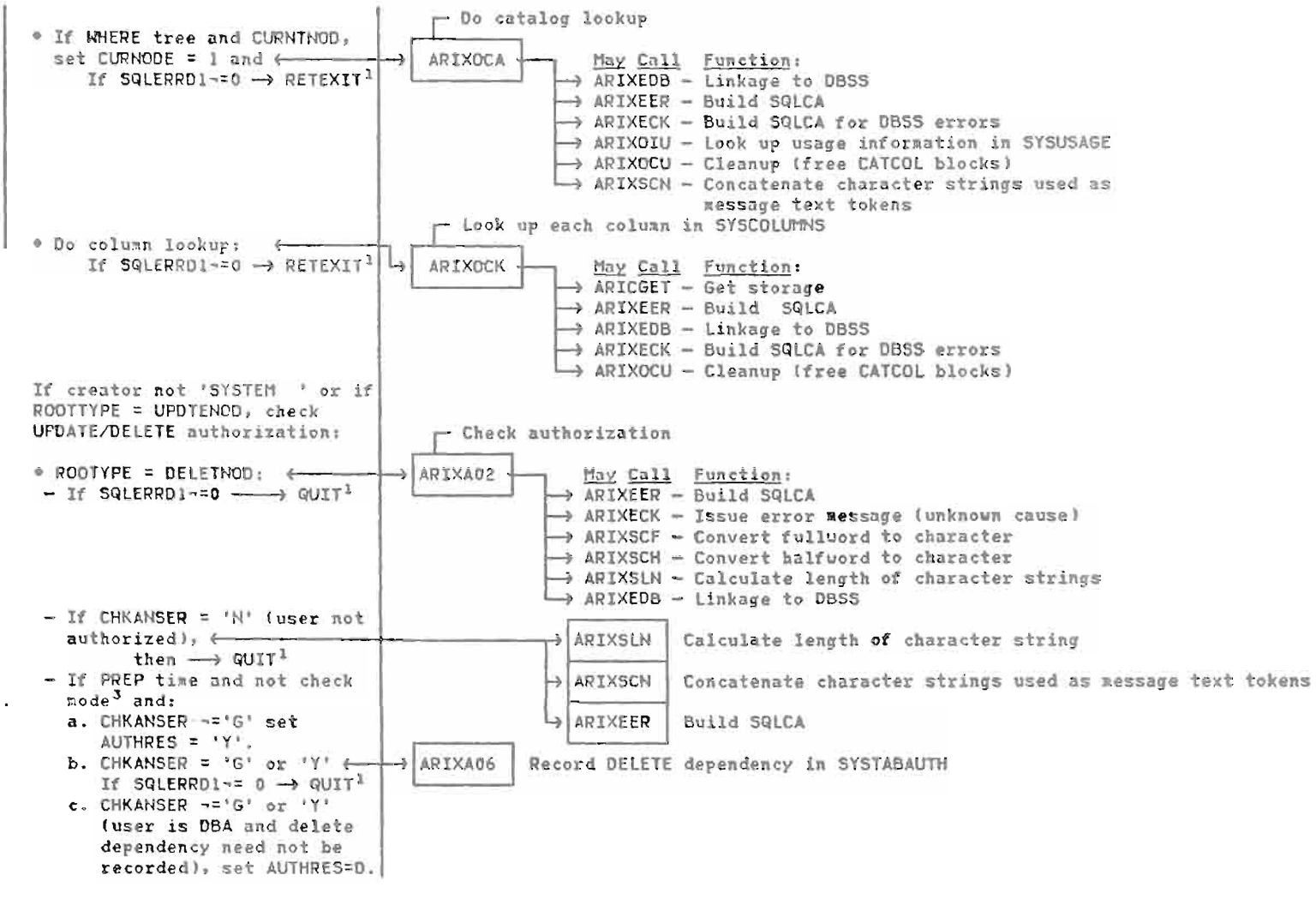
[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.



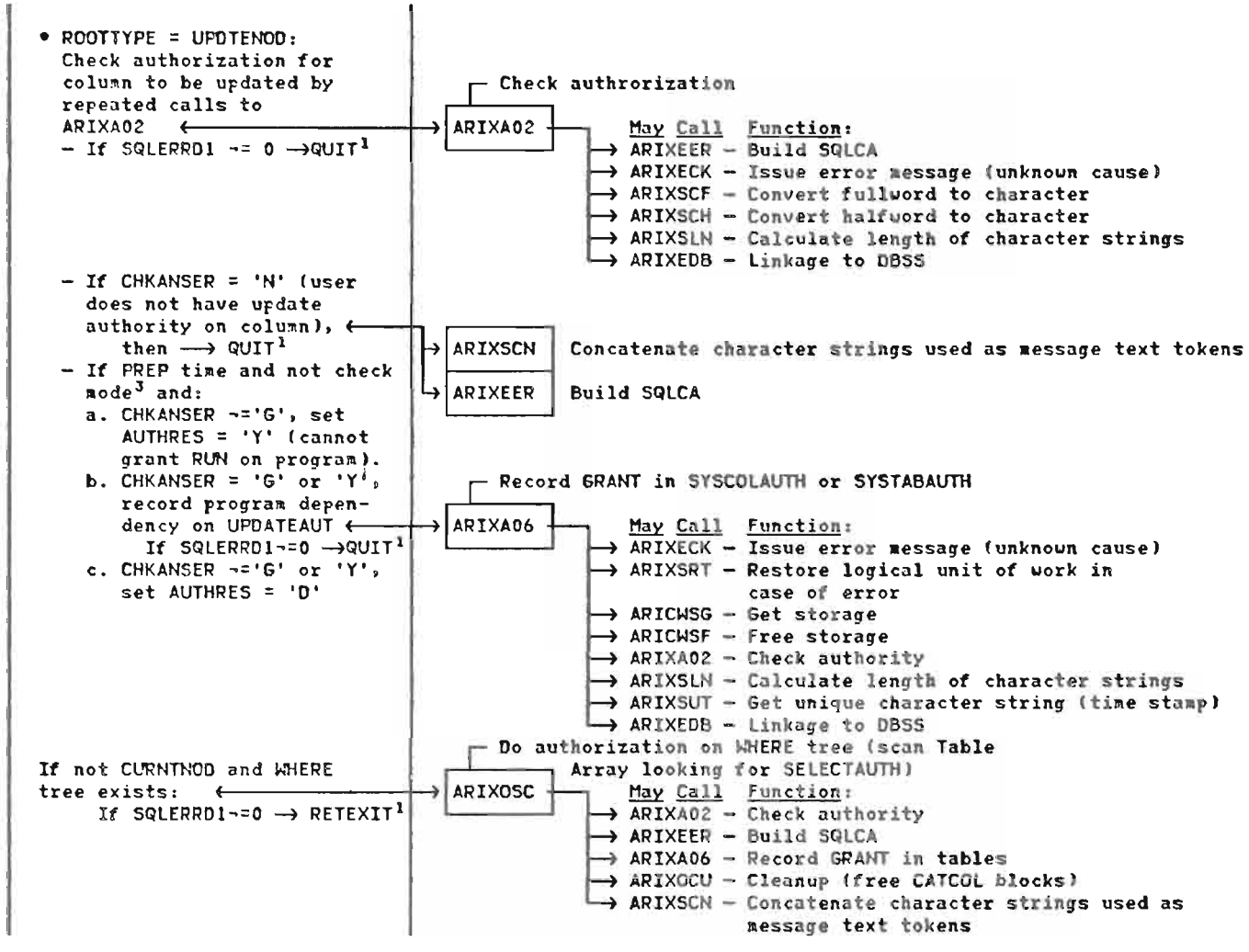
[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.



[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.



[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.



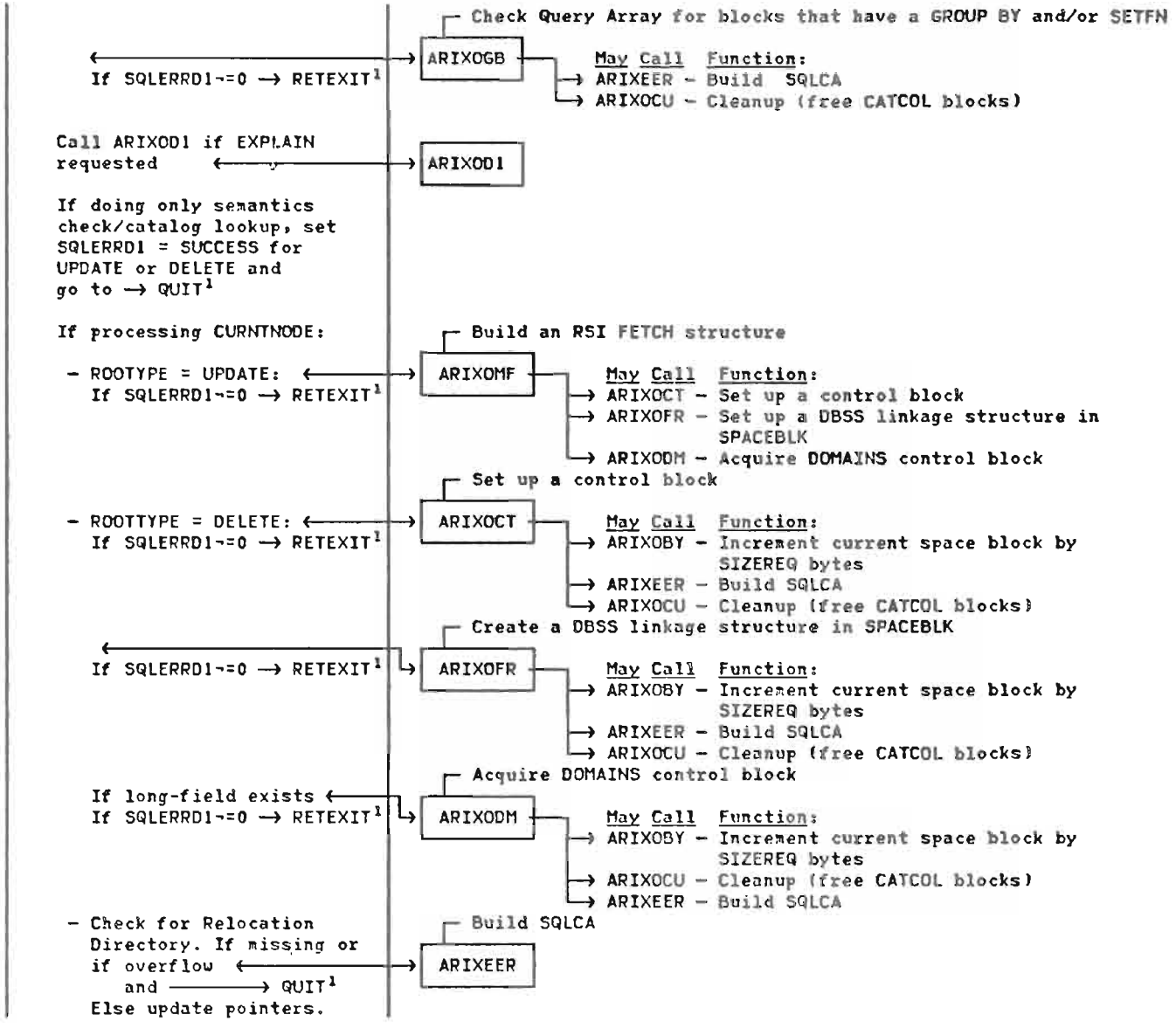
[3] "not check mode" means "if no errors encountered on any previous SQL statement in the user's program"

ARIXOOP (continued)

[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.

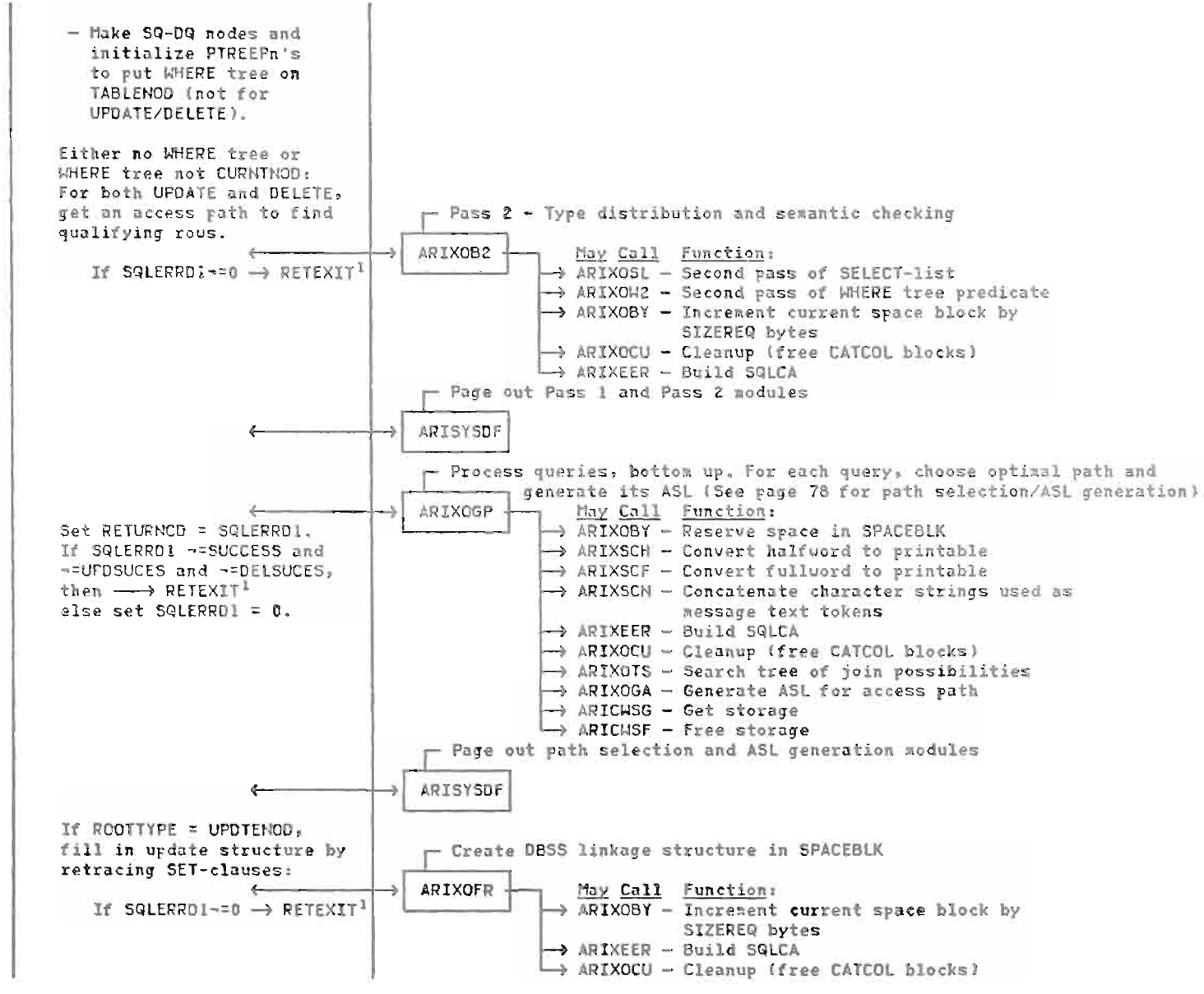
[3] "not check mode" means "if no errors encountered on any previous SQL statement in the user's program"

[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.



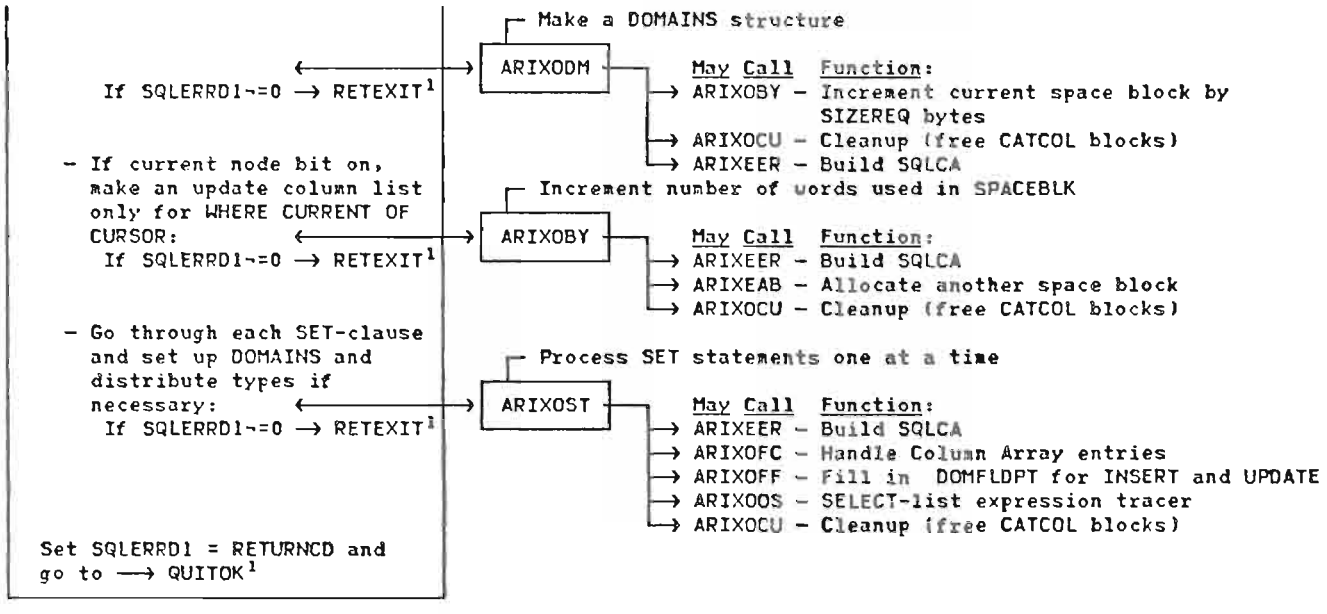
[3] "not check mode" means "if no errors encountered on any previous SQL statement in the user's program"

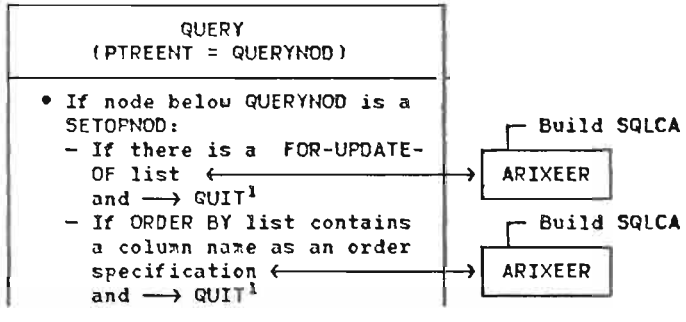
Σ For labels QUIT, QUITOK, and RETEXIT, see page 77.



[3] "not check mode" means "if no errors encountered on any previous SQL statement in the user's program"

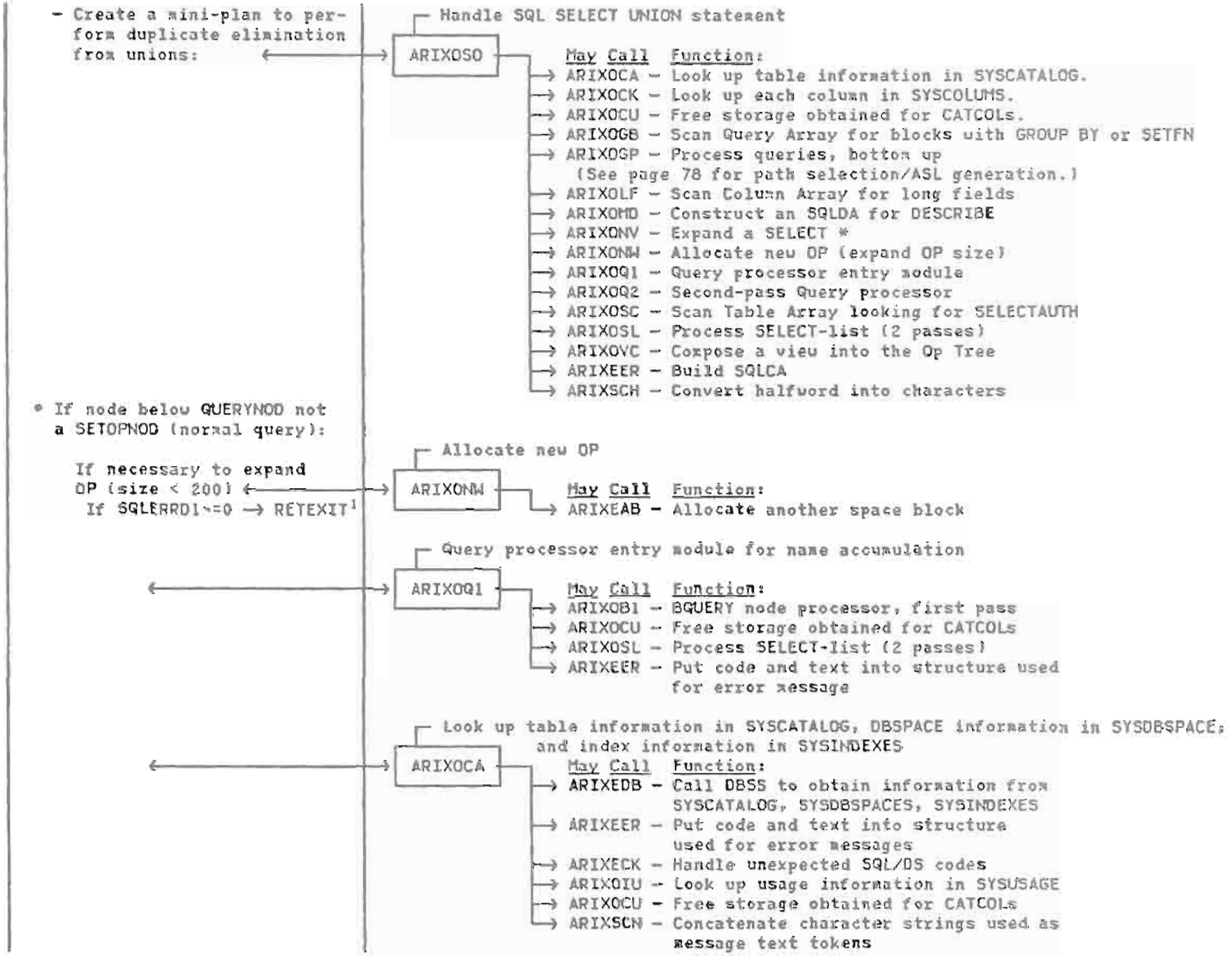
[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.





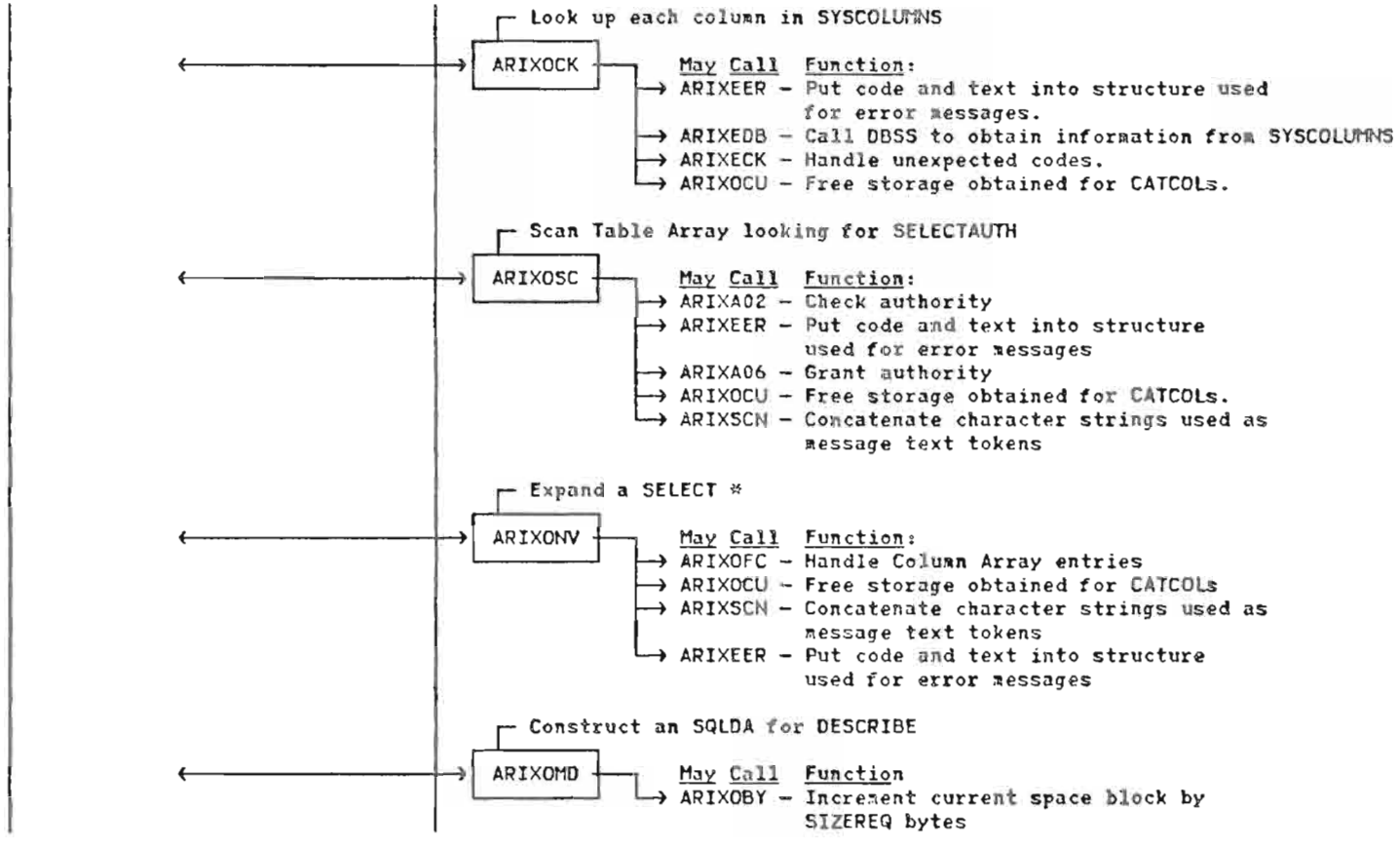
[3] "not check mode" means "if no errors encountered on any previous SQL statement in the user's program"

[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.



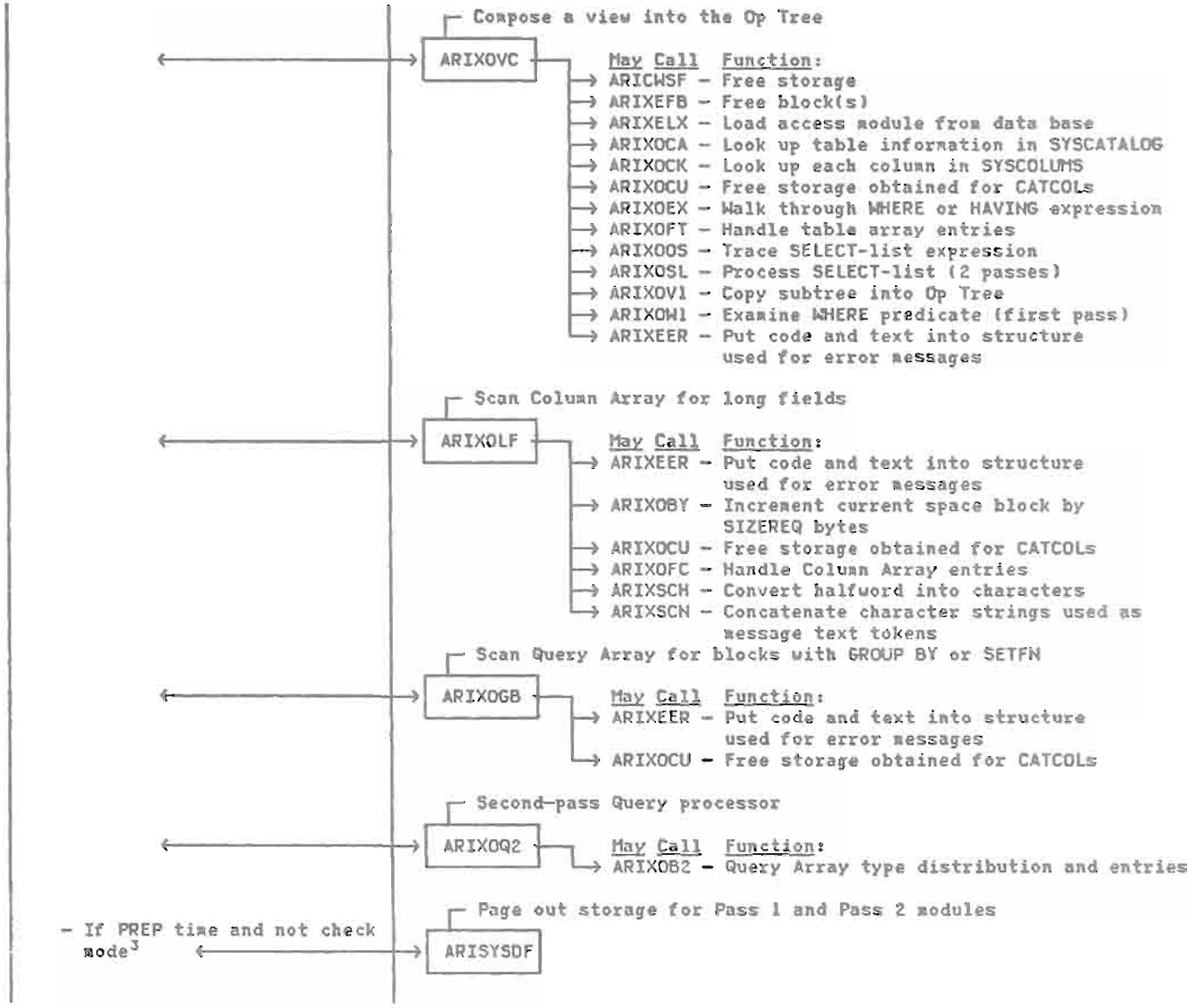
[3] "not check mode" means "if no errors encountered on any previous SQL statement in the user's program"

[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.



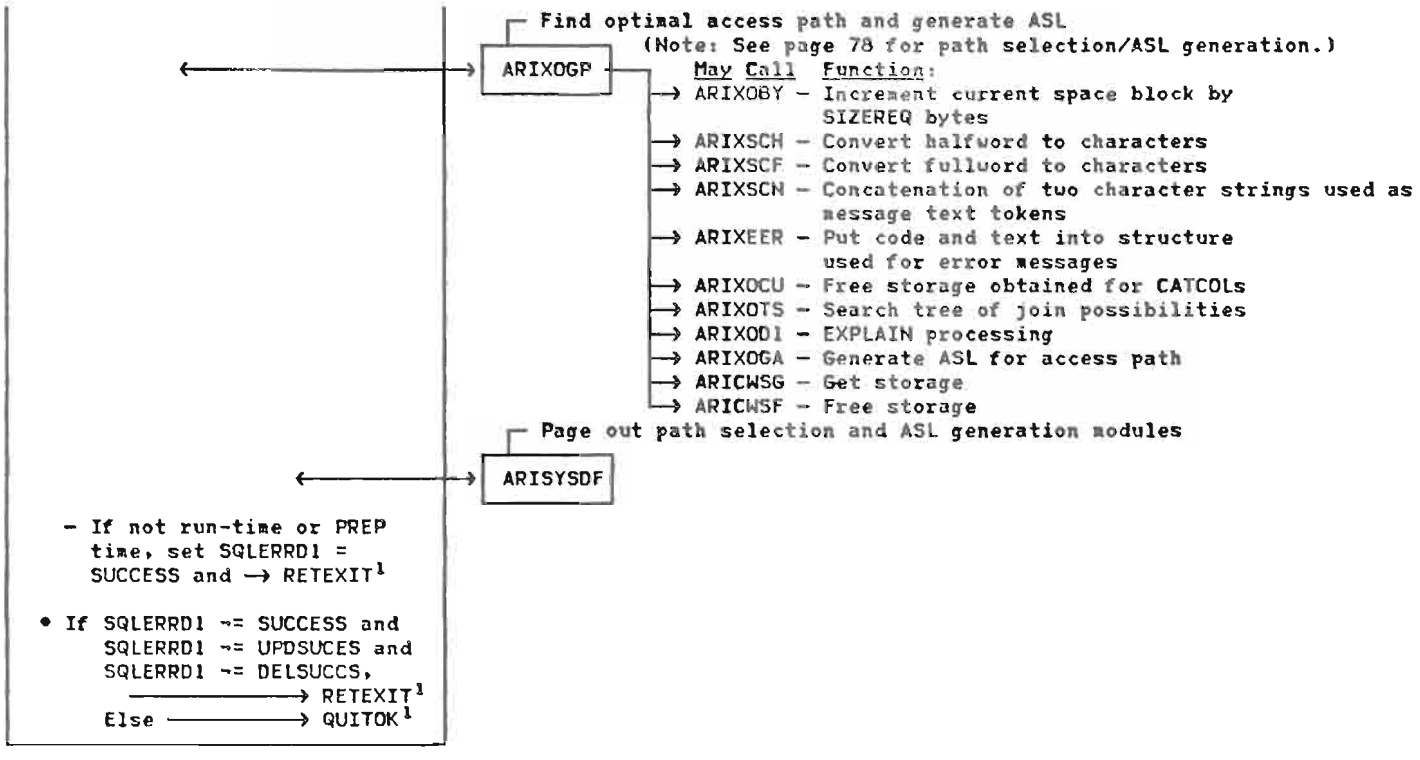
[3] "not check mode" means "if no errors encountered on any previous SQL statement in the user's program"

[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.



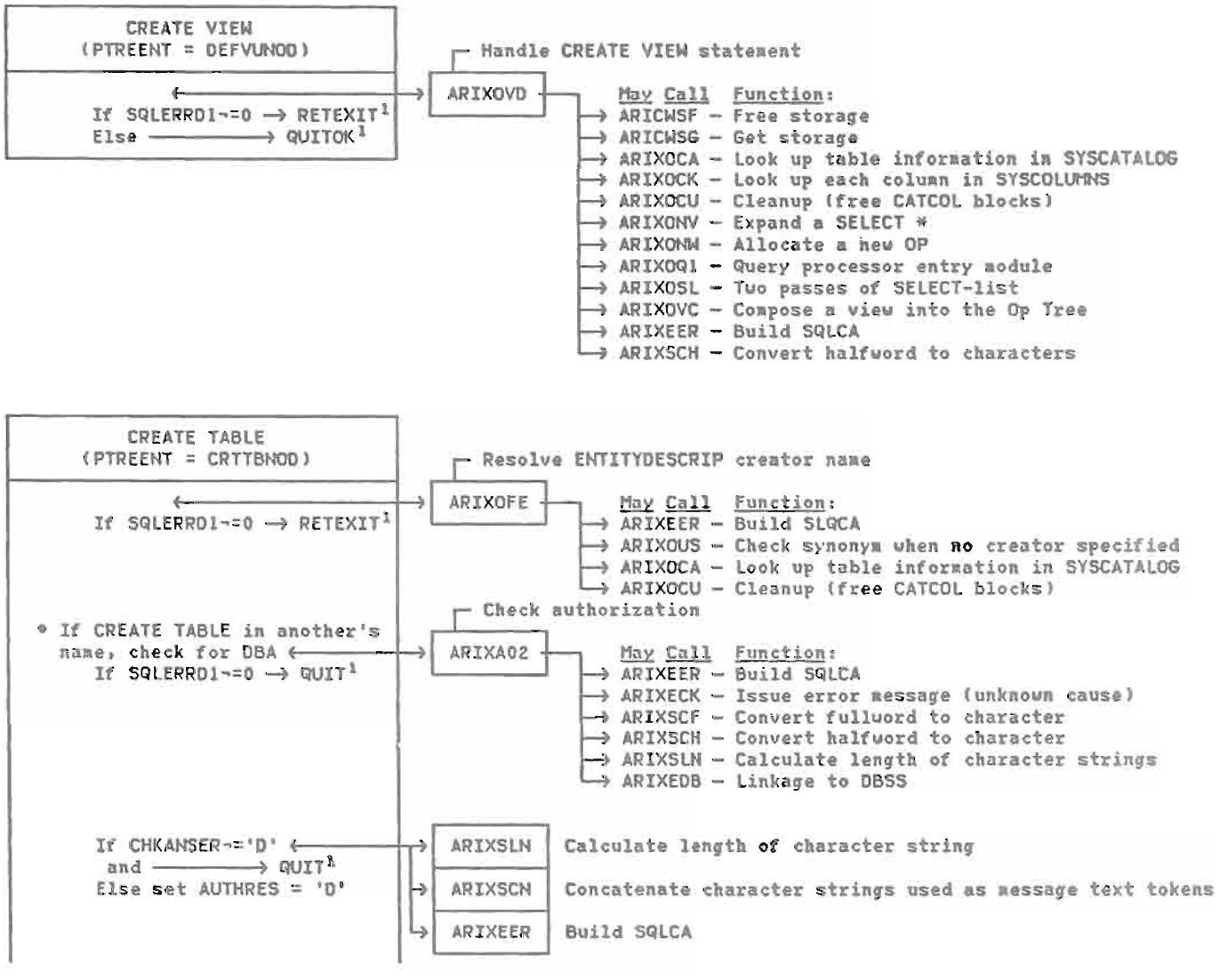
[3] "not check mode" means "if no errors encountered on any previous SQL statement in the user's program"

[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.



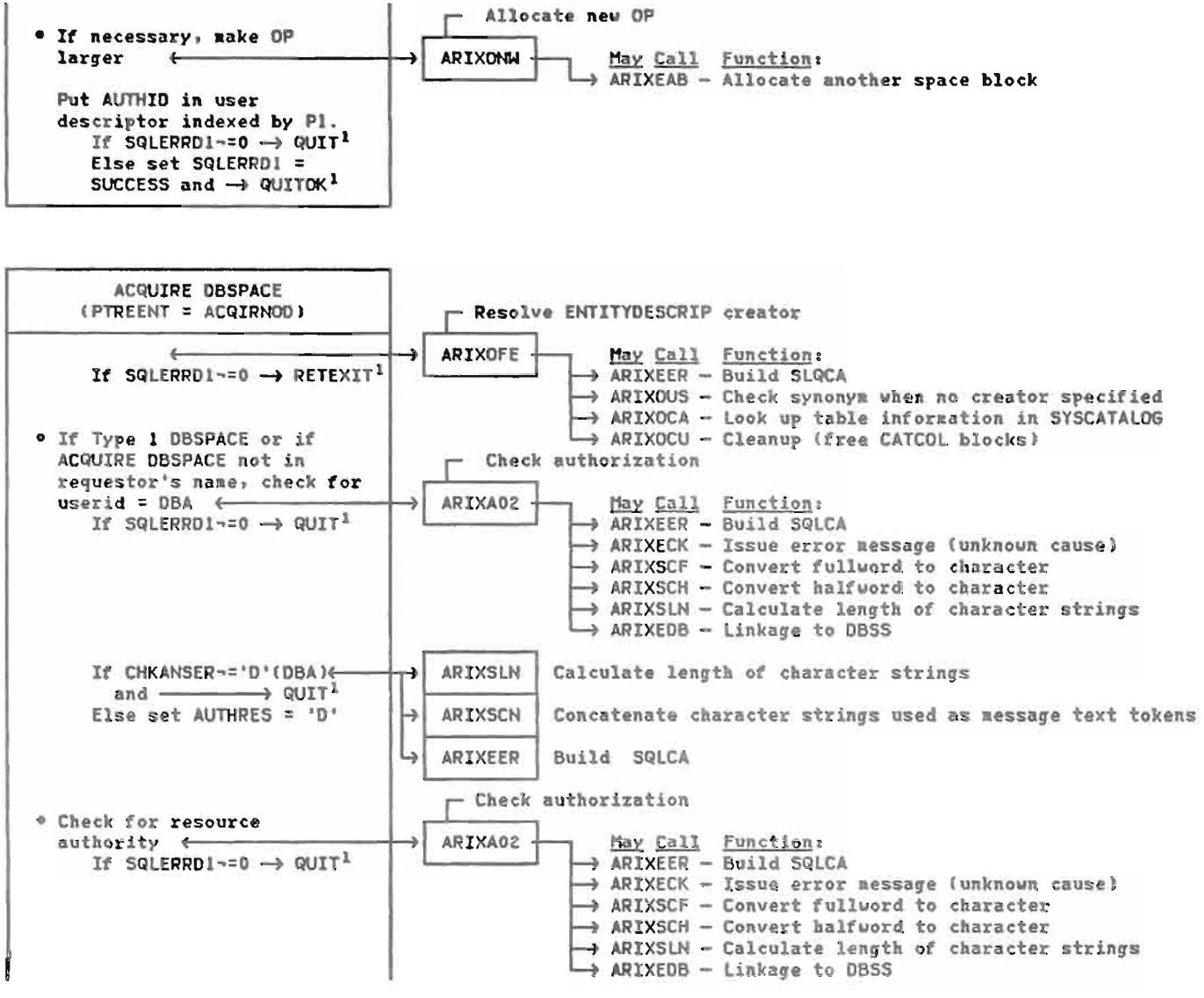
[3] "not check mode" means "if no errors encountered on any previous SQL statement in the user's program"

[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.



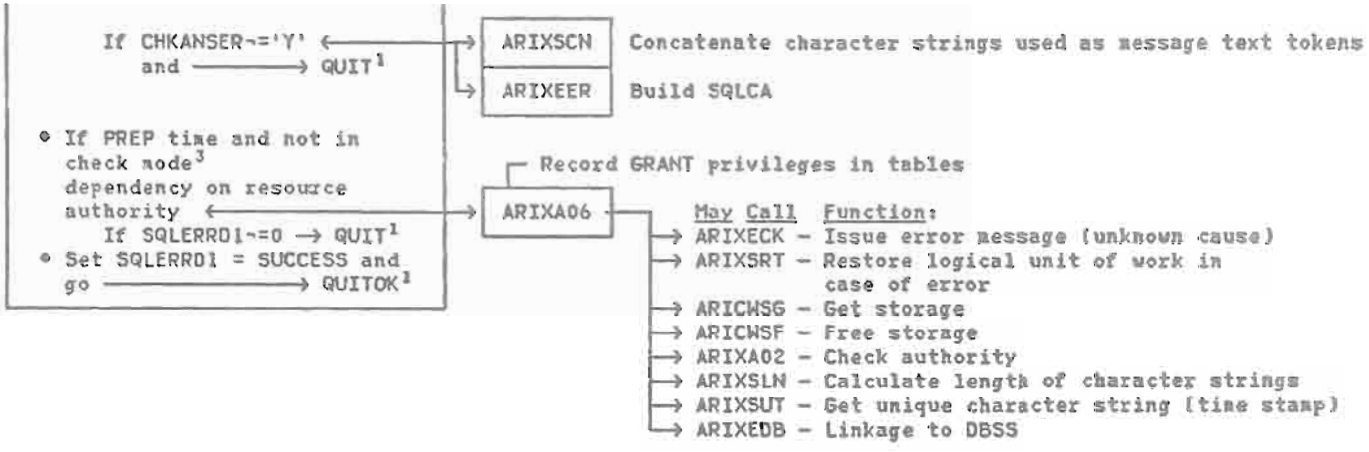
[3] "not check mode" means "if no errors encountered on any previous SQL statement in the user's program"

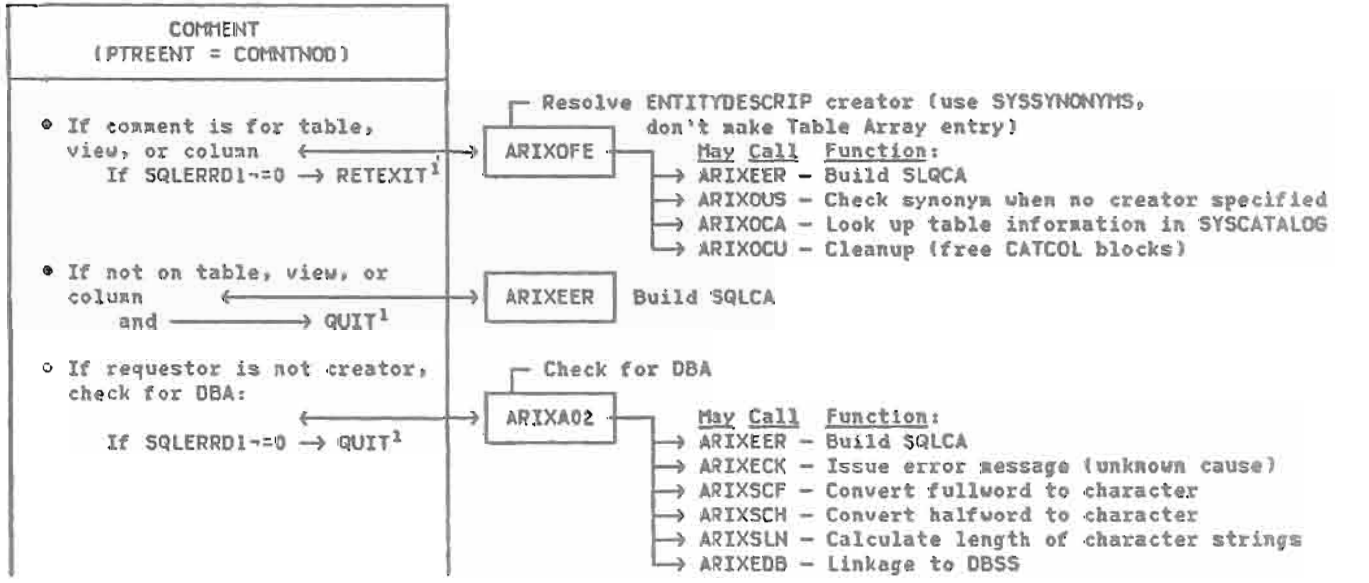
[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.



[3] "not check mode" means "if no errors encountered on any previous SQL statement in the user's program"

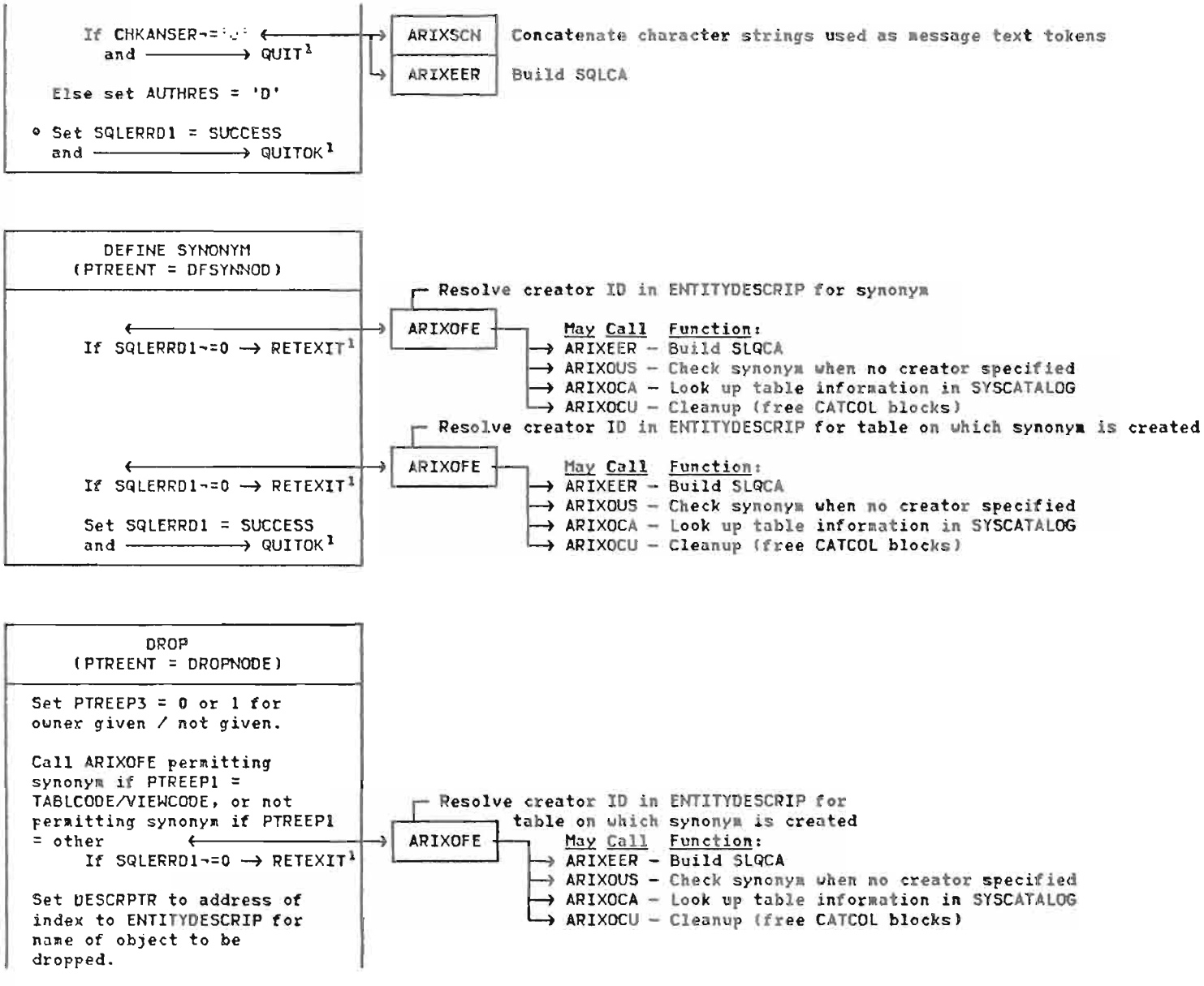
[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.





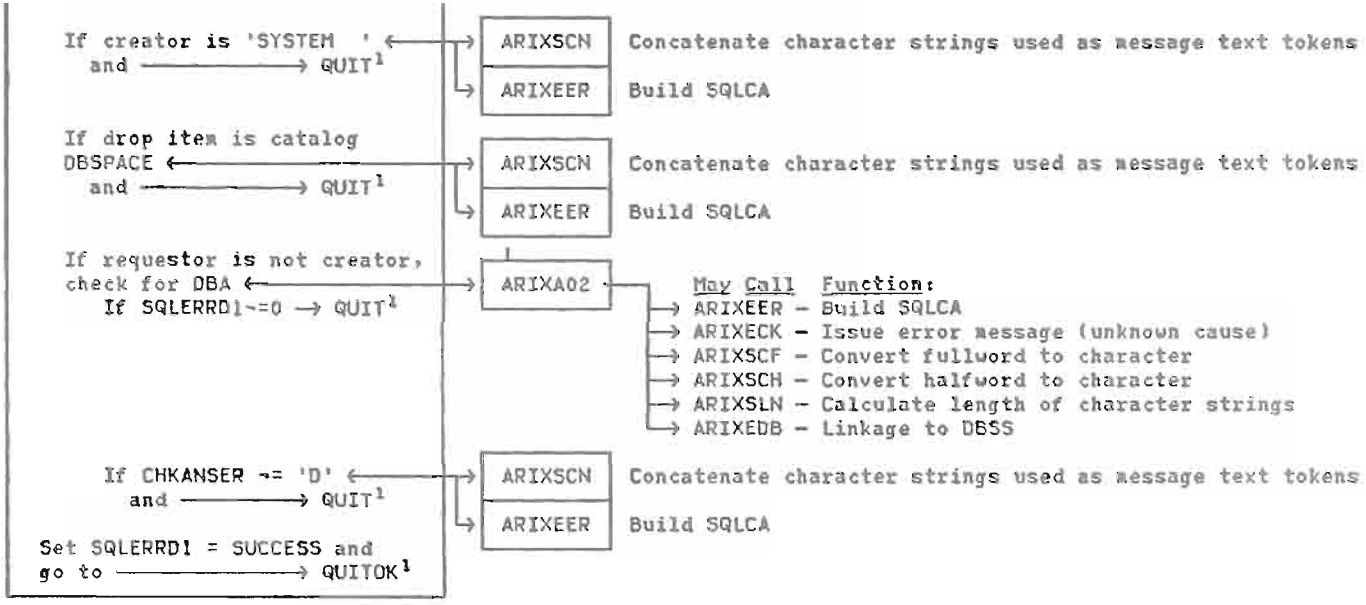
[3] "not check mode" means "if no errors encountered on any previous SQL statement in the user's program"

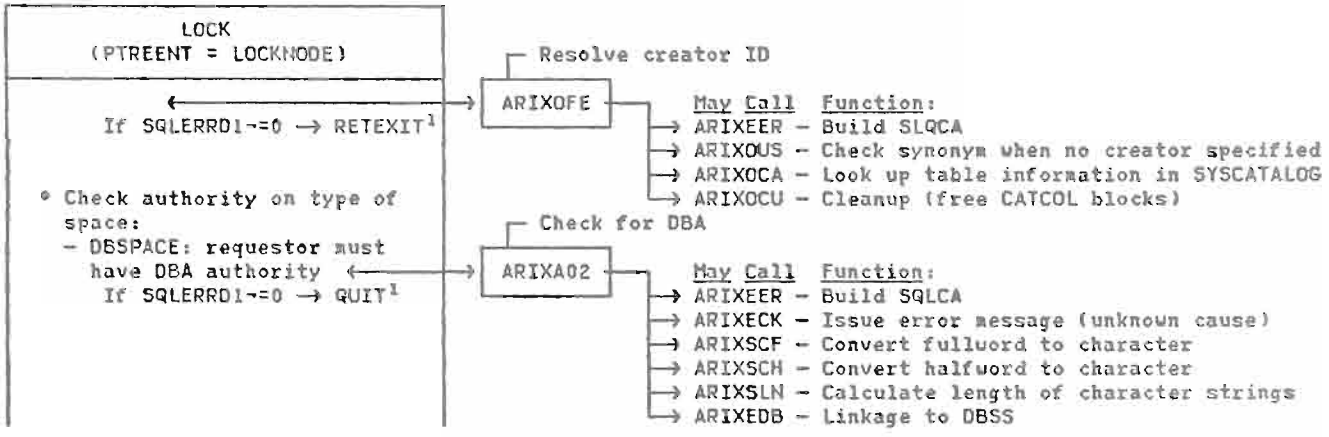
[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.



[3] "not check mode" means "if no errors encountered on any previous SQL statement in the user's program"

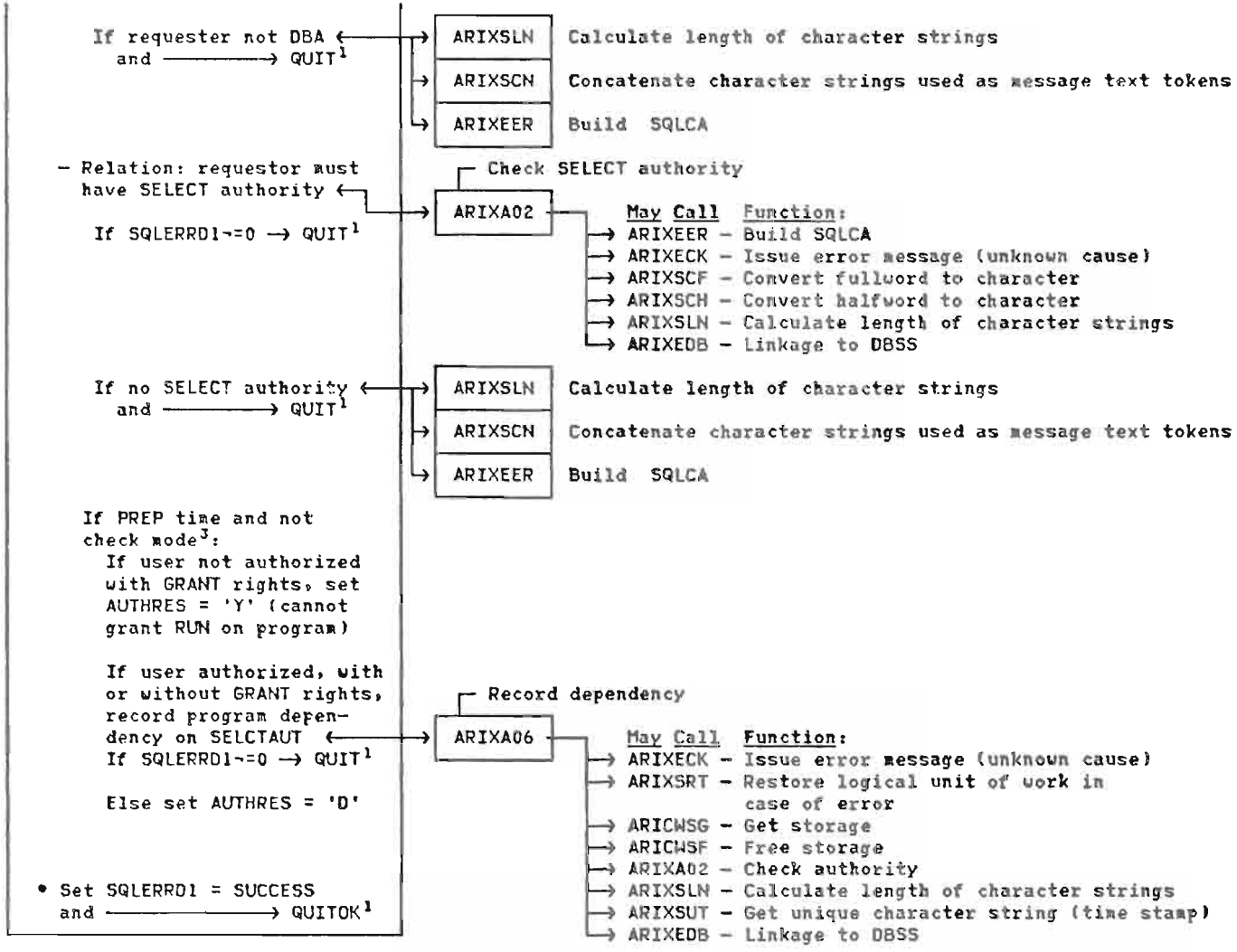
[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.





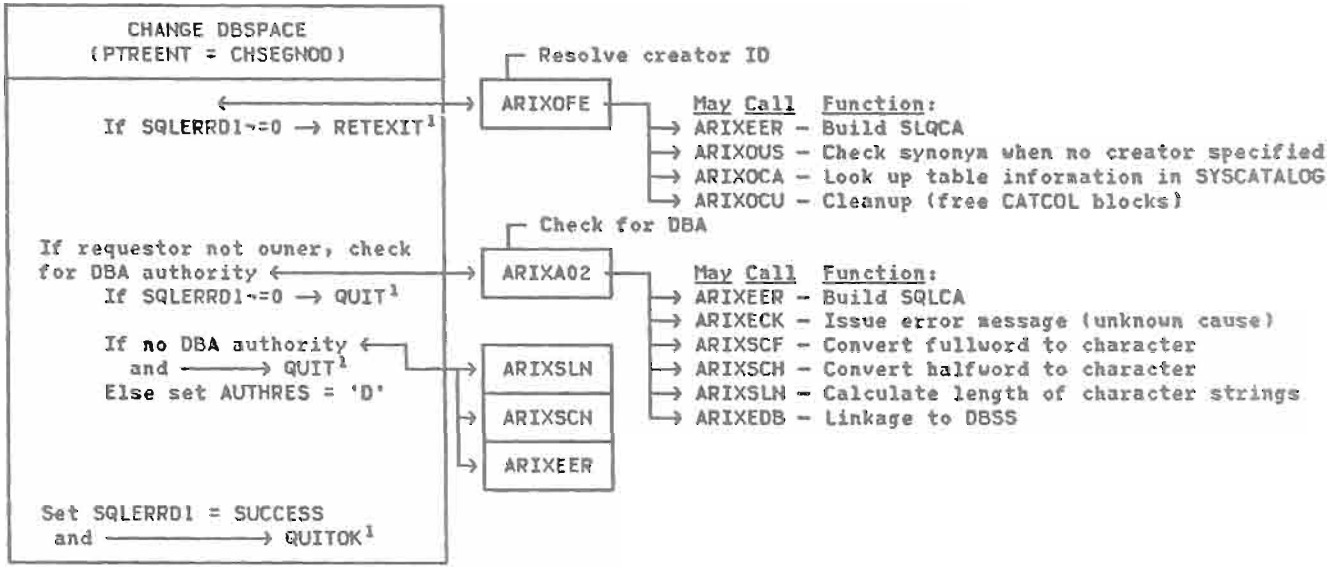
[3] "not check mode" means "if no errors encountered on any previous SQL statement in the user's program"

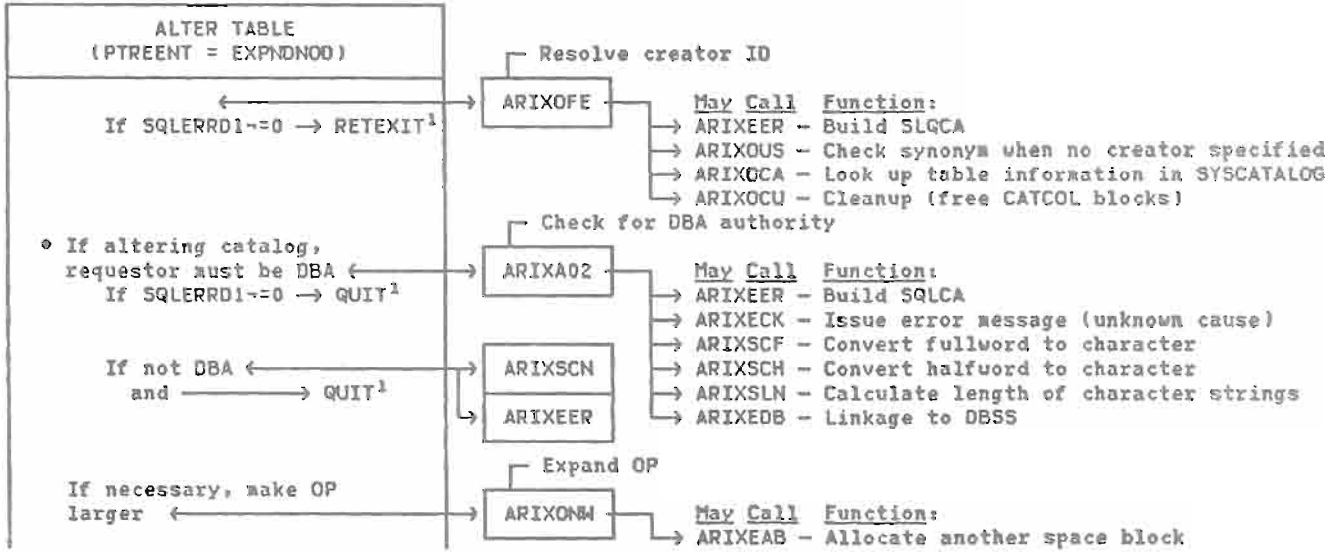
[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.



[3] "not check mode" means "if no errors encountered on any previous SQL statement in the user's program"

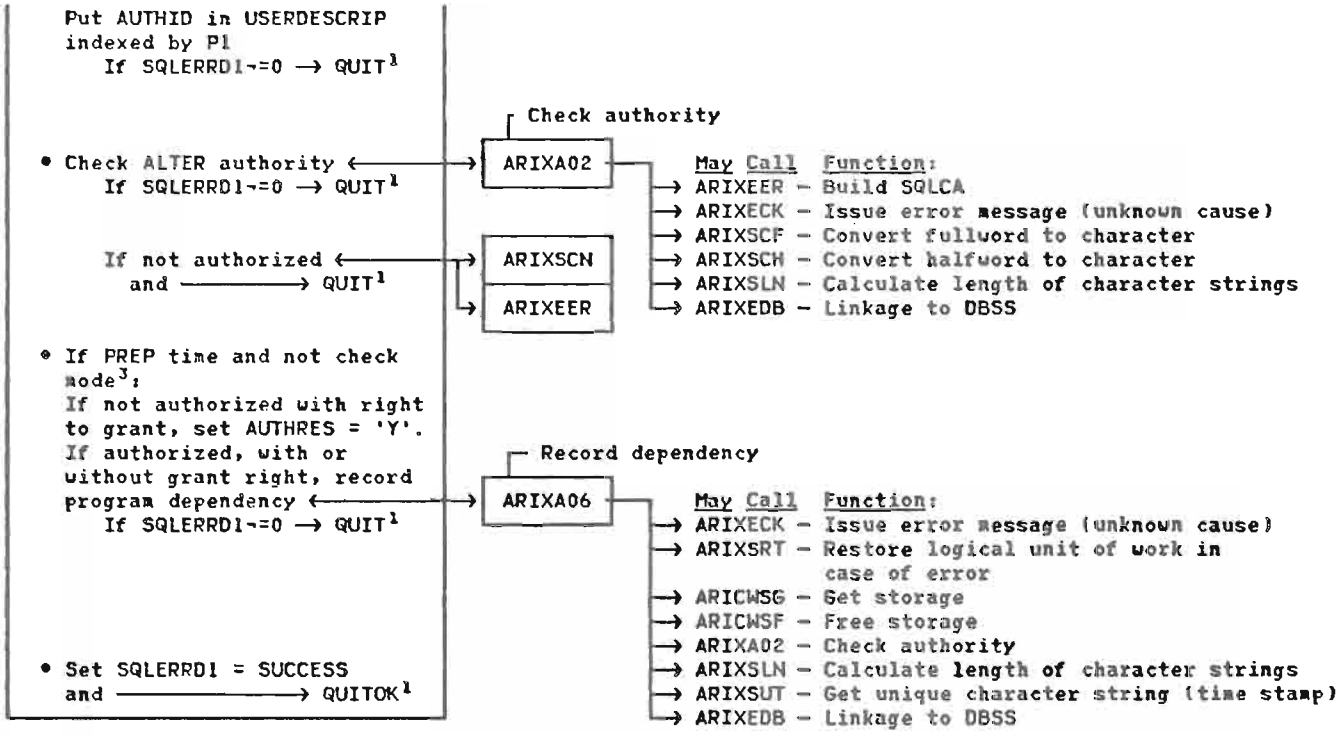
[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.

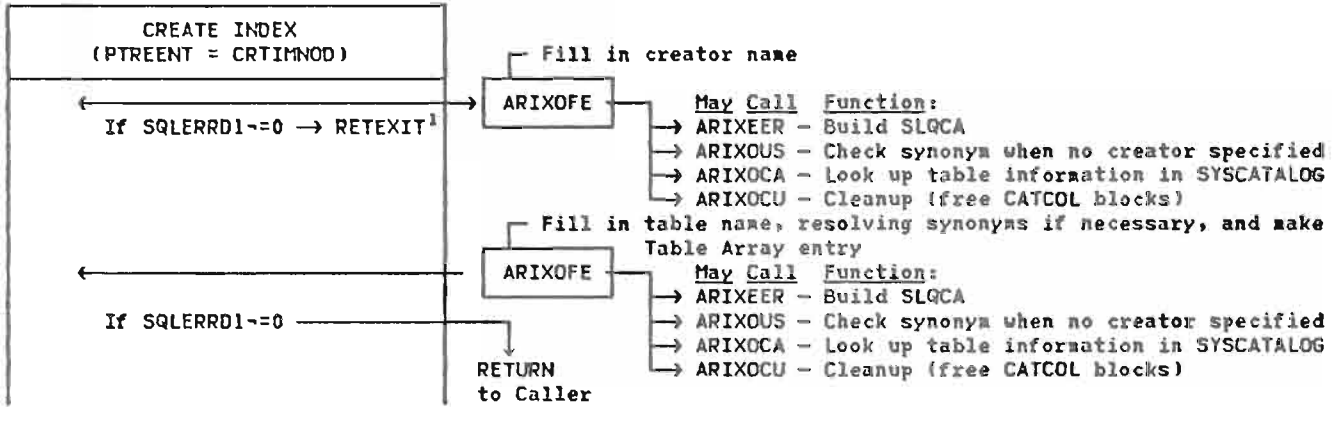




[3] "not check mode" means "if no errors encountered on any previous SQL statement in the user's program"

[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.

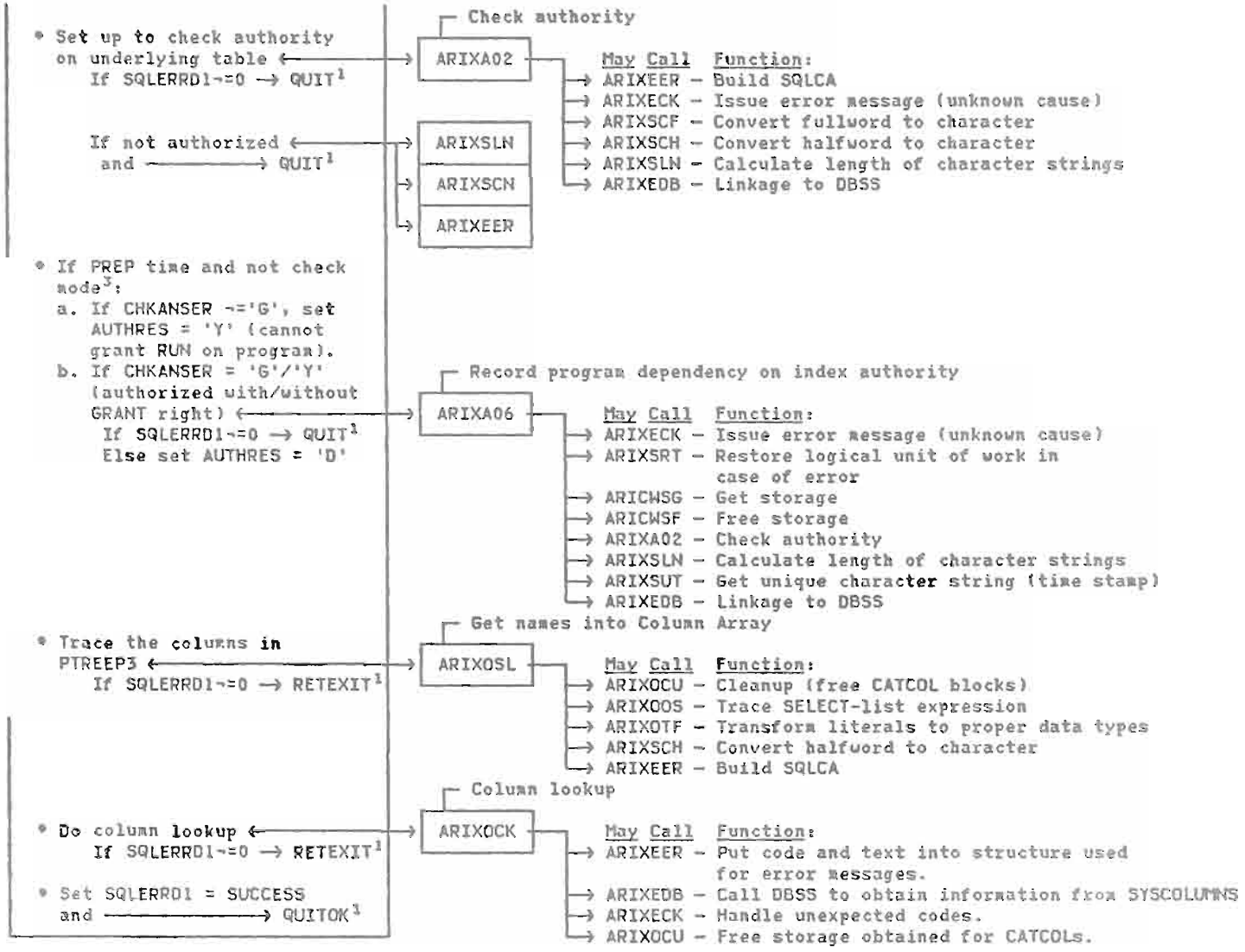




[3] "not check mode" means "if no errors encountered on any previous SQL statement in the user's program"

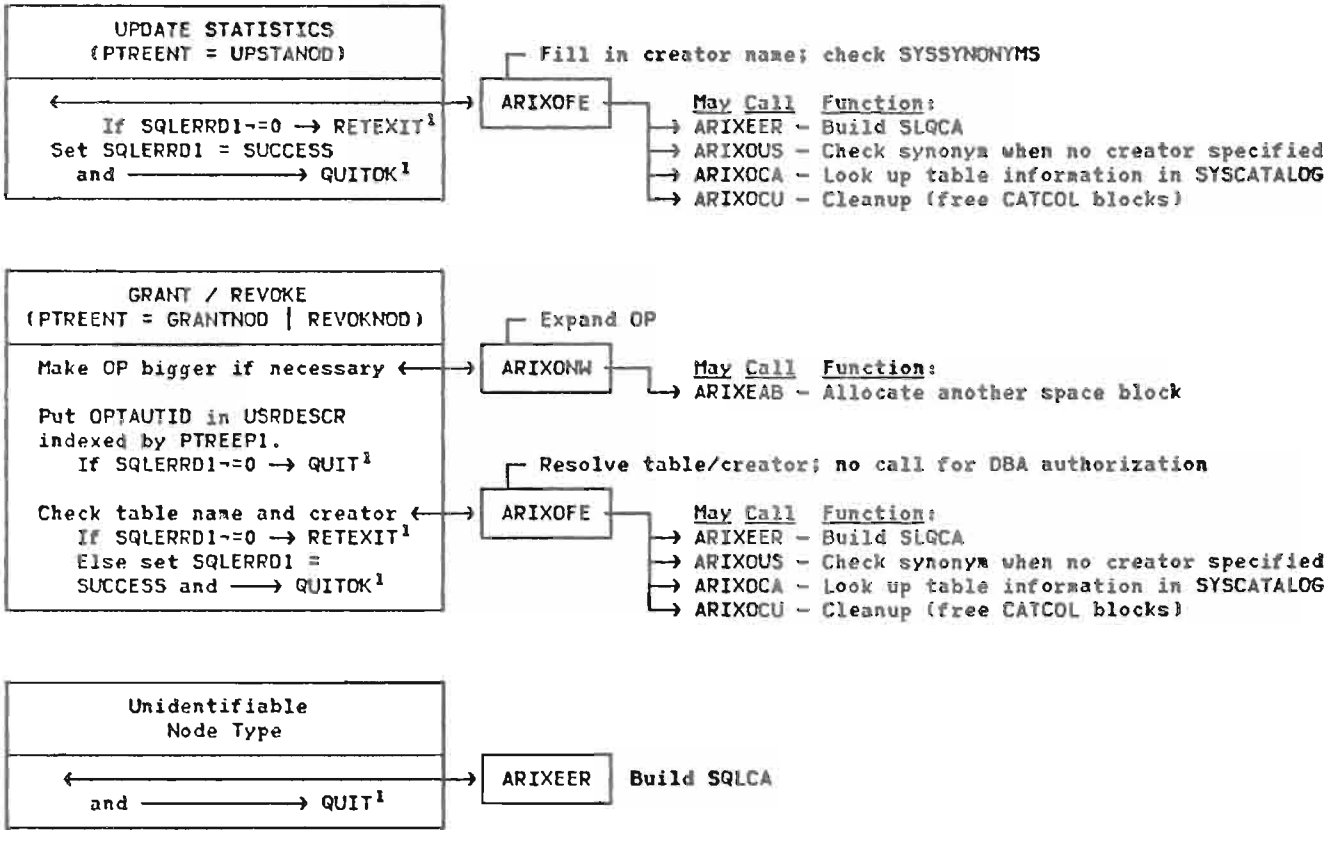
ARIXOOP (continued)

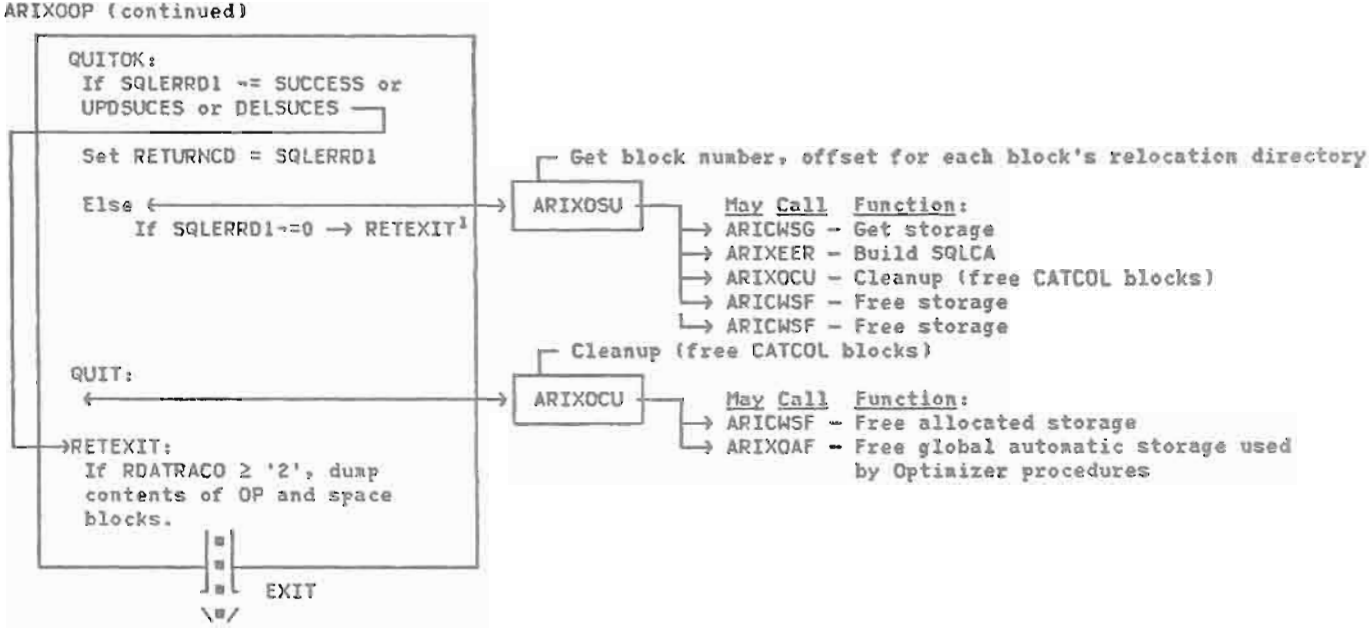
[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.

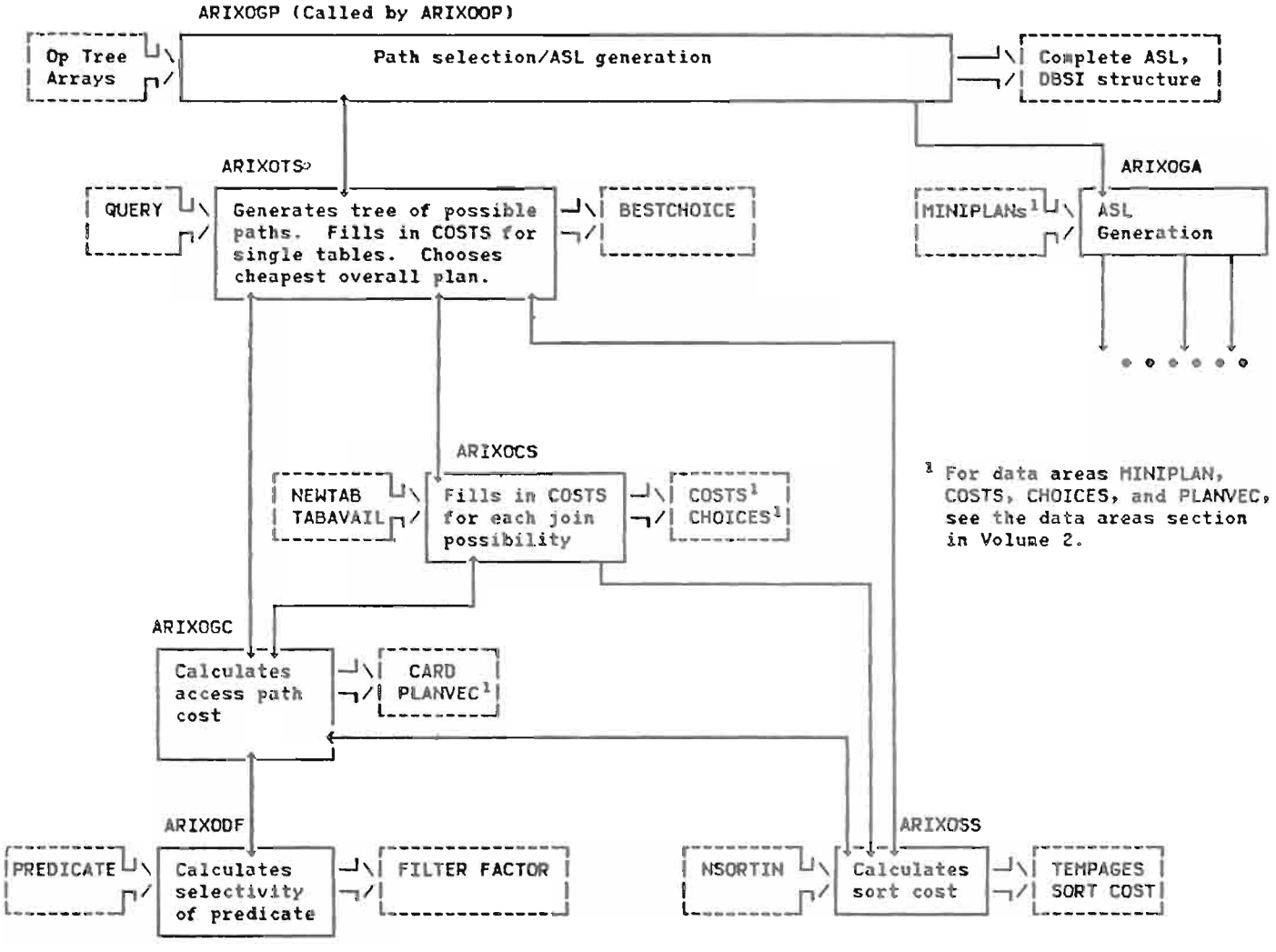


[3] "not check mode" means "if no errors encountered on any previous SQL statement in the user's program"

[1] For labels QUIT, QUITOK, and RETEXIT, see page 77.




ARIXOOP (continued)
QUITOK:
If SQLERRD1 ¬= SUCCESS or
UPDSUCES or DELSUCES
Set RETURNCD = SQLERRD1
Else
If SQLERRD1¬=0 → RETEXIT[1]
QUIT:
→RETEXIT:
If RDATRACO ≥ '2', dump
contents of OP and space
blocks.
EXIT
Get block number, offset for each block's relocation directory
ARIXOSU
May Call Function:
→ ARICWSG - Get storage
→ ARIXEER - Build SQLCA
→ ARIXOCU - Cleanup (free CATCOL blocks)
→ ARICWSF - Free storage
→ ARICWSF - Free storage
Cleanup (free CATCOL blocks)
ARIXOCU
May Call Function:
→ ARICWSF - Free allocated storage
→ ARIXOAF - Free global automatic storage used
by Optimizer procedures



CODE GENERATOR FUNCTION SUMMARY

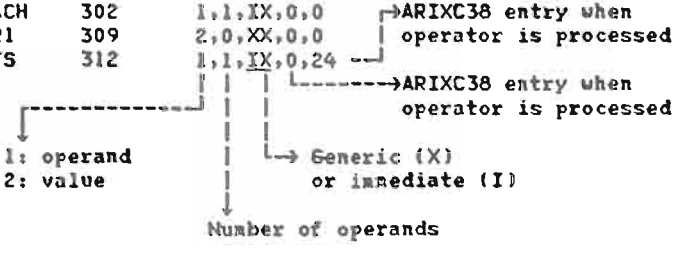
ARIXC37
Passes to ARIXC38:
- PCGDCL – † Code Generator Communication Area
- X – Subcase routine number
- Y – Syntactic stack index
- CGSEMSTX – Semantic stack index (template number)

ARIXC38

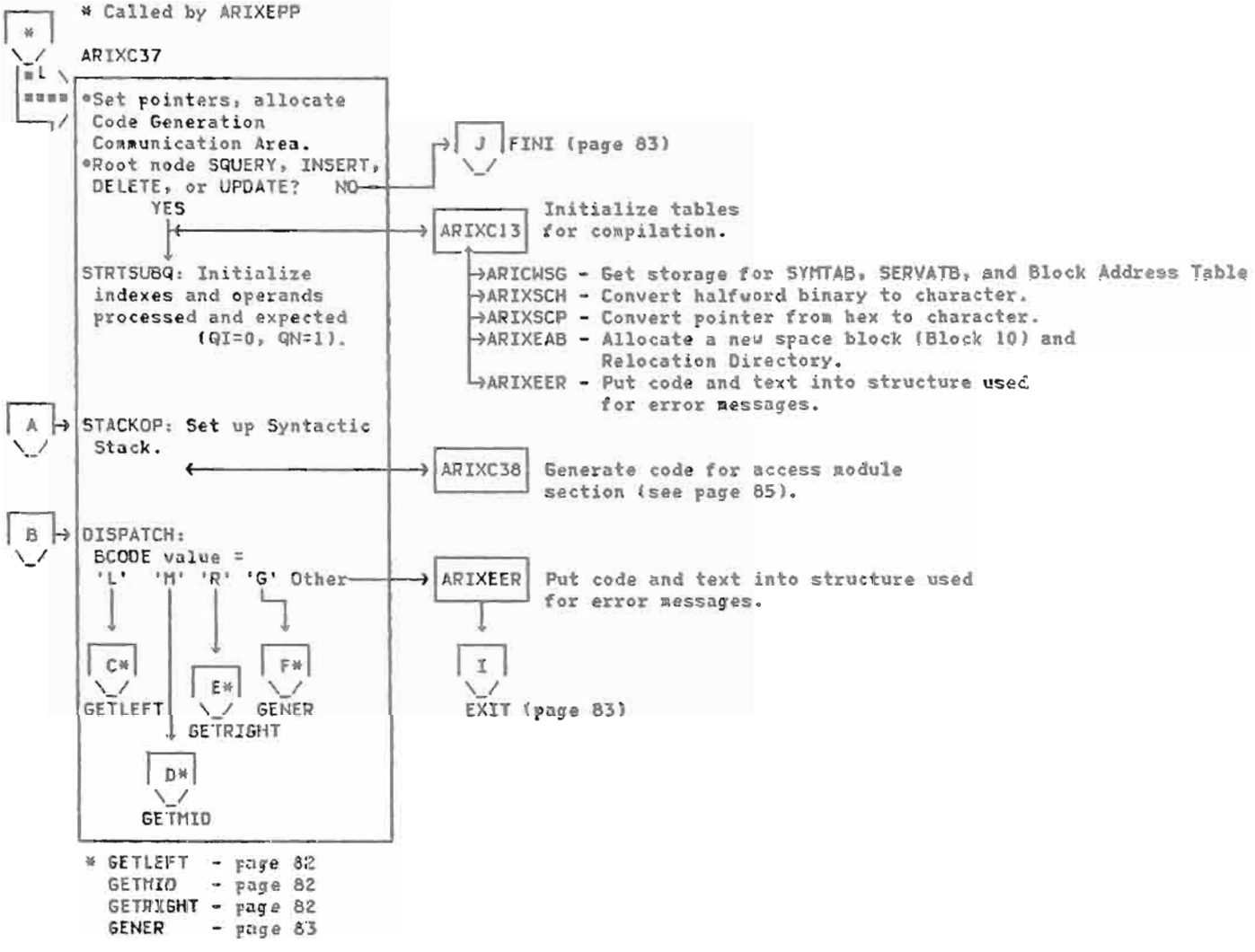
ENTRY	FUNCTION	CALLS
LAB00, LAB01	NO FUNCTION, MERELY RETURNS	- - - - - - -
LAB2	PROCESS SQUERY NODE	(PROCESS INLINE)
LAB4	PROCESS INSERT OF LITERAL	ARIXC29 TO PROCESS
LAB9 LAB9	PROCESS FIRST INVOCATION OF BQUERY INSERT, DELETE, UPDATE	DETERMINE TYPE OF STATEMENT AND TEMPLATE TO BE USED
LAB10	RE-INVOCATION FOR BQUERY	COMPLETE PROCESS STARTED IN LAB9
LAB12	PROCESS FOR ISNULL NODE	ARIXC16 TO PROCESS
LAB13	PROCESS FOR SETNODE FOR UPDATE	ARIXC35 TO PROCESS
LAB14, LAB15	EVALUATION OF BOOLEAN EXPRESSION	PROCESS INLINE
LAB19	PROCESS MINUS OPERATOR	ARIXC34 TO PROCESS
LAB20	PROCESS ARITHMETIC OPERATION	ARIXC02 TO PROCESS
LAB21	PROCESS NOT NODE	PROCESS INLINE
LAB22	PROCESS ARITHLIST NODE	PROCESS INLINE
LAB24	PROCESS A VALUE COMPARISON BLOCK	ARIXC05 TO PROCESS
LAB25	PROCESS SET FUNCTION NODE, FIRST INVOCATION	PROCESS INLINE
LAB26	PROCESS SET FUNCTION NODE, SECOND INVOCATION	ARIXC28 TO PROCESS
LAB27	PROCESS SETFN1 NODE	ARIXC06 TO PROCESS
LAB39	EVALUATE BOOLEAN EXPRESSION	PROCESS INLINE

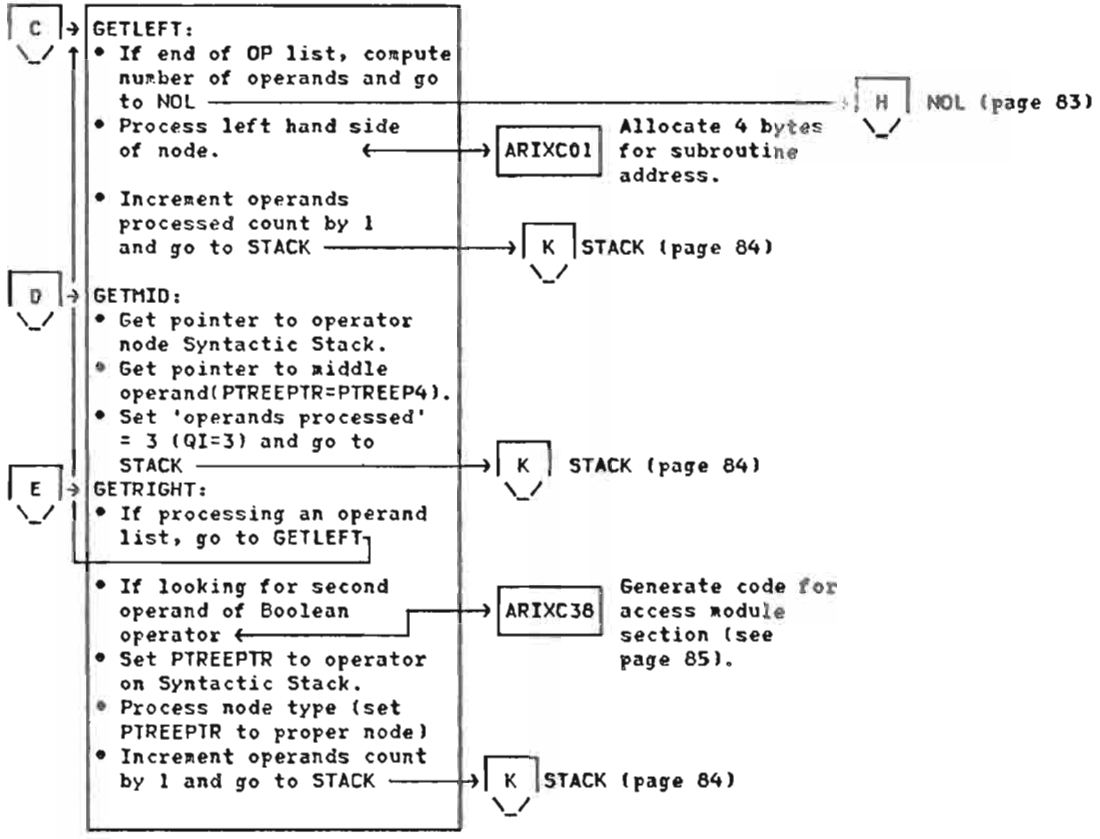
The following figure, the OPTABLE, defines operator nodes in the ASL tree. The OPTABLE is used by ARIXC37 to determine which routine in ARIXC38 is to be called to process an operator in the optimized ASL tree.

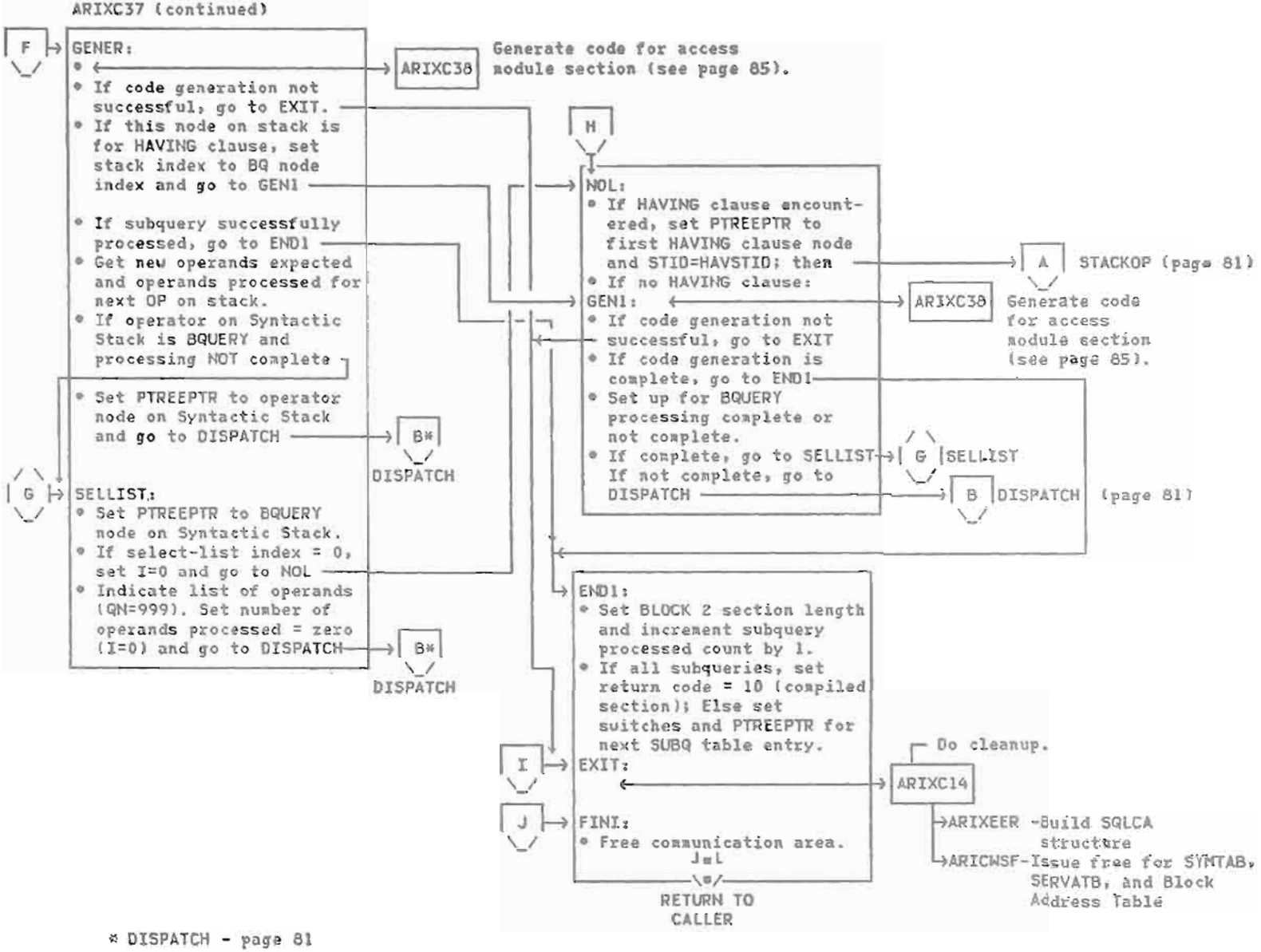
SYMBOL	CODE	OPERATOR
AVGFN	30	1,1,XX,25,26
SUMFN	31	1,1,XX,25,26
COUNTFN	32	1,1,XX,25,26
MAXFN	33	1,1,XX,25,26
MINFN	34	1,1,XX,25,26
PLUSCODE	40	1,2,XX,0,20
MINUS	41	1,2,XX,0,20
MULTCODE	42	1,2,XX,0,20
DIVIDE	43	1,2,XX,0,20
ANDCODE	70	1,2,XX,0,14
ORCODE	71	1,2,XX,0,15
LT	180	1,2,XX,0,24
LE	181	1,2,XX,0,24
GT	182	1,2,XX,0,24
GE	183	1,2,XX,0,24
EQ	184	1,2,XX,0,24
NE	185	1,2,XX,0,24
LIKE	186	1,2,XX,0,24
LTANY	190	1,2,XX,0,24
LEANY	191	1,2,XX,0,24
GTANY	192	1,2,XX,0,24
GEANY	193	1,2,XX,0,24
EQANY	194	1,2,XX,0,24
NEANY	195	1,2,XX,0,24
LTALL	200	1,2,XX,0,24
LEALL	201	1,2,XX,0,24
GTALL	202	1,2,XX,0,24
GEALL	203	1,2,XX,0,24
EQALL	204	1,2,XX,0,24
NEALL	205	1,2,XX,0,24
ISIN	222	1,2,XX,0,24
ISNOTIN	223	1,2,XX,0,24
INSERT	232	1,1,IX,0,4
DELETE	233	1,1,IX,5,6
UPDATE	234	1,1,IX,7,8
SETNODE	235	1,1,IX,0,13
COL	237	2,0,XX,0,0
BQUERY	239	1,1,IX,9,10
NOTNODE	242	1,1,IX,0,21
UMNSNODE	243	1,1,IX,0,19
VALCOMP	244	1,2,XX,0,0
BETWEEN	245	1,3,IX,0,24
BOOL	241	1,2,XX,0,0
TABCOMP	246	1,2,XX,0,0
ARITLIST	251	1,1,IX,0,22
ARITHND	253	1,2,XX,0,15
SETFN	252	1,1,XX,25,26
LITN	254	2,0,XX,0,0
LITTUP	255	2,0,XX,0,0
ISNULL	288	1,1,IX,0,12
SETFN1	299	1,1,IX,0,27
SQUERY	300	1,2,IX,1,2
DQUERY	301	1,2,IX,0,0
FOREACH	302	1,1,IX,0,0
SQUER1	309	2,0,XX,0,0
EXISTS	312	1,1,IX,0,24



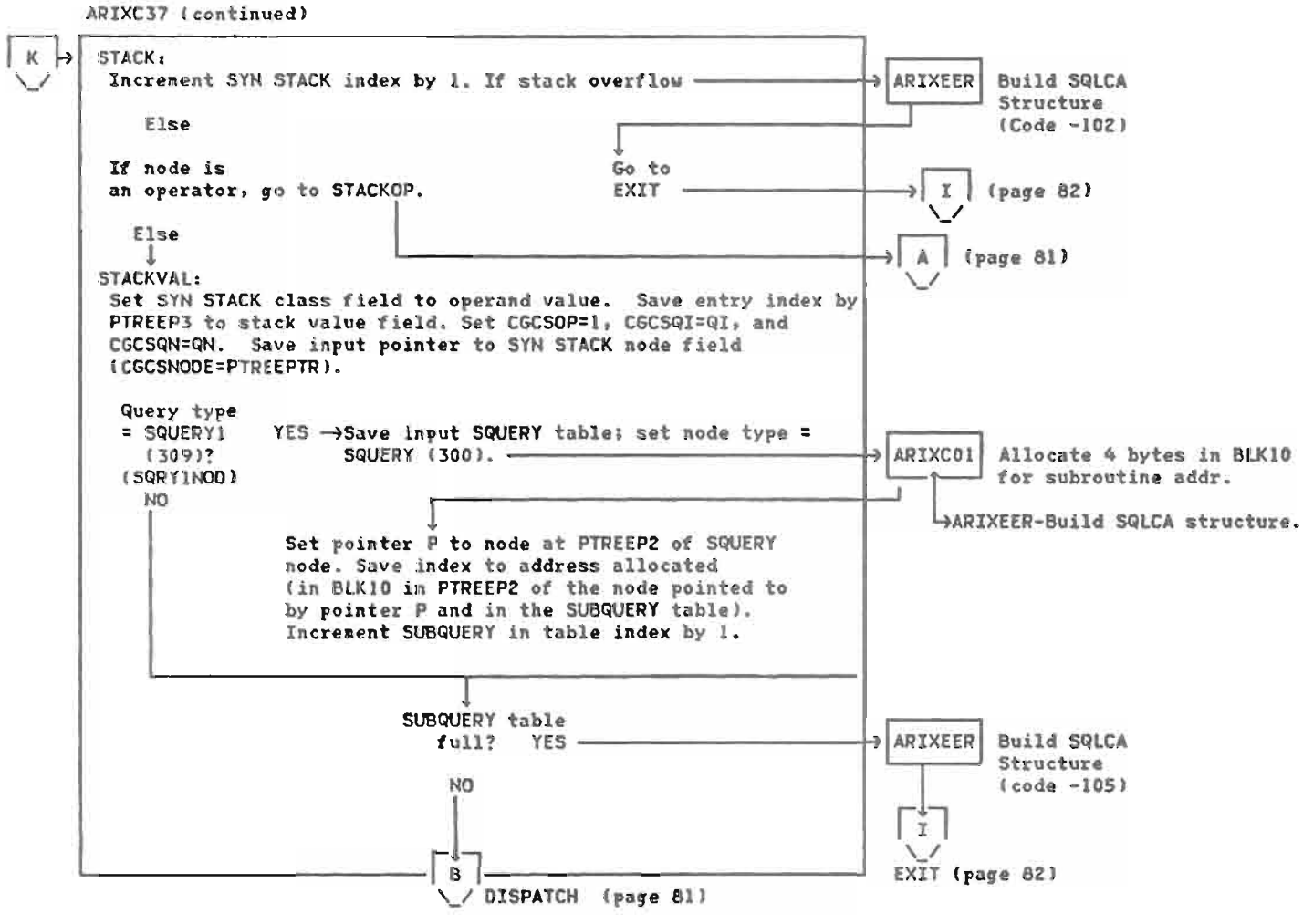
PREP-TIME CODE GENERATOR

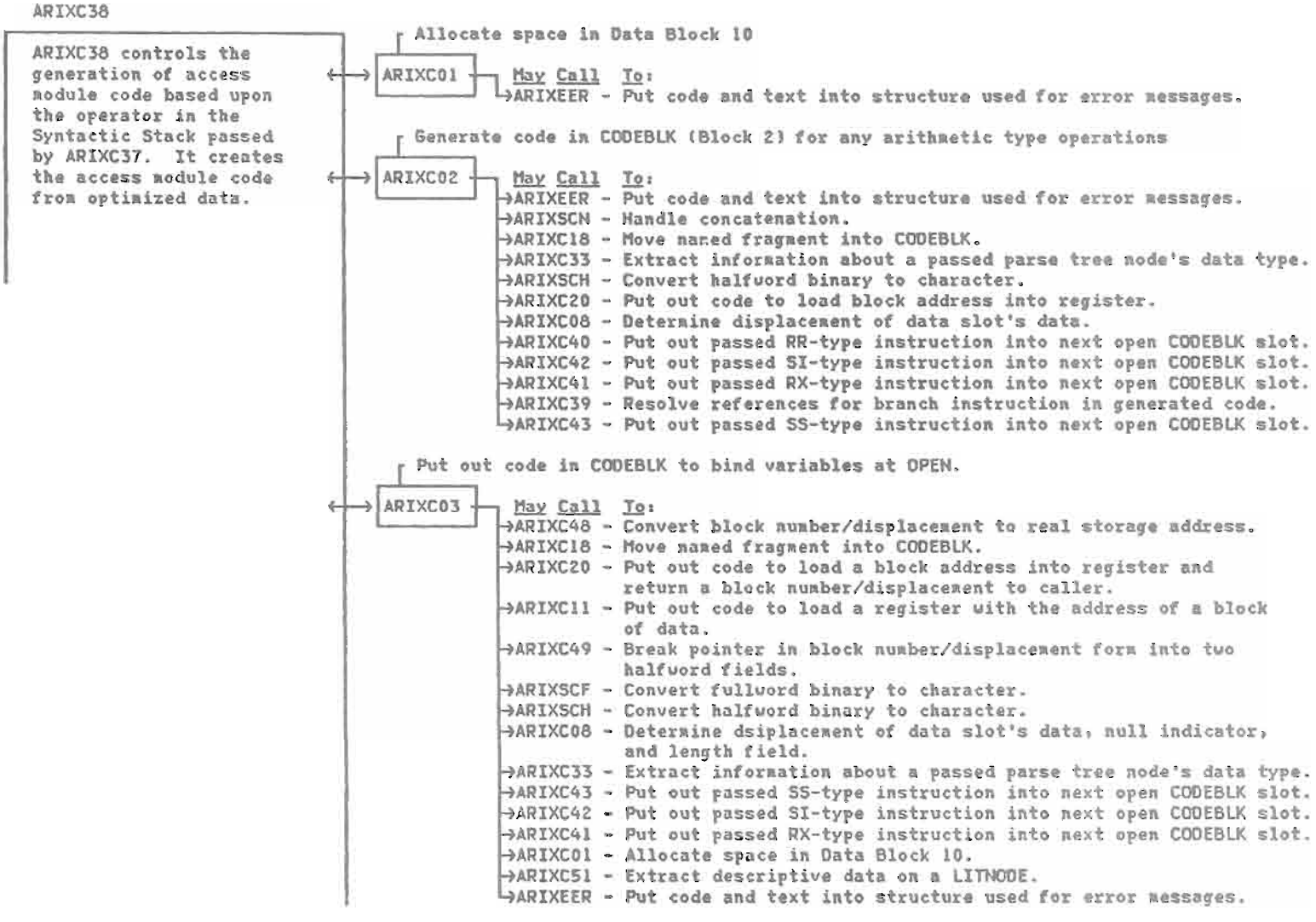


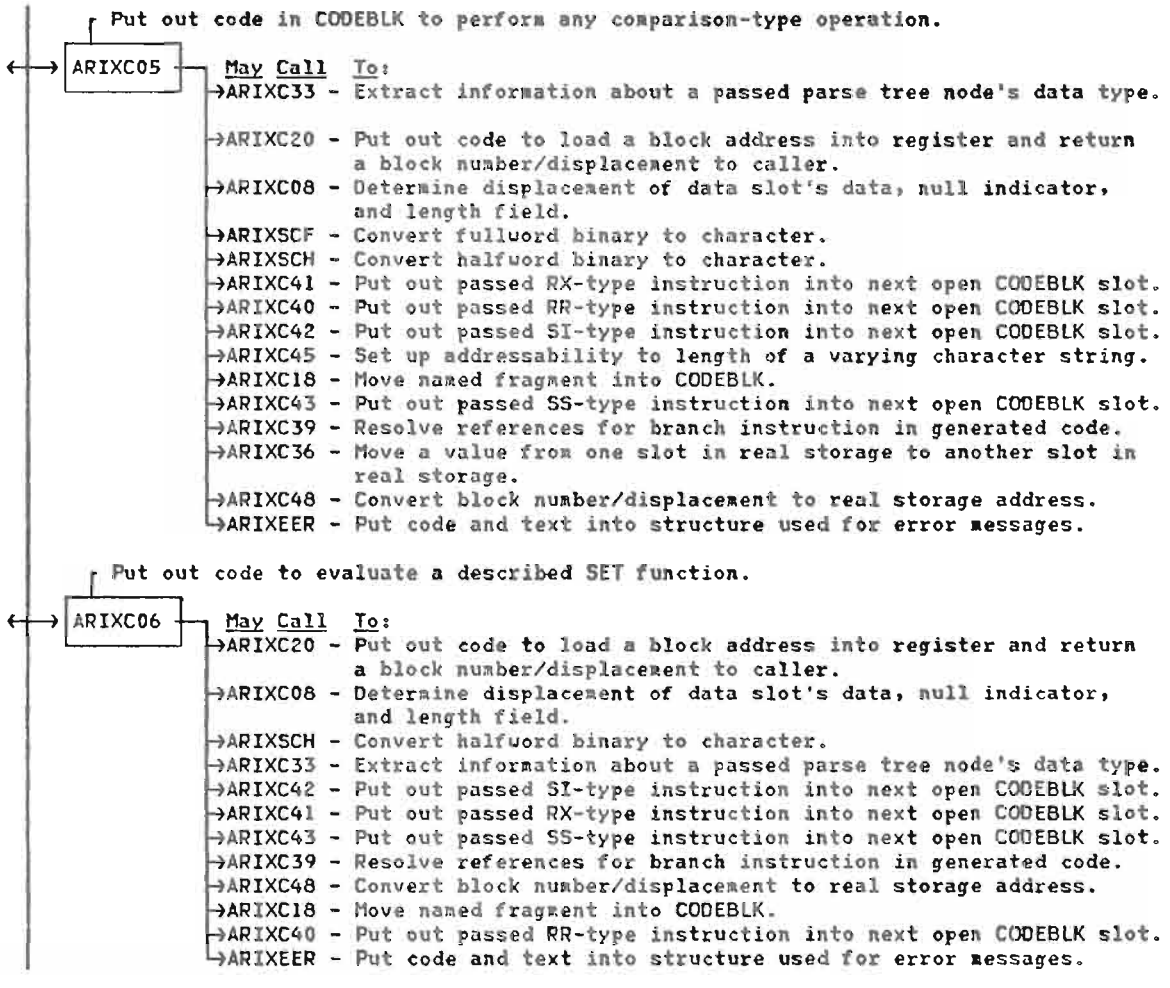


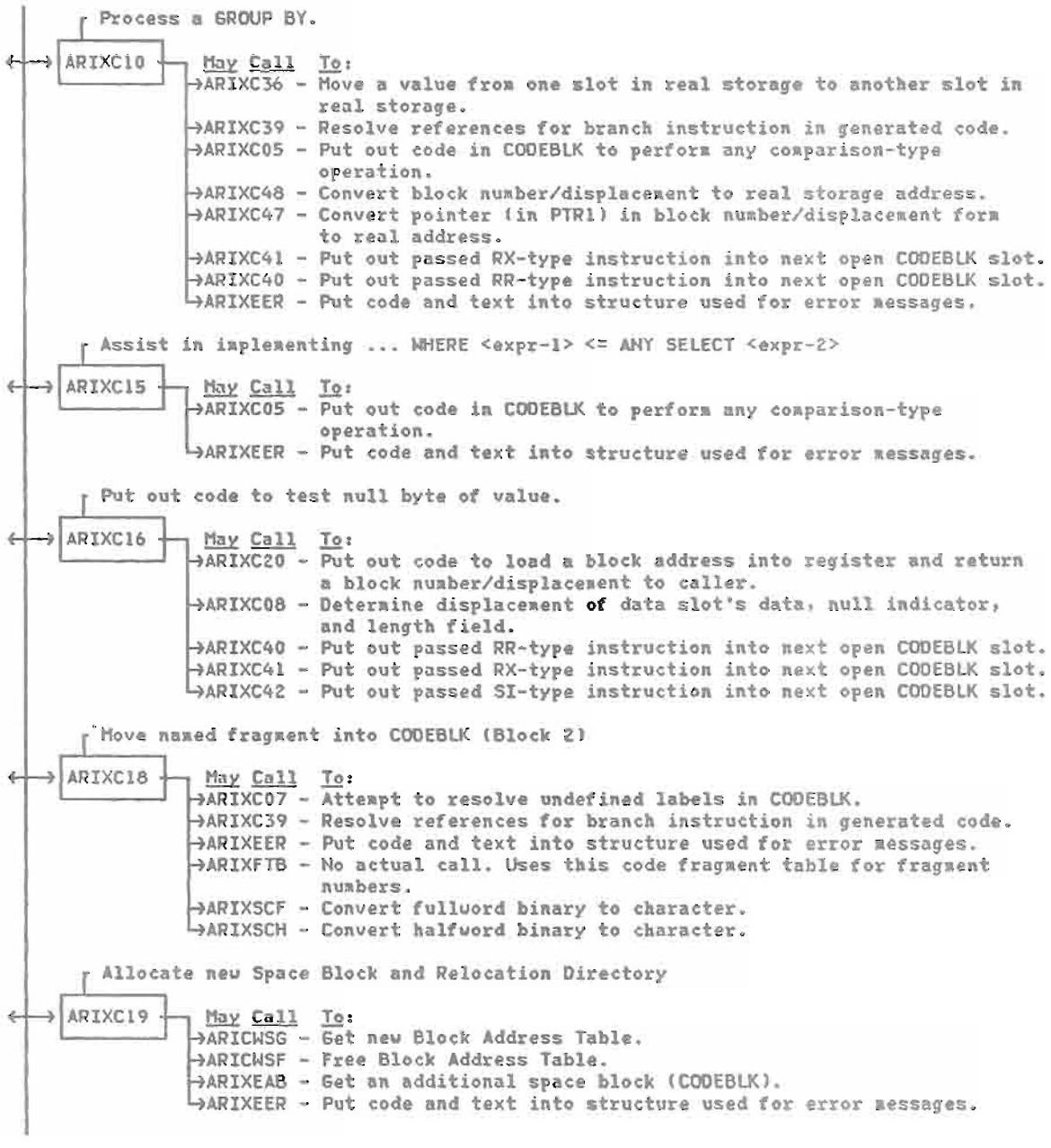


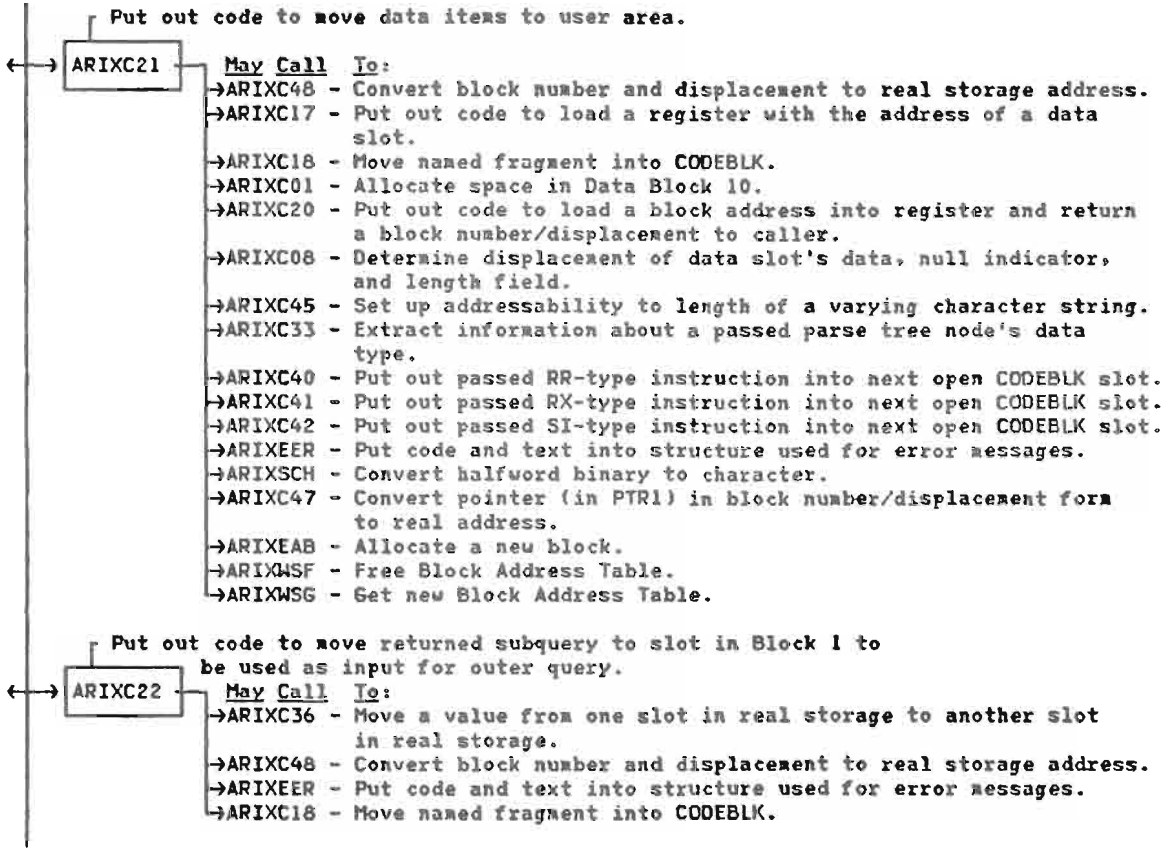
* DISPATCH - page 81



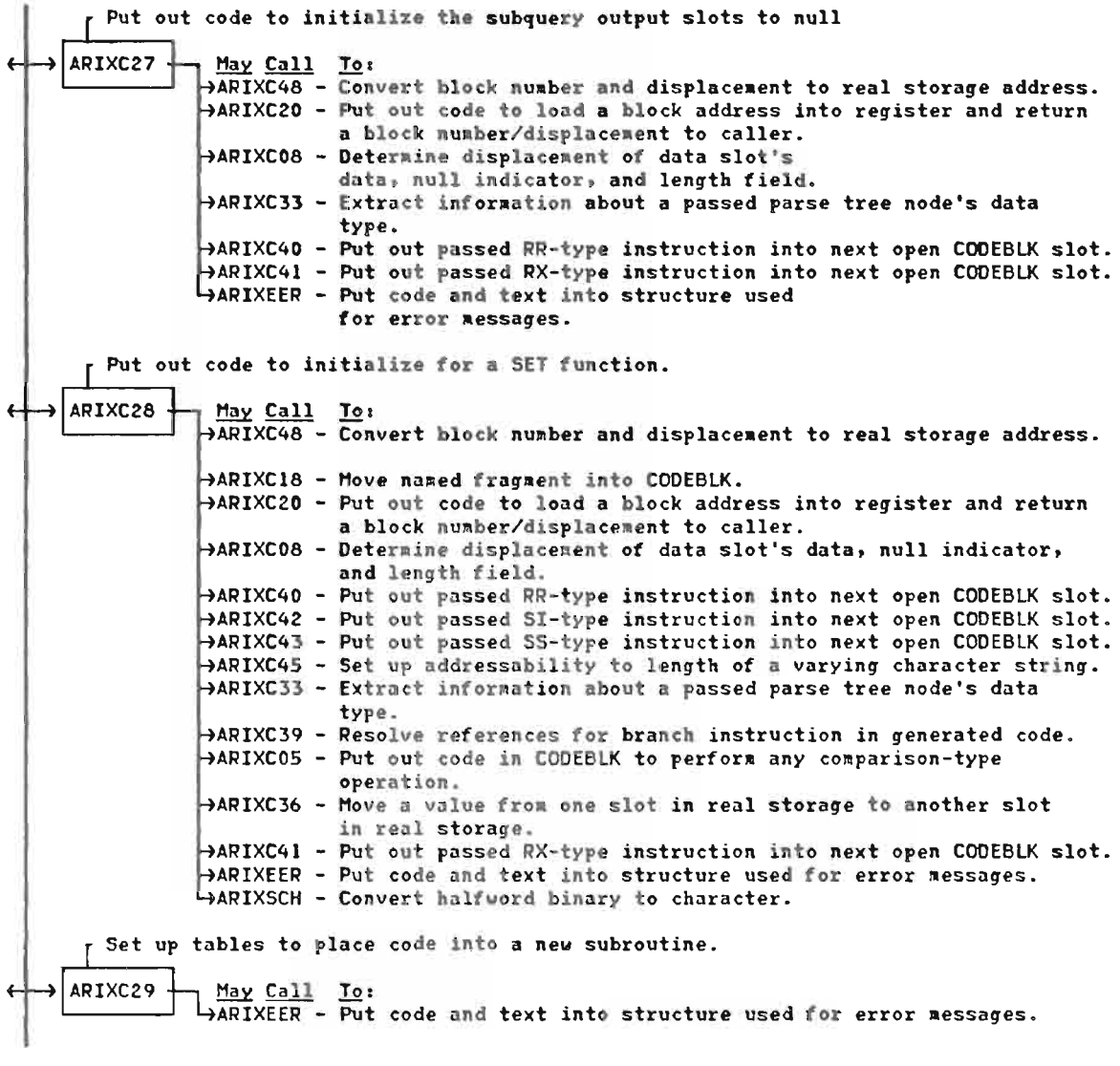

ARIXC38
ARIXC38 controls the generation of access module code based upon the operator in the Syntactic Stack passed by ARIXC37. It creates the access module code from optimized data.
Allocate space in Data Block 10
ARIXC01
May Call To:
→ARIXEER - Put code and text into structure used for error messages.
Generate code in CODEBLK (Block 2) for any arithmetic type operations
ARIXC02
May Call To:
→ARIXEER - Put code and text into structure used for error messages.
→ARIXSCN - Handle concatenation.
→ARIXC18 - Move named fragment into CODEBLK.
→ARIXC33 - Extract information about a passed parse tree node's data type.
→ARIXSCH - Convert halfword binary to character.
→ARIXC20 - Put out code to load block address into register.
→ARIXC08 - Determine displacement of data slot's data.
→ARIXC40 - Put out passed RR-type instruction into next open CODEBLK slot.
→ARIXC42 - Put out passed SI-type instruction into next open CODEBLK slot.
→ARIXC41 - Put out passed RX-type instruction into next open CODEBLK slot.
→ARIXC39 - Resolve references for branch instruction in generated code.
→ARIXC43 - Put out passed SS-type instruction into next open CODEBLK slot.
Put out code in CODEBLK to bind variables at OPEN.
ARIXC03
May Call To:
→ARIXC48 - Convert block number/displacement to real storage address.
→ARIXC18 - Move named fragment into CODEBLK.
→ARIXC20 - Put out code to load a block address into register and return a block number/displacement to caller.
→ARIXC11 - Put out code to load a register with the address of a block of data.
→ARIXC49 - Break pointer in block number/displacement form into two halfword fields.
→ARIXSCF - Convert fullword binary to character.
→ARIXSCH - Convert halfword binary to character.
→ARIXC08 - Determine dsiplacement of data slot's data, null indicator, and length field.
→ARIXC33 - Extract information about a passed parse tree node's data type.
→ARIXC43 - Put out passed SS-type instruction into next open CODEBLK slot.
→ARIXC42 - Put out passed SI-type instruction into next open CODEBLK slot.
→ARIXC41 - Put out passed RX-type instruction into next open CODEBLK slot.
→ARIXC01 - Allocate space in Data Block 10.
→ARIXC51 - Extract descriptive data on a LITNODE.
→ARIXEER - Put code and text into structure used for error messages.

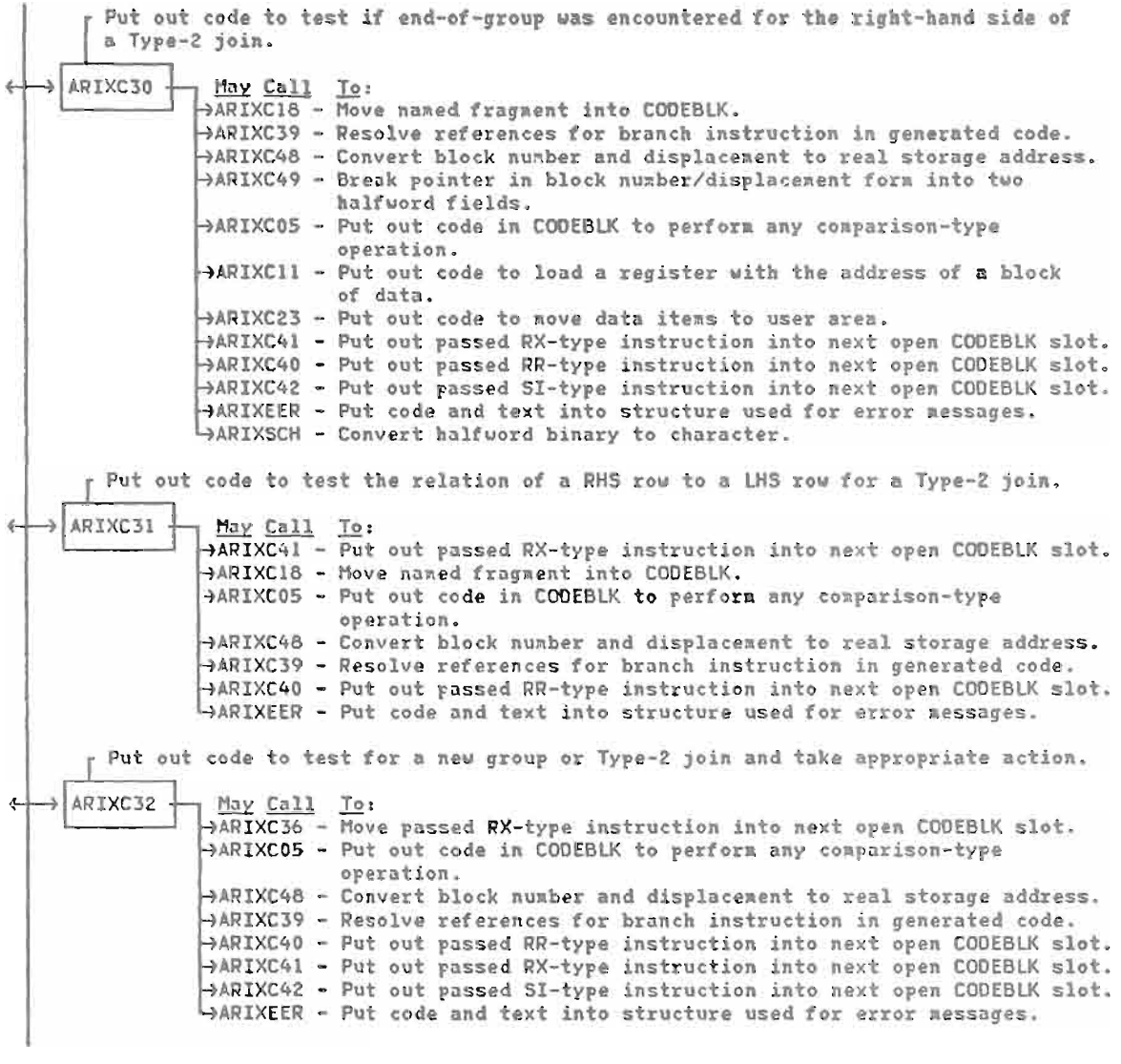


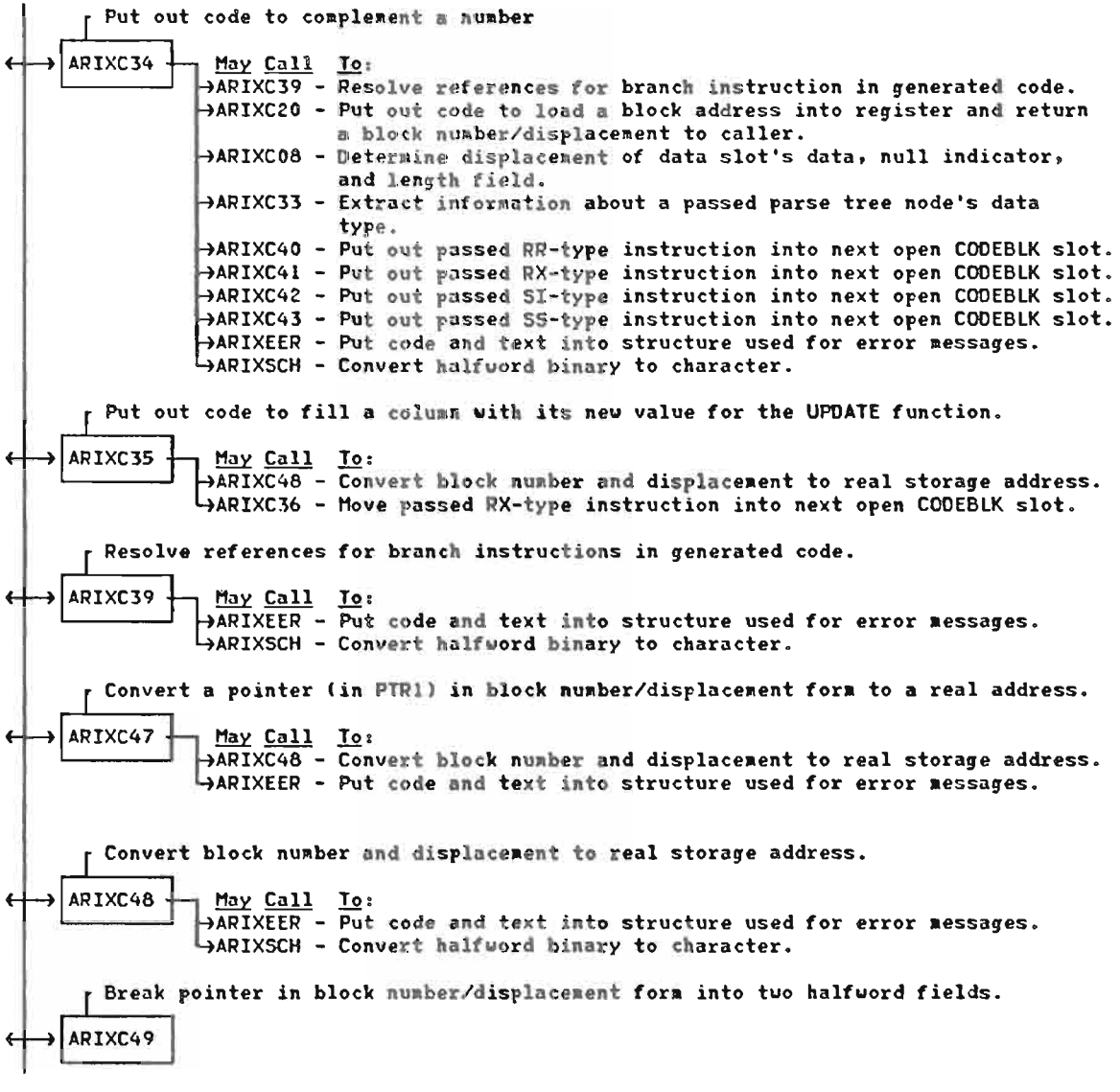

Process a GROUP BY.
ARIXC10
May Call To:
→ARIXC36 - Move a value from one slot in real storage to another slot in real storage.
→ARIXC39 - Resolve references for branch instruction in generated code.
→ARIXC05 - Put out code in CODEBLK to perform any comparison-type operation.
→ARIXC48 - Convert block number/displacement to real storage address.
→ARIXC47 - Convert pointer (in PTR1) in block number/displacement form to real address.
→ARIXC41 - Put out passed RX-type instruction into next open CODEBLK slot.
→ARIXC40 - Put out passed RR-type instruction into next open CODEBLK slot.
→ARIXEER - Put code and text into structure used for error messages.
Assist in implementing ... WHERE <expr-1> <= ANY SELECT <expr-2>
ARIXC15
May Call To:
→ARIXC05 - Put out code in CODEBLK to perform any comparison-type operation.
→ARIXEER - Put code and text into structure used for error messages.
Put out code to test null byte of value.
ARIXC16
May Call To:
→ARIXC20 - Put out code to load a block address into register and return a block number/displacement to caller.
→ARIXC08 - Determine displacement of data slot's data, null indicator, and length field.
→ARIXC40 - Put out passed RR-type instruction into next open CODEBLK slot.
→ARIXC41 - Put out passed RX-type instruction into next open CODEBLK slot.
→ARIXC42 - Put out passed SI-type instruction into next open CODEBLK slot.
Move named fragment into CODEBLK (Block 2)
ARIXC18
May Call To:
→ARIXC07 - Attempt to resolve undefined labels in CODEBLK.
→ARIXC39 - Resolve references for branch instruction in generated code.
→ARIXEER - Put code and text into structure used for error messages.
→ARIXFTB - No actual call. Uses this code fragment table for fragment numbers.
→ARIXSCF - Convert fullword binary to character.
→ARIXSCH - Convert halfword binary to character.
Allocate new Space Block and Relocation Directory
ARIXC19
May Call To:
→ARICWSG - Get new Block Address Table.
→ARICWSF - Free Block Address Table.
→ARIXEAB - Get an additional space block (CODEBLK).
→ARIXEER - Put code and text into structure used for error messages.



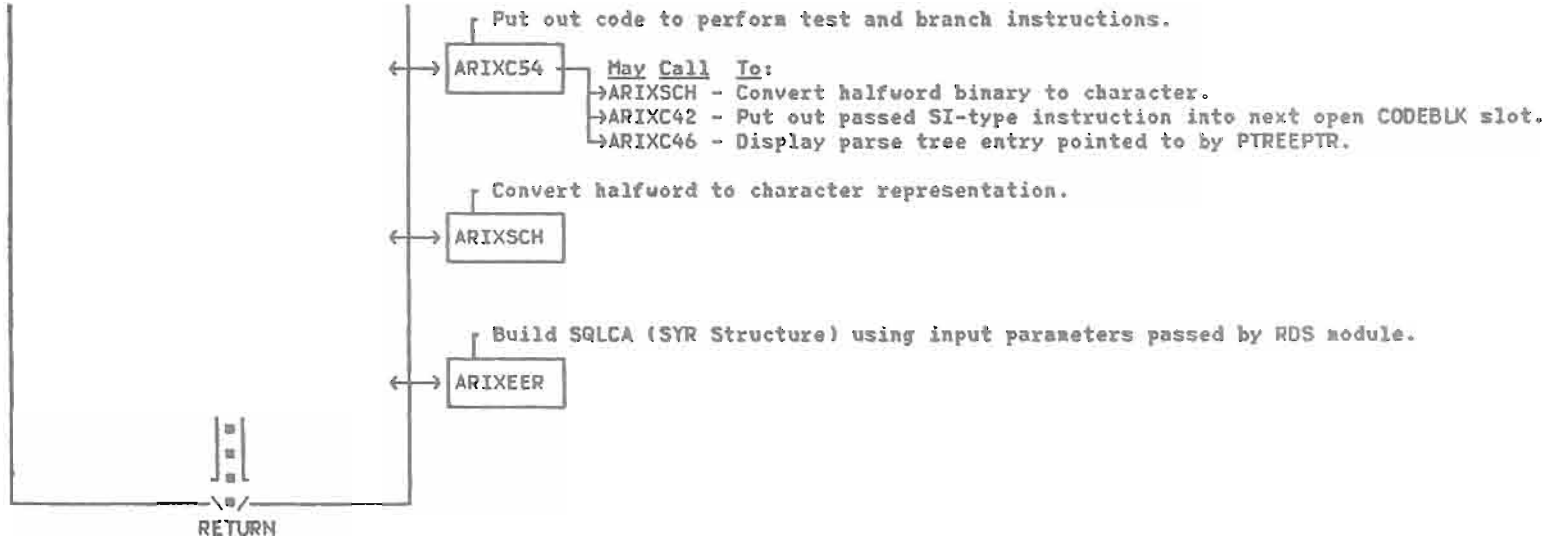

Put out code to bind table
ARIXC23
May Call To:
→ARIXSCH - Convert halfword binary to character.
→ARIXC11 - Put out code to load a register with the address of a block of data.
→ARIXC43 - Put out passed SS-type instruction into next open CODEBLK slot.
→ARIXC36 - Move a value from one slot in real storage to another slot in real storage.
→ARIXC47 - Convert pointer (in PTR1) in block number/displacement form to real address.
→ARIXC48 - Convert block number and displacement to real storage address.
→ARIXC49 - Break pointer in block number/displacement form into two halfword fields.
→ARIXC46 - Display parse tree entry pointed to by PTREEPTR.
→ARIXEER - Put code and text into structure used for error messages.
Put out code to call subroutines for subqueries
ARIXC24
May Call To:
→ARIXSCH - Convert halfword binary to character.
→ARIXC11 - Put out code to load a register with the address of a block of data.
→ARIXC18 - Move named fragment into CODEBLK.
→ARIXC23 - Put out code to move data items to user area.
→ARIXC40 - Put out passed RR-type instruction into next open CODEBLK slot.
→ARIXC41 - Put out passed RX-type instruction into next open CODEBLK slot.
→ARIXC42 - Put out passed SI-type instruction into next open CODEBLK slot.
→ARIXC46 - Display parse tree entry pointed to by PTREEPTR.
→ARIXC47 - Convert pointer (in PTR1) in block number/displacement form to real address.
→ARIXC48 - Convert block number and displacement to real storage address.
→ARIXEER - Put code and text into structure used for error messages.
Put out code to process the binding operation for a RE-OPEN ON CURSOR situation
ARIXC25
May Call To:
→ARIXC39 - Resolve references for branch instruction in generated code.
→ARIXC23 - Put out code to move data items to user area.
→ARIXC07 - Attempt to resolve undefined labels in CODEBLK.
→ARIXC40 - Put out passed RR-type instruction into next open CODEBLK slot.
→ARIXC41 - Put out passed RX-type instruction into next open CODEBLK slot.




Put out code to test if end-of-group was encountered for the right-hand side of a Type-2 join.
ARIXC30
May Call To:
→ARIXC18 - Move named fragment into CODEBLK.
→ARIXC39 - Resolve references for branch instruction in generated code.
→ARIXC48 - Convert block number and displacement to real storage address.
→ARIXC49 - Break pointer in block number/displacement form into two halfword fields.
→ARIXC05 - Put out code in CODEBLK to perform any comparison-type operation.
→ARIXC11 - Put out code to load a register with the address of a block of data.
→ARIXC23 - Put out code to move data items to user area.
→ARIXC41 - Put out passed RX-type instruction into next open CODEBLK slot.
→ARIXC40 - Put out passed RR-type instruction into next open CODEBLK slot.
→ARIXC42 - Put out passed SI-type instruction into next open CODEBLK slot.
→ARIXEER - Put code and text into structure used for error messages.
→ARIXSCH - Convert halfword binary to character.
Put out code to test the relation of a RHS row to a LHS row for a Type-2 join.
ARIXC31
May Call To:
→ARIXC41 - Put out passed RX-type instruction into next open CODEBLK slot.
→ARIXC18 - Move named fragment into CODEBLK.
→ARIXC05 - Put out code in CODEBLK to perform any comparison-type operation.
→ARIXC48 - Convert block number and displacement to real storage address.
→ARIXC39 - Resolve references for branch instruction in generated code.
→ARIXC40 - Put out passed RR-type instruction into next open CODEBLK slot.
→ARIXEER - Put code and text into structure used for error messages.
Put out code to test for a new group or Type-2 join and take appropriate action.
ARIXC32
May Call To:
→ARIXC36 - Move passed RX-type instruction into next open CODEBLK slot.
→ARIXC05 - Put out code in CODEBLK to perform any comparison-type operation.
→ARIXC48 - Convert block number and displacement to real storage address.
→ARIXC39 - Resolve references for branch instruction in generated code.
→ARIXC40 - Put out passed RR-type instruction into next open CODEBLK slot.
→ARIXC41 - Put out passed RX-type instruction into next open CODEBLK slot.
→ARIXC42 - Put out passed SI-type instruction into next open CODEBLK slot.
→ARIXEER - Put code and text into structure used for error messages.

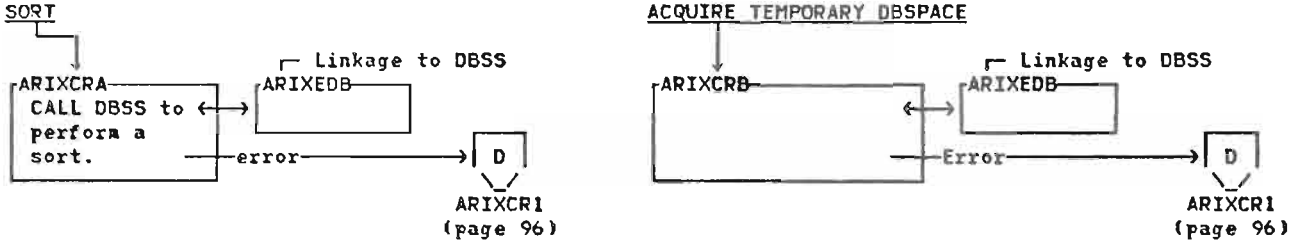


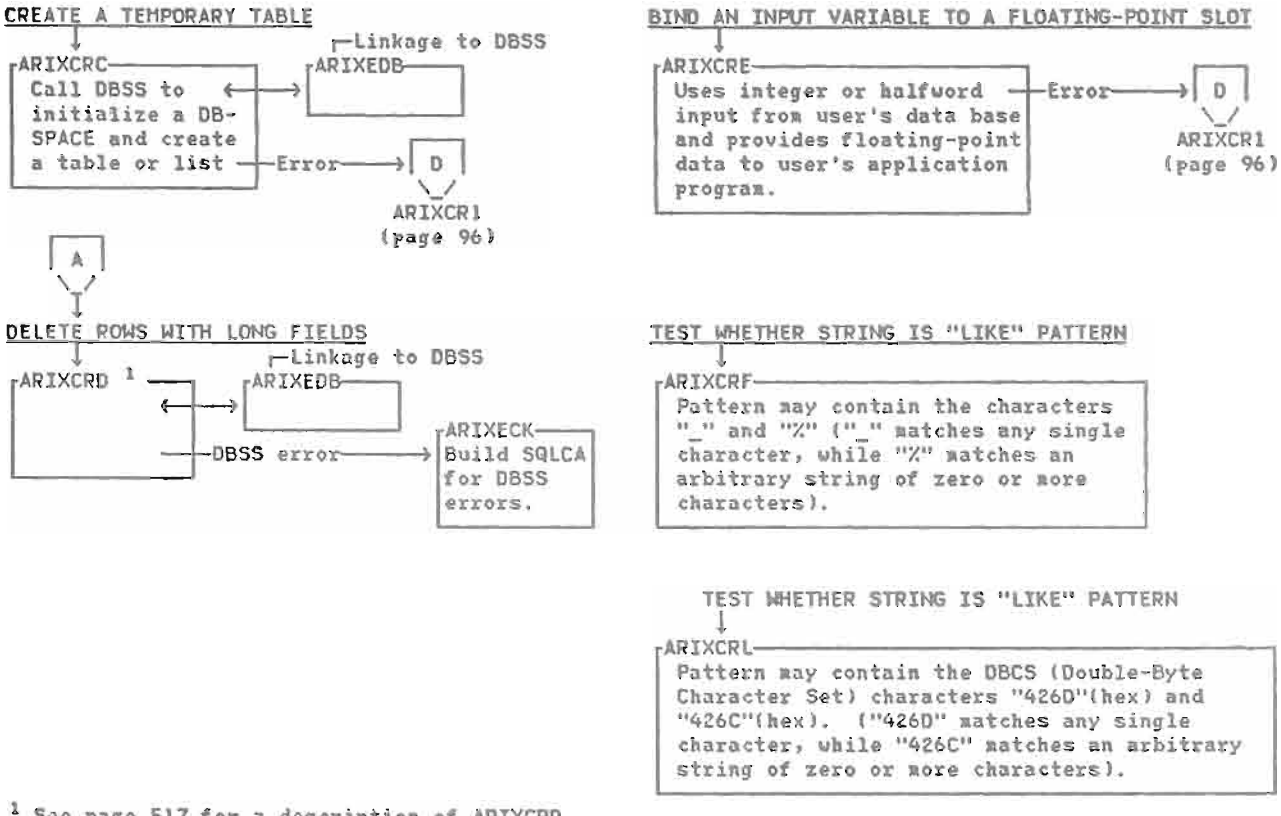
ARIXC38 (continued)



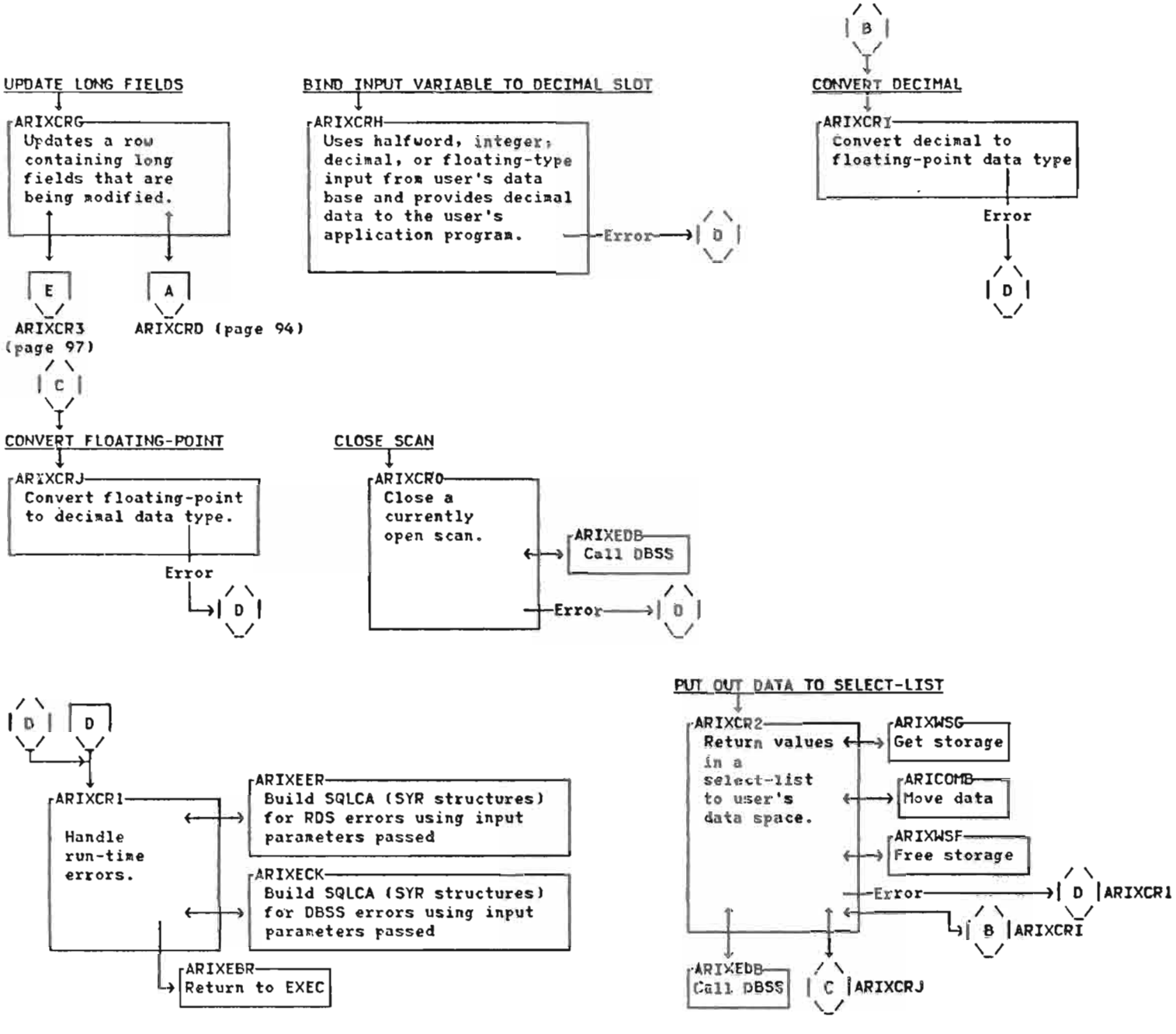
The run-time Code Generator modules are called via ARIXFTB (from fragments or other run-time routines), which contains the vector table for fragments and run-time routines.

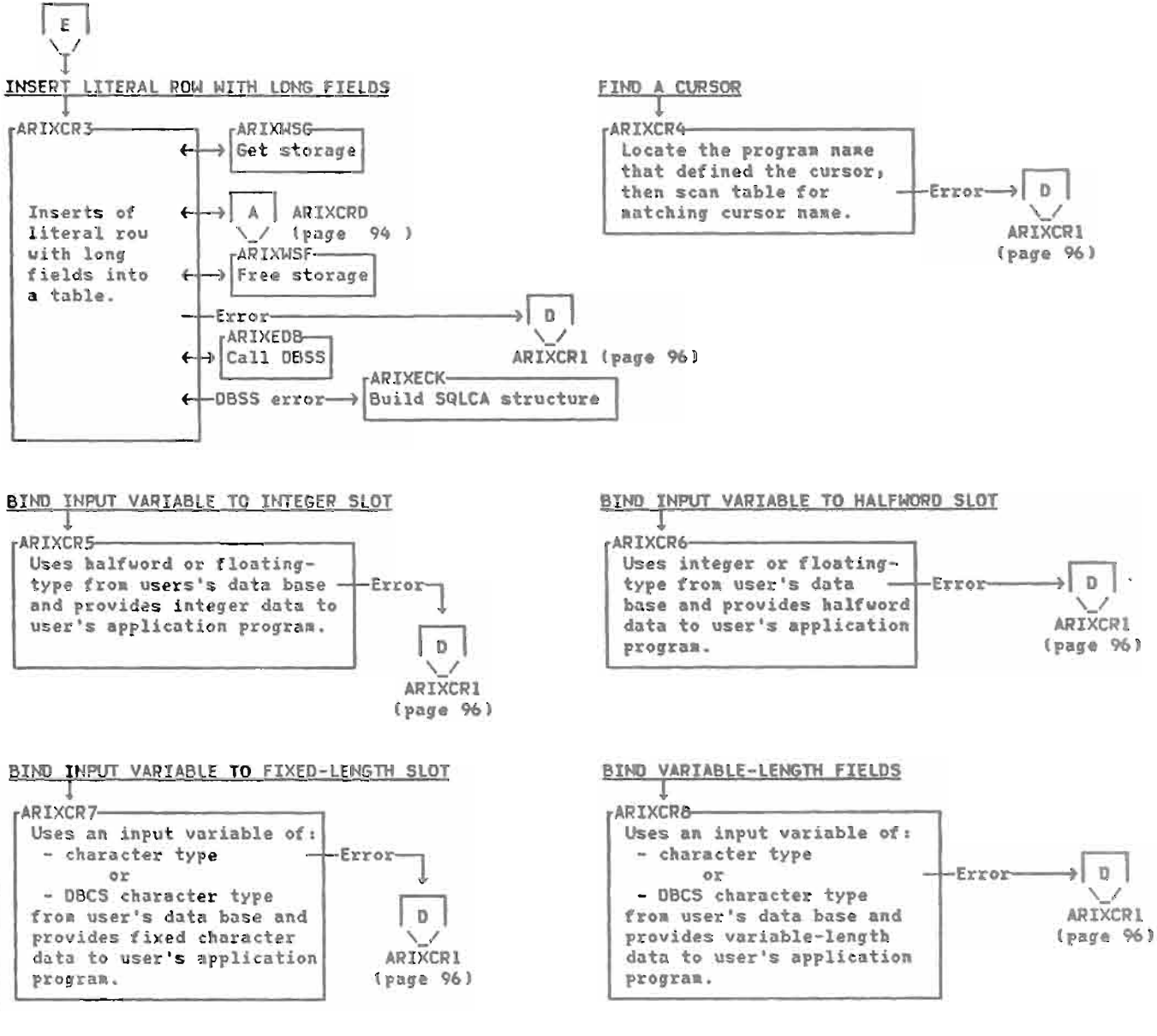
The routine called is based upon the requested function:



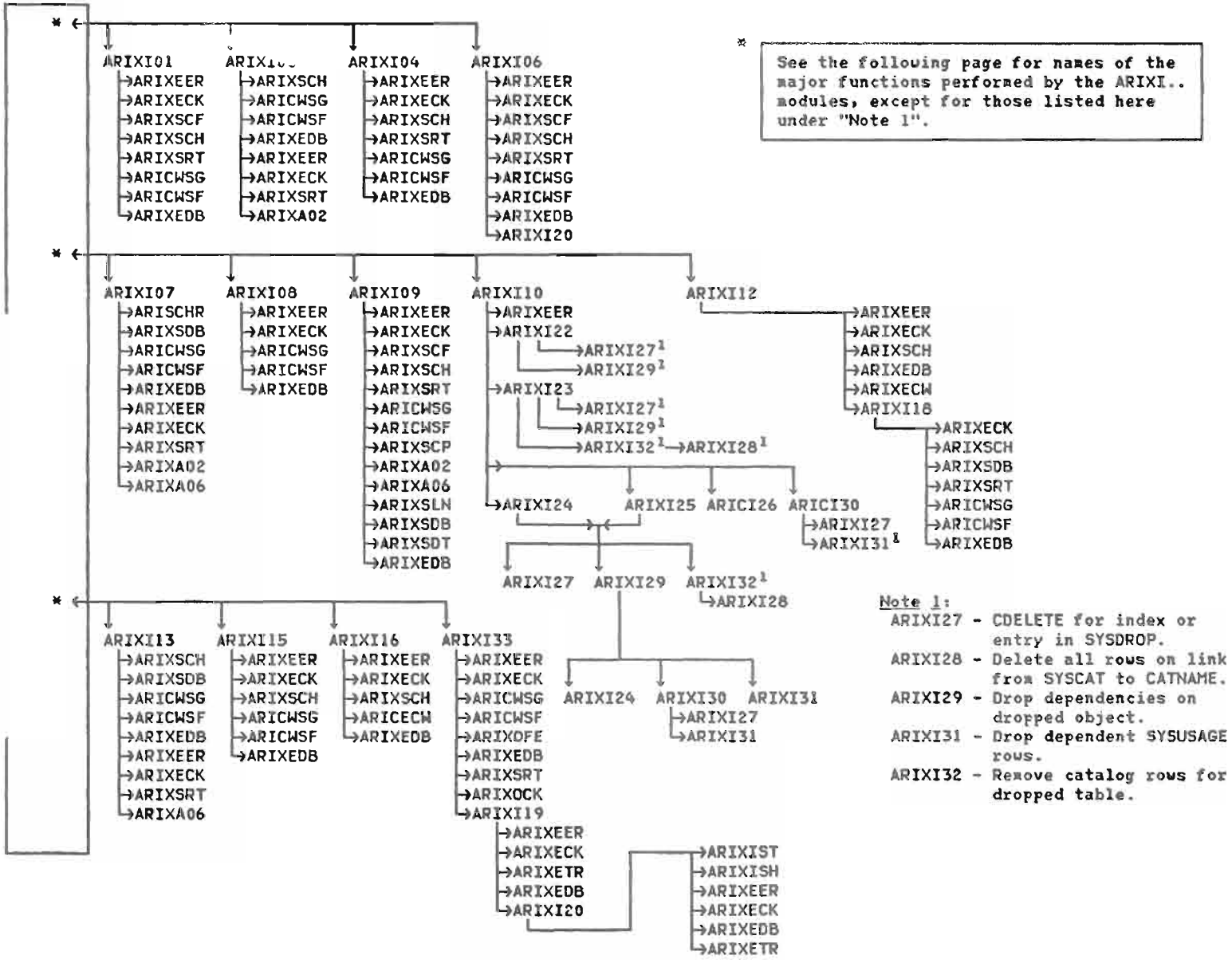


[1] See page 517 for a description of ARIXCRD.




E
INSERT LITERAL ROW WITH LONG FIELDS
ARIXCR3
Inserts of literal rou with long fields into a table.
ARIXWSG
Get storage
A
ARIXCRD
(page 94)
ARIXWSF
Free storage
Error
D
ARIXCR1 (page 96)
ARIXEDB
Call DBSS
DBSS error
ARIXECK
Build SQLCA structure
FIND A CURSOR
ARIXCR4
Locate the program name that defined the cursor, then scan table for matching cursor name.
Error
D
ARIXCR1
(page 96)
BIND INPUT VARIABLE TO INTEGER SLOT
ARIXCR5
Uses halfword or floating-type from users's data base and provides integer data to user's application program.
Error
D
ARIXCR1
(page 96)
BIND INPUT VARIABLE TO HALFWORD SLOT
ARIXCR6
Uses integer or floating-type from user's data base and provides halfword data to user's application program.
Error
D
ARIXCR1
(page 96)
BIND INPUT VARIABLE TO FIXED-LENGTH SLOT
ARIXCR7
Uses an input variable of:
- character type
or
- DBCS character type
from user's data base and provides fixed character data to user's application program.
Error
D
ARIXCR1
(page 96)
BIND VARIABLE-LENGTH FIELDS
ARIXCR8
Uses an input variable of:
- character type
or
- DBCS character type
from user's data base and provides variable-length data to user's application program.
Error
D
ARIXCR1
(page 96)

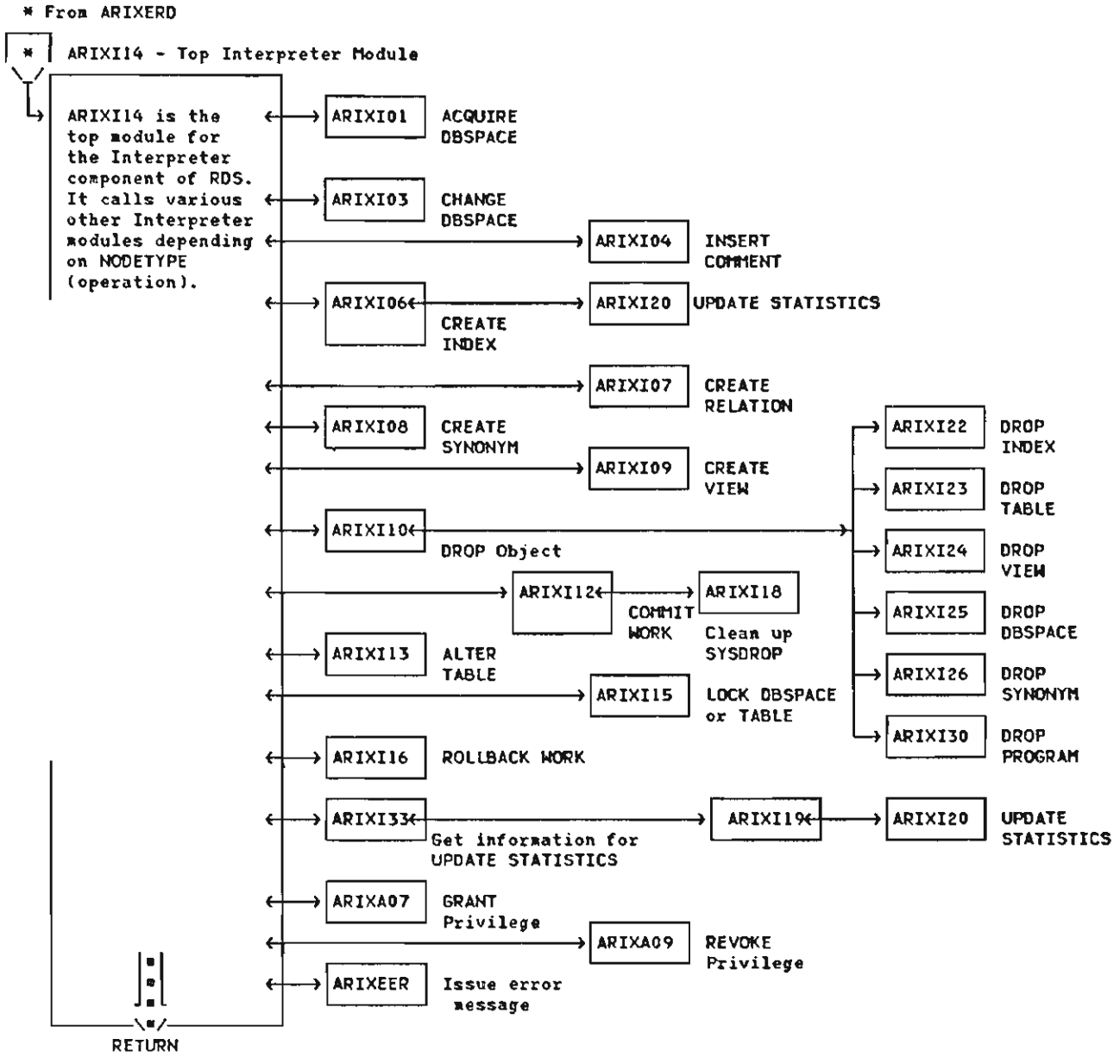
ARIXI14 - Called From ARIXERD (Topmost RDS Module)



* See the following page for names of the major functions performed by the ARIXI.. modules, except for those listed here under "Note 1".

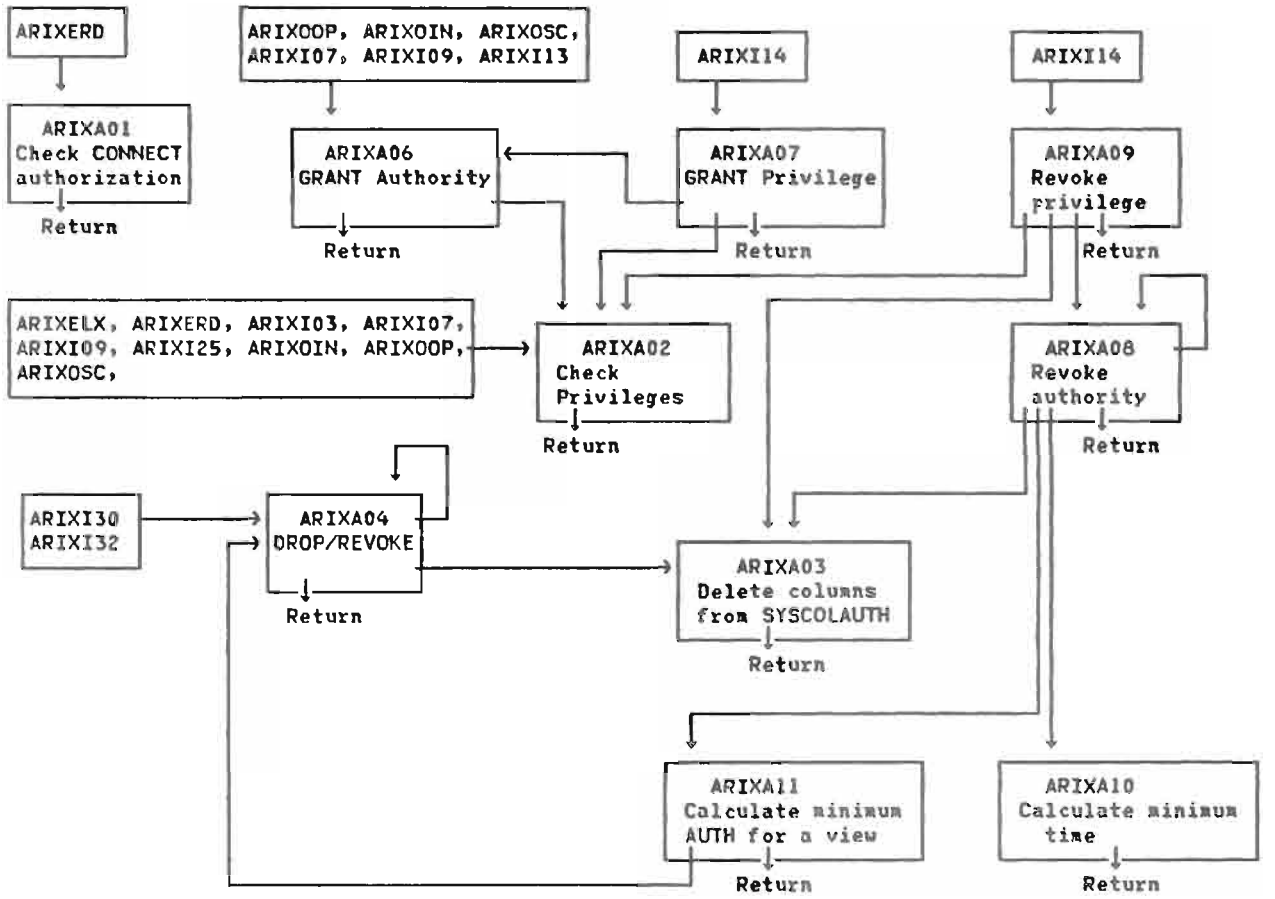
Note 1:

- ARIXI27 - CDELETE for index or entry in SYSDROP.
- ARIXI28 - Delete all rows on link from SYSCAT to CATNAME.
- ARIXI29 - Drop dependencies on dropped object.
- ARIXI31 - Drop dependent SYSUSAGE rows.
- ARIXI32 - Remove catalog rows for dropped table.

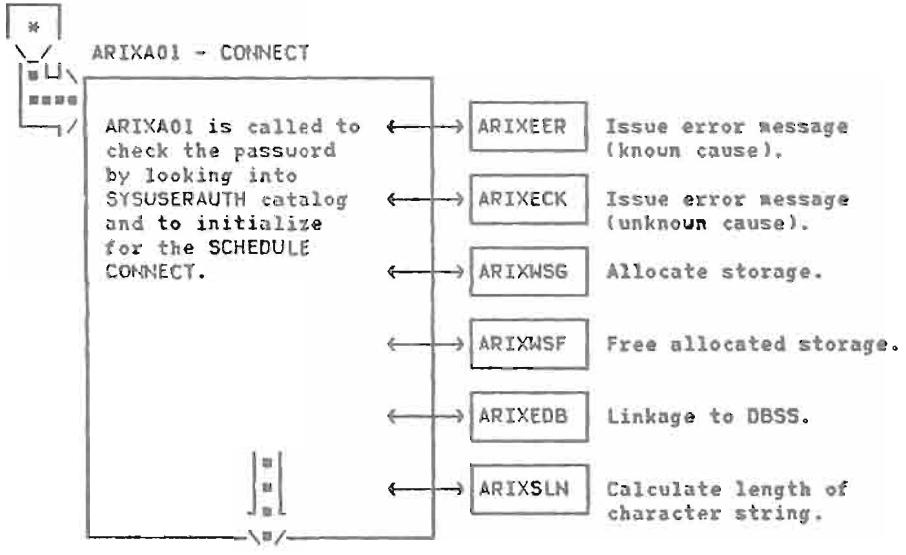

* From ARIXERD
*
ARIXI14 - Top Interpreter Module
ARIXI14 is the top module for the Interpreter component of RDS. It calls various other Interpreter modules depending on NODETYPE (operation).
ARIXI01
ACQUIRE DBSPACE
ARIXI03
CHANGE DBSPACE
ARIXI04
INSERT COMMENT
ARIXI06
CREATE INDEX
ARIXI20
UPDATE STATISTICS
ARIXI07
CREATE RELATION
ARIXI08
CREATE SYNONYM
ARIXI09
CREATE VIEW
ARIXI10
DROP Object
ARIXI12
COMMIT WORK
ARIXI18
Clean up SYSDROP
ARIXI13
ALTER TABLE
ARIXI15
LOCK DBSPACE or TABLE
ARIXI16
ROLLBACK WORK
ARIXI33
Get information for UPDATE STATISTICS
ARIXI19
ARIXI20
UPDATE STATISTICS
ARIXA07
GRANT Privilege
ARIXA09
REVOKE Privilege
ARIXEER
Issue error message
RETURN
ARIXI22
DROP INDEX
ARIXI23
DROP TABLE
ARIXI24
DROP VIEW
ARIXI25
DROP DBSPACE
ARIXI26
DROP SYNONYM
ARIXI30
DROP PROGRAM

AUTHORIZATION

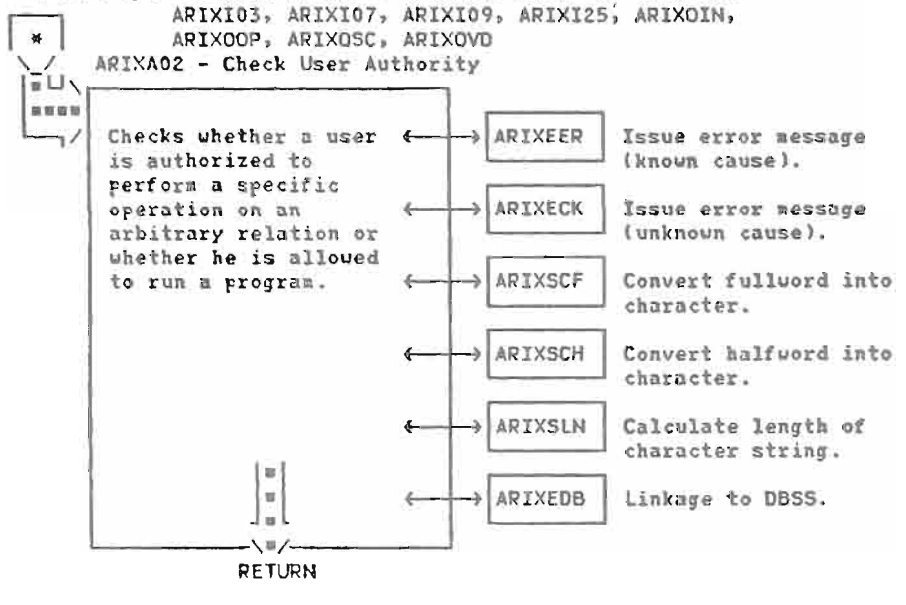
The following diagrams show from where the authorization modules are called and the interactions between the authorization modules. More detailed diagrams follow this one. For a description of the individual modules, see Section 3, Program Organization.

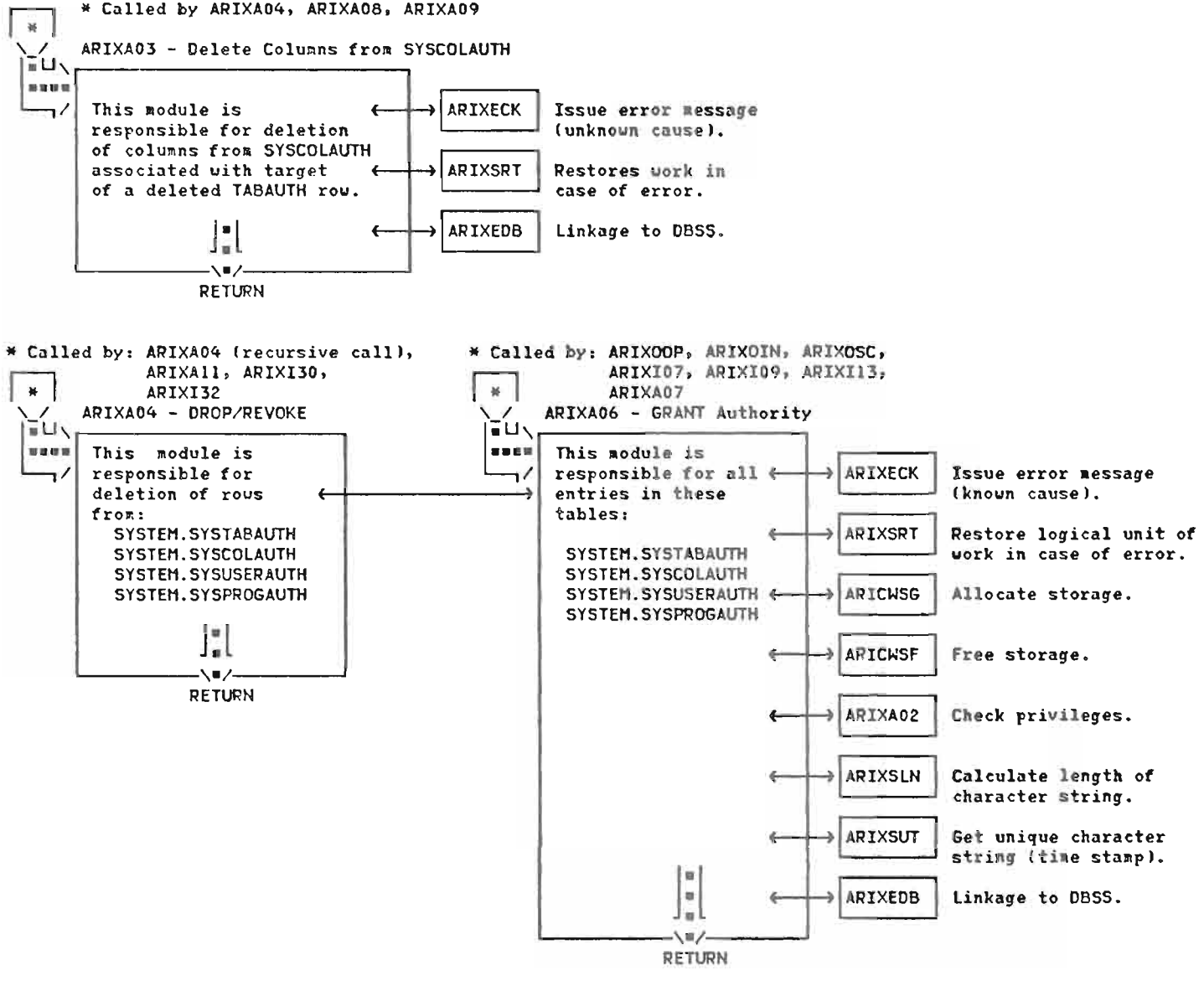


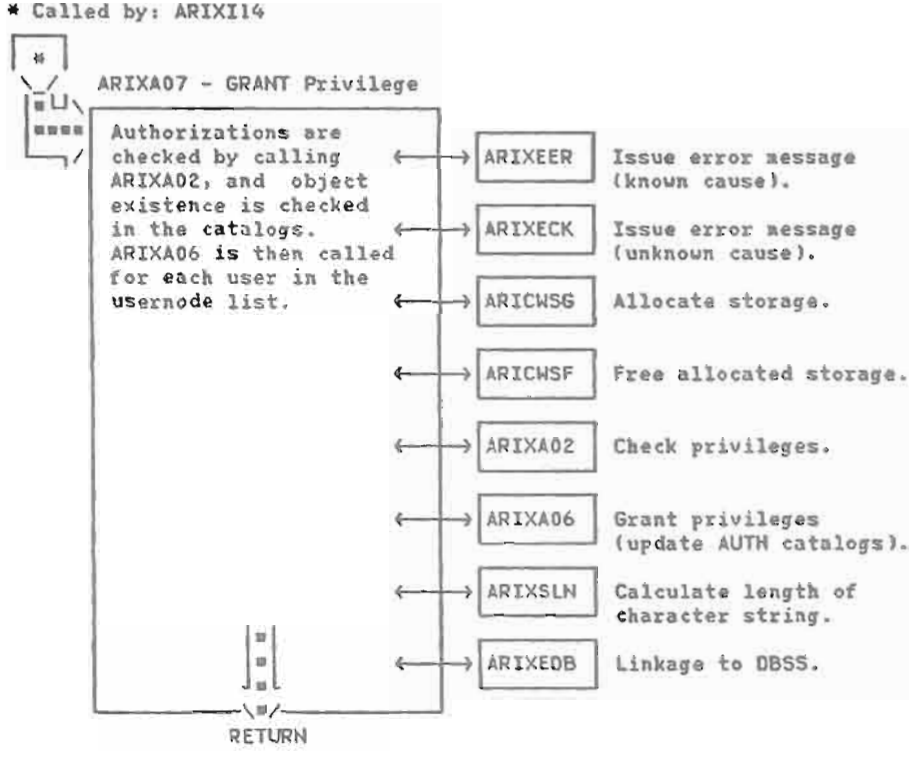
* Called by ARIXERD

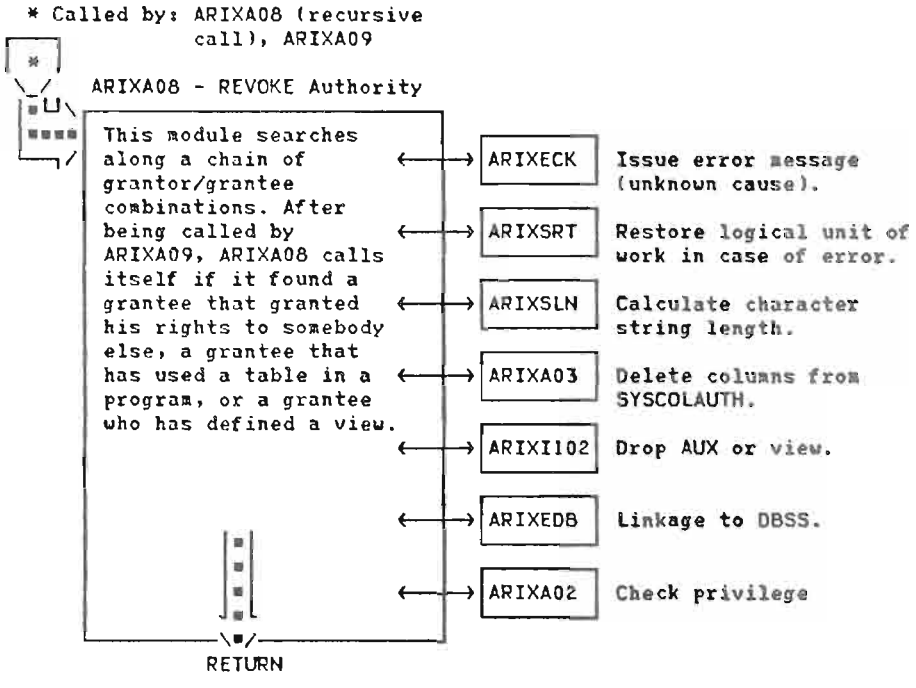


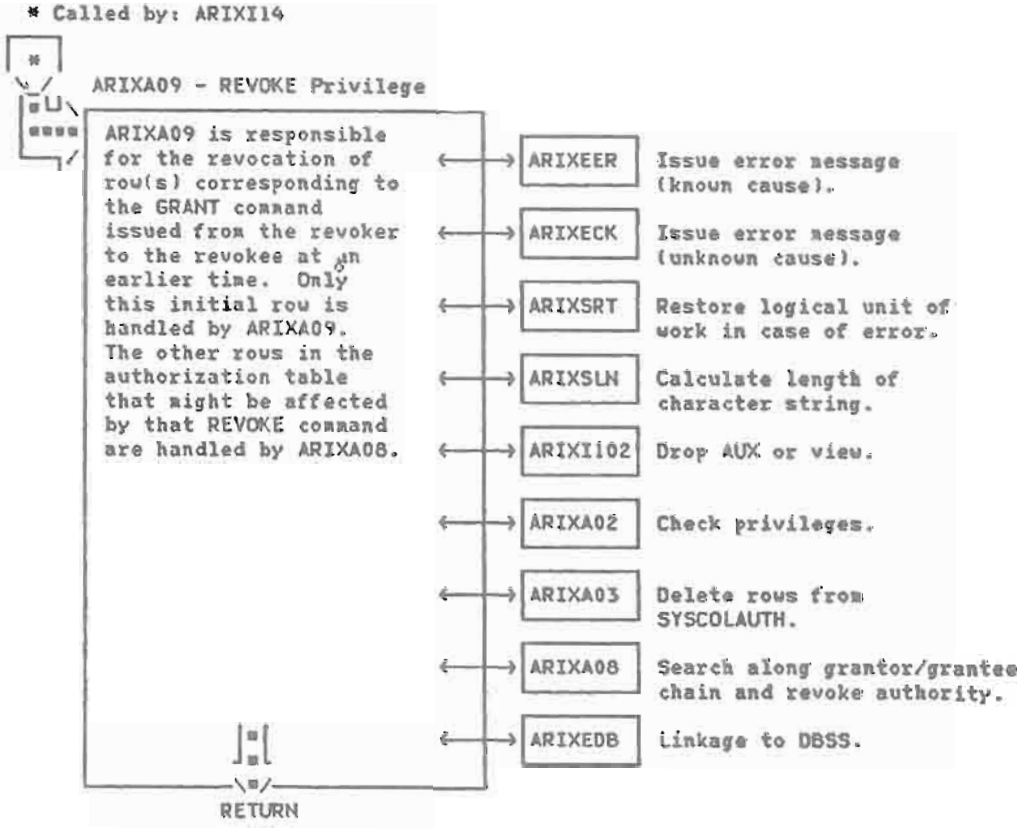
* Called by: ARIXA05, ARIXA07, ARIXA09, ARIXELX, ARIXERD,
ARIXI03, ARIXI07, ARIXI09, ARIXI25, ARIXOIN,
ARIXOOP, ARIXQSC, ARIXOVD



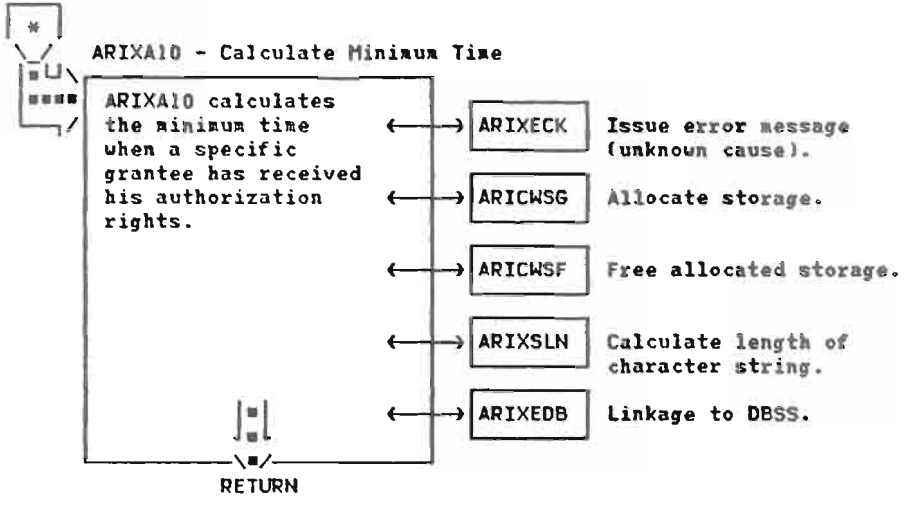



* Called by: ARIXI14
ARIXA07 - GRANT Privilege
Authorizations are checked by calling ARIXA02, and object existence is checked in the catalogs. ARIXA06 is then called for each user in the usernode list.
ARIXEER
Issue error message (known cause).
ARIXECK
Issue error message (unknown cause).
ARICWSG
Allocate storage.
ARICWSF
Free allocated storage.
ARIXA02
Check privileges.
ARIXA06
Grant privileges (update AUTH catalogs).
ARIXSLN
Calculate length of character string.
ARIXEDB
Linkage to DBSS.
RETURN

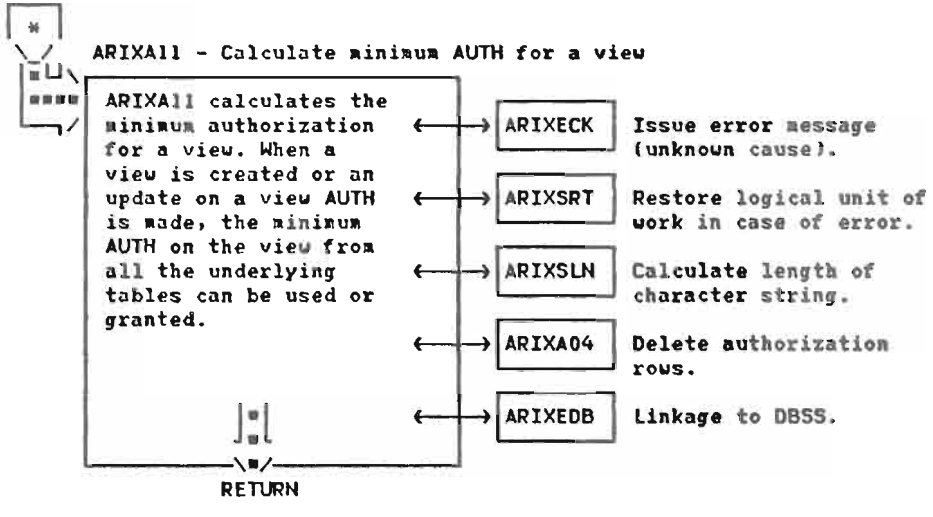



* Called by: ARIXI14
ARIXA09 - REVOKE Privilege
ARIXA09 is responsible for the revocation of row(s) corresponding to the GRANT command issued from the revoker to the revokee at an earlier time. Only this initial row is handled by ARIXA09. The other rows in the authorization table that might be affected by that REVOKE command are handled by ARIXA08.
ARIXEER
Issue error message (known cause).
ARIXECK
Issue error message (unknown cause).
ARIXSRT
Restore logical unit of work in case of error.
ARIXSLN
Calculate length of character string.
ARIXI102
Drop AUX or view.
ARIXA02
Check privileges.
ARIXA03
Delete rows from SYSCOLAUTH.
ARIXA08
Search along grantor/grantee chain and revoke authority.
ARIXEDB
Linkage to DBSS.
RETURN

* Called by: ARIXA08



* Called by: ARIXA08



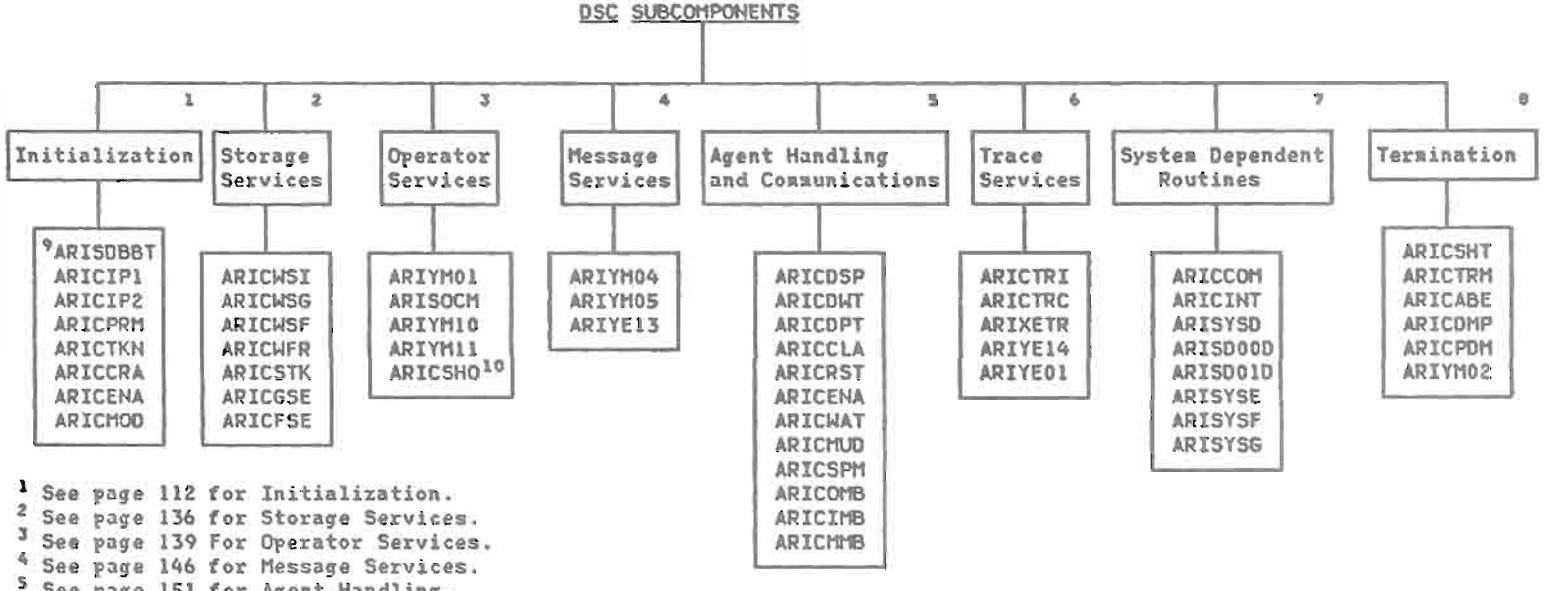
DATA SYSTEM CONTROL (DSC)

The Data System Control (DSC) component provides the control services for SQL/DS. It:

- Controls the initialization and termination of SQL/DS
- Provides for the other SQL/DS components
  - Storage Services
  - Message Services
  - Operator Services
  - Trace Services
- Provides Agent Handling and Communications capabilities (that is, it allows other subcomponents of SQL/DS to handle multiple concurrent users and to communicate with them)
- Provides system-dependent routines that shield the other subcomponents from the VSE/Advanced Functions and VM/SP system interfaces (that is, it allows the other subcomponents of SQL/DS to be system independent).

DSC OVERVIEW

For a high-level view of how DSC fits in to the SQL/DS scheme, refer back to page 6.



1 See page 112 for Initialization.
2 See page 136 for Storage Services.
3 See page 139 For Operator Services.
4 See page 146 for Message Services.
5 See page 151 for Agent Handling.
6 See page 188 for Trace Services.
7 See page 106 for the DSC system dependent module ARICCOM, page 513 for ARISYSD, page 109 for ARISD00D and ARISD01D, page 109 for ARISYSE, page 109 for ARISYSF, and page 109 for ARISYSG. (D-suffix modules are VSE/Advanced Functions only, and C-suffix modules are VM/SP only.)
8 See page 209 for Termination
9 VM environment only. (This is the ARISDBK module after a "GENMOD" under the name ARISDBBT.)
10 VM environment only.

Initialization Modules

ARISDBBT - ('GENMODed' version of ARISDBK.) In the VM environment, this bootstrap module loads the DBSS/DSC and RDS code into the specified area and then branches to Initialization Phase 1 (ARICIP1).

ARICIP1 - Initialization Phase 1. Establish the initial environment.

ARICIP2 - Initialization Phase 2. Complete the initialization.

ARICPRM - Parameter Processor. Process the SQL/DS parameters.

ARICTKN - Tokenizer Routine. Get a token from the parameter string (keyword = xxx).

ARICCRA - Create Agent Structure. Allocate and create the agent structure control blocks.

ARICENA - Enable Agent Structure. Makes an agent structure dispatchable. In VSE, for MPM, it will "connect the cross-partition link".

ARICMOD - Phase (no code) "loaded" by Batch Resource Manager to determine if the environment is MPM or SPM for VSE, or MVMM or SVMM for VM.

Storage Services Modules

ARICWSI - Initialize Working Storage. Allocate and initialize a working storage pool for an Agent Structure.

ARICWSG - Get Working Storage. Get working storage from the agent's pool.

ARICWSF - Free Working Storage. Return working storage to the agent's pool.

ARICWFR - Free Extended Working Storage Pools. Unconditionally free all extended working storage pools.

ARICSTK - Initialize Stack Storage. Allocate and initialize stack storage pool.

ARICGSE - Get Stack Extension. Allocate and initialize extensions to the stack storage pool.

ARICFSE - Free Stack Extension. Free an extension to the stack storage pool.

Operator Services Modules

ARICSHO - Displays users connected to SQL/DS (VM only). ARICSHO enables the operator to display the VM userid and the SQL id of the users connected to a real agent, waiting for a real agent, and inactive.

ARIYM01 - Establishes Supervisor linkage for operator command processing and issues SQL/DS initialization complete message.

ARISOCM - Entered due to VSE/Advanced Functions MSG operator command. Causes the operator agent to be 'waked up' and dispatched to perform operator command processing (VSE only).

ARIYM10 Processes all SQL/DS operator commands (system operator and ISQL). Prompts/reads command (system operator), tokenizes and validates command, routes it for command processing and issues error or completion message.

ARIYM11 - Routes operator command (system operator or ISQL) to command processing module. Validates name for SHOW commands.

Message Services Modules

ARIYM04 - Called (via MSG or RMSG macro) to retrieve, format and output a SQL/DS message for DBSS, DSC, RDS, and Batch and On-line Resource Managers. Supports prompting (WTOR) messages.

ARIYM05 - Retrieves (from message modules) and edits/formats SQL/DS message lines including translation (for example, binary to decimal) and substitution of message variables. Called by ARIYM04.

ARIYE13 - Routes display lines and messages. Routes to ISQL for ISQL command processing. Routes to ARISYSDH for output under control of the SQL/DS DSPLYDEV parameter, or to ARIYM50 for operator output, or to ARIYM51 for operator output with reply. Routing flags control output destination.

Agent Handling and Communication Modules

ARICDSP - SQL/DS Dispatcher. Dispatch agents, provide the operating system wait, and handle cross-partition (VSE) or Inter-User Communication Vehicle (IUCV) (VM) link termination.

ARICDWT - Dispatcher Wait Routine. Provide the DSC/DBSS linkage to the Dispatcher.

ARICDPT - Post an Agent Structure. Take an agent out of General or Lock Wait.

ARICCLA - Clean Up Agent Routine. Disconnect agent from XPTN or IUCV link. If SHUTDOWN issued, post the checkpoint agent when all agent structures are inactive.

ARICRST - Reset Storage Routine. Frees up storage used by an agent (stack storage, working storage, etc).

ARICENA - Enable Agent Structure. For VSE re-enable agent structure, "connect cross-partition link". For VM it will deallocate a "pseudo agent" from a real

agent and allocate a "waiting pseudo agent" to a real agent whenever a real agent completes a LUW.

ARICWAT - Cross-partition Wait Routine (RDS). Provide the RDS linkage to the Dispatcher for cross-partition (VSE) or inter-user communication vehicle (VM) wait. It will also drive the "rollback" process (backout).

ARICMUD - Cross-partition (VSE) or inter-user communication vehicle (VM) Message Receive Routine (RDS). Receive data (message) sent by the Resource Managers.

ARICSPM - Single-Partition Mode Linkage Routine. For VSE, load (CDLOAD) the user application, GENCAT, ADDSEG, DBS Utility, etc, and transfer control to it (BALR 14,15). In VM, the routine has been loaded, it only invokes the routine.

ARICOMB - Output Mailbox Processor (RDS). Put the reply (message) into an output mailbox and send it (REPLY) to the Resource Manager(s).

ARICIMB - Input Mailbox Processor (Resource Manager). Build the input mailbox for the Resource Manager(s). Input mailbox to be sent (SENDR) to the SQL/DS partition.

ARICMMB - Move Data Routine (Resource Manager). Move data to the user application area.

Trace Services Modules

ARICTRI - Called if TRACDBSS or TRACRDS parameters are specified. ARICTRI prompts the operator to ready trace tape, opens trace output file and enables SQL/DS tracing.

ARICTRC - Called to process the TRACE operator command to enable or disable SQL/DS tracing. For TRACE ON, ARICTRC prompts for various tracing options. Entry point ARICTRC1 is called to shut down/close trace during SQL/DS termination.

ARIXETR - Called (via XTRACE macro) to process an RDS trace point and generate and write trace output.

ARIYE14 - Called (via TRACE macro) to process a DBSS trace point and generate and write trace output.

ARIYE01 - Called by ARIYM00 if DBSS entry or DBSS exit tracing is active. Produces level 1 trace output (via TRACE macro) or calls (if it exists) level 2 trace module for the DBSS operation code.

System-Dependent Routines

ARICCOM - Communication Manager. Provide the VSE Cross-Partition Communication or VM Inter-User Communication Vehicle interface for SQL/DS.

ARICINT - SQL/DS Interrupt Handler. Handles external interrupts which occur as the result of using the IUCV.

ARISYSD - Provides a system-dependent interface to either the various VSE functions or VM functions used by SQL/DS. It contains multiple entry points (described in Section 3, Program Organization starting on page 512).

ARISD00D - VSE/Advanced Functions Root Phase for I/O Modules. Used to determine what type of SAM file processing is required and loads and invokes the appropriate I/O module (ARISDnn). This module performs SAM file processing for SYSIPT, SYSLST, and SYSPCH files. Invoked via entry point ARISYSD5 in module ARISYSDD.

ARISD01D - VSE/Advanced Functions Source Statement Library Read Access Routine. Used to find and read a member in the Source Statement Library. Invoked via entry point ARISYSD8 in module ARISYSDD.

ARISYSE - SQL/DS VM/SP System-Dependent Routine. Provides SAM file processing (including work files for PREP) for all SQL/DS components except ISQL. It is always invoked via the entry point ARISYSD5. (Also see page 515.)

ARISYSF - SQL/DS VM/SP System-Dependent Routine. It is the counterpart to VSE/Advanced Functions module ARISD01D. Used to find and read a "card image" CMS file. It is always invoked via entry point ARISYSD8. (Also see page 515.)

ARISYSG - SQL/DS VM/SP System-Dependent Routine. Called during SQL/DS initialization and termination to display information such as messages and mini-dumps to the virtual machine terminal. It exists because of the VSE/Advanced Functions SQL/DS DSPLYDEV parameter, which is ignored by VM/SP SQL/DS. It is always invoked via entry point ARISYSDH. (This module is also briefly described on page 515.)

DSC Termination Modules

ARICSHT - SQL/DS Shutdown Routine. Handle the system operator SHUTDOWN command (NORMAL, ARCHIVE, or QUICK).

ARICTRM - SQL/DS Termination Routine. "Close" SQL/DS and return to the operating system (BR 14).

ARICABE - SQL/DS ABEND Handler. VSE "STXIT AB" or VM DMSABN routine to handle abnormal termination and program check conditions.

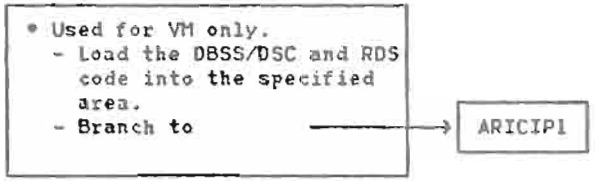
ARICDMP - SQL/DS Dump Routine. "Dump" selected areas of the SQL/DS partition (VSE) or SQL/DS virtual machine (VM) to the printer and/or console.

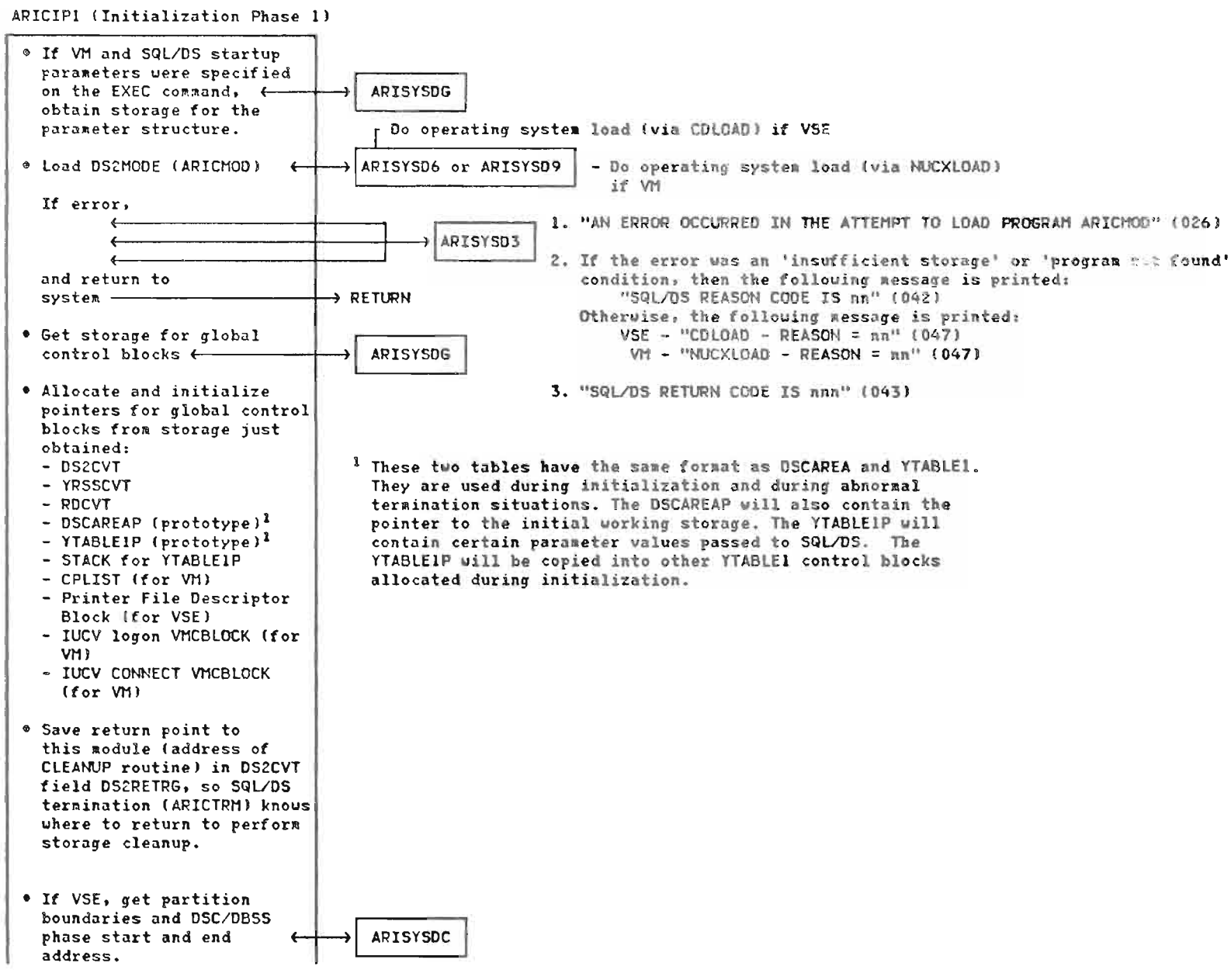
ARICPDM - SQL/DS Snap Dump Routine. VSE "PDUMP" the entire SQL/DS partition or VM "CP DUMP" the entire virtual machine, or the partition/virtual machine minus the the major phases/programs (DBSS/DSC, RDS, Batch Resource Manager, etc.).

ARIYM02 - SQL/DS System Error Routine. Called by SQL/DS routines to simulate an "ABEND" condition.

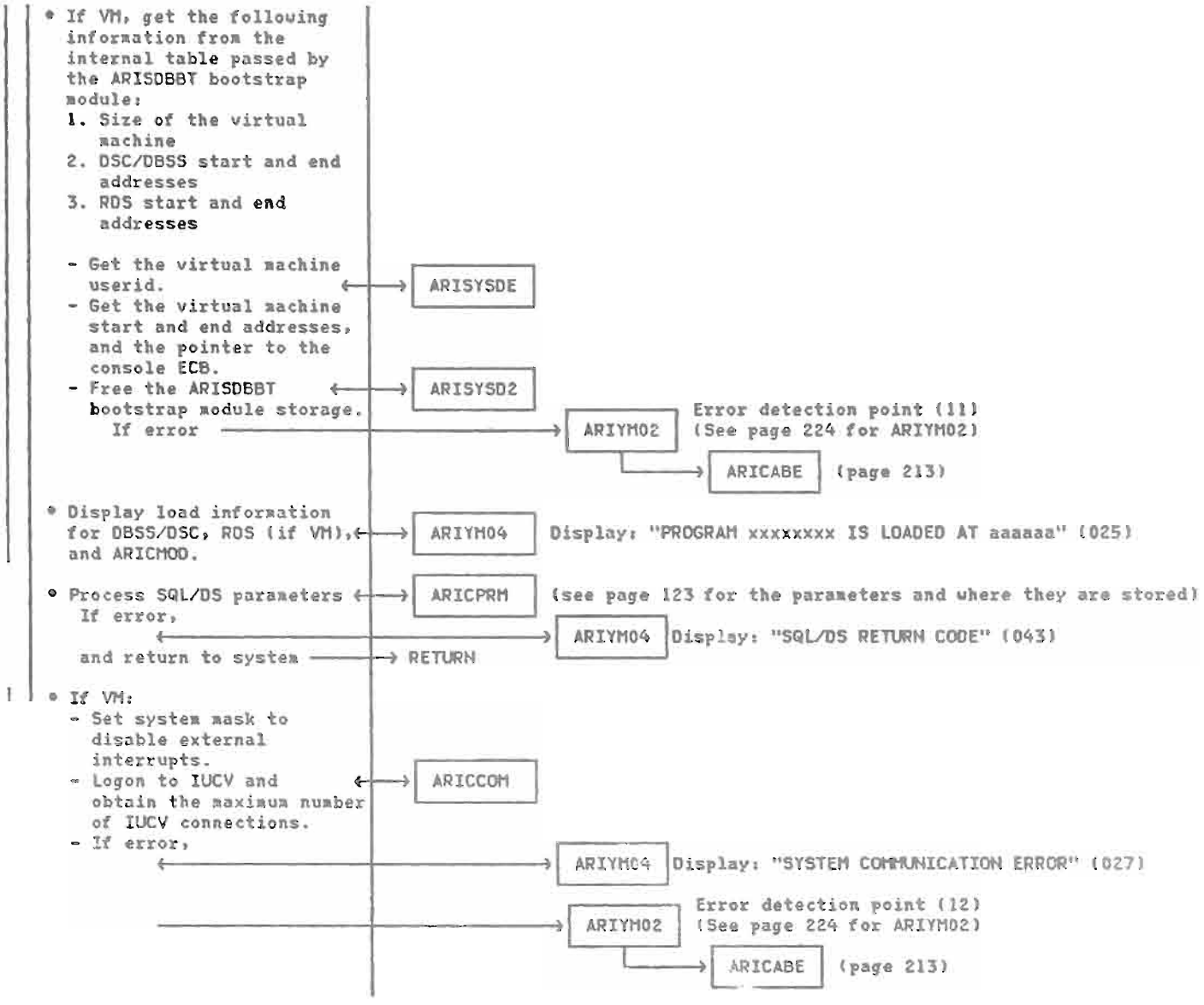
DSC INITIALIZATION

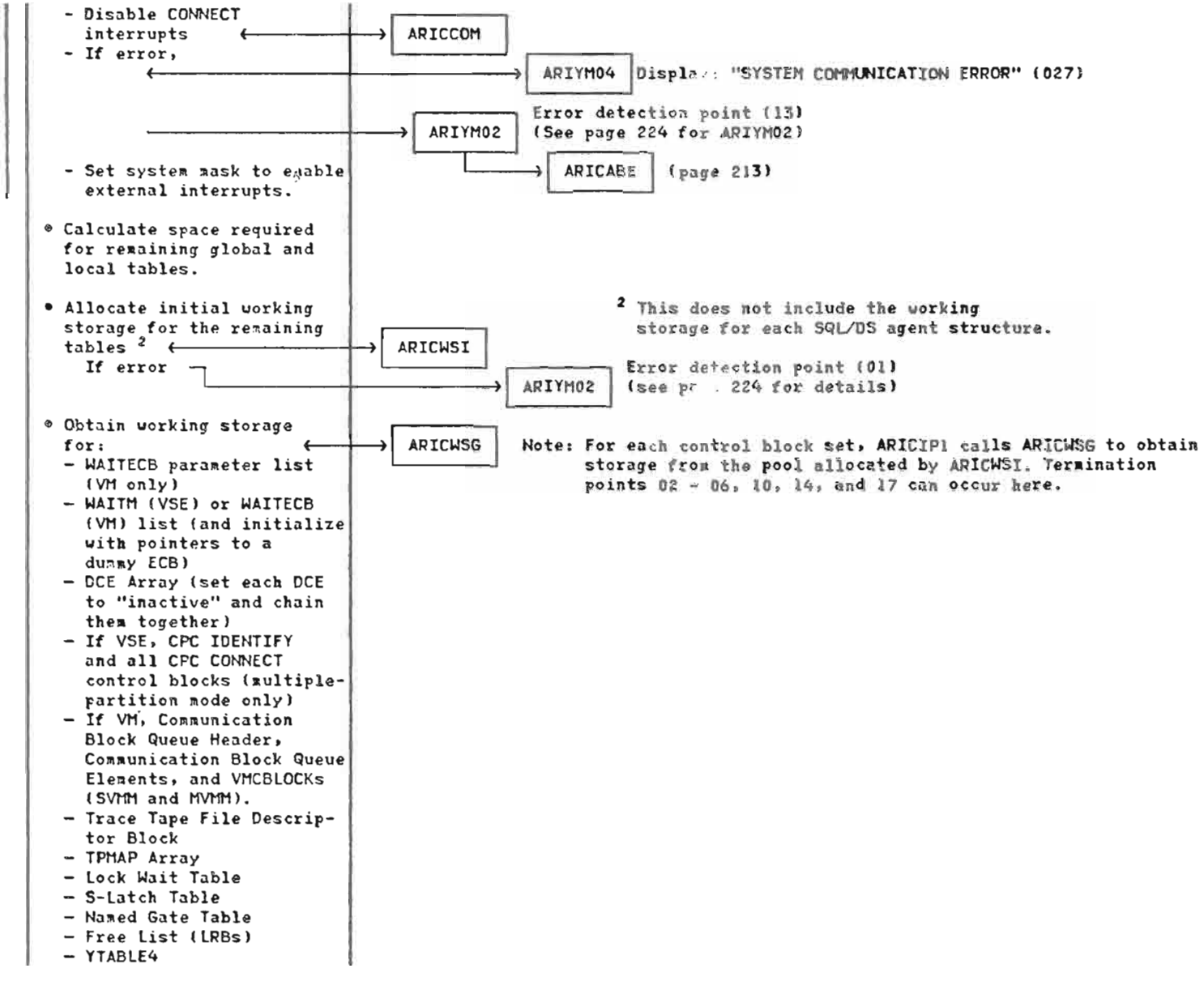
ARISDBBT - Bootstrap module ('GENMODed' version of ARISDBKC)



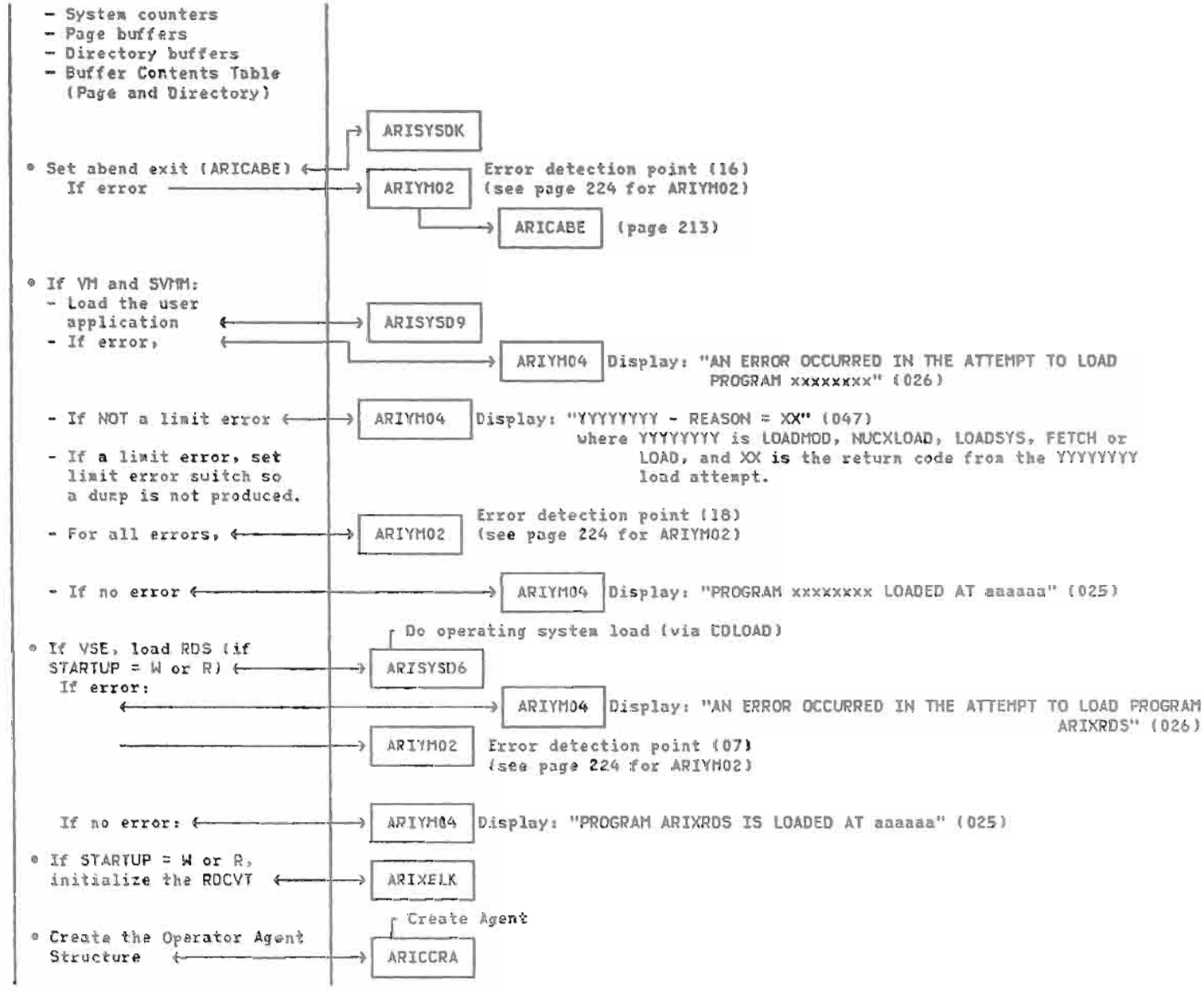


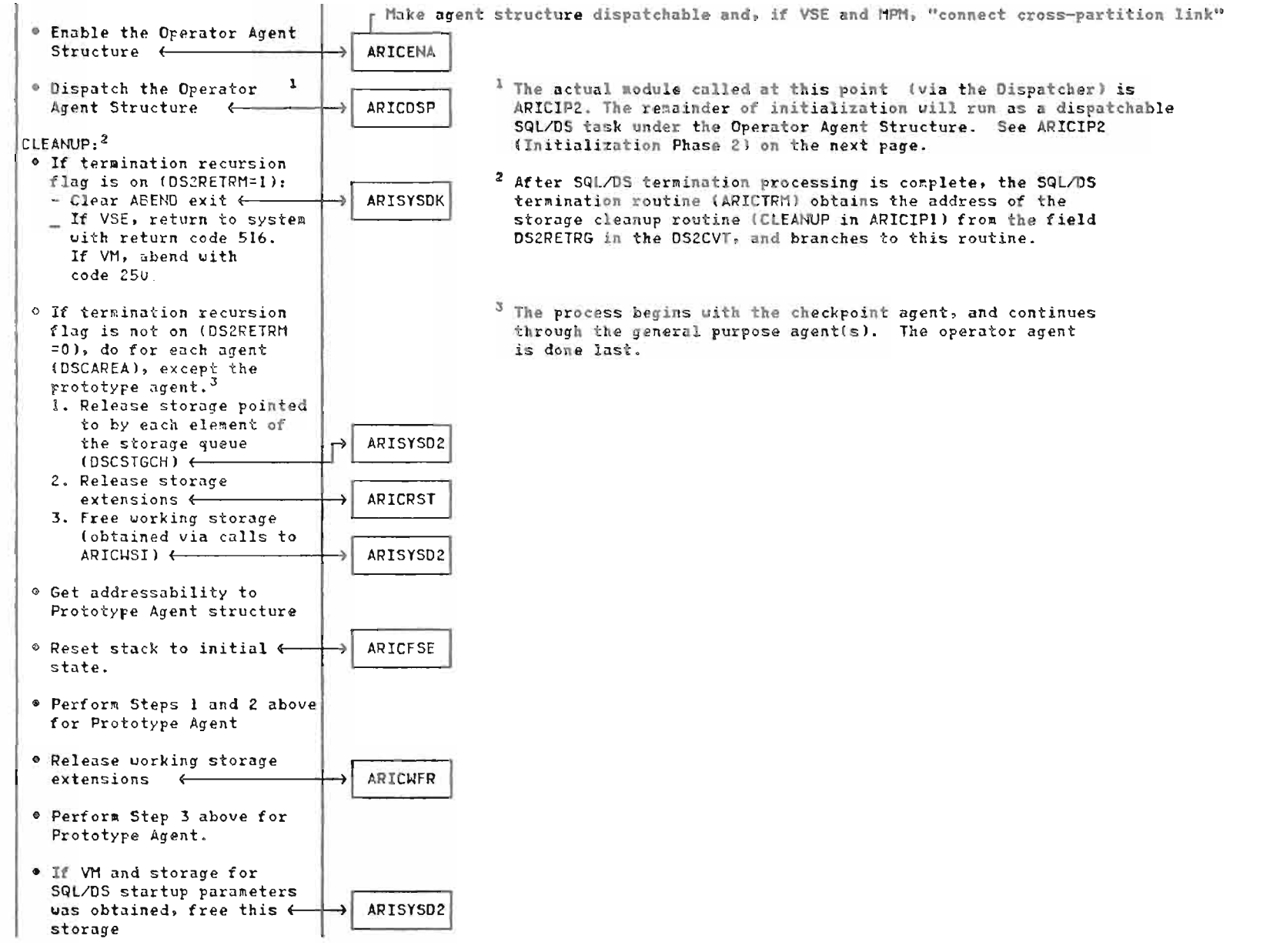
DSC Initialization (continued)





DSC Initialization (continued)



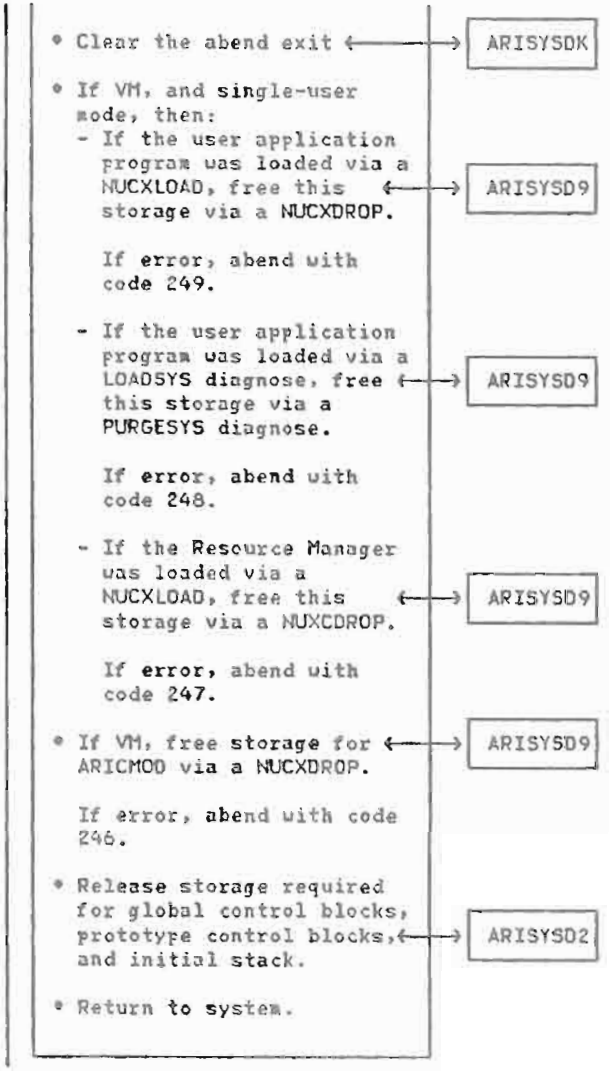


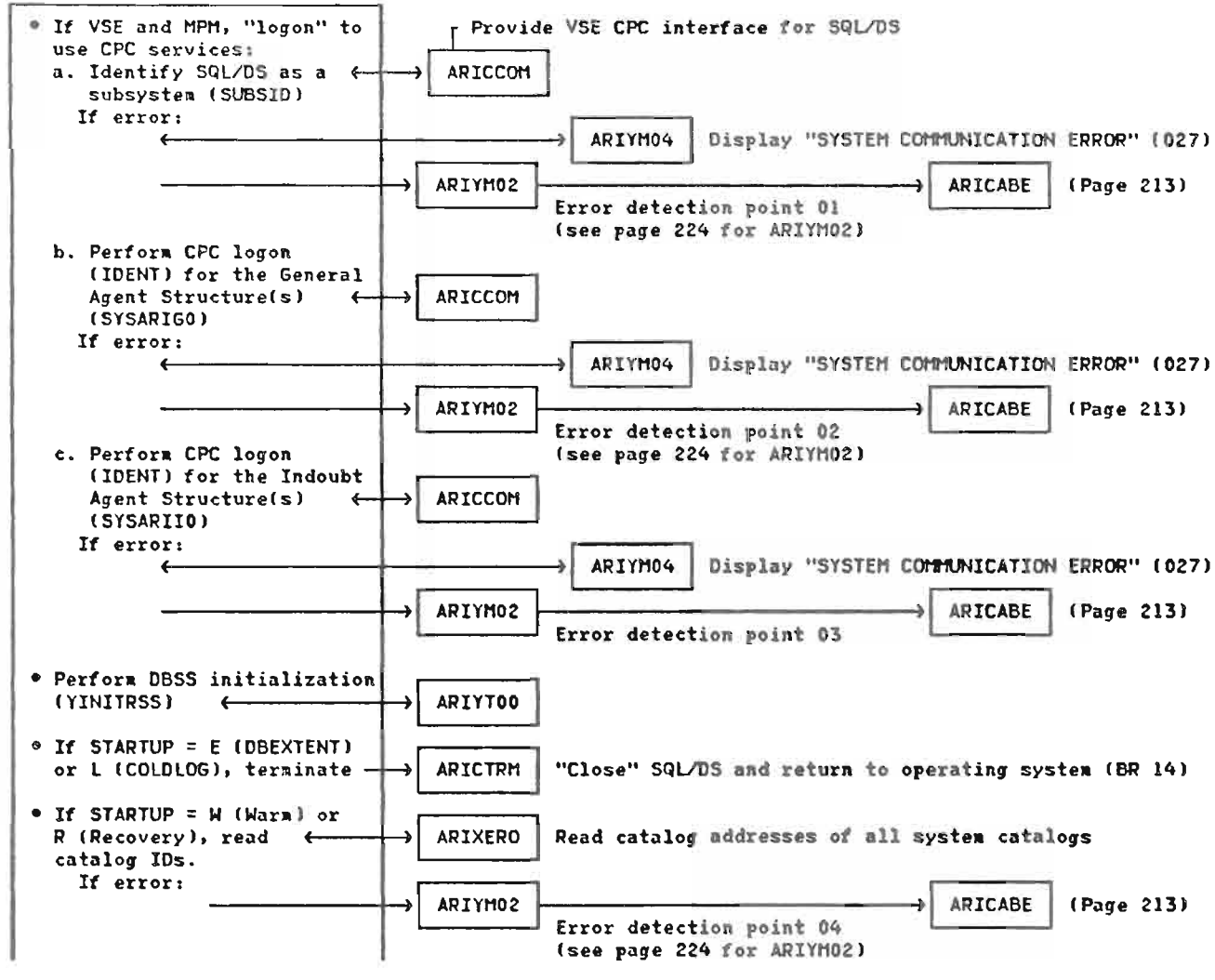
[1] The actual module called at this point (via the Dispatcher) is ARICIP2. The remainder of initialization will run as a dispatchable SQL/DS task under the Operator Agent Structure. See ARICIP2 (Initialization Phase 2) on the next page.

[2] After SQL/DS termination processing is complete, the SQL/DS termination routine (ARICTRM) obtains the address of the storage cleanup routine (CLEANUP in ARICIP1) from the field DS2RETRG in the DS2CVT, and branches to this routine.

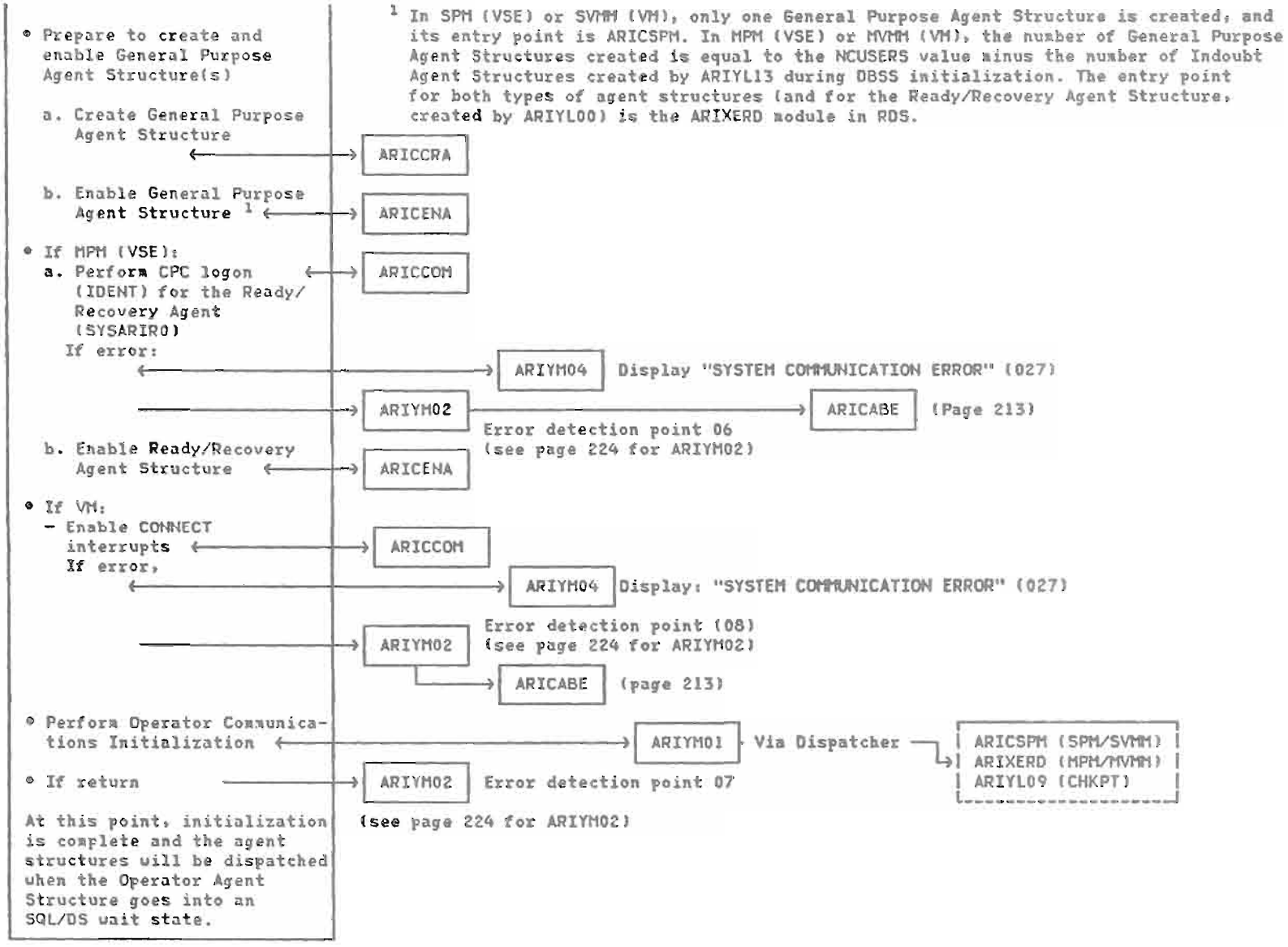
[3] The process begins with the checkpoint agent, and continues through the general purpose agent(s). The operator agent is done last.

DSC Initialization (continued)



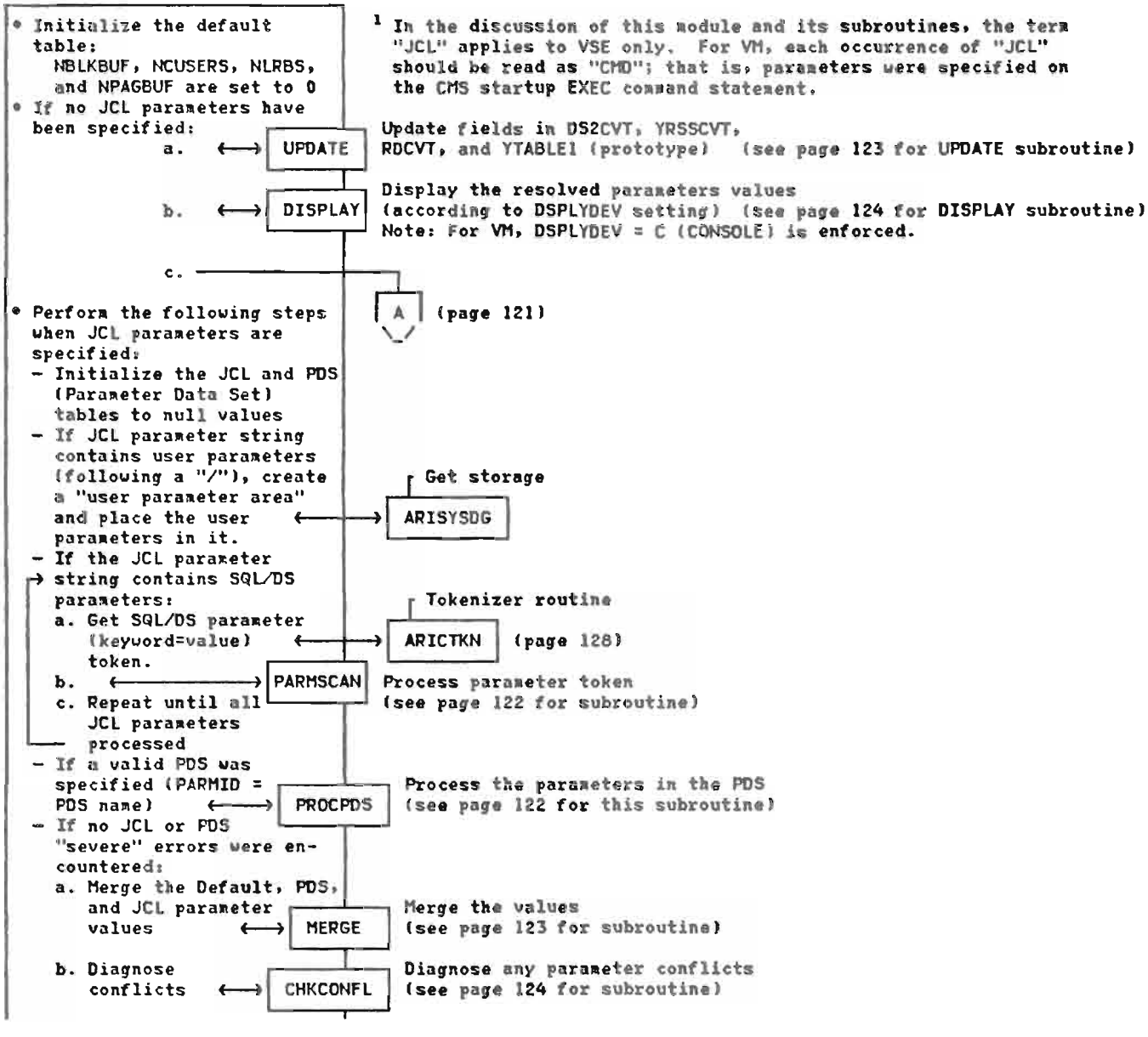


DSC Initialization (continued)



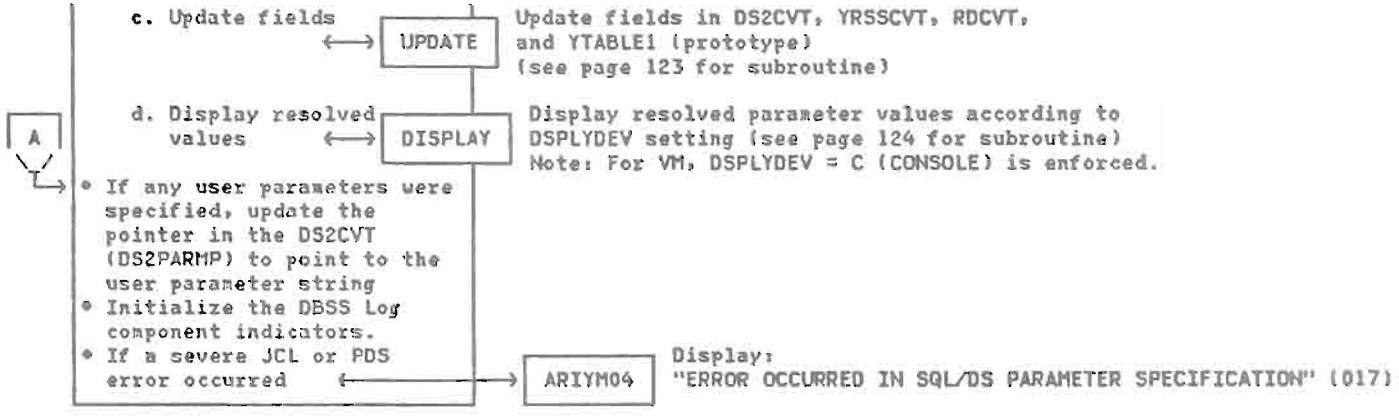
[1] In SPM (VSE) or SVMM (VM), only one General Purpose Agent Structure is created, and its entry point is ARICSPM. In MPM (VSE) or MVMM (VM), the number of General Purpose Agent Structures created is equal to the NCUSERS value minus the number of Indoubt Agent Structures created by ARIYL13 during DBSS initialization. The entry point for both types of agent structures (and for the Ready/Recovery Agent Structure, created by ARIYL00) is the ARIXERD module in RDS.

ARICPRM (Process SQL/DS Parameters)[1]

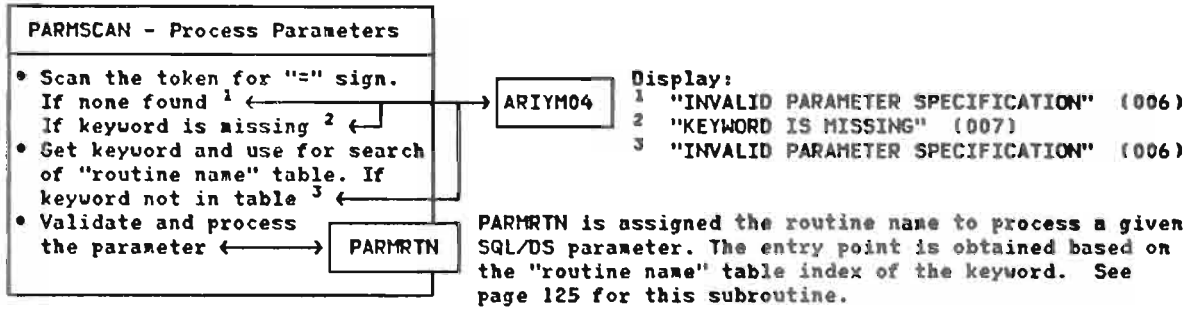


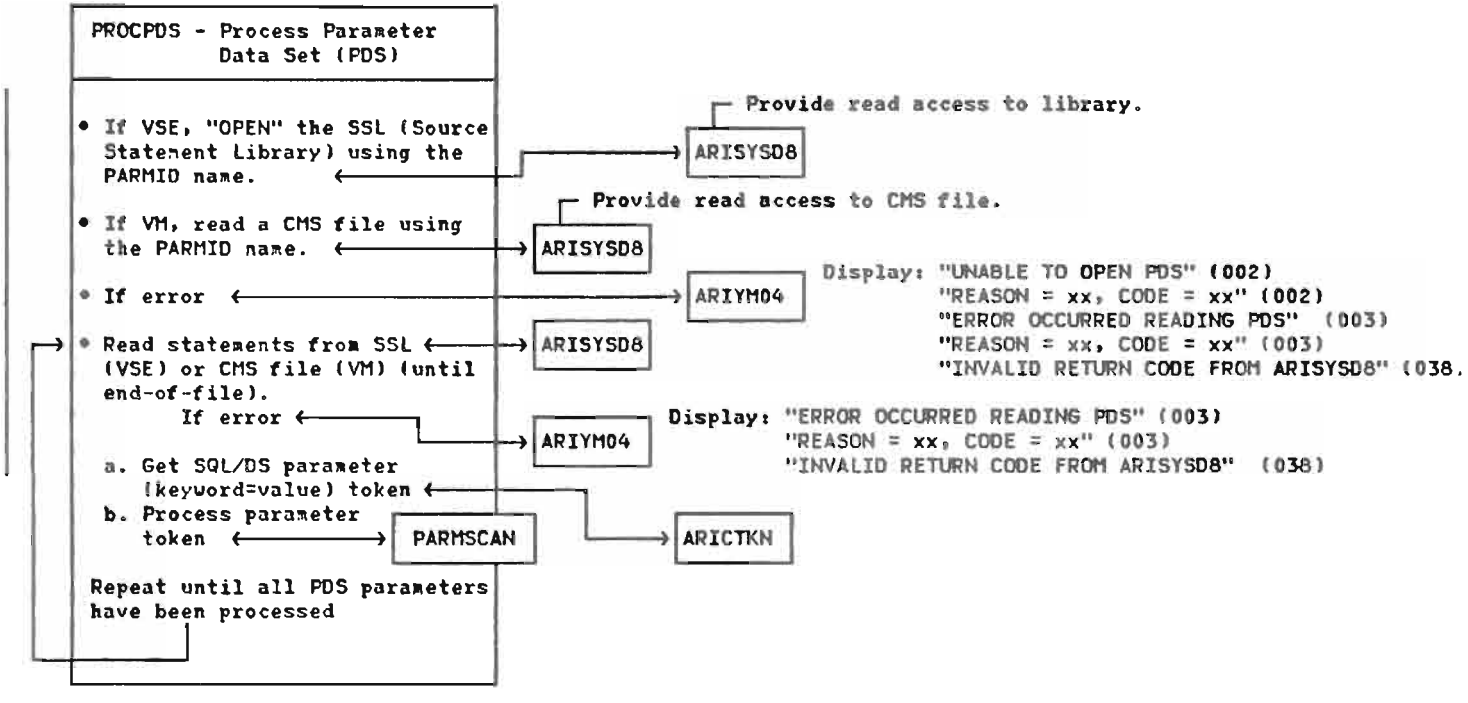
[1] In the discussion of this module and its subroutines, the term "JCL" applies to VSE only. For VM, each occurrence of "JCL" should be read as "CMD"; that is, parameters were specified on the CMS startup EXEC command statement.

DSC Initialization (continued)



Note: JCL parameters are processed before the PDS parameters and override the PDS parameters. Therefore, PDS errors will be diagnosed, but they will be ignored if there were corresponding JCL parameters. It is assumed that the JCL parameters correct the PDS parameter errors.





MERGE - Merge Default, PDS, and JCL Parameters

- Move PDS parameter values into Default Table
- Move JCL parameter values into Default Table
- Set "latent" defaults for NCUSERS, NLRBS, NPAGBUF, and NDIRBUF if not specified in JCL or PDS

UPDATE - Update Tables [1]

- Move updated values in the Default Table (after merge of JCL and PDS) into the DS2CVT, YRSSCVT, RDCVT, and YTABLE1 (prototype)
- If TRACE parameters were specified, set trace open required switch (DS2TROPN)
- Set internal SQL/DS defaults
- Calculate the Named Gate Hashing Value (NNAMGATE). It is the first prime number greater than (NCUSERS * NLRBU) / 4.

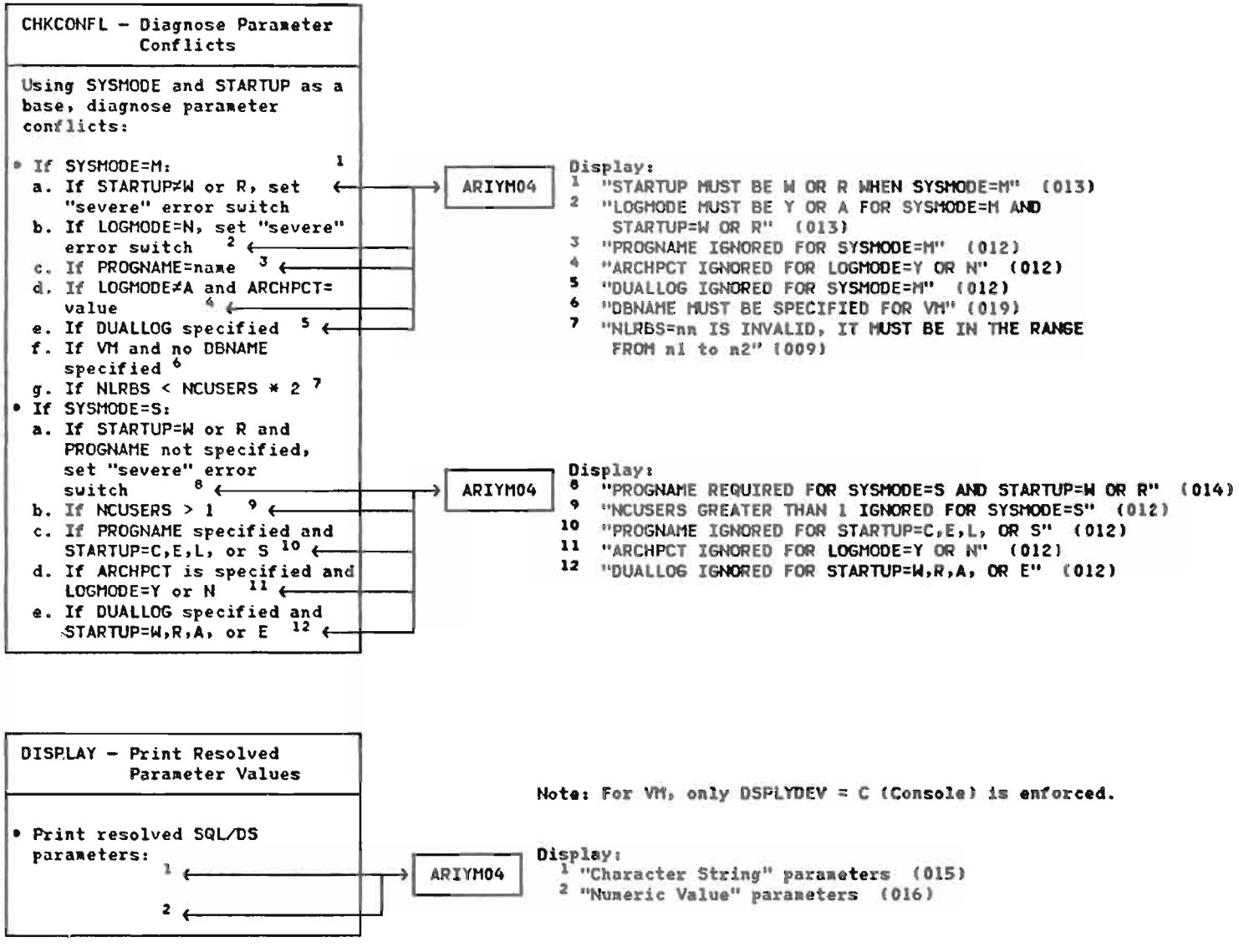
[1] See "Parameter Values - Where Stored" following this diagram for a list of the parameter values and where they are stored.

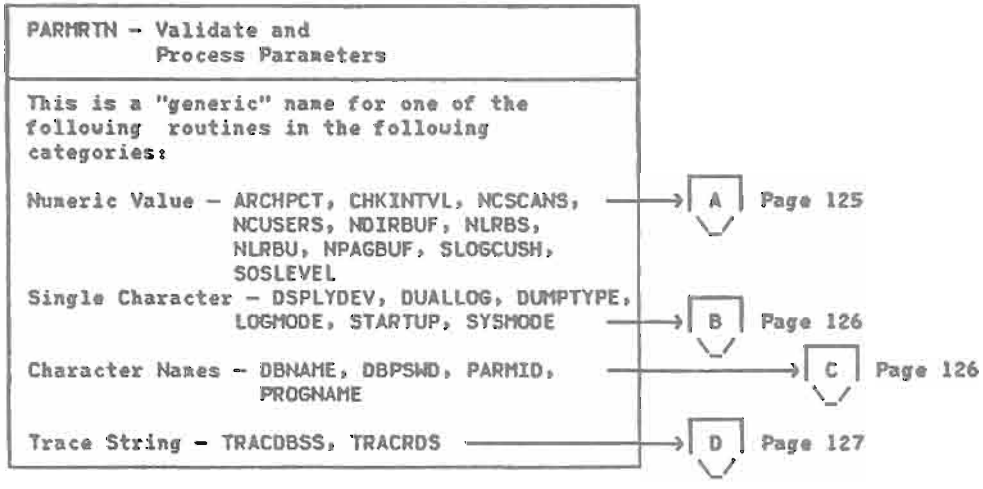
Parameter Values - Where Stored

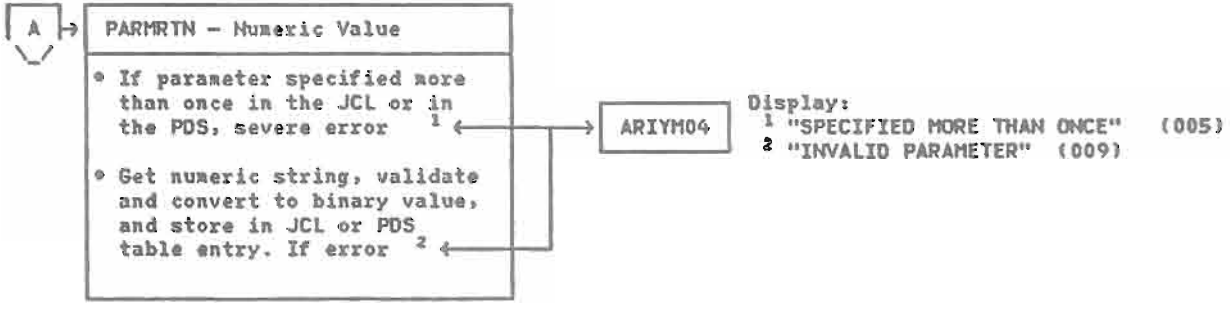
Parameter	Field	Control Block
ARCHPCT	ARCHPCT	YRSSCVT
CHKINTVL	AUTOCNT, NPAGSCHK	YRSSCVT
DBNAME	YRSDBNME	YRSSCVT
DBPSWD	DBPSWD	YRSSCVT
DSPLYDEV	DS2DSPDV	DS2CVT
DUALLOG	DUALLOG	YRSSCVT
DUMPTYPE	DS2DUMPT	DS2CVT
LOGMODE	LOGMODE (LOGARCH)	YRSSCVT
NCSCANS	NCSCANS	YRSSCVT
NCUSERS	NTRANS + number of system agents (number of system agents is 2 if SUM or 3 if MUM)	YRSSCVT
NDIRBUF	YTABNBB	YRSSCVT

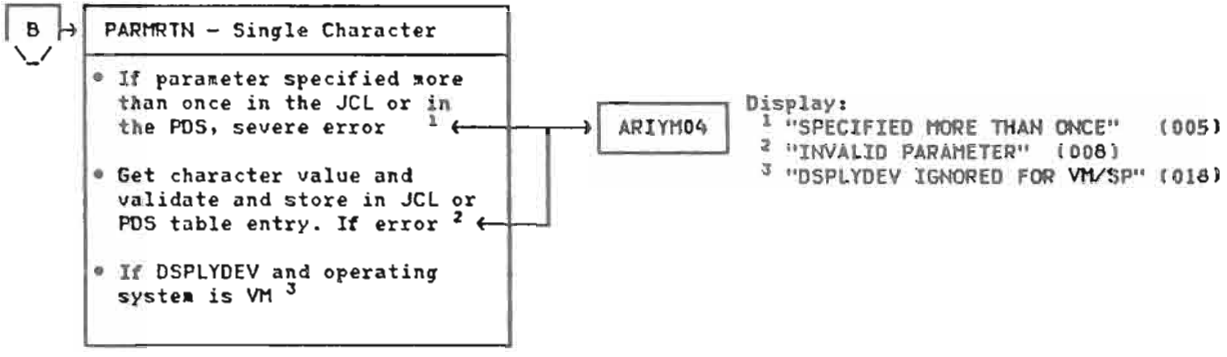
Parameter	Field	Control Block
NLRBS	NLRBS	YRSSCVT
NLRBU	NLRBU	YRSSCVT
NPAGBUF	YTABNPB	YRSSCVT
PARMID	Handled internally by ARICPRM	
PROGNAME	PROGNAME	YRSSCVT
SLOGCUSH	SLOGCUSH	YRSSCVT
SOSLEVEL	YSOSLEV	YRSSCVT
STARTUP	YRSSTART	YRSSCVT
SYSMODE	SYSMODE	YTABLE1 (Proto)
TRACDBSS	YT1TRAC	YTABLE1 (Proto)
TRACRDS	RDCTRAC(RDATRAC)	RDCVT (RDAREA) *

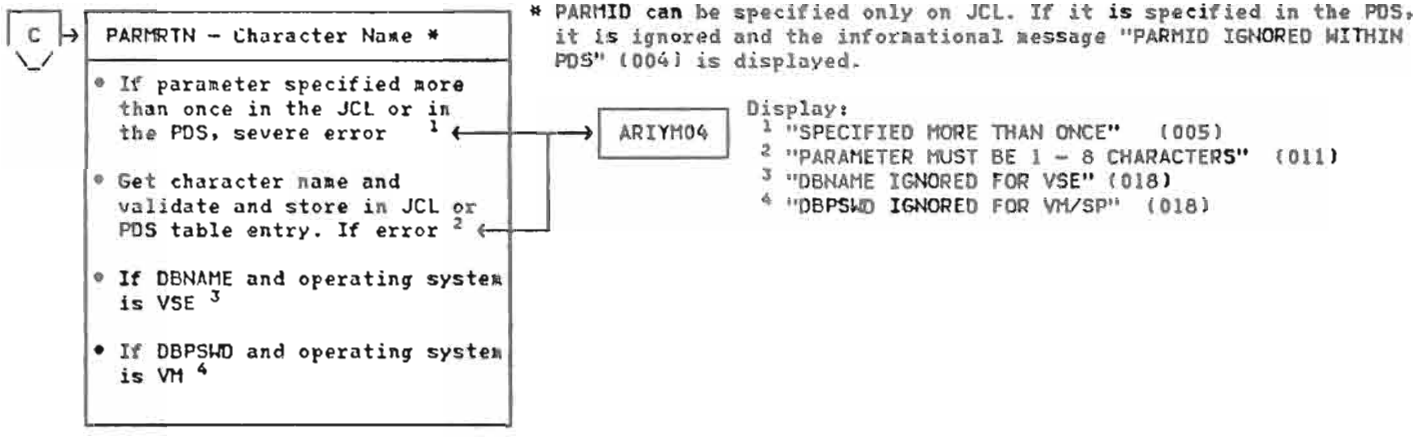
* The TRACRDS values are temporarily stored in the RDCVT. When the agent structure is created, these values are moved from the RDCVT into the RDAREA.




PARMRTN - Validate and
Process Parameters
This is a "generic" name for one of the following routines in the following categories:
Numeric Value - ARCHPCT, CHKINTVL, NCSCANS, NCUSERS, NDIRBUF, NLRBS, NLRBU, NPAGBUF, SLOGCUSH, SOSLEVEL
A
Page 125
Single Character - DSPLYDEV, DUALLOG, DUMPTYPE, LOGMODE, STARTUP, SYSMODE
B
Page 126
Character Names - DBNAME, DBPSWD, PARMID, PROGNAME
C
Page 126
Trace String - TRACDBSS, TRACRDS
D
Page 127

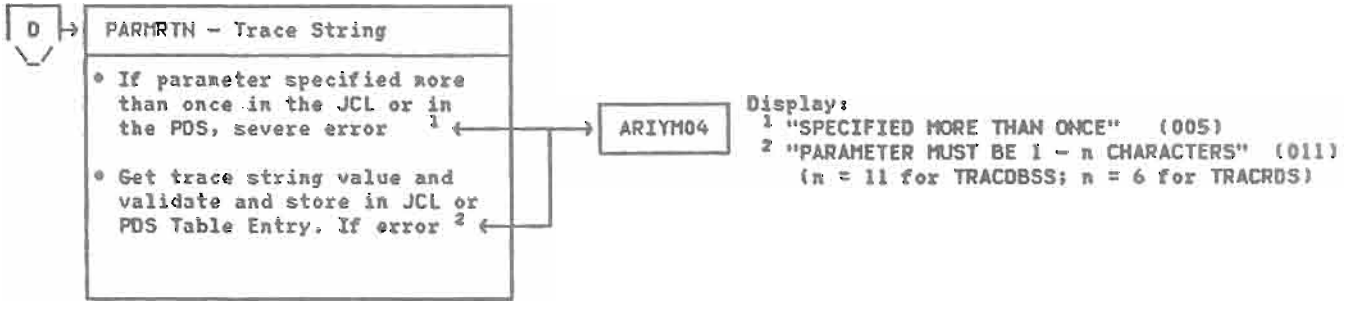

A
PARMRTN - Numeric Value
• If parameter specified more than once in the JCL or in the PDS, severe error 1
• Get numeric string, validate and convert to binary value, and store in JCL or PDS table entry. If error 2
ARIYM04
Display:
1 "SPECIFIED MORE THAN ONCE" (005)
2 "INVALID PARAMETER" (009)



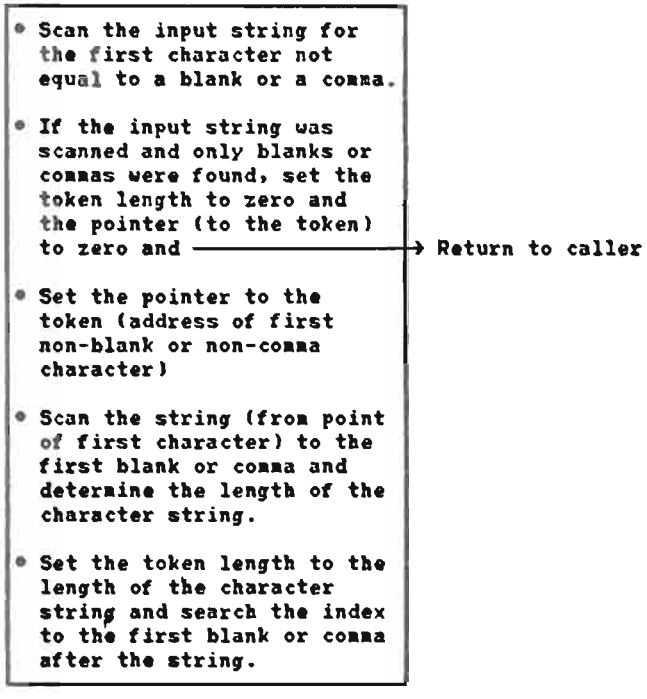


* PARMID can be specified only on JCL. If it is specified in the PDS, it is ignored and the informational message "PARMID IGNORED WITHIN PDS" (004) is displayed.

DSC Initialization (continued)

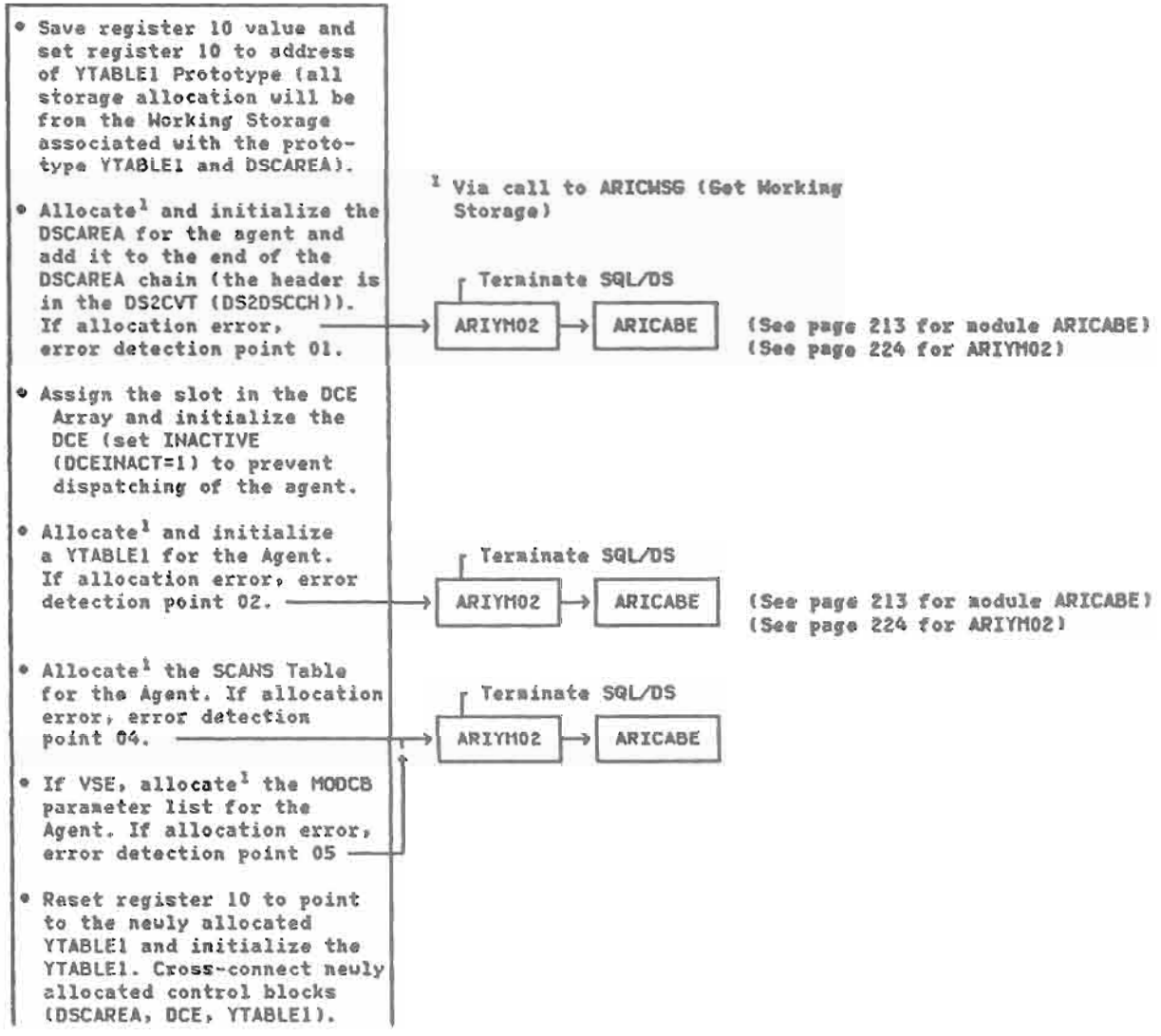


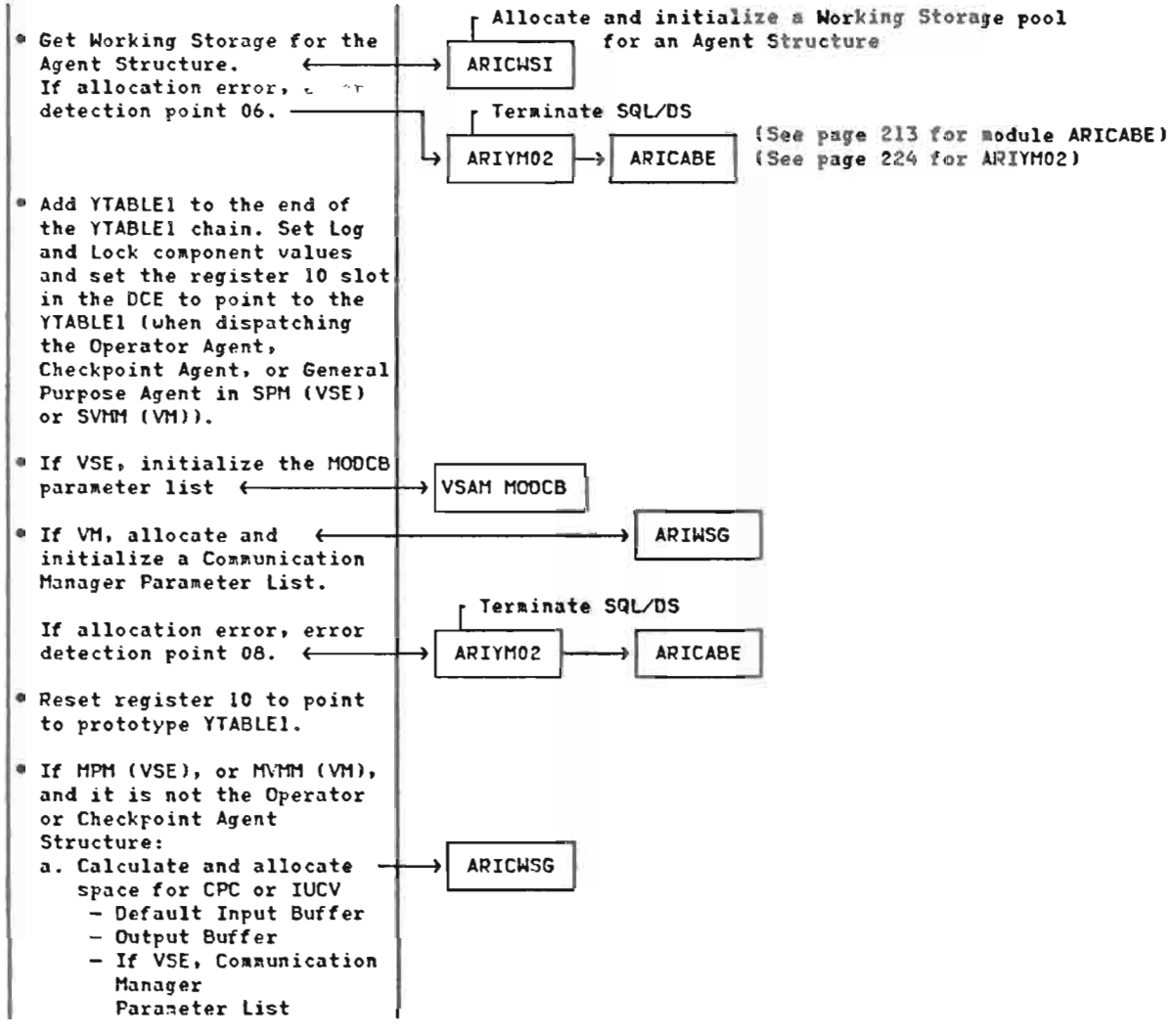
ARICTKN (Tokenizer)



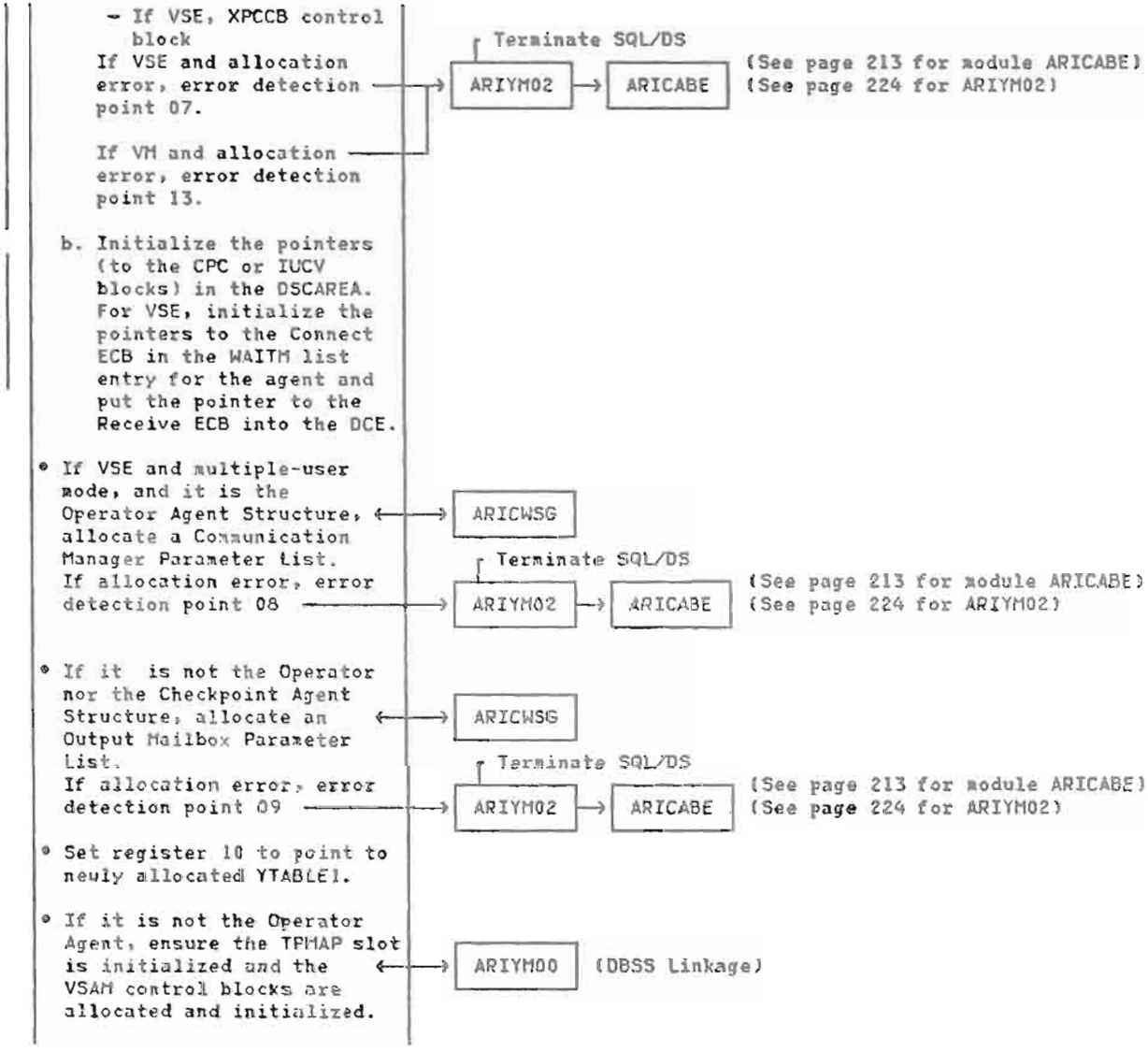
DSC Initialization (continued)

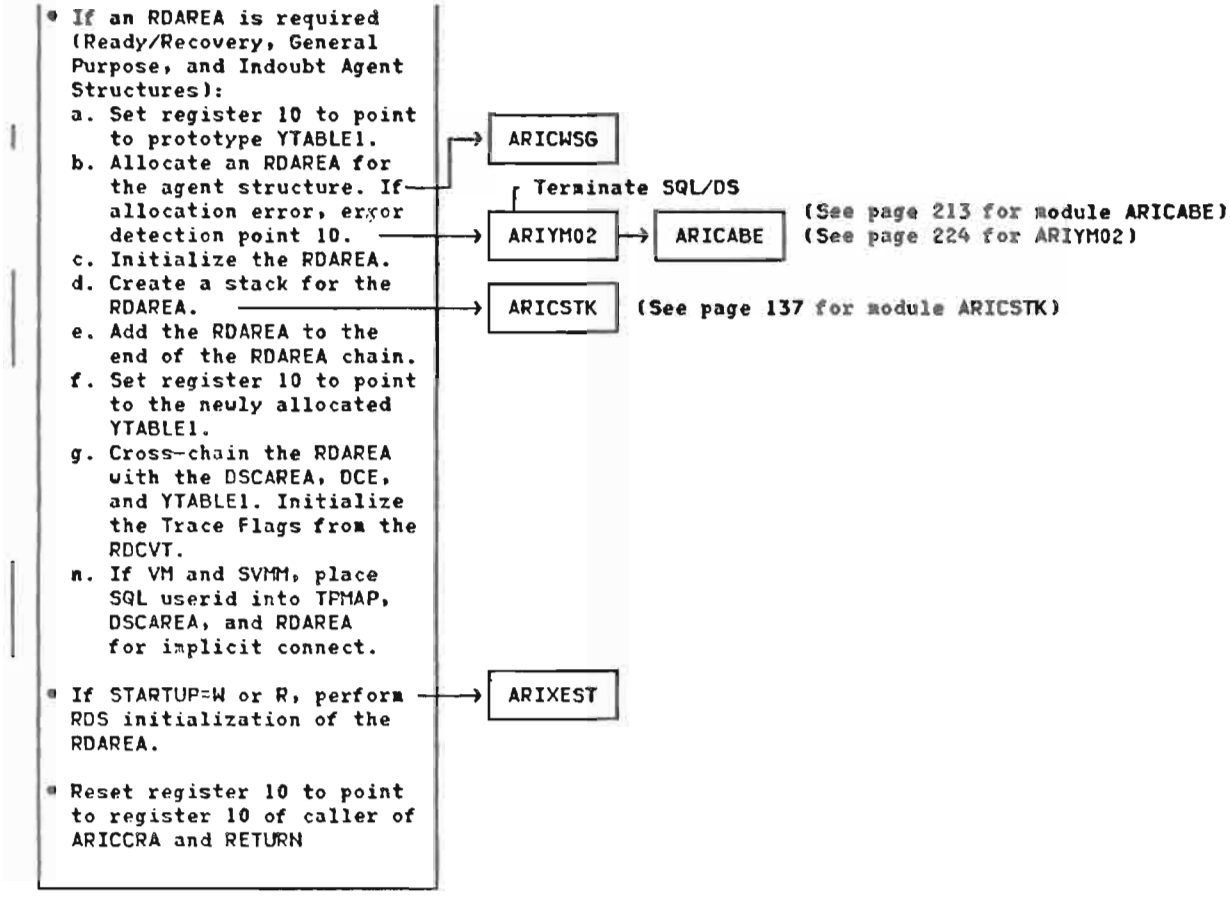
ARICCRA (Create Agent Structure)





DSC Initialization (continued)





ARICENA (Enable Agent Structure)

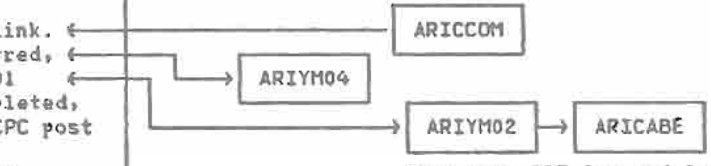
- Initialize the DCE, ensuring that the DCE is dispatchable and the link flags are cleared. Set register 13 to point to save area to be used by agent when initially dispatched (the save area is the DSCAREA (DSCAREA (DSCDBSSA)). Set registers 14 and 15 to the entry point of the agent (14 is used as the return point from the Dispatcher and 15 is used as the initial base register by the entry point of the agent).

- If the entry point is in RDS (ARIXERD), set the register 10 slot to point to the RDAREA.

- For VSE:
  - If the Cross-Partition Communication (CPC) links are to be connected:
    a. Initialize the Communication Manager Parameter List to connect the correct link (General Purpose, Indoubt, or Ready/Recovery).
    b. Issue a CONNECT for the CPC link.
       - If a severe CPC error occurred, and error detection point 01
       - If the connect was not completed, set the DCE waiting for a CPC post to occur (DCEXPTN=1).
       - If the connect was completed, "unpost" the connect ECB and set the DCE connect flag (DCECNCT=1).

- For VM:
  - If the pseudo agent is to be detached from the real agent (PFDSCPA=1), then perform the following:
    (Establish addressability to the pseudo-agent control blocks. If the IUCV communication link has been disconnected (SEVERed) either by the user (VMCSEVCD ≠ 0) or by SQL/DS abnormal termination processing (PFSQLAB = 1), then perform Steps 1 - 5.
    1. Remove pseudo agent from in-use pseudo agent queue.
    2. Add pseudo agent to start of available pseudo agent queue.
    3. Increment available pseudo agent counter.

If this routine is called by the SQL/DS Dispatcher and the CPC or IUCV link is to be reset (not disconnected/reconnected), then the Dispatcher will set DCECNCT=1 (in the link flag byte DCELFLAG).



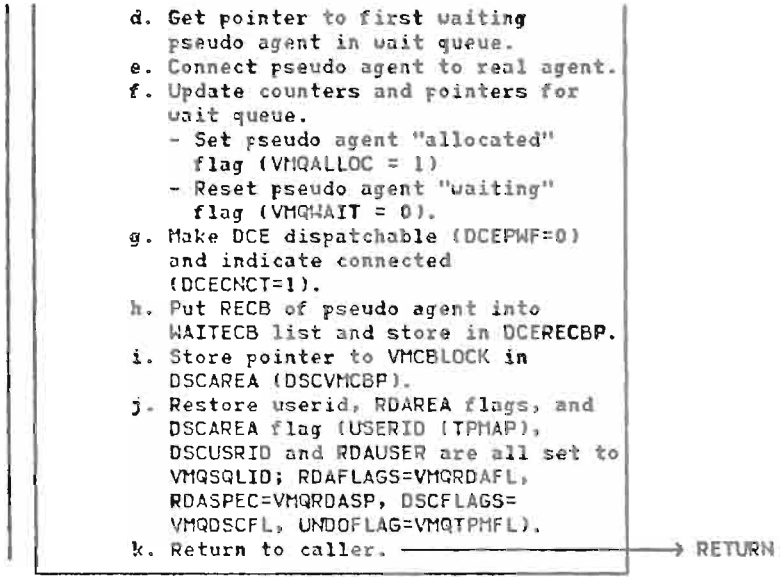
(See page 213 for module ARICABE)
(See page 224 for ARIYM02)

(This does not occur at initialization time. It is part of the Dispatcher/Agent Handling process.)

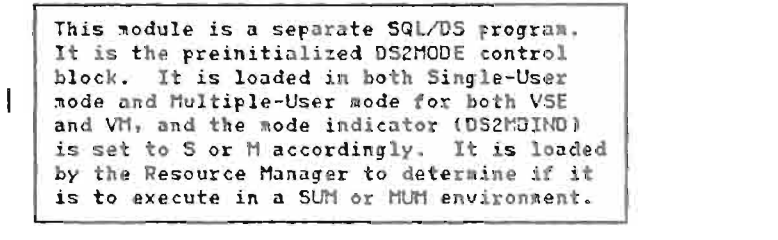
4. Decrement in-use pseudo agent counter.
5. Set the pseudo agent available flag (VMQAVAIL = 1) and reset the in-use and allocated flags (VMQINUSE = 0 and VMQALLOC = 0).
6. If waiting pseudo agents (VMHCWTPA¬=0) or current user has not sent another message (RECB not posted), then disconnect pseudo agent (VMCADDR=0) from real agent (DCEADRPA=0) and clear pseudo agent allocated flag (VMQALLOC=0). Otherwise, return.[1]
7. If there are no waiting pseudo agents:
   a. Remove real agent from in-use real agent queue.
   b. Place real agent at the top of the available queue.
   c. Increment available real agent counter.
   d. Decrement in-use real agent counter.
   e. Ensure DCE not dispatchable (DCEINACT = 1), not connected (DCECNCT = 0), and that a 'posted' ECB is not in the WAITECB list.
   f. Save the userid (RDAUSER) in VMQSQLID, the RDAREA flags (VMQRDAFL=RDAFLAGS, VMQRDASP=RDASPEC), the DSCAREA flag (VMQDSCFL=DSCFLAGS), and the TPMAP flag (VMQTPMFL=UNDOFLAG).
   g. Return to caller. ——→ RETURN
8. Else, if there are any waiting pseudo agents:
   a. If the current user sent another message (RECB posted) then:
      - Move current pseudo agent to bottom of wait queue.
      - Increment waiting pseudo agent counter (VMHCWTPA).
      - Set waiting flag in current pseudo agent (VMQWAIT=1).
   b. Save the userid (RDAUSER) in VMQSQLID, the RDAREA flags (VMQRDAFL=RDAFLAGS, VMQRDASP=RDASPEC), the DSCAREA flag (VMQDSCFL=DSCFLAGS), and the TPMAP flag (VMQTPMFL=UNDOFLAG).
   c. Allocate first waiting pseudo agent to real agent.

[1] (This is the case where there are no waiting pseudo agents and the current user has sent another message. The current user retains ownership of the real agent for another logical unit of work.)

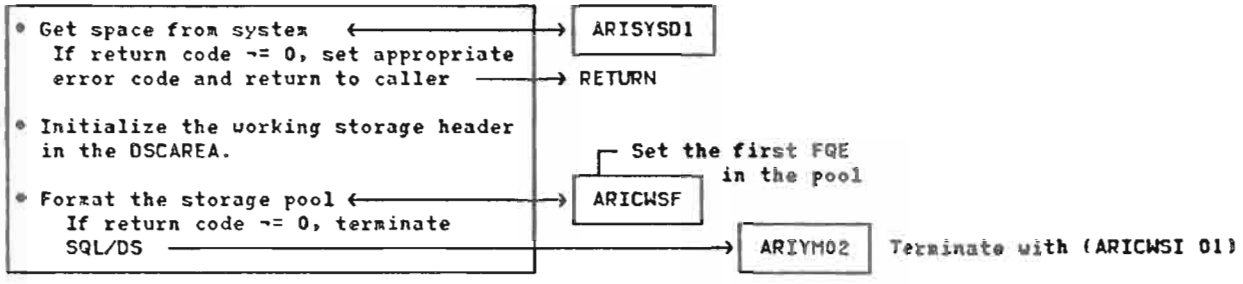
DSC Initialization (continued)



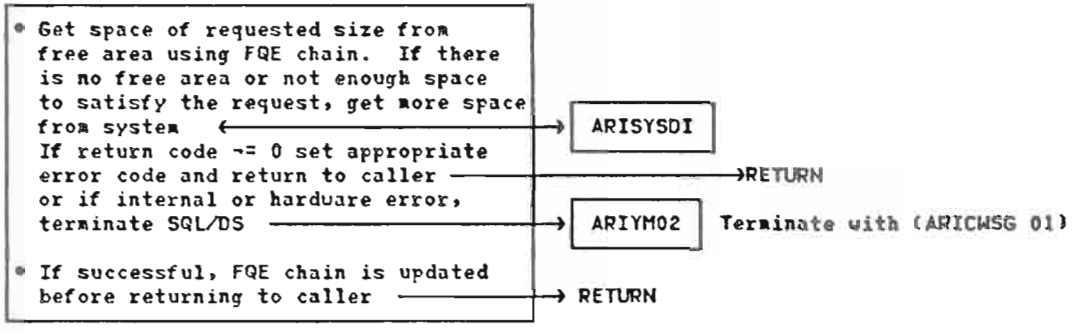
ARICMOD (SQL/DS Mode Indicator)



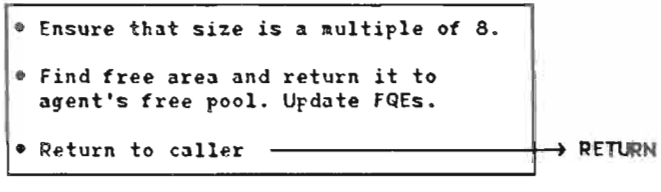
* From ARICIP1 and ARICCRA
ARICWSI (Initialize Working Storage)



* From many modules, including ARICIP1 and ARICCRA (see Section 6, "Module-to-Module Cross Reference", in Volume 2 for a list of calling modules)
ARICWSG (Get Working Storage from Agent's Pool)



* From many modules (see Section 6, "Module-to-Module Cross Reference", in Volume 2 for a list of calling modules)
ARICWSF (Free Storage)

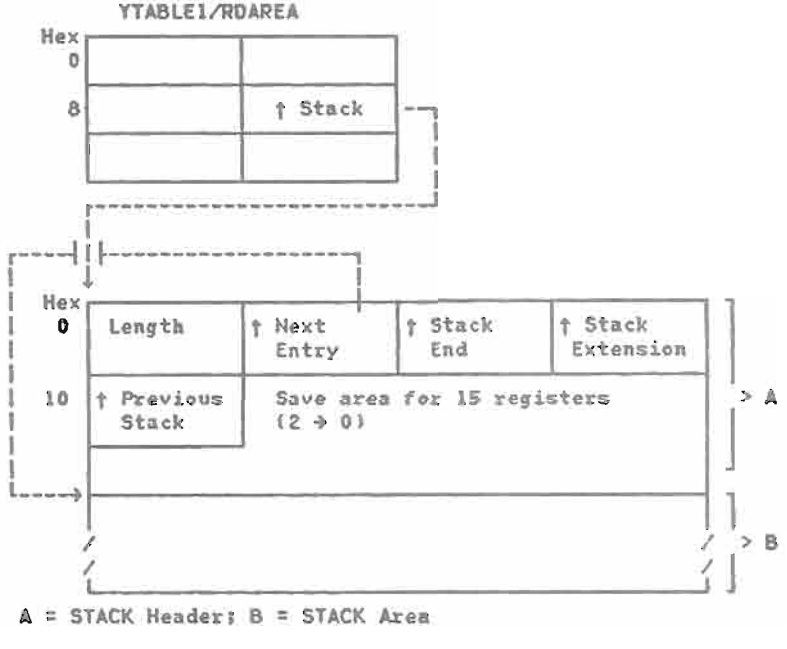


* From ARIYT06
ARICWFR (Free Extended Pool)

- If no attached (extended) pools, return to caller → RETURN
- Locate last attached pool in chain and call ARISYSD2 to release pool to operating system ↔ ARISYSD2
  If non-0 return code, terminate SQL/DS → ARIYM02
- Return to caller → RETURN

ARICSTK

- ARICSTK calls ARICWSG to get working storage ↔ ARICWSG
  If return code from ARICWSG is not zero, return to caller with the return code → RETURN
- ARICSTK then puts the stack pointer in YTABLE1/RDAREA and sets up as in the diagram at the right.

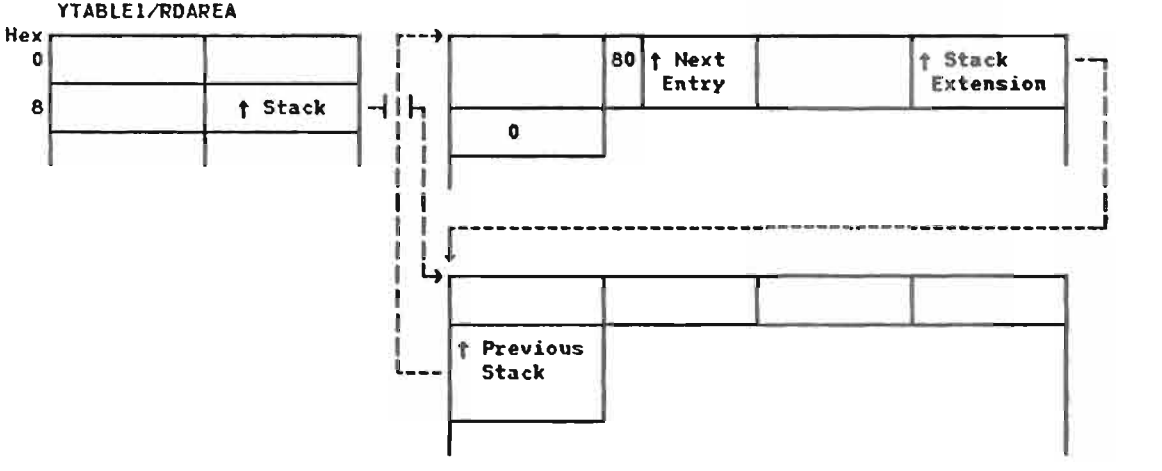


A = STACK Header; B = STACK Area

ARICGSE (Get Stack Extension)

- ARICGSE calls ARISYSDG to get storage to extend the stack. The setup of the extension is indicated in the next diagram ---------

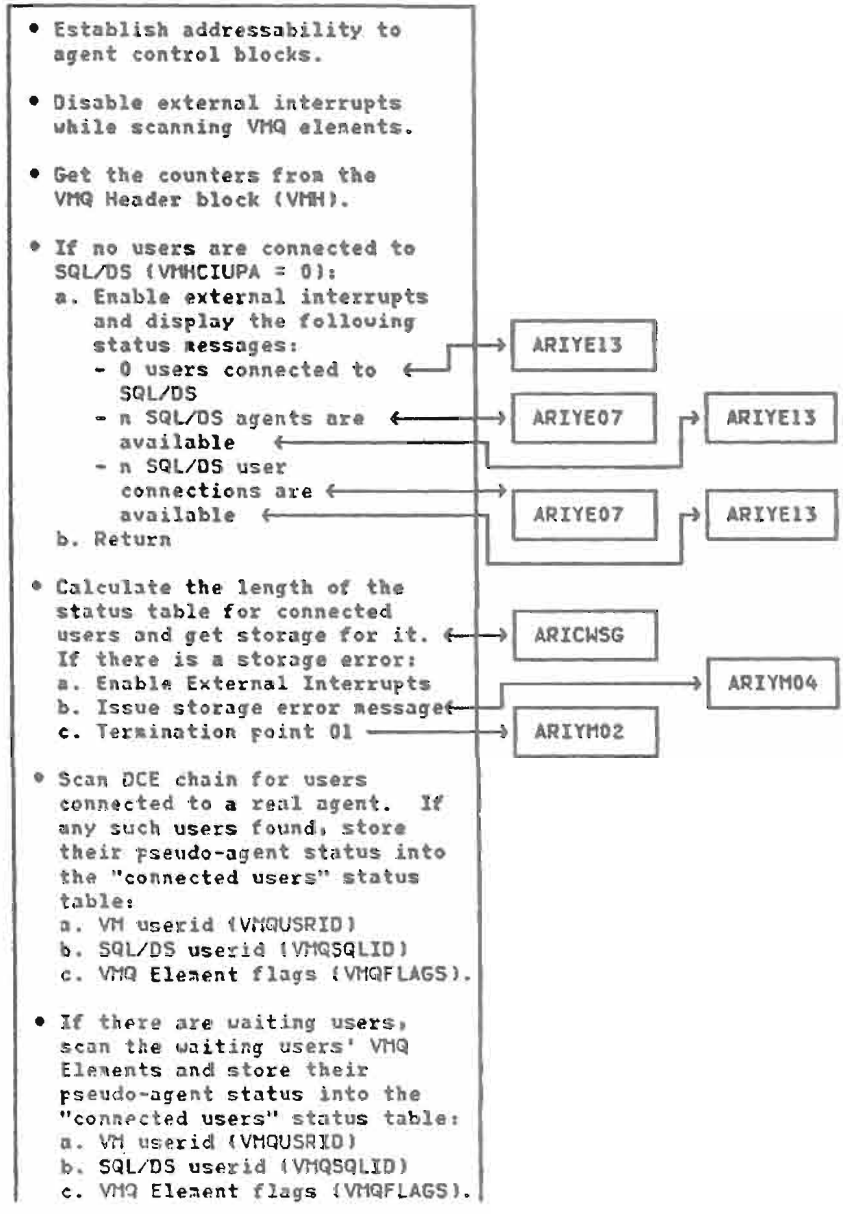
→ ARISYSDG

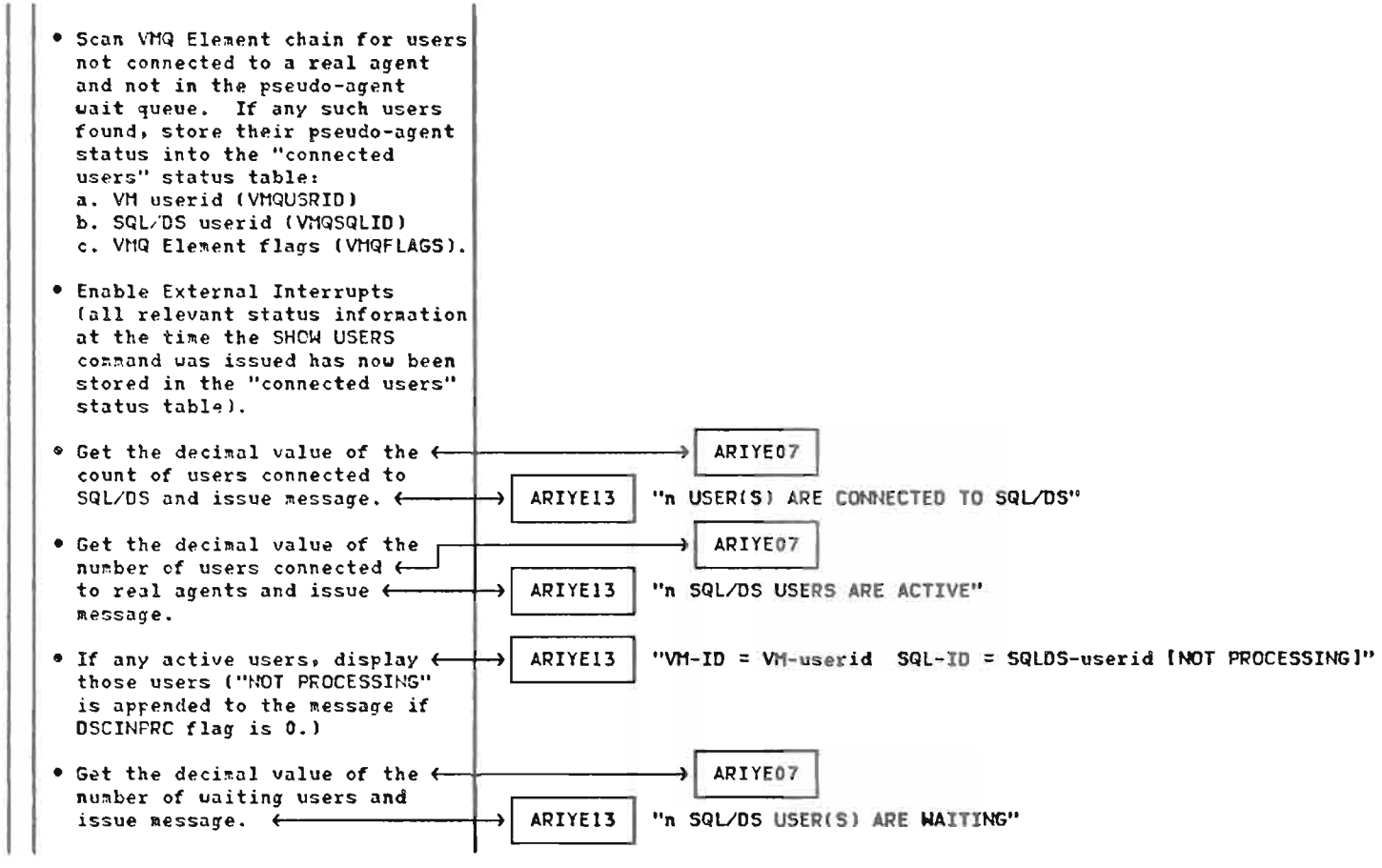


ARICFSE (Free Stack Extension)

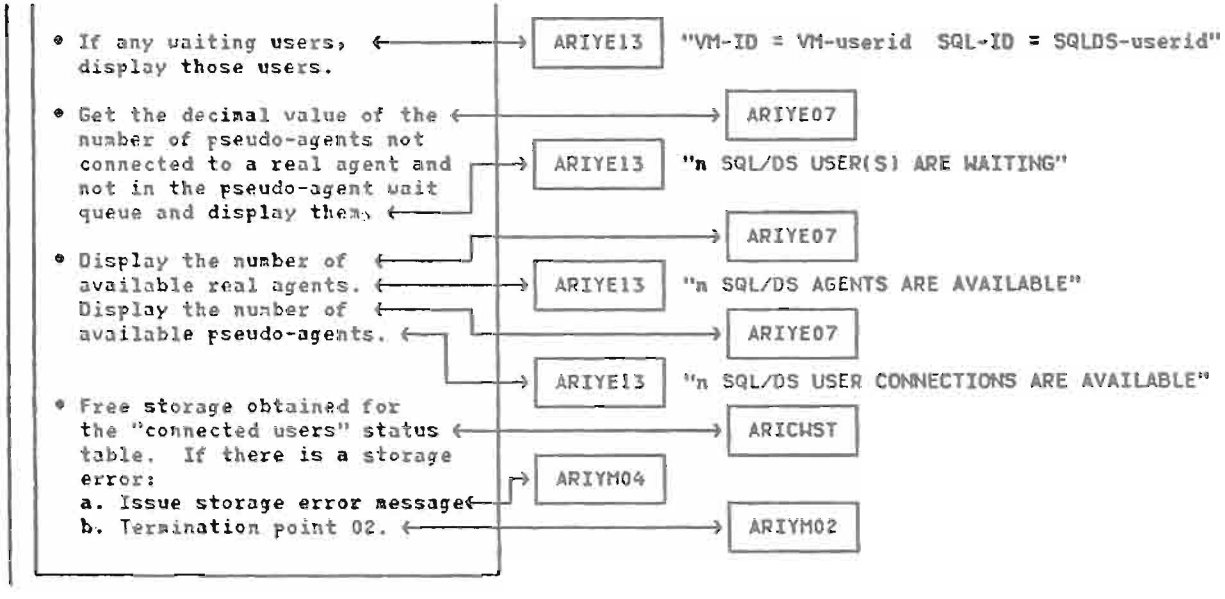
- Get pointer to current stack from register 1.
- Free the current stack and rechain pointers back to previous stack. (This module utilizes pointers in YTABLE1/RDAREA and the Stack (see the diagram on the previous page).
- Return to caller → RETURN

ARICSHO (Called by ARIYM11)

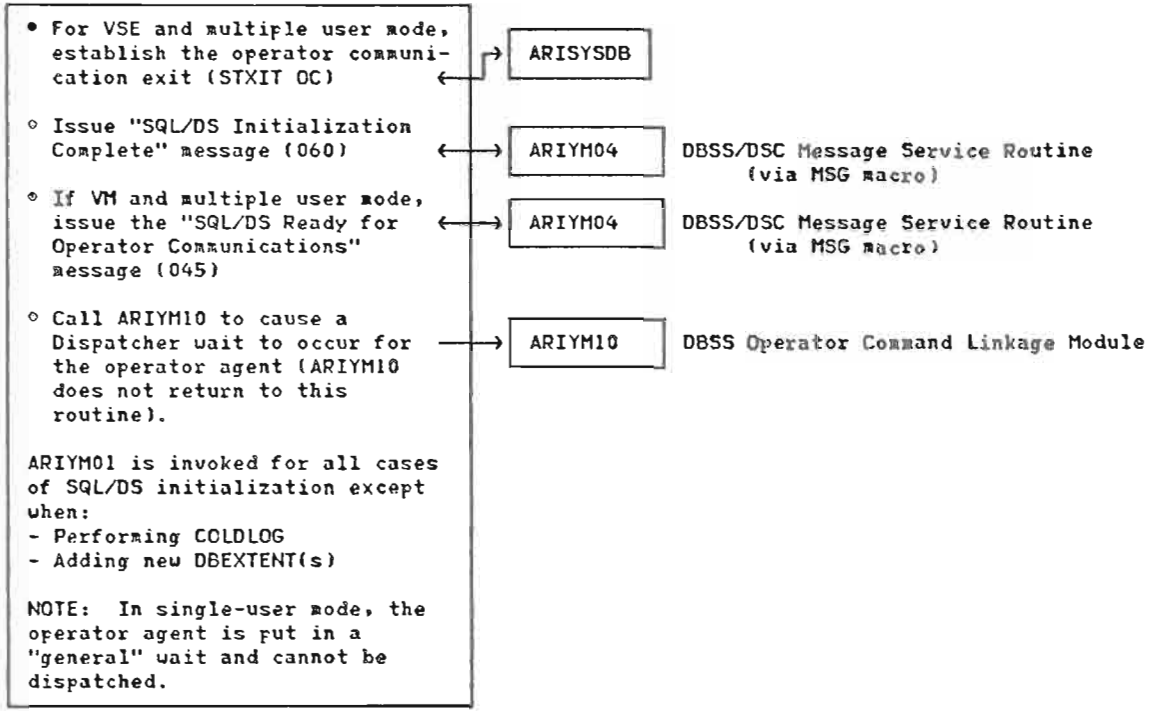




DSC Operator Services (continued)



ARIYM01 (Called by ARICIP2)



DSC Operator Services (continued)

ARISOCM (Entered as a result of the VSE MSG supervisor command
to 'wake up' the SQL/DS Operator Agent for command entry)

ARISOCM provides the linkage between the VSE operator console and the SQL/DS command linkage module ARIYM10 when the operator wishes to enter a SQL/DS operator command. ARISOCM can be entered only in multi-partition mode (no supervisor linkage in single-partition mode). The module is entered when the system operator enters the VSE attention command MSG, which notifies the VSE Attention Routine that he wishes to communicate with the SQL/DS partition. The Operator Agent structure is posted by turning off the DCEGENRL bit in the agent DCE.

A 'busy' bit is also turned on in the DS2CVT to prevent entry into SQL/DS command processing in the event that the operator enters another attention request while the command is being processed. A dummy ECB is inserted into the SQL/DS Dispatcher Wait List and SVC-posted to ensure that VSE will dispatch SQL/DS and that module ARIYM10 in the Operator Agent will be given control.

The VSE macro EXIT OC returns control to the operating system.

ARISOCM calls ARISYSD6 to obtain the address of DS2MODE from the partition phase load list. (DS2MODE is a control block and phase that gives addressability to SQL/DS control blocks.)

To obtain load address of phase ARICMOD (DS2MODE control block)

ARIYM10 (Entry point ARIYM10 entered at 'SQL/DS initialization complete'
(called by ARIYM01); entry point ARIYM101 entered from ARIYM00)

ARIYM10 is entered at SQL/DS 'initialization complete' to place the Operator Agent in SQL/DS wait state waiting for a system operator command request.

ARIYM101 processes commands from ISQL terminal users.

The system operator or ISQL operator is notified of the success or failure of the command processing.

In single-user mode, ARIYM10 places the Operator Agent in permanent SQL/DS Dispatcher wait state at the completion of SQL/DS initialization

Modules called from this module are:

- To execute a valid (command name) SQL/DS operator command from the system operator or ISQL terminal user → ARISYSDJ — Read an operator command
- → ARIYM11 — DBSS operator command router (see page 145)
- To issue command completion or error messages to the system operator or ISQL terminal user → ARIYM04 — DBSS/DSC message service routine (MSG macro)
- To tokenize the command input line → ARIYE04 — Tokenizer routine
- To put Operator Agent in wait state:
  - at end of SQL/DS initialization (waiting for system operator request for command entry), or
  - at end of command processing from system operator

  → ARIYK18 — Put Operator Agent in wait state

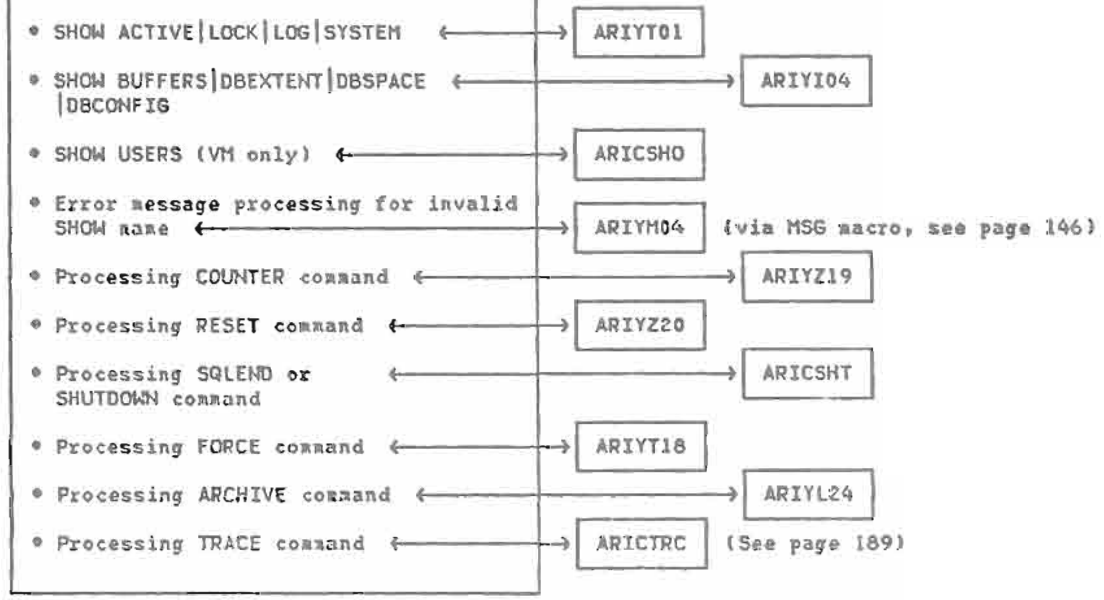
Note: ARIYM10 entry logic is re-dispatchable by the SQL/DS Dispatcher when the system operator requests command entry via the VSE MSG command.

DSC Operator Services (continued)

ARIYM11 (Called by ARIYM10)

ARIYM11 is called by ARIYM10 when it has received a valid SQL/DS operator command from the operator or from the ISQL terminal user. ARIYM11 provides the interaction between module ARIYM10 and the SQL/DS operator command processing modules. Also, ARIYM11 validates the SHOWNAME operand (operand 1) of SHOW commands.

ARIYM11 calls the following modules for:



* From DBSS, DSC, RDS and Batch or On-line Resource Manager Modules
via MSG Macro for DBSS/DSC/RM (or RMSG macro for RDS)
ARIYM04 (Message Services)

ARIYM04 is invoked to write a DBSS, DSC, or RDS message to the operator (VSE system or virtual machine terminal, or ISQL) and optionally read a reply and/or to write the message to the system print file or to write a Resource Manager message to the VSE system or virtual machine terminal operator and optionally read a reply.

ARIYM04 builds the structures MSGID and BUFFER and invokes the message formatter for message retrieval and variable substitution. ARIYM04 calls ARIYM05 to retrieve and edit all lines of a requested message or to retrieve an error message if there was a message formatting error. After the message is formatted, it is passed to ARIYE131 ARIYM50 or ARIYM51 for display (ARIYM50 and ARIYM51 are used for reply type messages only).

- ARIYM05 — Message Formatter (Page 147)
- ARIYM50 — Reply Messages (Multi-line and not last line)
- ARIYE131 — (Page 149)
- ARIYM51 — Reply Messages (Last line)

For multi-line messages, ARIYM04 loops displaying each line of the message.

- If a reply is requested, on the last (or only) line of the message ARIYM51 is invoked to display the message line and read the reply.
  - ARIYM51 — Reply Messages (last line only)
- If a reply is requested and this is not the last line, ARIYM50 is called to ensure that the line is directed to the VSE system or virtual machine terminal operator.
  - ARIYM50 — Reply Messages (all but last line)

If the message formatter signals an incomplete message, an additional message is displayed to inform the operator or print file of this error condition (with the same calls as shown above). If the message formatter returns an error

message (return code 16), it is displayed in place of the requested message with any reply request suppressed and the reply buffer cleared to blanks.

* From ARIYM04
ARIYM05 (Message Formatter)

ARIYM05 is used to retrieve a message (or line of a multi-line message) and substitute into the message (or line) any caller provided message variables. Using the caller provided message number or SQL code, the message index or SQL index table is searched to locate the address of the message module containing the message. Then, the message module directory is searched for an entry matching the requested message/SQL code number.

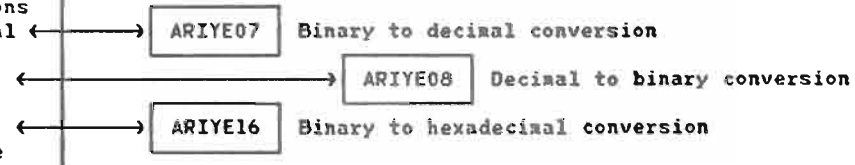
- If the message is not a multi-line message, the message directory entry gives the address of the message text structure.

- If the message is a multi-line message, the message directory entry gives the address of the the message sequence structure which contains an entry for each line of the message. The message sequence structure is searched for a line number entry matching the line sequence number provided by the caller. The line number entry gives the address of the line message text structure and a last line flag. ARIYM05 updates the caller line sequence number for the next line request.

The message formatter now processes the message text moving it to the caller provided message buffer and performing variable substitutions (with optional binary to decimal or hexadecimal conversion with zero suppression and low order blank suppression for normal string substitution). Variable requests are denoted in message text by &nnn. or &nnnalpha. (nnn is request for nth passed variable) and the variables are passed via the VARLIST structure.

The length of the formatted message line is returned to the caller. Return codes indicate:

- If there are more lines to the message
- If a formatting error occurred
- If the message could be found.



* From Command Processing Modules and other Routines
  Entry Point ARIYE131 is called by ARIYM04.
  ARIYE13 (Line Display)

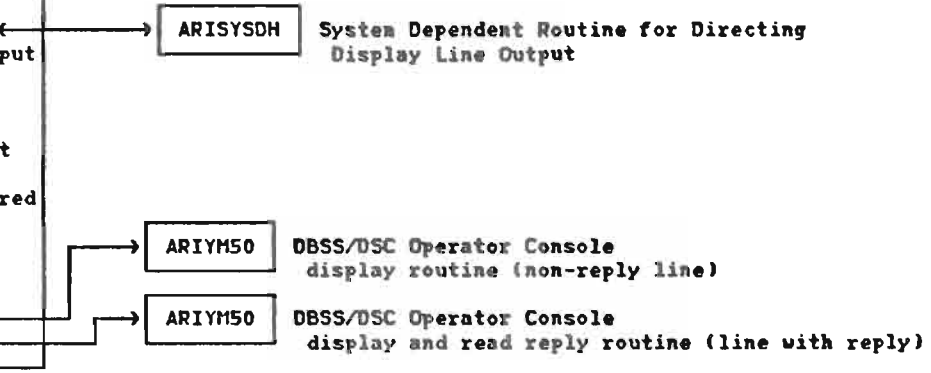
ARIYE13 will display a line of data (or a SQL/DS message).

- Main entry ARIYE13 is used for displaying a line of data which is not a SQL/DS message. A flag (DSCOMODE), in the DSCAREA table of SQL/DS, directs ARIYE13 to display the line locally or to send the line to the ISQL linkage for display on a CICS terminal (VSE) or the ISQL virtual machine terminal (this is for support of SQL/DS commands entered from ISQL).
- Secondary entry ARIYE131 is used exclusively by the DBSS/DSC message processor for displaying SQL/DS messages. Output lines are directed to either local output or to ISQL for CICS Terminal output (VSE) or virtual machine terminal output.

During SQL/DS operator command processing, only messages originating from the operator command linkage and processing modules go to the ISQL linkage, all others go to local output.

- If output is ISQL, ARICOMB is called to route the output line to ISQL → ARICOMB — ISQL Output Mailbox Linkage
- If output is to ISQL and the SQL/DS Mailbox is full, ARICMUD is called. → ARICMUD — To SQL/DS Dispatcher to wait for Mailbox transfer to ISQL to complete.

SQL/DS has an initialization parameter DSPLYDEV which directs all output lines routed to this module to be displayed on either the system print file or the operator's console or to both (for VM, the DSPLYDEV parameter has no effect and the output goes to the virtual machine terminal). A routing flag in the DS2CVT indicates if line display is to be controlled by the DSPLYDEV parameter value. If the flag is on (output is under control of the DSPLYDEV initialization parameter), the line and the routing code is passed to module entry ARISYSDH for display on the specified output device(s)/file(s). This routing function is active only during SQL/DS initialization and termination. It does not conflict with ISQL routing since SQL/DS operator commands cannot be entered during initialization or termination. When the DSPLYDEV routing function is not active (flag is off) local output is routed to the operator display modules ARIYM50 and ARIYM51.



DSC AGENT HANDLING AND COMMUNICATIONS

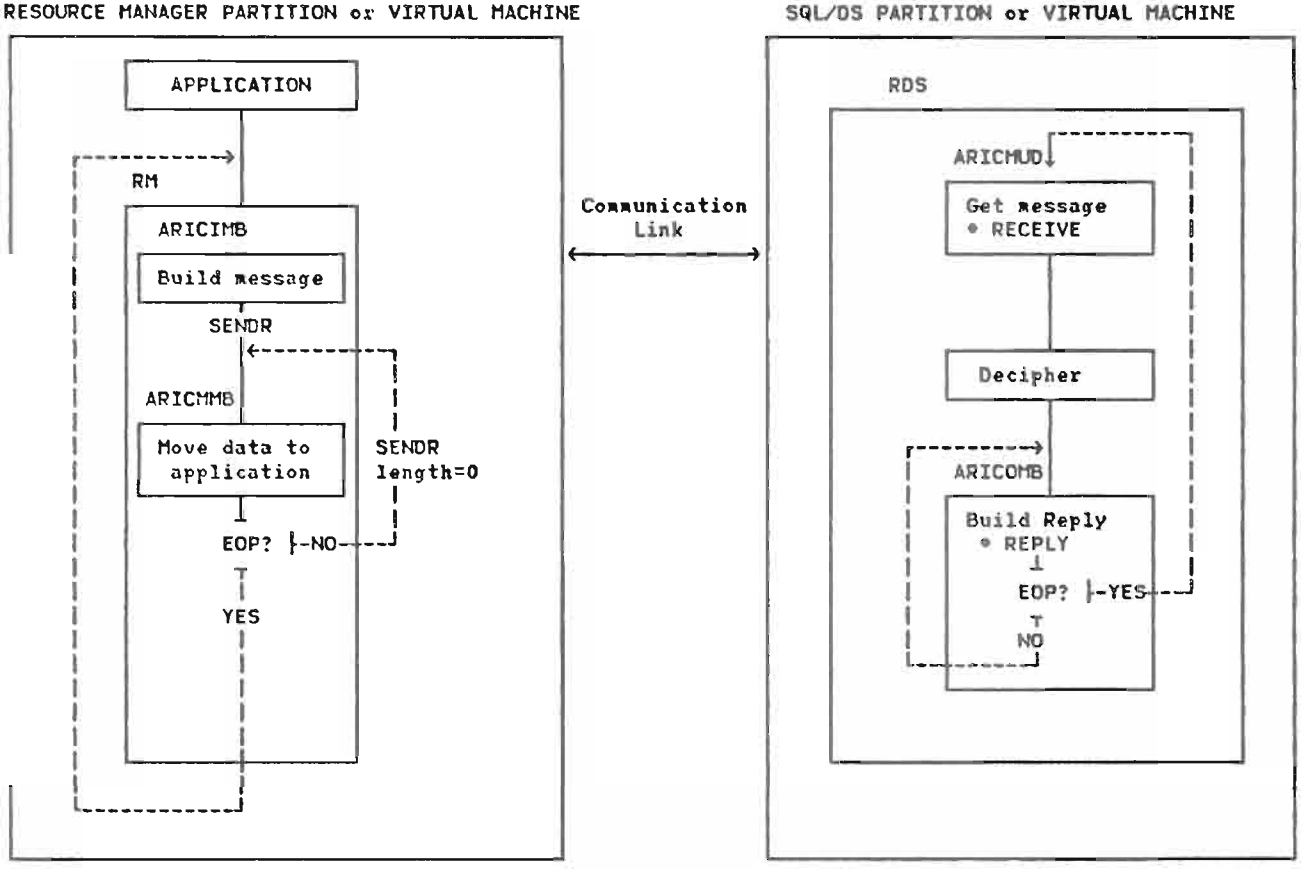
The following series of diagrams provide an overview of the interaction and relationship of agent handling and communications with some of the other components/functions of SQL/DS as it applies to the multi-user mode environment.

In single-user mode, the flow is similar except ARICMUD is not called by RDS because it (RDS) can receive the "message" directly from the Batch Resource Manager.

The Resource Manager (Batch or On-line) will call ARICIMB to build the message to be sent to the SQL/DS partition via the communication (CPC or IUCV) link. The message will be received by ARICMUD and passed on to RDS for processing.

RDS will then call ARICOMB to send a reply message back to the Resource Manager partition. The Resource Manager will then call ARICMMB to move the reply message into the application's data areas.

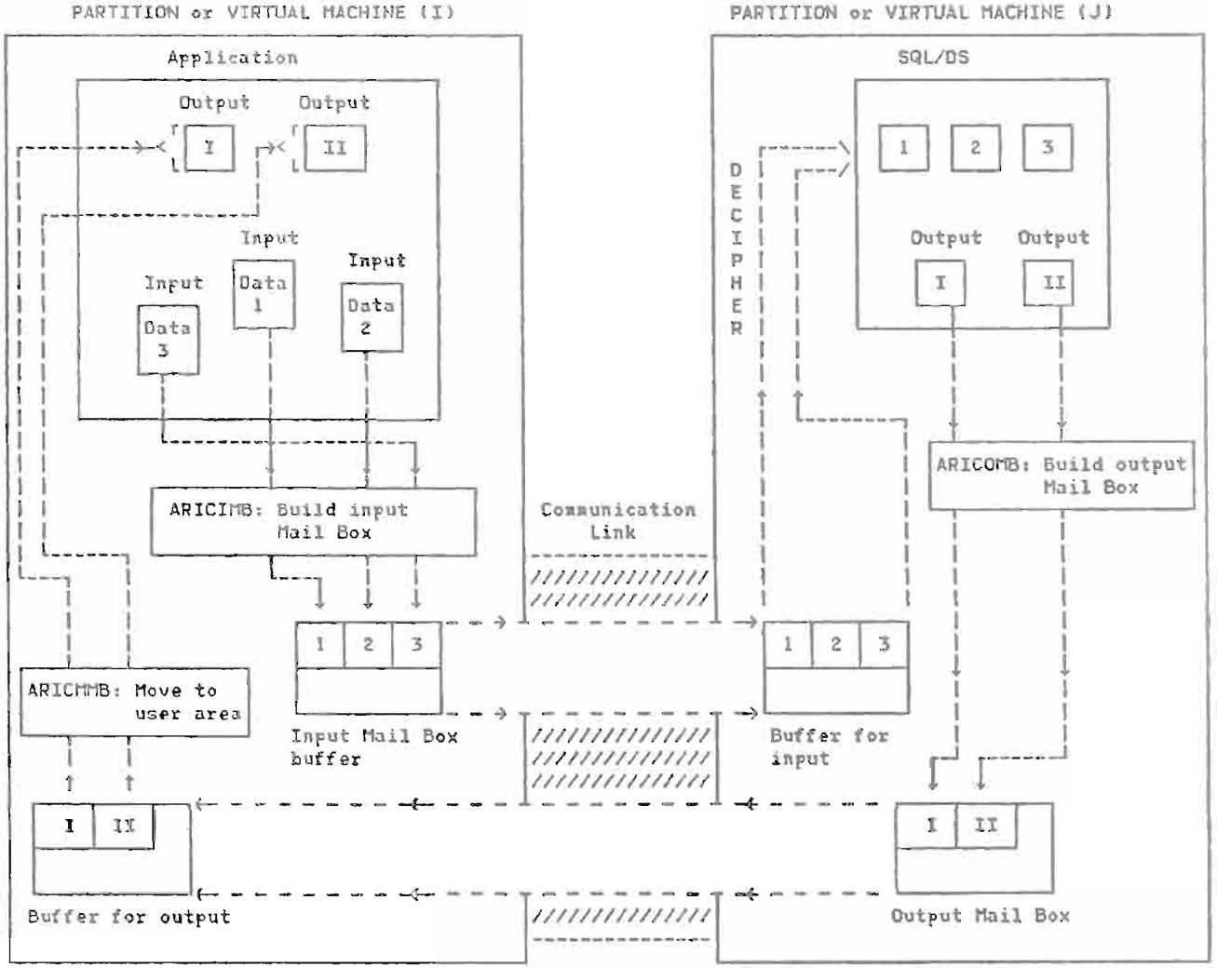
DSC/Resource Manager Interaction



Note: ARICIMB, ARICMMB, and ARICOMB are Mail Box modules. The Mail Box/Resource Manager interaction is further described on the next page.

Mail Box/Resource Manager/RDS Interaction

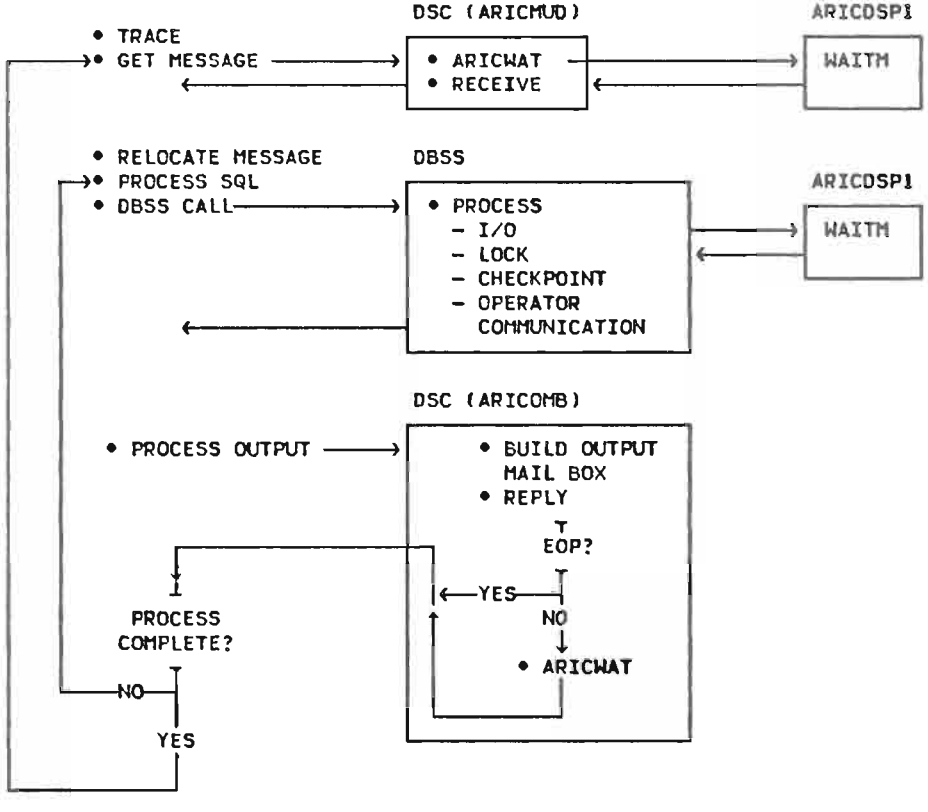
Mail Box is a facility specifically designed for SQL/DS cross-partition (CPC) or inter-user communication vehicle (IUCV) support. It collects a group of data from one partition/virtual machine and passes it on to another partition/virtual machine in a one-move procedure. The actual data transfer is done by CPC (for VSE) or IUCV (for VM) link since SQL/DS is assumed to be run in a VSE or VM environment. However, the Mail Box itself does not have any system dependency. The following diagram gives an overview of the Mail Box process. For a more detailed flow, see page 183. Descriptions of the modules ARICIMB, ARICMMB, and ARICOMB are included in Section 3. For the data areas associated with the Mail Box facility, see "Mail Box Data Areas" in Section 5 of Volume 2.



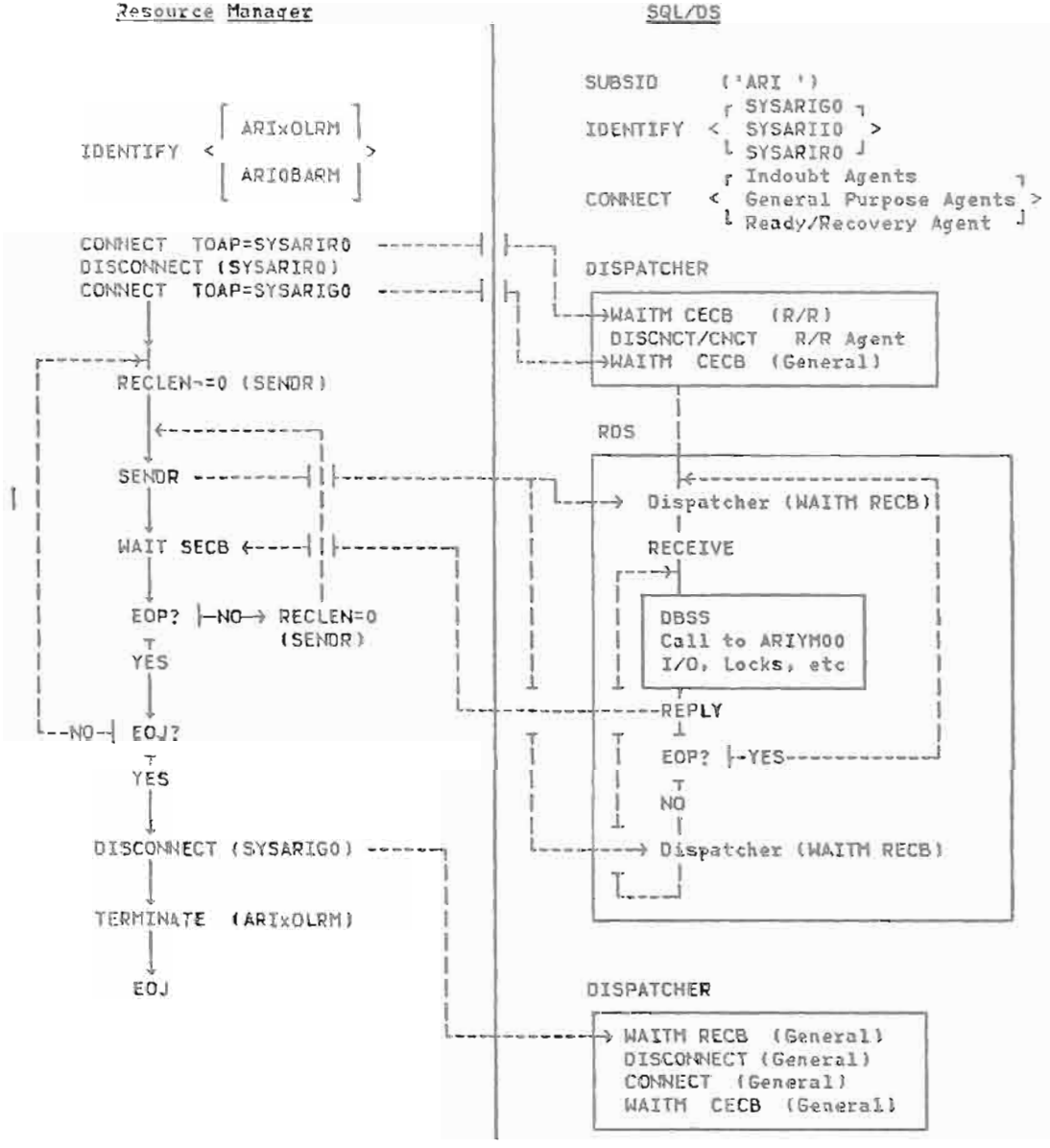
DSC/RDS Interaction

This diagram illustrates the relationship between DSC and RDS. When an agent structure is initially dispatched, control is given to the RDS module, ARIXERD. (This does not apply to the Operator or Checkpoint Agents nor the General Purpose agent in single-partition mode or single virtual machine mode). If trace is activated for ARIXERD (RD$), the trace routine will be called. ARIXERD will then call ARICMUD to receive the message sent by the Resource Manager partition. (In single-partition mode or single virtual machine mode, ARIXERD does not call ARICMUD, but gets the message directly from the Resource Manager routine.) RDS will now decipher the message and perform one or more DBSS calls to process the input message and call ARICOMB to send a reply message back to the Resource Manager partition. If the reply message can be sent as one message the ARIXERD will indicate that the reply is also the end of the reply process to ARICOMB. If the reply message can not be sent back as one message, then RDS will pass back the amount of data that has been processed and indicate to ARICOMB that it is not "end-of-process". ARICOMB will then send that portion of the message processed back to the Resource Manager partition or virtual machine and wait for an acknouledgement by the Resource Manager partition (a SENDR with zero data length). This process continues until RDS indicates "end-of-process" to ARICOMB.

RDS (ARIXERD)



Cross-Partition Communication Protocol (VSE only)

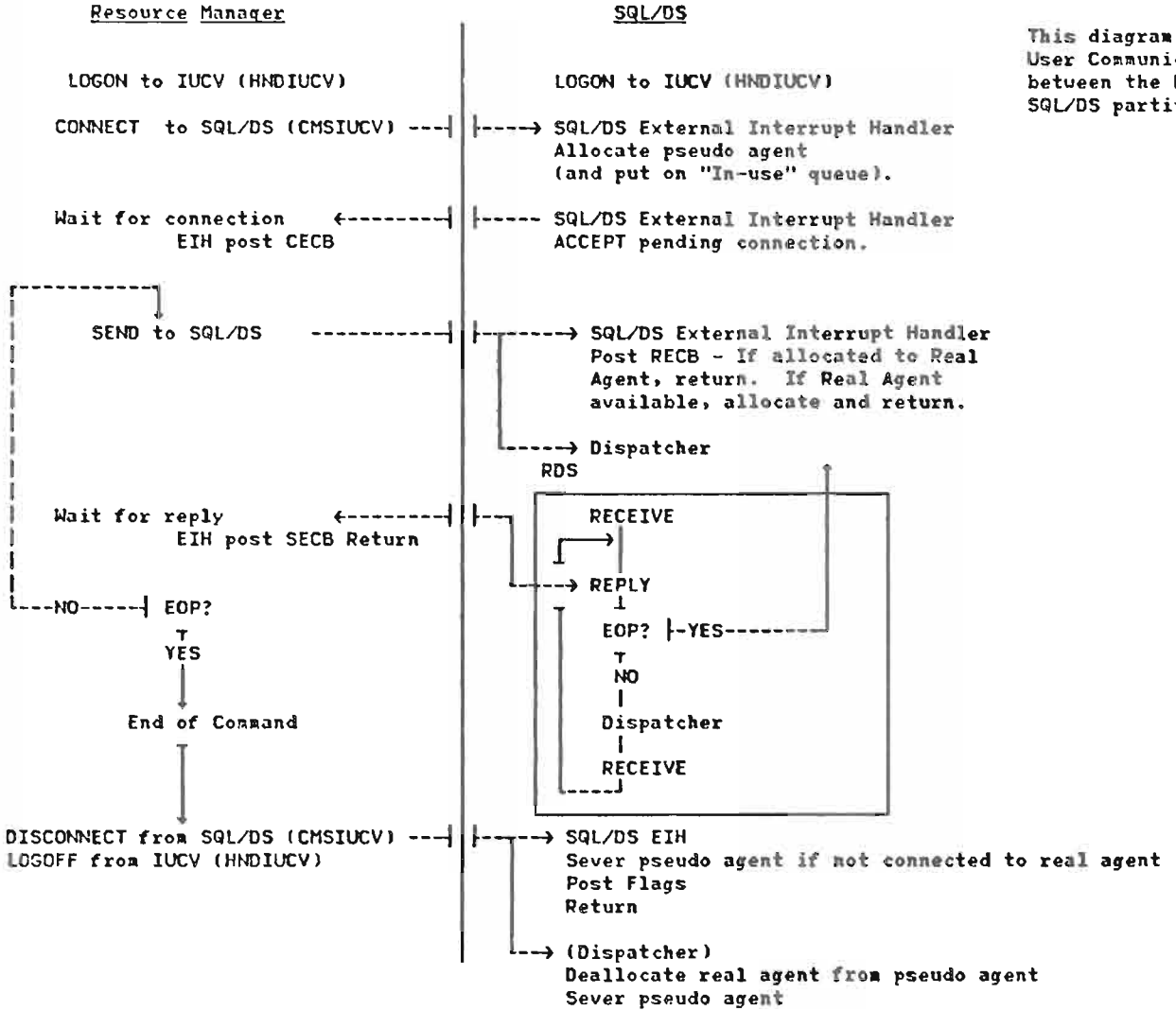


This diagram illustrates the Cross-Partition Communication (CPC) flow between the Resource Manager and SQL/DS partitions. The SQL/DS partition issues a SUBSID macro instruction to identify itself to VSE/Advanced Functions as a subsystem. It then issues three CPC IDENTIFY instructions, one for each category of agent structures (General Purpose, Indoubt, and Ready/Recovery). After SQL/DS has identified itself to VSE/Advanced Functions as a subsystem, it issues a CPC CONNECT for each agent structure (General Purpose, Indoubt, and Ready/Recovery) and then waits for the Resource Manager partition(s) to complete the connection and send messages. The SENDR/RECEIVE protocol flow is as described above (DSC/RDS Interaction).

The Resource Manager partition(s) will also issue a CPC IDENTIFY and then attempt to "CONNECT" to the SQL/DS Ready/Recovery Agent. This "connect" is used to determine if SQL/DS is "ready" to communicate with the Resource Manager partition(s). If the connection is completed, the Resource Manager disconnects from the SQL/DS Ready/Recovery agent and attempts to connect to a General Purpose (or Indoubt, Online Resource Manager only) agent structure. If that connection is successful, it will then send messages to the SQL/DS partition following the protocol as described above (DSC/Resource Manager Interaction).

When all messages have been processed (the Resource Manager partition has issued a COMMIT RELEASE, gone to EOJ, etc) its CPC link will be disconnected. The SQL/DS partition will then disconnect its CPC link and reconnect it for any other potential communicator.

Inter-User Communication Vehicle Protocol (VM only)



This diagram illustrates the Inter-User Communication Vehicle (IUCV) flow between the Resource Manager and SQL/DS partitions.

Dispatcher Overview and Control Blocks

The SQL/DS Dispatcher is a non-preemptive, round-robin dispatcher. It dispatches SQL/DS Agents (SQL/DS tasks). A SQL/DS Agent has an associated set of control blocks called an agent structure. The control block in this structure that is used to drive the dispatcher is the Dispatcher Control Element (DCE). In addition to the DCE, there is a WAITM (VSE) or WAITECB (VM) list containing the pointers to the various IORBs (CCBs) and ECBs that are posted by the VSE operating system or the SQL/DS External Interrupt Handler (VM). The Dispatcher will issue a WAITM (VSE) or WAITECB (VM) on this list when no agents are dispatchable.

There are five basic categories of SQL/DS agent structures:

- The Operator Agent structure, which is used for communication between SQL/DS and the VSE system operator or SQL/DS operator (VM). (Since this agent has no interaction with the RDS components it has no RDAREA.)
- The Checkpoint Agent structure, which controls the SQL/DS checkpoint and archiving process (it also has no RDAREA).
- For VSE only:
  - The Ready/Recovery Agent structure, which is queried by the Resource Manager(s) to determine if SQL/DS is ready to communicate with other partitions. In addition, the Online Resource Manager uses this agent structure to transmit recovery data to SQL/DS (that is, to synchronize the CICS and SQL/DS logs). This agent structure only exists when the parameter SYSMODE=M is specified.
  - The Indoubt Agent structure, which is used to either commit or rollback previous units of work that were prepared-to-commit or rollback when SQL/DS unexpectedly terminated. These agent structures exist only in a VSE multi-partition mode environment. After "Indoubt processing" has been completed, these agent structures are converted to a General Purpose Agent structure.
- The General Purpose Agent structure which is used to perform normal SQL/DS processing. There is one General Purpose Agent structure built in single-partition/single virtual machine mode. The number of General Purpose Agent structures built in multi-partition/multiple virtual machine mode is equal to the "NCUSERS" parameter value minus the number of Indoubt Agent structures created. In VM this number is zero. Note: The NCUSERS value must always be large enough to allow creation of all Indoubt Agent structures.

An agent is dispatchable when the DCEPWF field (in the DCE) is zero. If this field is non-zero, then it can be one of five states (which are mutually exclusive):

- I/O wait, the agent is waiting for an I/O operation to complete. When the I/O operation completes, the pointer to the IORB (CCB - VSE or ECB - VM) in the WAITM (VSE) or WAITECB (VM) list will be replaced with a pointer to a dummy ECB (which is not posted). This prevents a WAITM (VSE) or WAITECB (VM) request from being satisfied by a previously-posted IORB or ECB.
- Lock wait, the agent is waiting to obtain a lock held by another agent.
- Cross-Partition Communication (XPTN) or Inter-User Communication Vehicle (IUCV) wait, the agent is waiting for its connect ECB to be posted (to establish a CPC link) or for its receive ECB to be posted (both CPC and IUCV).
- General wait. This is the normal wait state condition for the Operator and Checkpoint Agents whenever they complete a given function. In addition, this wait state condition is entered by other agents whenever the Checkpoint Agent is activated.
- Inactive. This condition occurs only at initialization time for VSE (while the agent structure is being created) and when the agent is disconnected and reconnected to the CPC or IUCV link.

The above wait state conditions and dispatchability of an agent are tested in the following order:

1. I/O wait (DCEIOWAT=1)
2. Dispatchable (DCEPWF = 0) (or a suspend wait)
3. Lock wait (DCELOCK = 1)
4. XPTN wait (DCEXPTN = 1)
5. General Wait (DCEGENRL = 1) (or Inactive - DCEINACT = 1)

In addition to the Dispatcher wait flags (DCEPWF), there is a second set of Dispatcher flags which control the Dispatcher flow. They are as follows:

- CPC or IUCV link connected (DCECNCT). The agent is connected via a CPC or IUCV link with the Resource Manager partition or virtual machine.
- Rollback requested (DCELKTRM). This indicator is set whenever ARIYT23 is called to begin the rollback (backout) process. (It will also be set if a commit is required and the Resource Manager partition or virtual machine went to a normal end-of-job and a logical unit of work (LUW) was not in process (DSCINPRC = 0). In VM, this indicator is also set by ARICMUD whenever RDS indicates that a logical unit of work has been completed (DSCELUW = 1). In conjunction with DCELKTRM, the DCERESET indicator results in the current user (pseudo agent) associated with the real agent structure being deallocated from the real agent, and allowing a new user (pseudo agent) to be allocated to the real agent for a logical unit of work.
- Disconnect/reconnect of the CPC or IUCV link requested (DCEFORCE). This flag indicator is set when a FORCE DISABLE command has been issued. It will cause the CPC or IUCV link to be disconnected and then reconnected for use by any new communicator.

- Reset the agent (DCERESET). This indicator is set whenever a FORCE command (but not FORCE DISABLE) is issued (or the Resource Manager partition issued a CPC CLEAR command (VSE only)). VSE only - The current LUW will be backed out (rolled back), and the agent structure re-initialized as if a CPC connection had just been completed. However, is this case, no CPC disconnect/reconnect will be done.
  In VM, this indicator is also set by ARICMUD whenever RDS indicates a logical unit of work has been completed (DSCELUW = 1). In conjunction with DCERESET, the DCELKTRM indicator results in the current user (pseudo agent) associated with the real agent structure being deallocated from the real agent and allowing a new user (pseudo agent) to be allocated to the real agent for a logical unit of work.
- SQL/DS ABEND (DCESQLAB). This indicator is set by ARICABE (MUM only) to show that SQL/DS abnormally severed the IUCV communication link. This bit is tested by ARICCLA. If on, ARICCLA passes the keyword "SQLABEND" in the user data field for the IUCV sever function, prior to disconnecting the communication link. This is used to inform the Resource Manager that the SQL/DS virtual machine abnormally severed the communication link and the logical unit work for that user was rolled back.

The basic dispatching flow is as follows: The DCEs are chained together in the order that the agent structures were created. The Dispatcher will scan the DCE chain testing its associated IORB/ECBs (using the pointers in the WAITM (VSE) or WAITECB (VM) list) to determine if they are posted. The various wait state conditions are tested in the priority stated earlier. Whenever a posted ECB is encountered, the corresponding DCE wait bit is reset and the agent is dispatched. The DS2CRDCE field in the DS2CVT will contain the pointer to the DCE of the agent that was dispatched. Whenever the Dispatcher is called to perform a SQL/DS wait for an agent, that agent will give up control. (Its DCE wait flag having already been set, unless it is a "suspend wait". In this case, the agent has given up control to let other agents have a chance to run). The next DCE in the chain will be obtained and determined if dispatchable as described above. When all agents are in a SQL/DS wait state, then a WAITM (VSE) or WAITECB (VM) will be issued on the WAITM or WAITECB list.

DCE

DCE			
Length (X'80')	PWF[1]	LFLAG[2]	INDEX
† DS2CVT	† DSCAREA		
† YTABLE1	† RDAREA		
† Next DCE	† Wait-Multiple list (for this DCE)		
† RECB (in XPCCB)	Dummy ECB		
Registers 0 → 15 (of ARICDWT/ARICWAT) Note: Register 9 contains pointer to save area (ARICDWT/ARICWAT); Register 10 contains pointer to YTABLE1.			
Address next available real agent	Address next in-use real agent		
Address of previous in-use real agent	Address of associated pseudo-agent		

[1] DCEPWF (Priority)

DCEGENRL	X'80'	- General Wait	(5)
DCEIOWAT	X'40'	- I/O Wait	(1)
DCELOCK	X'20'	- Lock Wait	(3)
DCEXPTN	X'10'	- XPTN Wait	(4)
DCEINACT	X'08'	- Inactive	
	X'00'	- Dispatchable (or Suspended)	(2)

[2] DCELFLAG

DCECNCT	X'80'	- XPTN Link Connected
DCELKTRM	X'40'	- Rollback Started
DCEFORCE	X'20'	- Force (Disable)
DCERESET	X'10'	- Force (Abort)
DCESQLAB	X'08'	- Comm link terminated by SQL/DS

Note: X'20' causes XPTN link to be disconnected and reconnected.
X'10' causes XPTN link to be "reset".

Dispatcher parameter lists (for Calls) are in the DSCAREA (DSCSTBKP/DSCSTBKL and DSCPELSP/DSCPELST).

Whenever ARICENA is called at initialization time, or whenever an agent is reset (or disconnected/reconnected) by the Dispatcher, it sets register slots in the DCE:

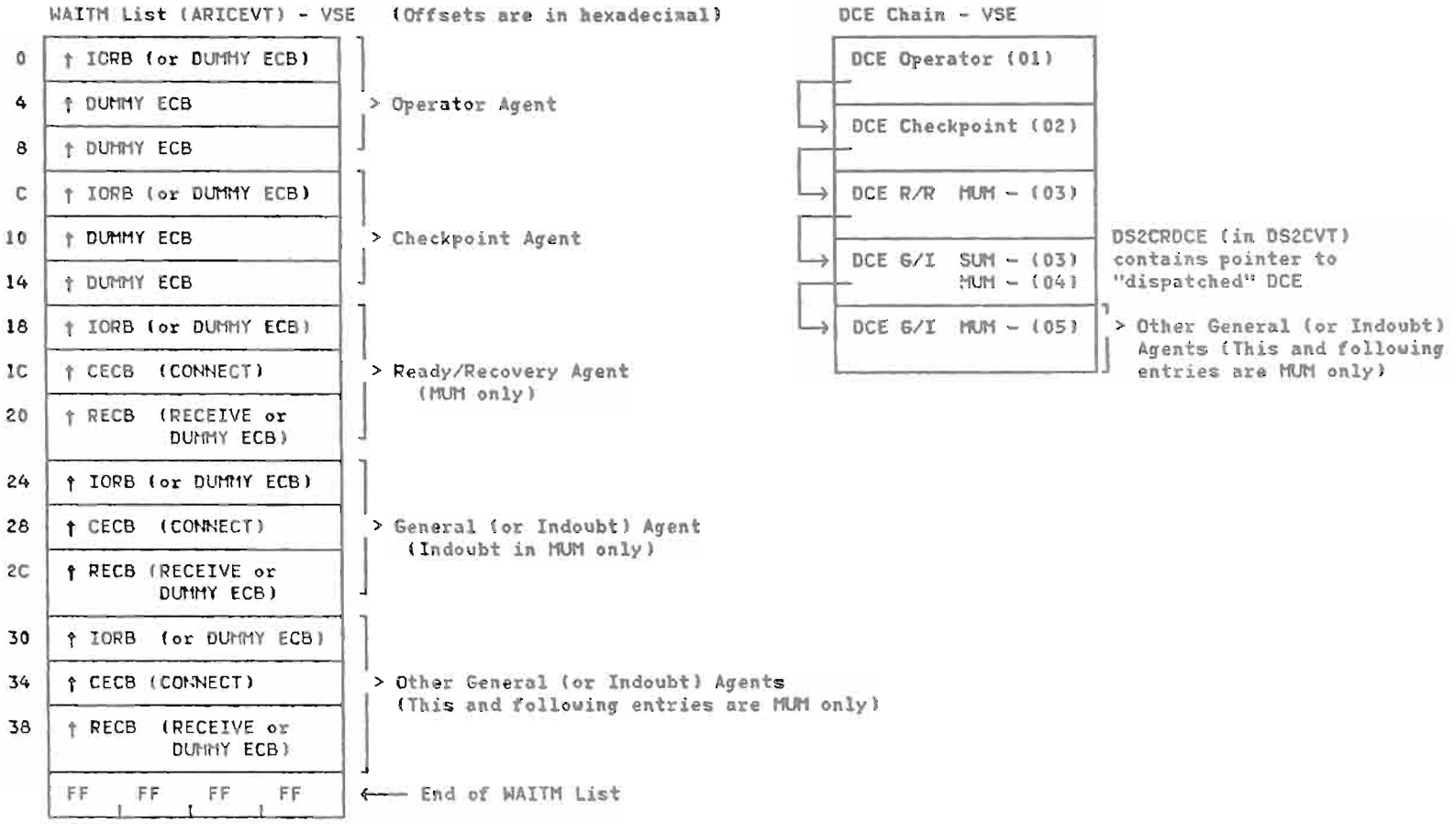
10 - Pointer to YTABLE1/RDAREA (RDAREA in MPM/MVMM)

13 - Pointer to DSCOBSSA (DSCAREA) (for STM on "Agent entry")
14 - Pointer to Agent "Entrypoint" (acts as Dispatcher return register)
15 - Pointer to Agent "Entrypoint" (Base Register - module call BALR 14,15)

Pseudo-Agent and Real Agent Structures

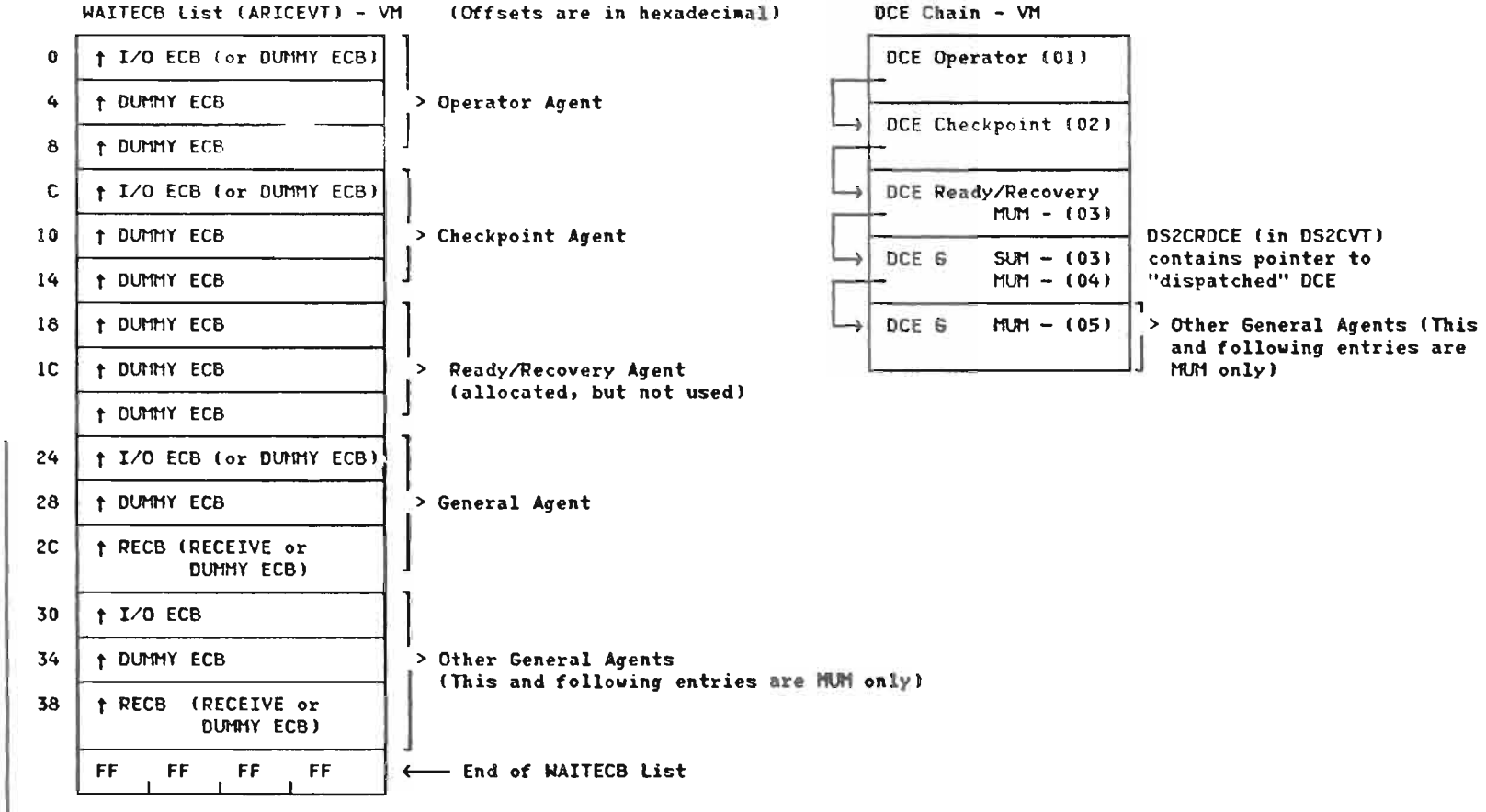
SQL/DS uses a control block called an agent structure (or real agent) to service requests from multiple users to access a common data base. Each real agent requires approximately 200K bytes of storage. Since it would not be practical in terms of storage and performance to allocate a real agent for each user, pseudo-agents that use approximately 300 bytes of storage were developed. Pseudo-agents allow many users to share (but not concurrently) a few real agent structures. (In VSE, ISQL and user applications use CICS to do the terminal processing. CICS routes multiple users through one SQL/DS agent structure, allowing the number of real agent structures to be kept low. VM does not use CICS but needs a similar technique to minimize storage requirements.)

Pseudo-agents are allocated to the real agent structure in round-robin fashion. Each user connecting to the SQL/DS service machine (via IUCV CONNECT) is allocated a pseudo-agent. Each user connecting to the SQL/DS service machine (via IUCV CONNECT), is allocated a pseudo-agent. The pseudo-agent is then placed in an 'in-use' queue, until the user associated with that pseudo-agent sends a message (IUCV SEND with reply) to the SQL/DS service machine. That pseudo-agent is then allocated a real agent (assuming one is available). If all real agents are in use, any users having sent messages to the SQL/DS machine will have their pseudo-agents placed on a 'wait' queue until a real agent is available. A real agent becomes available whenever an active user (one whose pseudo-agent already owns a real agent) completes a SQL/DS unit-of-work. At this point, the first waiting pseudo-agent is allocated to the newly available real agent. If the pseudo-agent which just completed the unit-of-work has sent another message, it will be added to the end of the waiting pseudo-agent queue.



* ARICWAT puts pointer to RECB into WAITM slot on XPTN Wait.
* ARICDWT puts pointer to IORB into WAITM slot on I/O Wait and pointer to RECB into WAITM slot on Lock Wait (allows "ISQL Operator" to force an agent in Lock Wait).

* ARICDSP replaces IORB and RECB pointers with pointer to a dummy ECB; this keeps "processed ECBs" out of the list.

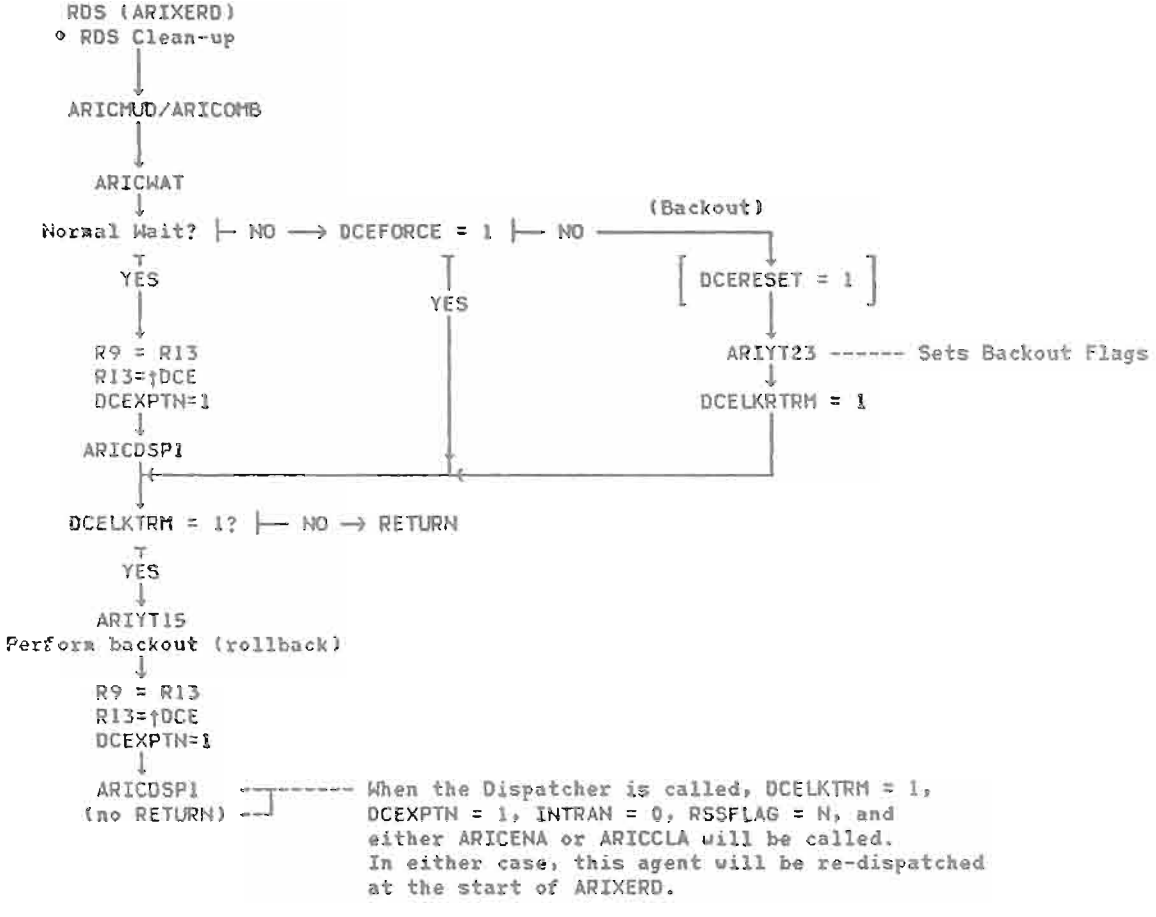


- ARICWAT puts pointer to RECB into WAITECB slot on XPTN Wait.
- ARICDWT puts pointer to I/O ECB into WAITECB slot on I/O Wait and pointer to RECB into WAITM slot on Lock Wait (allows "ISQL Operator" to force an agent in Lock Wait).

- ARICDSP replaces I/O ECB and RECB pointers with pointer to a dummy ECB; this keeps "processed ECBs" out of the list.

Backout Flow Initiated by ARICWAT

The following diagram provides a general overview of the backout (rollback) process as intiated by ARICWAT and an example of the backout process is also provided. ARICWAT will initiate the backout process whenever it is called with a explicit backout request or it finds the DCEFORCE indicator set.



Example of Backout Flow Through SQL/DS (FORCE Issued)

RDS:

- Issues a DBSS (ARIYM00) call (RSSFLAG=Y,INTRAN=1,DCEPWF=0)
- DBSS does I/O and issues I/O wait(s) (DCEIOWAT=1)

  * Operator issues FORCE DISABLE

- Operator Agent dispatched (process FORCE)
  * FORCE calls ARIYT23. Initiates backout, and sets DCELKTRM=1, DCEFORCE=1, DCEXPTN = 0, DCELOCK = 0; returns to Dispatcher to wait next command (DCEGENRL = 1).

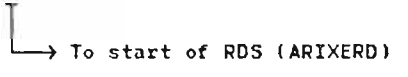
- I/O operation(s) complete (all I/O done) return is made to ARIYM00
- ARIYM00 finds the backout (rollback) process has been started, calls ARIYT15 to do the rollback, and sends a return code to RDS.

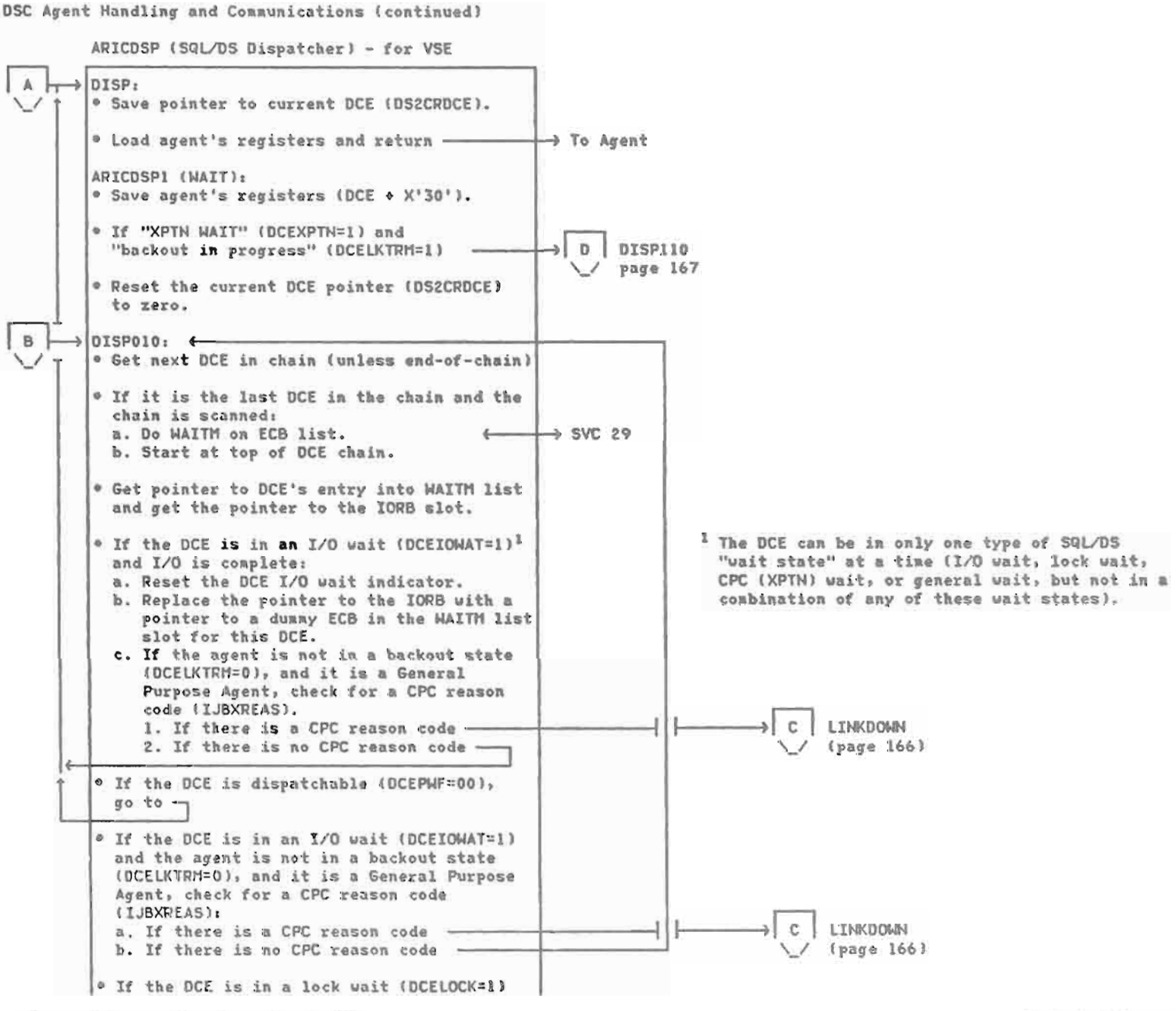
RDS:

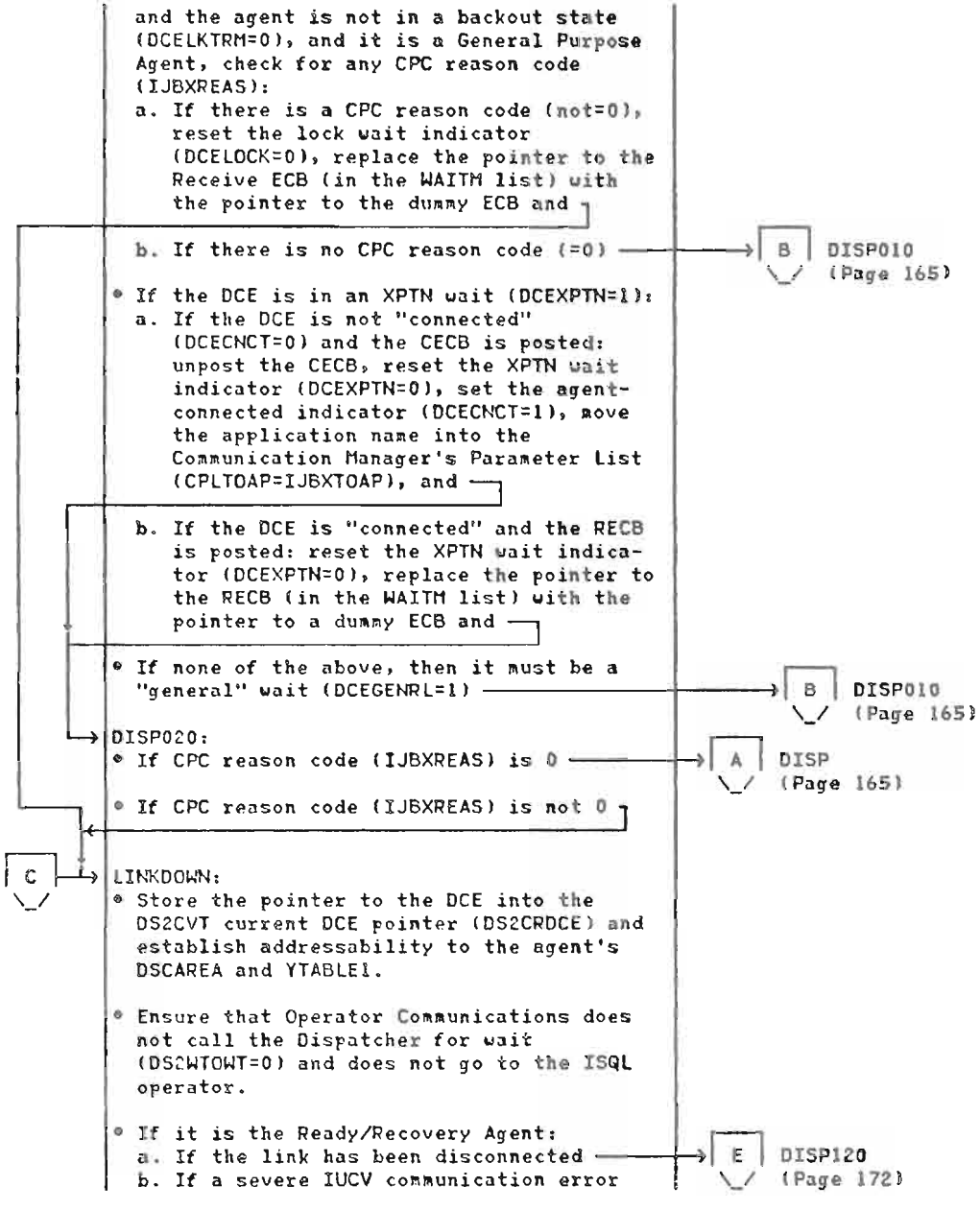
- Issues a reply message with EOP (end-of-process) (ARICOMB)
- Goes to get the next message (ARICMUD)
  - ARICMUD calls ARICWAT
    - ARICWAT finds DCEFORCE=1:
      a. Calls ARIYT15 (NOP as it has been previously called)
      b. Calls ARICDSP1 (Dispatcher wait) with DCEXPTN=1

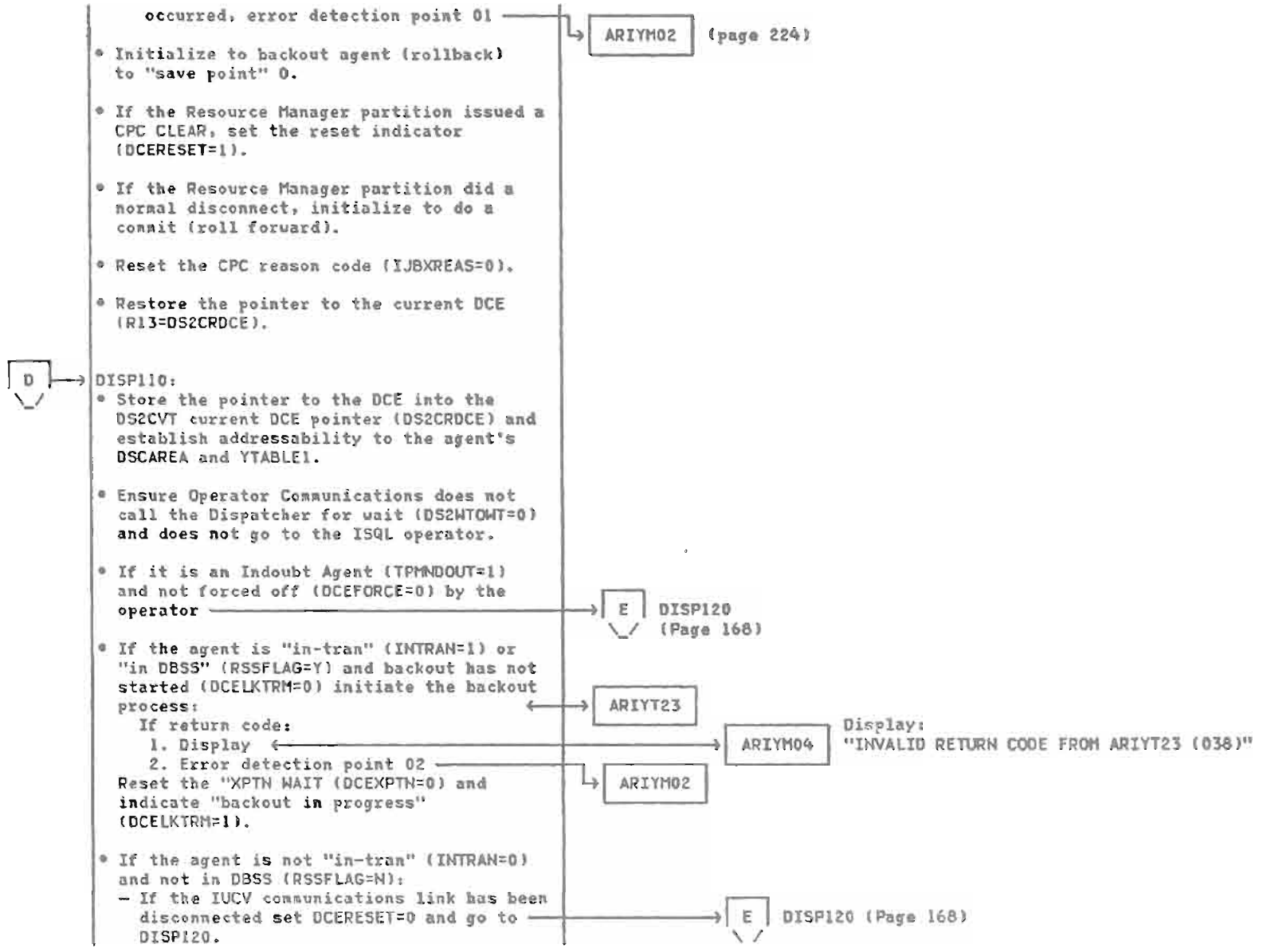
Dispatcher Wait:

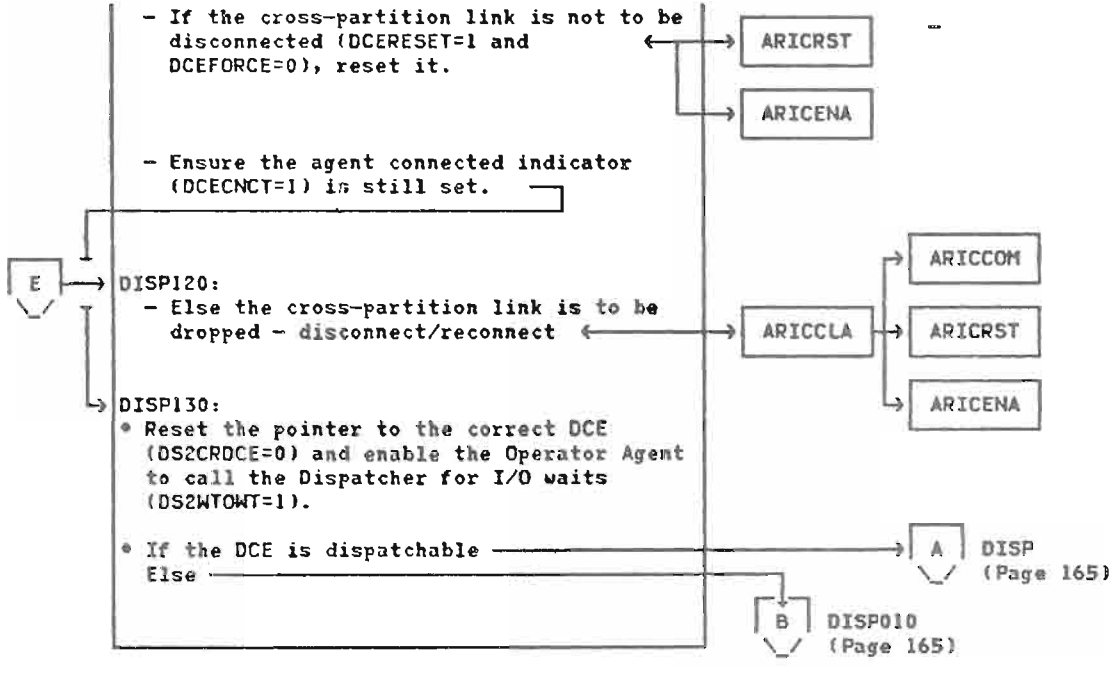
- Finds DCELKTRM=1 and DCEXPTN=1; goes to DISP110.
- DISP110 finds "not INTRAN" and "RSSFLAG=N" (from rollback process)
  - Calls ARICCLA to disconnect/connect the CPC or IUCV link
    WAITM or WAITECB on CECB (or dispatches if connected)

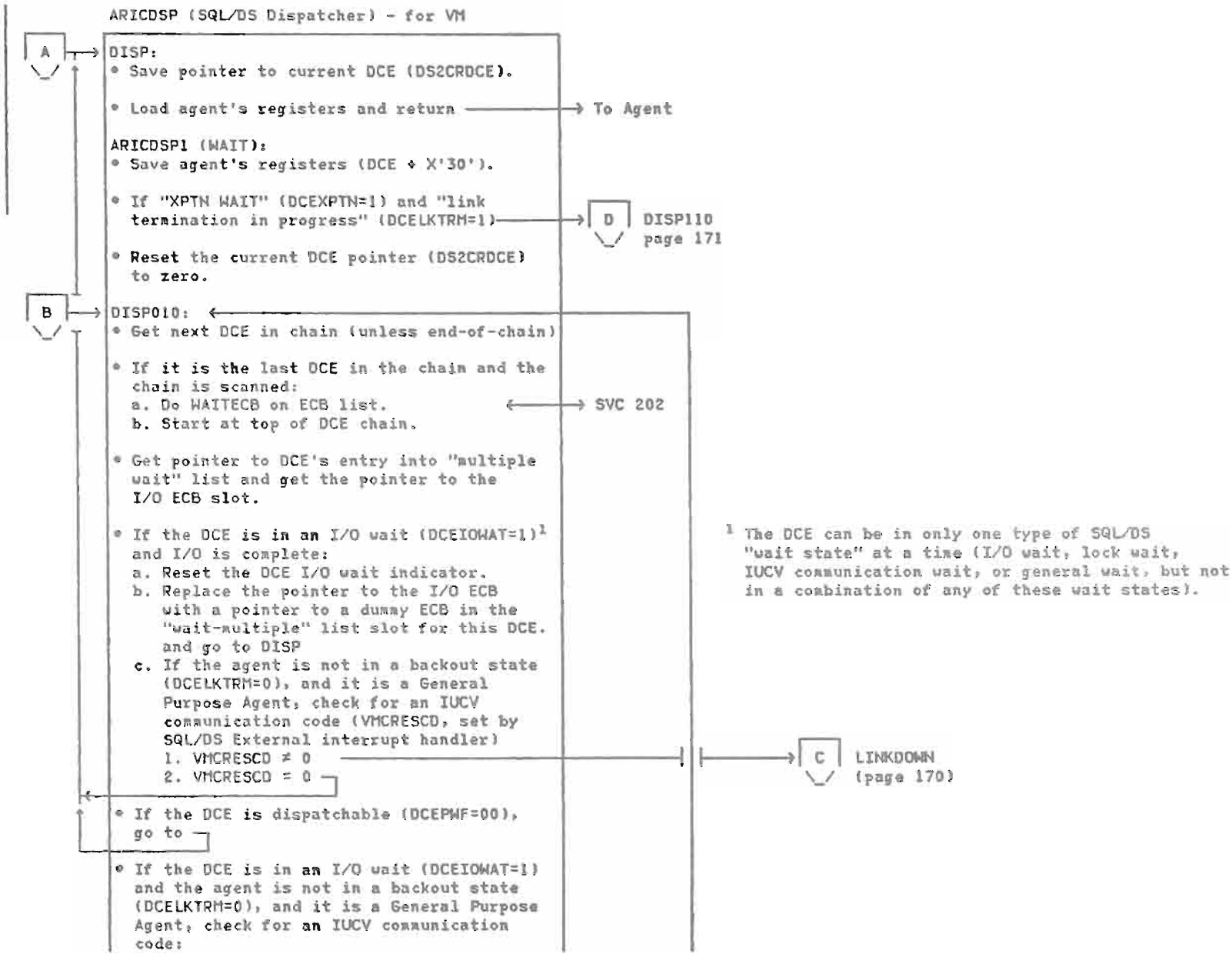


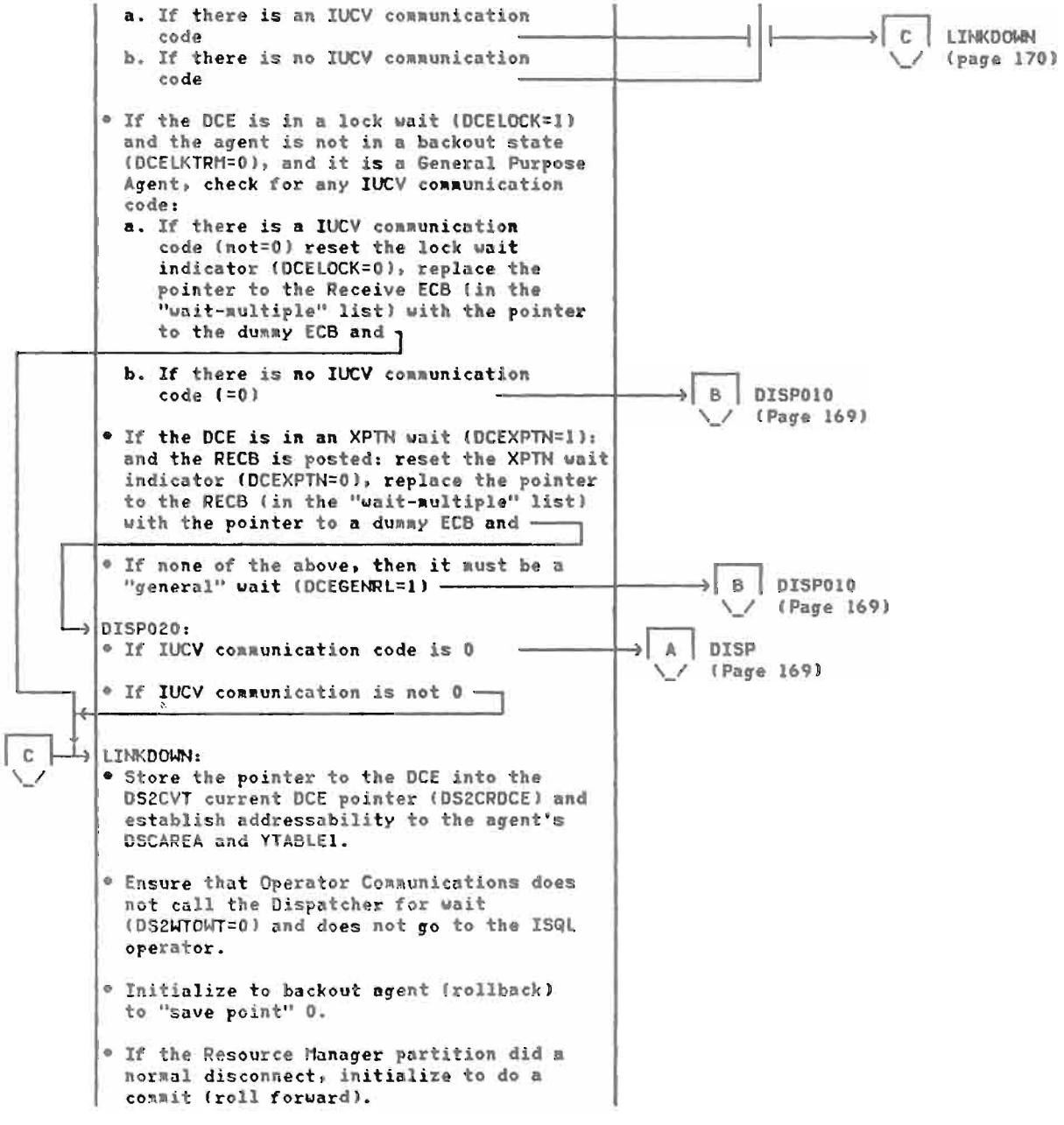

ARICDSP (SQL/DS Dispatcher) - for VSE
A
DISP:
• Save pointer to current DCE (DS2CRDCE).
• Load agent's registers and return
To Agent
ARICDSP1 (WAIT):
• Save agent's registers (DCE + X'30').
• If "XPTN WAIT" (DCEXPTN=1) and "backout in progress" (DCELKTRM=1)
D
DISP110
page 167
• Reset the current DCE pointer (DS2CRDCE) to zero.
B
DISP010:
• Get next DCE in chain (unless end-of-chain)
• If it is the last DCE in the chain and the chain is scanned:
a. Do WAITM on ECB list.
SVC 29
b. Start at top of DCE chain.
• Get pointer to DCE's entry into WAITM list and get the pointer to the IORB slot.
• If the DCE is in an I/O wait (DCEIOWAT=1)[1] and I/O is complete:
a. Reset the DCE I/O wait indicator.
b. Replace the pointer to the IORB with a pointer to a dummy ECB in the WAITM list slot for this DCE.
c. If the agent is not in a backout state (DCELKTRM=0), and it is a General Purpose Agent, check for a CPC reason code (IJBXREAS).
1. If there is a CPC reason code
2. If there is no CPC reason code
C
LINKDOWN
(page 166)
• If the DCE is dispatchable (DCEPWF=00), go to
• If the DCE is in an I/O wait (DCEIOWAT=1) and the agent is not in a backout state (DCELKTRM=0), and it is a General Purpose Agent, check for a CPC reason code (IJBXREAS):
a. If there is a CPC reason code
b. If there is no CPC reason code
C
LINKDOWN
(page 166)
• If the DCE is in a lock wait (DCELOCK=1)
[1] The DCE can be in only one type of SQL/DS "wait state" at a time (I/O wait, lock wait, CPC (XPTN) wait, or general wait, but not in a combination of any of these wait states).

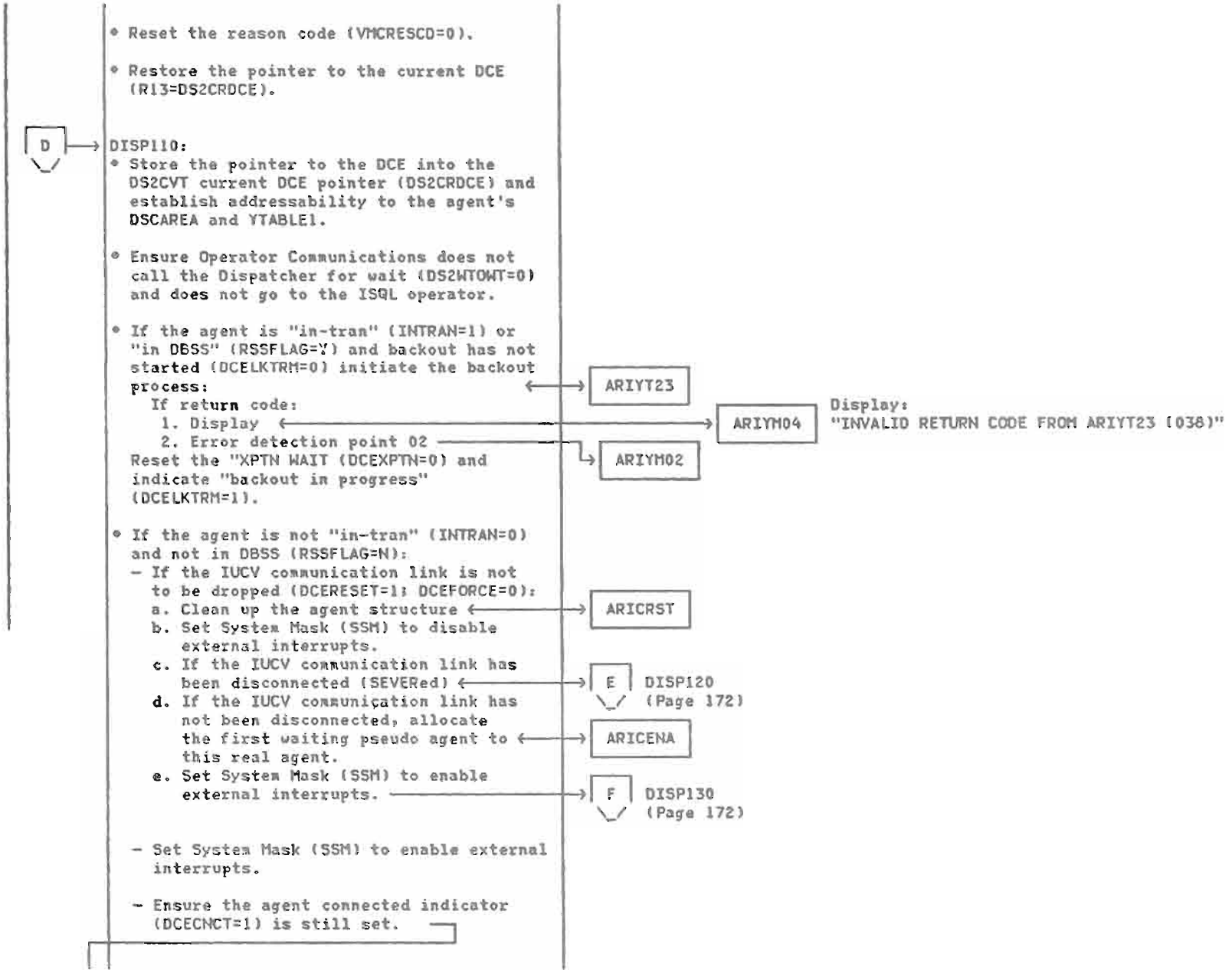


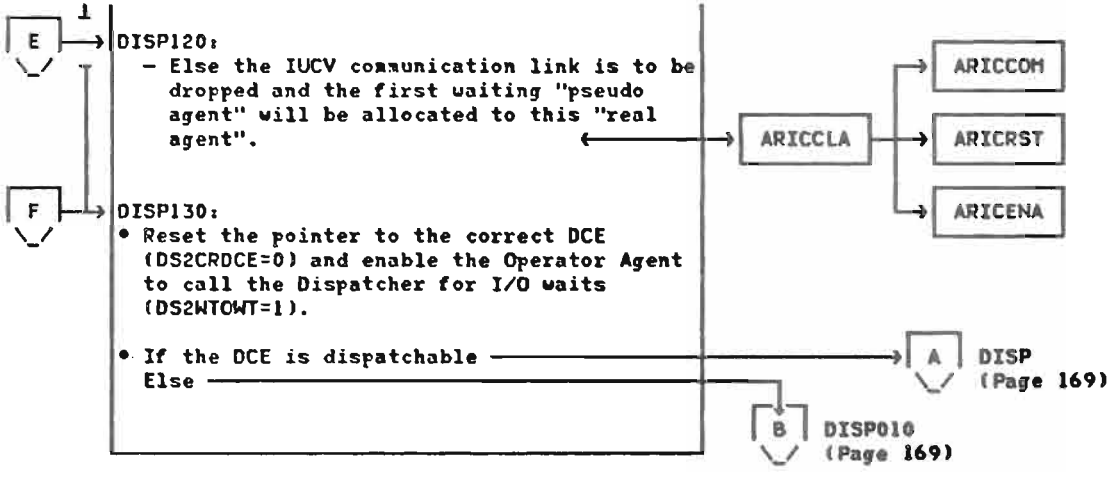

occurred, error detection point 01
ARIYM02
(page 224)
• Initialize to backout agent (rollback) to "save point" 0.
• If the Resource Manager partition issued a CPC CLEAR, set the reset indicator (DCERESET=1).
• If the Resource Manager partition did a normal disconnect, initialize to do a commit (roll forward).
• Reset the CPC reason code (IJBXREAS=0).
• Restore the pointer to the current DCE (R13=DS2CRDCE).
D
DISP110:
• Store the pointer to the DCE into the DS2CVT current DCE pointer (DS2CRDCE) and establish addressability to the agent's DSCAREA and YTABLE1.
• Ensure Operator Communications does not call the Dispatcher for wait (DS2WTOWT=0) and does not go to the ISQL operator.
• If it is an Indoubt Agent (TPMNDOUT=1) and not forced off (DCEFORCE=0) by the operator
E
DISP120
(Page 168)
• If the agent is "in-tran" (INTRAN=1) or "in DBSS" (RSSFLAG=Y) and backout has not started (DCELKTRM=0) initiate the backout process:
ARIYT23
If return code:
1. Display
ARIYM04
Display:
"INVALID RETURN CODE FROM ARIYT23 (038)"
2. Error detection point 02
ARIYM02
Reset the "XPTN WAIT (DCEXPTN=0) and indicate "backout in progress" (DCELKTRM=1).
• If the agent is not "in-tran" (INTRAN=0) and not in DBSS (RSSFLAG=N):
- If the IUCV communications link has been disconnected set DCERESET=0 and go to DISP120.
E
DISP120 (Page 168)




ARICDSP (SQL/DS Dispatcher) - for VM
A
DISP:
• Save pointer to current DCE (DS2CRDCE).
• Load agent's registers and return
To Agent
ARICDSP1 (WAIT):
• Save agent's registers (DCE + X'30').
• If "XPTN WAIT" (DCEXPTN=1) and "link termination in progress" (DCELKTRM=1)
D
DISP110
page 171
• Reset the current DCE pointer (DS2CRDCE) to zero.
B
DISP010:
• Get next DCE in chain (unless end-of-chain)
• If it is the last DCE in the chain and the chain is scanned:
a. Do WAITECB on ECB list.
SVC 202
b. Start at top of DCE chain.
• Get pointer to DCE's entry into "multiple wait" list and get the pointer to the I/O ECB slot.
• If the DCE is in an I/O wait (DCEIOWAT=1)[1] and I/O is complete:
a. Reset the DCE I/O wait indicator.
b. Replace the pointer to the I/O ECB with a pointer to a dummy ECB in the "wait-multiple" list slot for this DCE. and go to DISP
c. If the agent is not in a backout state (DCELKTRM=0), and it is a General Purpose Agent, check for an IUCV communication code (VMCRESCD, set by SQL/DS External interrupt handler)
1. VMCRESCD ≠ 0
2. VMCRESCD = 0
C
LINKDOWN
(page 170)
• If the DCE is dispatchable (DCEPWF=00), go to
• If the DCE is in an I/O wait (DCEIOWAT=1) and the agent is not in a backout state (DCELKTRM=0), and it is a General Purpose Agent, check for an IUCV communication code:
[1] The DCE can be in only one type of SQL/DS "wait state" at a time (I/O wait, lock wait, IUCV communication wait, or general wait, but not in a combination of any of these wait states).




* Reset the reason code (VMCRESCD=0).
* Restore the pointer to the current DCE (R13=DS2CRDCE).
D
DISP110:
* Store the pointer to the DCE into the DS2CVT current DCE pointer (DS2CRDCE) and establish addressability to the agent's DSCAREA and YTABLE1.
* Ensure Operator Communications does not call the Dispatcher for wait (DS2WTOWT=0) and does not go to the ISQL operator.
* If the agent is "in-tran" (INTRAN=1) or "in DBSS" (RSSFLAG=Y) and backout has not started (DCELKTRM=0) initiate the backout process:
ARIYT23
If return code:
1. Display
ARIYM04
Display:
"INVALID RETURN CODE FROM ARIYT23 (038)"
2. Error detection point 02
ARIYM02
Reset the "XPTN WAIT (DCEXPTN=0) and indicate "backout in progress" (DCELKTRM=1).
* If the agent is not "in-tran" (INTRAN=0) and not in DBSS (RSSFLAG=N):
- If the IUCV communication link is not to be dropped (DCERESET=1; DCEFORCE=0):
a. Clean up the agent structure
ARICRST
b. Set System Mask (SSM) to disable external interrupts.
c. If the IUCV communication link has been disconnected (SEVERed)
E
DISP120
(Page 172)
d. If the IUCV communication link has not been disconnected, allocate the first waiting pseudo agent to this real agent.
ARICENA
e. Set System Mask (SSM) to enable external interrupts.
F
DISP130
(Page 172)
- Set System Mask (SSM) to enable external interrupts.
- Ensure the agent connected indicator (DCECNCT=1) is still set.



ARICDWT (Dispatcher Wait Routine)

- If the wait request is a "gate wait" or "latch wait", set the lock wait indicator (DCELOCK = 1) and store the pointer to the Receive ECB in a "wait-multiple" list RECB slot for the agent.
- If the wait request is an "I/O wait", set the I/O wait indicator (DCEIOWAT = 1) and store the pointer to the IORB (VSE) or I/O ECB (VM) in a "wait-multiple" list IORB or I/O ECB slot for the agent.
- If the wait request is not a "gate wait", [1] a "latch wait", an "I/O wait", nor a "suspend wait", set the "general wait" indicator (DCEGENRL).
- Ensure that all DCEs are scanned [2] (DS2DCESC = 1) and wait for the "event" to complete.

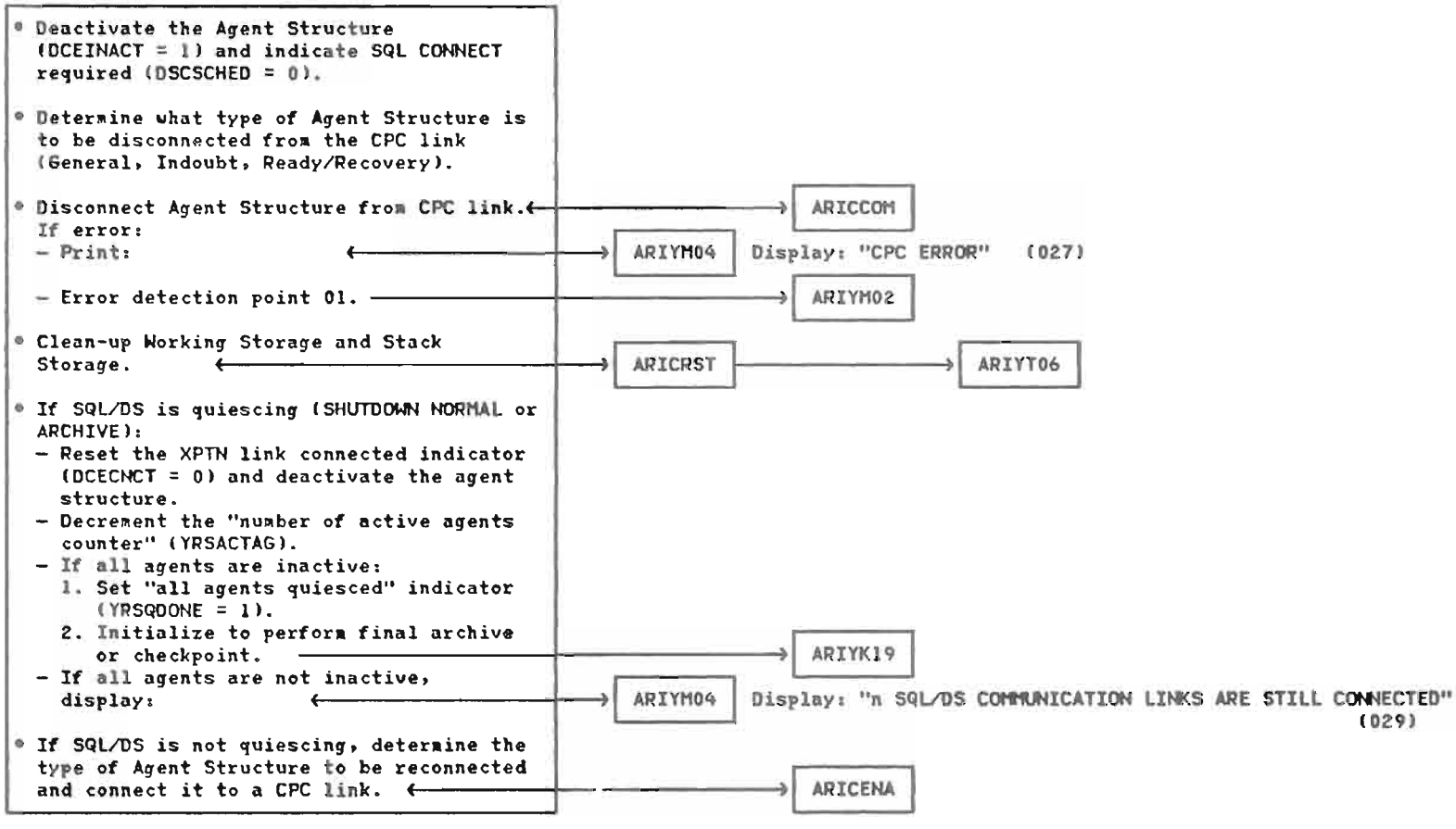
[1] A "suspend wait" request is a wait request that does not set the DCE wait indicators, that is, it leaves the DCE marked as dispatchable. Its purpose is to allow temporary suspension of a long running process (for example, index search or sort) to give up control to allow other SQL/DS agents to be dispatched.

[2] When control is returned to the SQL/DS Dispatcher, the next DCE in the chain is obtained and checked if dispatchable. Once the end of the DCE chain is reached, the DCE at the start of the chain will be obtained and checked if dispatchable. The scan will continue until the end of the DCE chain is reached a second time (assuming no dispatchable DCEs were found). At this point the dispatcher will issue a WAITM (VSE) or WAITECB (VM) macro instruction on the list of ECBs in the a "wait-multiple" list.

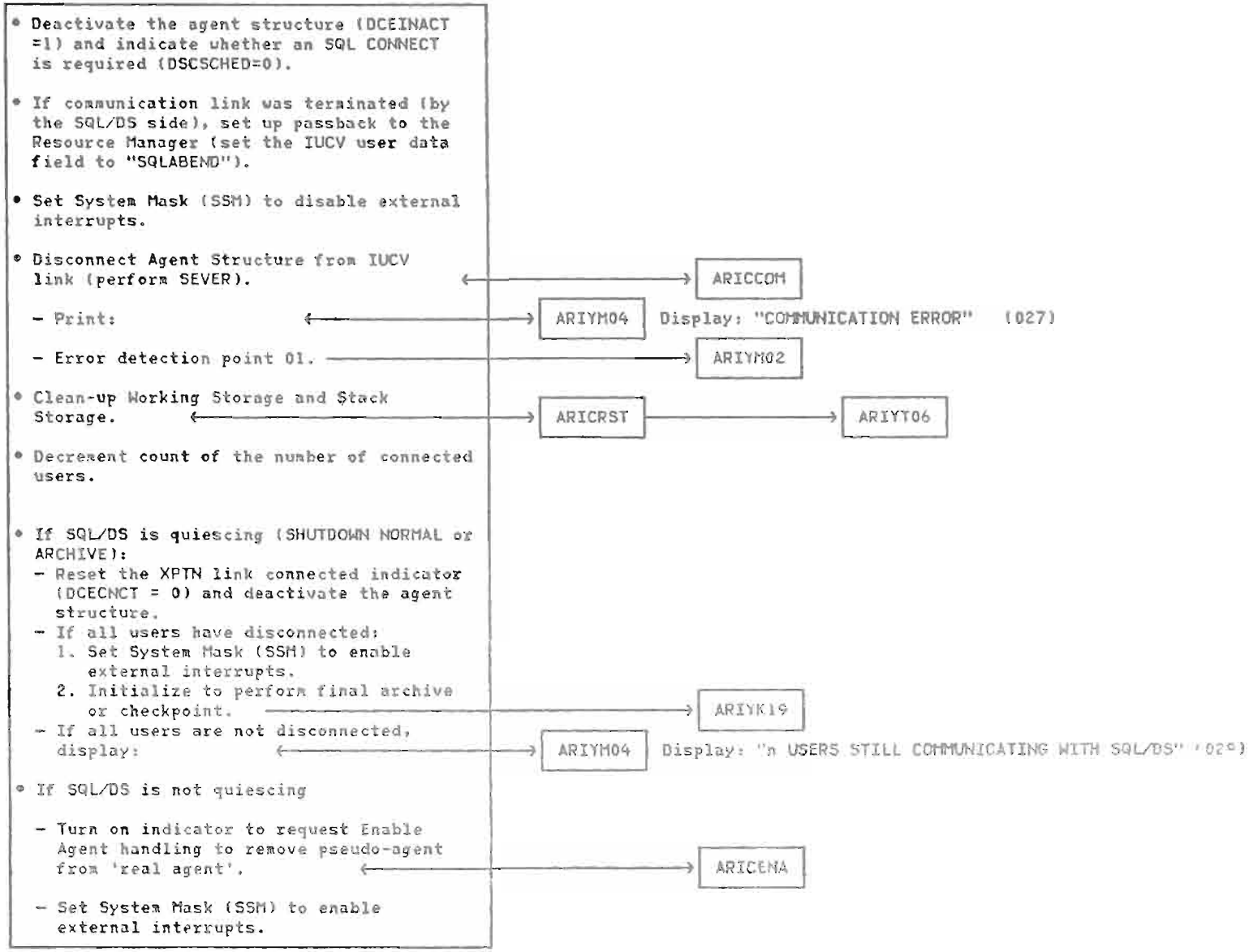
ARICDPT (Dispatcher Post Routine)

- If the request is not for a "gate wait" or "latch wait" post, reset the "general wait" indicator (DCEGENRL = 0).
- If the request is for a "gate wait" or "latch wait" post, reset the "lock wait" indicator (DCELOCK = 0) and put a pointer to the dummy ECB into a "wait-multiple" list receive ECB slot for this agent.
- Tell the dispatcher to scan the DCE chain for a dispatchable agent (DS2DCESC = 1).

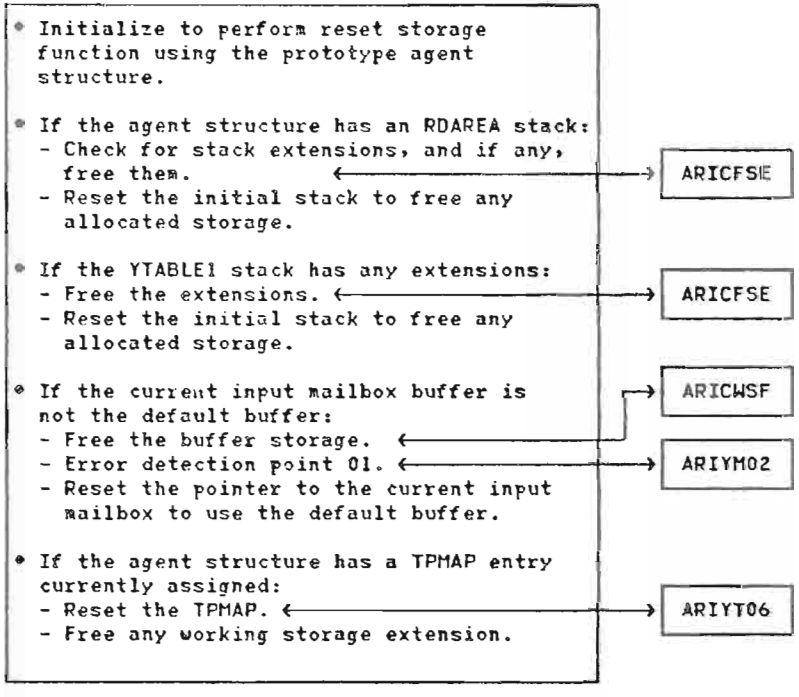
ARICCLA (Clean-up Agent) - for VSE



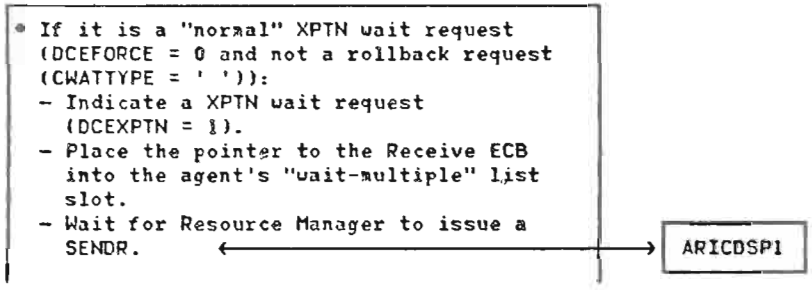
ARICCLA (Clean-up Agent) - for VM

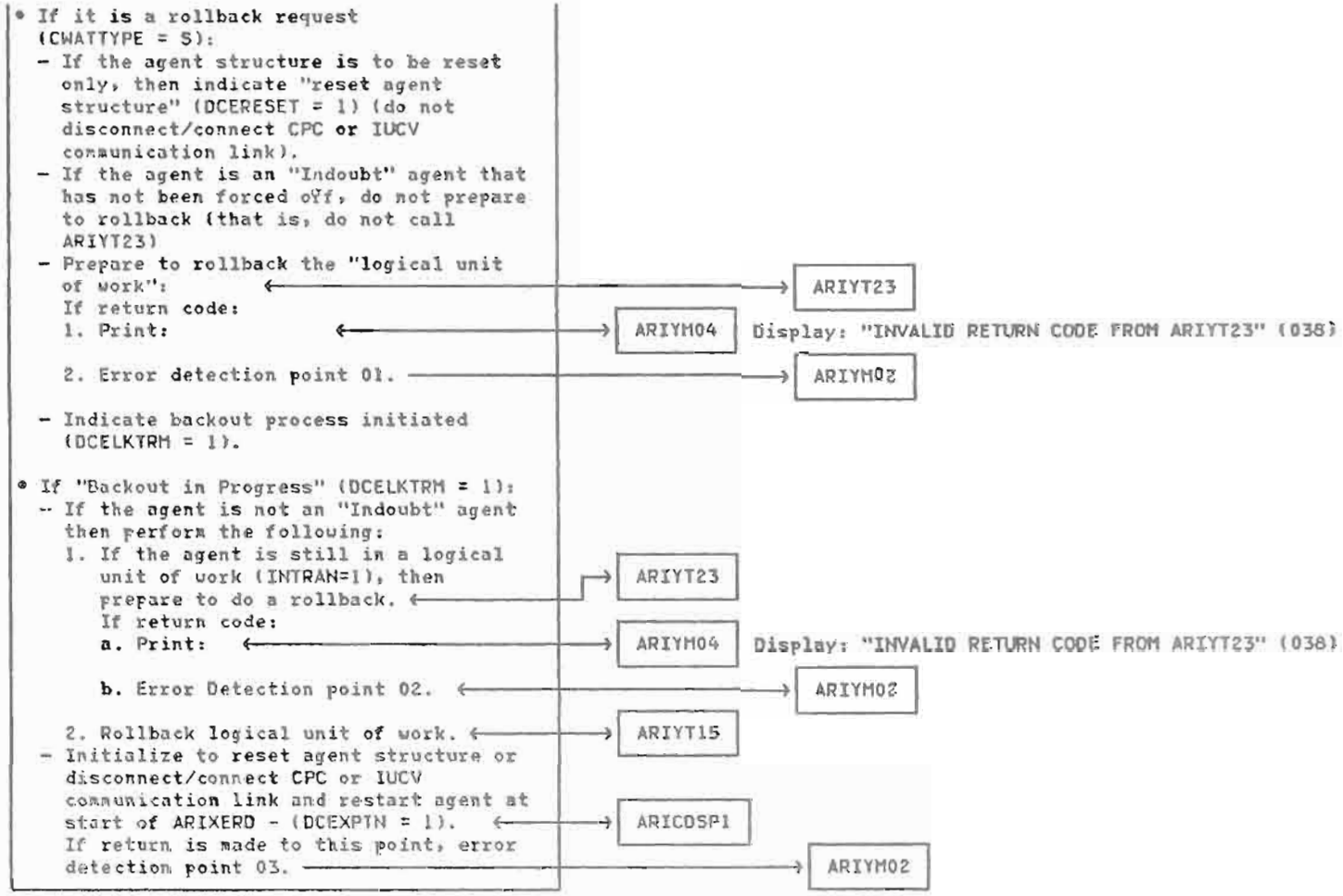


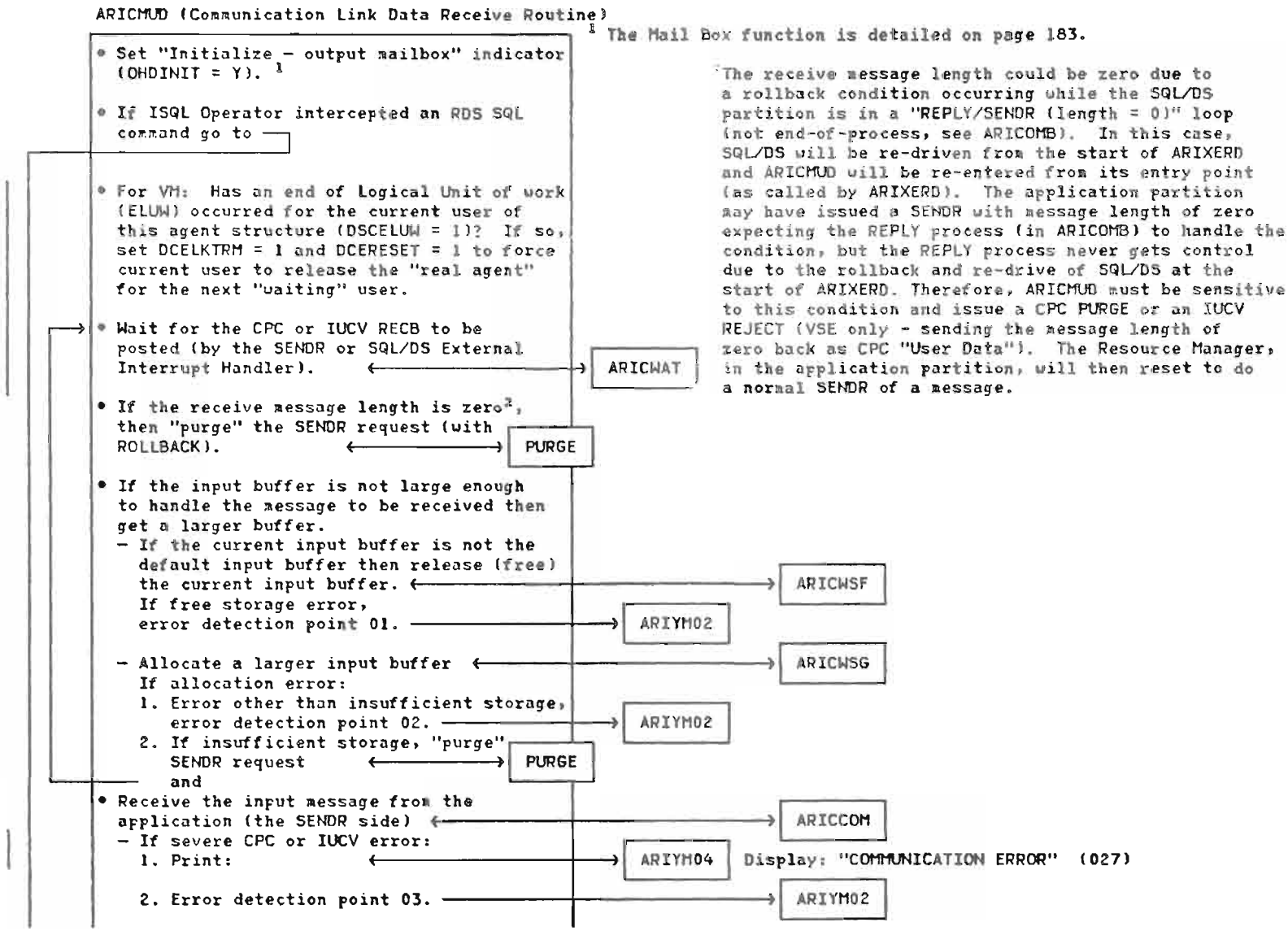
ARICRST (Reset Storage Routine)



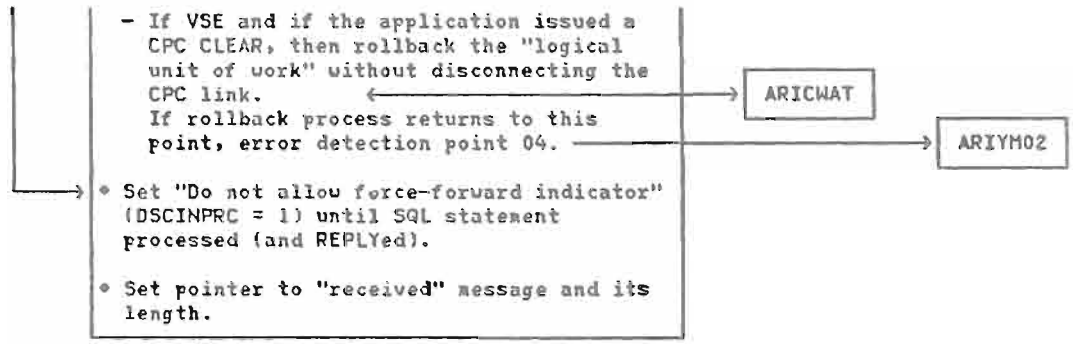
ARICWAT (Cross-Partition Wait Routine)



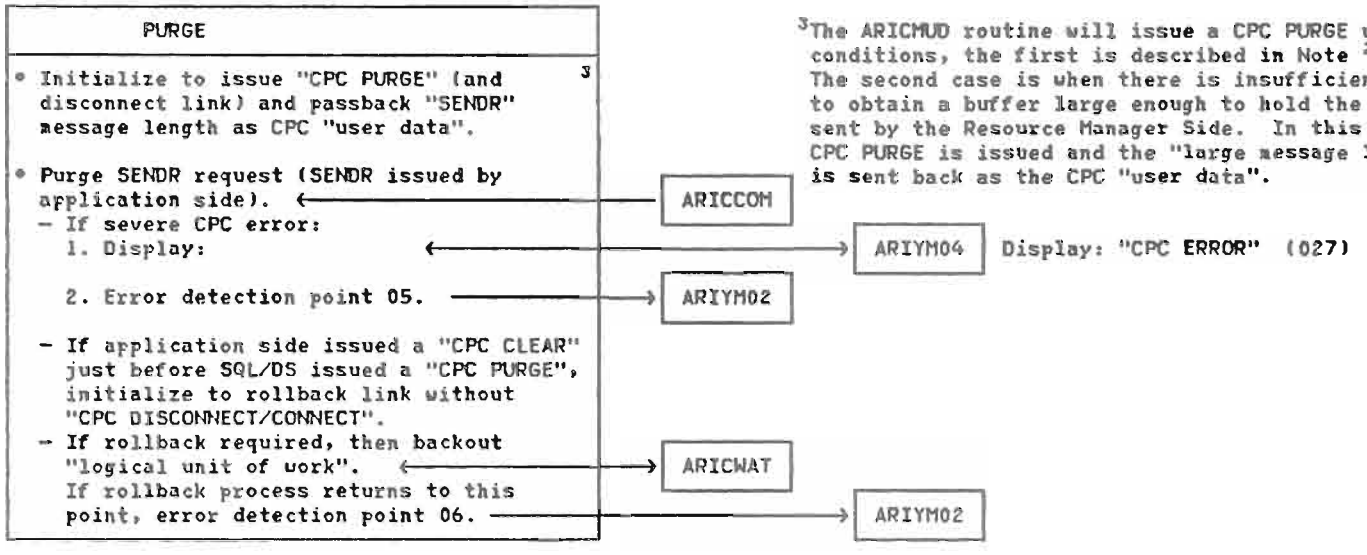

* If it is a rollback request (CWATTYPE = 5):
- If the agent structure is to be reset only, then indicate "reset agent structure" (DCERESET = 1) (do not disconnect/connect CPC or IUCV communication link).
- If the agent is an "Indoubt" agent that has not been forced off, do not prepare to rollback (that is, do not call ARIYT23)
- Prepare to rollback the "logical unit of work":
ARIYT23
If return code:
1. Print:
ARIYM04
Display: "INVALID RETURN CODE FROM ARIYT23" (038)
2. Error detection point 01.
ARIYM02
- Indicate backout process initiated (DCELKTRM = 1).
* If "Backout in Progress" (DCELKTRM = 1):
- If the agent is not an "Indoubt" agent then perform the following:
1. If the agent is still in a logical unit of work (INTRAN=1), then prepare to do a rollback.
ARIYT23
If return code:
a. Print:
ARIYM04
Display: "INVALID RETURN CODE FROM ARIYT23" (038)
b. Error Detection point 02.
ARIYM02
2. Rollback logical unit of work.
ARIYT15
- Initialize to reset agent structure or disconnect/connect CPC or IUCV communication link and restart agent at start of ARIXERD - (DCEXPIN = 1).
ARICOSP1
If return is made to this point, error detection point 03.
ARIYM02



DSC Agent Handling and Communications (continued)

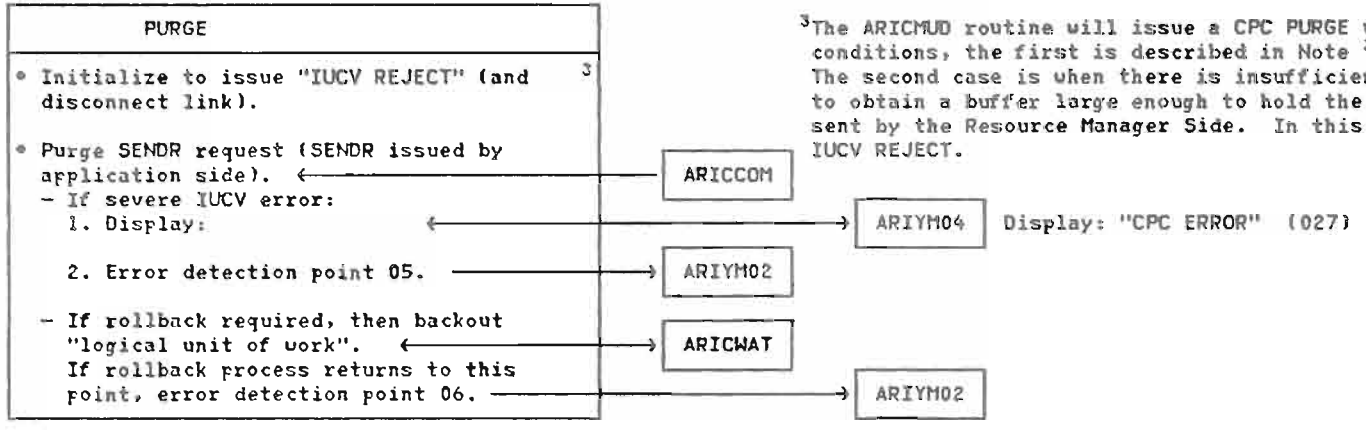


ARICMUD Subroutine (for VSE)



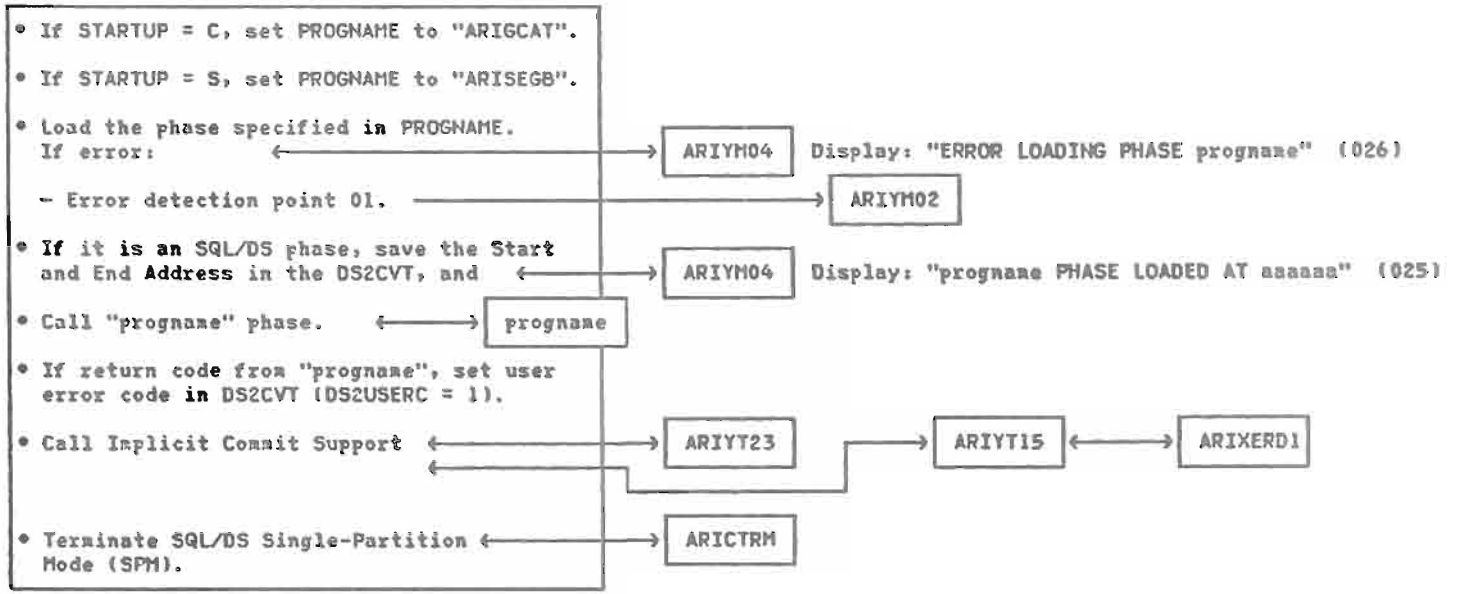
[3]The ARICMUD routine will issue a CPC PURGE under two conditions, the first is described in Note [2] above. The second case is when there is insufficient storage to obtain a buffer large enough to hold the message sent by the Resource Manager Side. In this case, a CPC PURGE is issued and the "large message length" is sent back as the CPC "user data".

ARICMUD Subroutine (for VM)

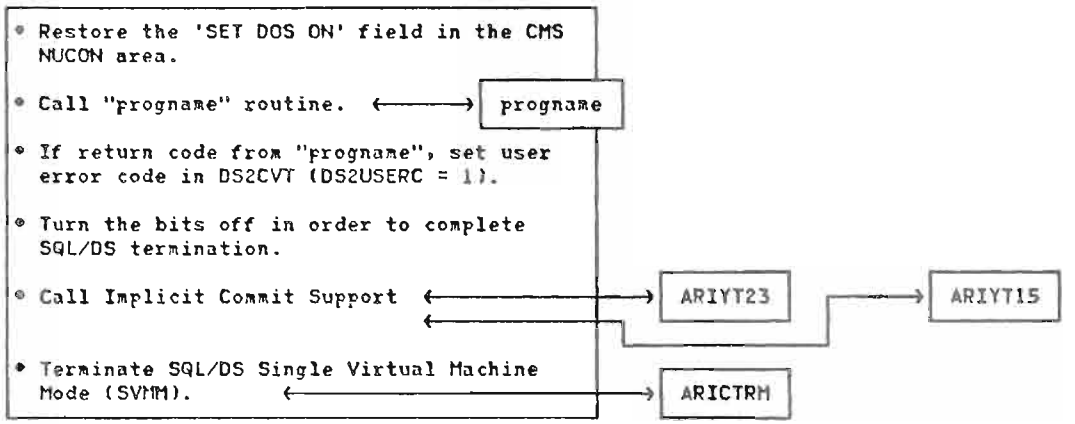


[3]The ARICMUD routine will issue a CPC PURGE under two conditions, the first is described in Note [2] above. The second case is when there is insufficient storage to obtain a buffer large enough to hold the message sent by the Resource Manager Side. In this case, a IUCV REJECT.

ARICSPM (Single Partition Mode Routine) - VSE

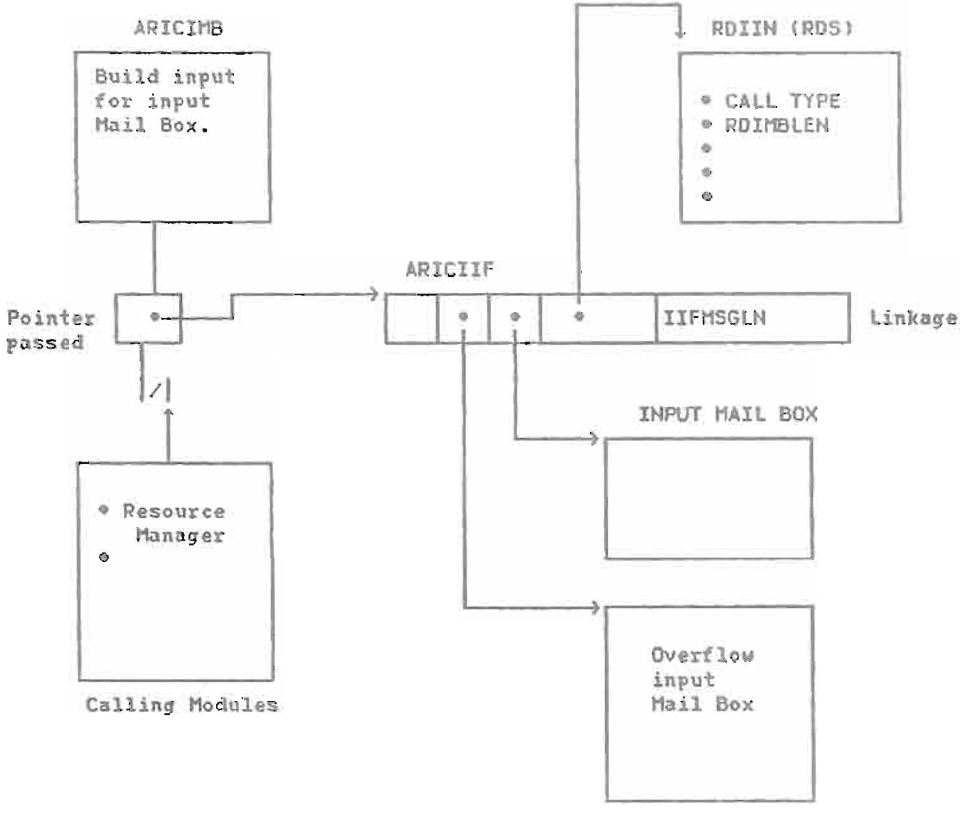


ARICSPM (Single Virtual Machine Mode) - VM



ARICIMB - Build Input Mail Box

ARICIMB resides in the application side partition. Its function is to compact all the input data (by user) and place it into a buffer (input Mail Box buffer) in the same sequence requested. This module also manages the buffer allocation and deallocation (input Mail Box) when the original buffer is not big enough for input request. (The original buffer is allocated by the Resource Manager.) ARICIMB builds the input Mail Box upon different opcodes passed to it (up to 14 opcodes). All the input data information exists in RDIIN control block (RDS). Pointer to RDIIN resides in ARICIIF.



The following are the opcodes (call types) passed to ARICIMB:

CALL TYPE (RDICTYPE)	MEANING
30	AUX CALL
35	SET UP CALL
40	DESCRIBE CALL
45	CLOSE CALL
50	OPEN CALL
120	SCHEDULE OR CONNECT CALL
125	RECOVERY CALL
130	PREP INIT CALL
131	CREATE PROGRAM CALL
132	DROP STATEMENT CALL
135	LOOKUP CALL
140	SQL CALL
145	FINISH CALL
155	OPERATOR MESSAGE CALL
160	RECOVERY (INDOUBT) CALL
170	OPERATOR CONTINUE CALL

ARICIMB has to know the size of input to Mail Box (RDIMBLEN in RDIIN) in order to use the right buffer for input mail box (overflow or default buffer size). Also, it has to inform the Resource Manager which buffer has been used for Mail Box. There is a message size length in ARICIIF (IIFMSGLN). If IIFMSGLN is greater than the default Mail Box size, then the overflow buffer is used as the mailbox, otherwise the default Mail Box. (Default buffer size is declared externally in ARICIMB). If RDIMBLEN has not been calculated, then ARICIMB will find the size of input (first pass).

After building the input Mail Box ARICIMB will return to the Resource Manager.

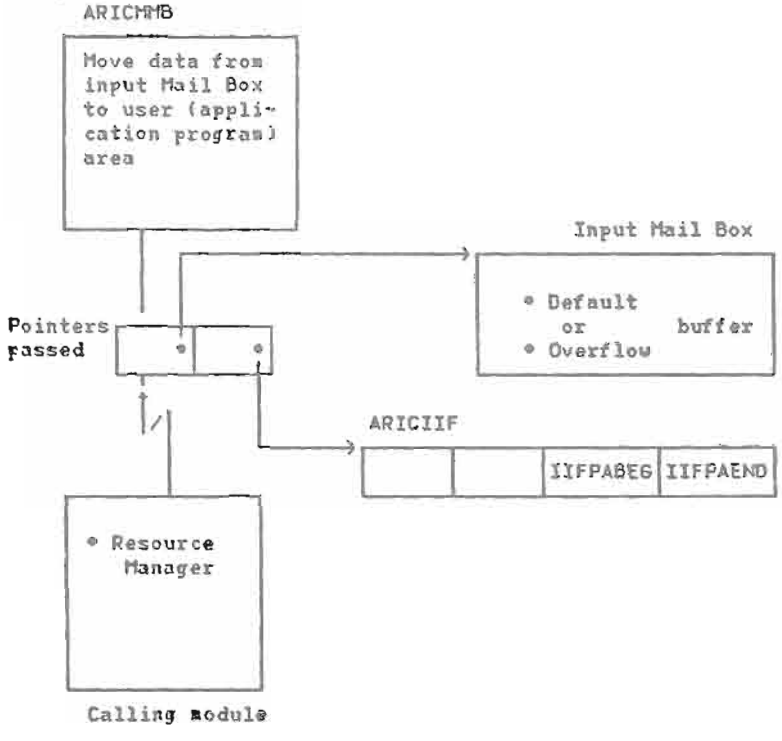
ARICMMB - Move data to user (application program) area (in application side)

ARICMMB resides in the Resource Manager partition or virtual machine. Its function is to decompact the data passed by SQL/DS to the input Mail Box and replace it in the user (application program) area.

The pointer to the input Mail Box is passed by the Resource Manager.

The Resource Manager is the linkage between the application program and ARICMMB.

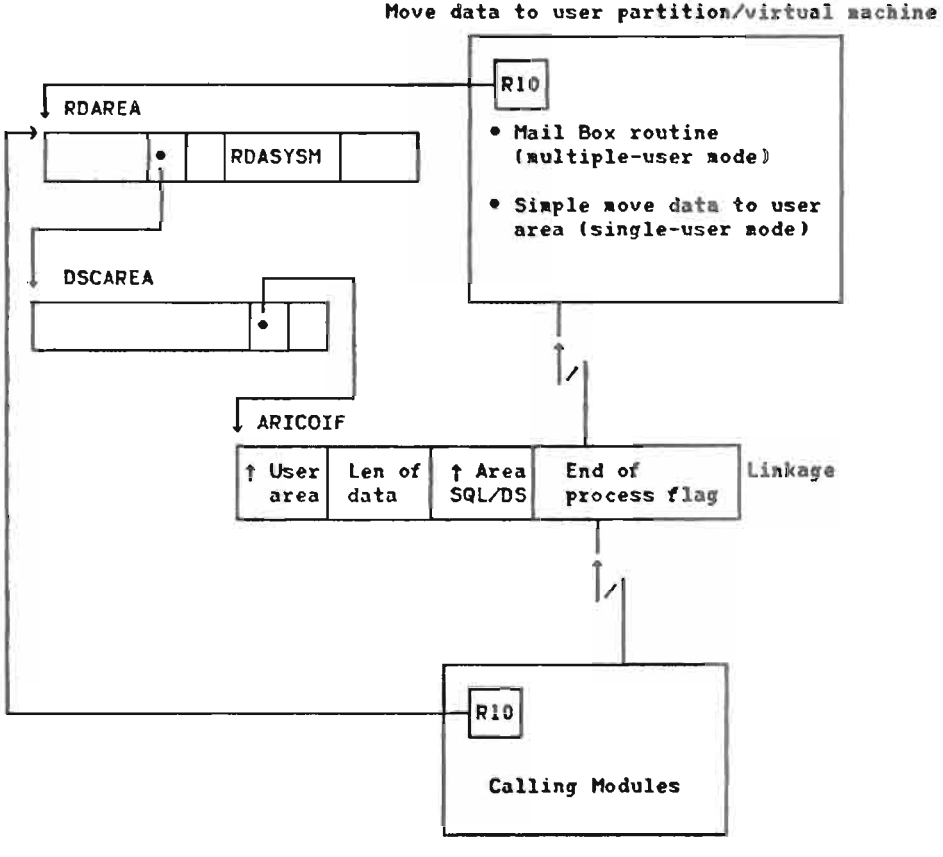
Return Codes are:
- 0 Successful
- -1 Address boundary violation



ARICOMB - Send data to user (application program) partition or virtual machine.

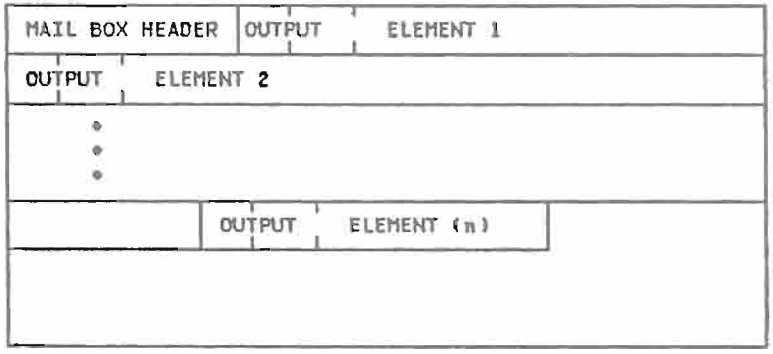
ARICOMB resides in the SQL/DS partition (VSE) or virtual machine (VM). Its function is to "get" the SQL/DS output to the user (application program) area. ARICOMB is sensitive to running-mode environment. There is a flag (RDASYSM) in RDAREA which states the running environment of the system ('S' = single-user mode or 'M' = multiple-user mode). If SQL/DS is running in single-user mode, then ARICOMB simply moves the data to the user area (no Mail Box). However, in a multiple-user mode environment, ARICOMB would use the Mail Box facility to pass the data to the user partition or virtual machine. The main functions when using the Mail Box facility in a multiple-user mode environment are:

1 - Build an output element in output Mail Box buffer
2 - Pass the output data (Mail Box) via the communication link.



Layout of Output Mail Box Contents

Output Mail Box Buffer



Output Element (in OUTBUILD Mail Box)

Transfer of output Mail Box would happen in two cases:

1. The process is done and complete.
2. It is not the end of process but the output Mail box buffer is full (SEGMENTATION).

The following would cause the second case (SEGMENTATION):

1 - The output message length is greater than output Mail Box buffer size
2 - There is not enough room in output Mail Box buffer

In the above cases ARICOMB would segment the output data to the Mail Box capacity and then ships the Mail Box cross over. This action would repeat until the whole output data is transferred. The wait is issued in segmentation process until control is returned. This will insure that output data is received in the other side and placed in the user area.

ARICOIF is the linkage between the caller and ARICOMB when ARICOMB is called to build an output element in the output Mail Box or to do a simple move (single user mode). The pointer to input parameters resides in DSCAREA (DSCOMBPP).

* From ARIYT00
ARICTRI (Trace Initialization)

ARICTRI is called during SQL/DS initialization if either the TRACDBSS or the TRACRDS parameter was specified as a SQL/DS initialization parameter. ARICTRI supports SQL/DS trace activation via the SQL/DS initialization process (as opposed to the TRACE operator command). ARICTRI prompts the operator to:

- Allocate a currently unassigned tape unit to SQL/DS for tracing
- Ready the unit (with a scratch volume)
- Reply with the tape unit cuu.

For hex to binary conversion (cuu message prompt input) call ←→ **ARIYE06** Hex to Binary Conversion

ARICTRI then:

- Causes the trace output file to be dynamically assigned (VSE only) and opened. ←→ **ARISYSD5** System Dependent Input/Output
  If the open fails an error message is diplayed and the operator is again prompted (see above).
- Updates the DS2CVT to indicate that trace is now active
- Informs the operator that trace is active
- Returns to the caller. → Return

If the message operator replies CANCEL rather than CUU to the prompting message, the DS2CVT is flagged to indicate that trace is 'OUTPUT DISABLED' (so that the trace service routines can deactivate tracing in each agent structure and so that the trace command processing/shutdown routine can function properly), and SQL/DS initialization continues.

ARIYM04 is called to issue messages to the System Operator for: ←→ **ARIYM04** DBSS/DSC Message Services (via MSG macro)

- Prompting message ARI055A - to

DSC Trace Services (continued)

ready trace tape and reply with cuu
- Error message ARI086E - if invalid reply to message ARI055A
- Message ARI056I - if reply to message ARI056A is CANCEL
- Message ARI094E - if dynamic assign/open for trace output file fails
- Message ARI095I - for normal completion

* From ARIYM11
Entry point ARICTRC1 is called by ARICTRM
ARICTRC (Trace Command Processing)

ARICTRC is invoked to process the SQL/DS TRACE operator command (primary entry point ARICTRC) or to shut down SQL/DS trace when the SQL/DS partition or virtual machine terminates normally or abnormally (secondary entry point ARICTRC1).

If the TRACE ON operator command is specified, the operator is prompted, via reply messages, for the following:
- Whether SQL/DS is to trace only activity for: a specific user-ID, a specific agent number or all agents
- Whether SQL/DS is to trace functions in the DBSS or the RDS or both
- If RDS tracing is requested, which functions are to be traced and the trace level for each
- If DBSS tracing is requested, which functions are to be traced and the trace level for each
- If the DUMP parameter was specified in the TRACE ON command, what trace point number is to cause a SNAP DUMP when first activated.

For TRACE ON, modules ARIYE06, ARIYE04, and ARIYE08 are called. ARIYE06 is used for cuu parameter. ARIYE04 is used to edit terminal input line into token array (for prompting message inputs). ARIYE08 is used for converting inputs agent number and trace point number.

→ ARIYE06 Hex to Binary Conversion

→ ARIYE04 Tokenize Input Line

→ ARIYE08 Decimal to Binary Conversion

When prompting is complete, unless trace is in OFF NOCLOSE state the routine ARISYSD5 is called to dynamically assign (VSE only) and open the trace output tape file. Next, the system control blocks which contain trace control information (YTABLE1 for DBSS agent local, RDAREA for RDS agent local and DS2CVT for SQL/DS global) are updated to indicate that trace is active and to control what DBSS and RDS functions are to be traced. The operator is informed that trace is active.

→ ARISYSD5 System Dependent Input/Output Routine

If the TRACE OFF operator command is specified, the trace control information in system control block (YTABLE1, RDAREA, or DS2CVT) is reset to reflect the TRACE OFF state. If NOCLOSE was specified, the operator is informed that trace is now off but the trace file remains open. If NOCLOSE was omitted, routine ARISYSD5 is called to close and dynamically unassign (VSE only) the trace output tape file, the DS2CVT control block is updated to reflect trace file closed, the block is updated to reflect trace file closed, the operator is informed that trace is off and closed.

→ ARISYSD5 System Dependent Input/Output Routine

If trace lock is held by another agent, ARICDWT is called.

→ ARICDWT DSC Dispatcher External Wait

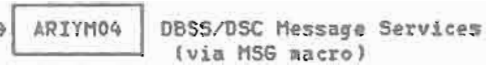
DSC Trace Services (continued)

When the ARICTRC is called during SQL/DS termination (secondary entry point ARICTRC1):

- If trace is not active and closed, the module returns.
- If trace is active or not closed, processing is functionally identical to the TRACE OFF operator command above except that the NOCLOSE specification does not apply.

ARIYM04 is called to issue messages to the operator for:

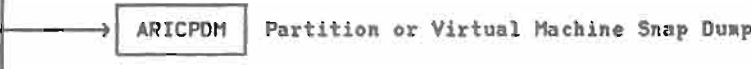
- Various prompting messages for additional TRACE ON parameters (entry point ARICTRC only)
- A message that command or trace shut down was processed successfully
- A message that command conflicts with existing trace status (entry point ARICTRC only)
- A message that command has invalid, missing or extra parameter (entry point ARICTRC only)
- A message that prompt message response is invalid (entry point ARICTRC only)
- A message that operator requested command termination (entry point ARICTRC only)
- A message that trace file dynamic assign (VSE only) or open failed (entry point ARICTRC only)
- A message that trace was output disabled and is now off
- A message that trace file close or dynamic unassign (VSE only) failed



* ARIXETR and ARIYE14 are 'twin' trace services routines for RDS and DBSS, respectively.
ARIXETR is called by a RDS active trace point (via XTRACE macro).
ARIYE14 is called by a DBSS active trace point (via TRACE macro).
ARIXETR/ARIYE14 (Trace Services)

Trace services routines are used to cause trace point data to be collected and written to the Trace Output File. This function is invoked via the XTRACE macro (RDS) or the TRACE macro (DBSS).

On entry tests are made to determine if:
- Trace is active at all
- Active for current User-ID
- Active for the invoking function (If not active, return with no output).
- To determine if a snap dump was requested (via TRACE command) (If snap dump was requested, the SQL/DS partition or virtual machine Snap Dump Routine is called and Snap Dump request is disabled (dump first time only))



Using the caller-provided trace point number, the corresponding trace point descriptor module is located and within the trace point descriptor module, the corresponding trace point descriptor structure (generated by macros TPOINT etc.) is located via the trace point descriptor directory (generated by the TPDIR macro). The trace point descriptor structure, supplemented by caller parameters, describes what data is to be collected and displayed for this trace point activation. A trace point header subrecord containing the trace point number, opcode, user-ID, agent number,

date, time and LUW (DBSS only) is created and written out. Various optional substructures within the trace point descriptor allow display of:

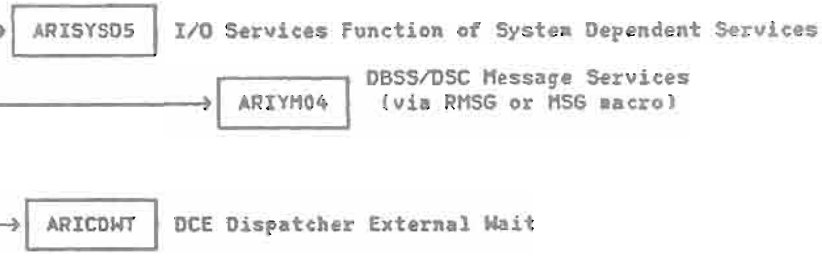
- Trace point module name or entry point name and whether trace point is module entry, module exit or other
- Literal strings which are message "Eye Catchers" for the trace point
- Return code values
- Local variables (addressability is module local)
- Global variables (addressability is global)

For variables, the trace point descriptor supplies:

- The keyword to be associated with the data
- The display format (decimal, character, hex, hex dump, special formatting)
- Minimum trace level required to display the variable.

The address and length of the variable may be supplied by the trace point descriptor or by the caller depending on data addressability and length characteristics. The trace point data is collected into trace point data records in a stylized internal format and written to the trace output file. For trace record output ARISYSD5 is called. If ARISYSD5 call returns an error, an error message is issued via a call to ARIYM04.

On module entry, after it is determined that trace point output is required, if internal trace lock is held by another agent, ARICDWT is called.



* From ARIYM00
ARIYE01 (Trace DBSS Opcode Entry and Exit)

The main entry point is called by ARIYM00 when ARIYM00 is called and DBSS entry tracing is active. Entry point ARIYE011 is called by ARIYM00 when ARIYM00 is returning (DBSS opcode call completion) and DBSS exit tracing is active.

If trace is not active for the current user-ID, the module simply returns.

For DBSS Entry:

- If the entry trace level is 2 and there is a DBSS entry trace routine for the opcode, the routine is called to display the 'opcode name' string, and opcode input variables via a DBSS entry trace point. → DBSS Entry level 2 Opcode-dependent trace routine
- If the entry trace level is 1 or there is no DBSS entry trace routine for the opcode, this module executes a trace point which displays:
  DBSS ENTRY, and
  L_OPCODE='opcode name'
  → ARIYE14 DBSS Trace Service Routine (via TRACE macro)

For DBSS Exit:

- If the exit trace level is 2 and there is a DBSS exit trace routine for the opcode, the routine is called to display the 'opcode name' string, the return code, and opcode output variables via a DBSS exit trace point. → DBSS Exit level 2 Opcode-dependent trace routine
- If the exit trace level is 1 or there is no DBSS exit trace routine for the opcode, this module executes a trace point which displays:
  DBSS EXIT,
  L_OPCODE='opcode name', and
  RETCODE=return-code-value.
  → ARIYE14 DBSS Trace Service Routine (via TRACE macro)

ARICINT (SQL/DS External Interrupt Handler) - VM only

Entry Point - ARICINT

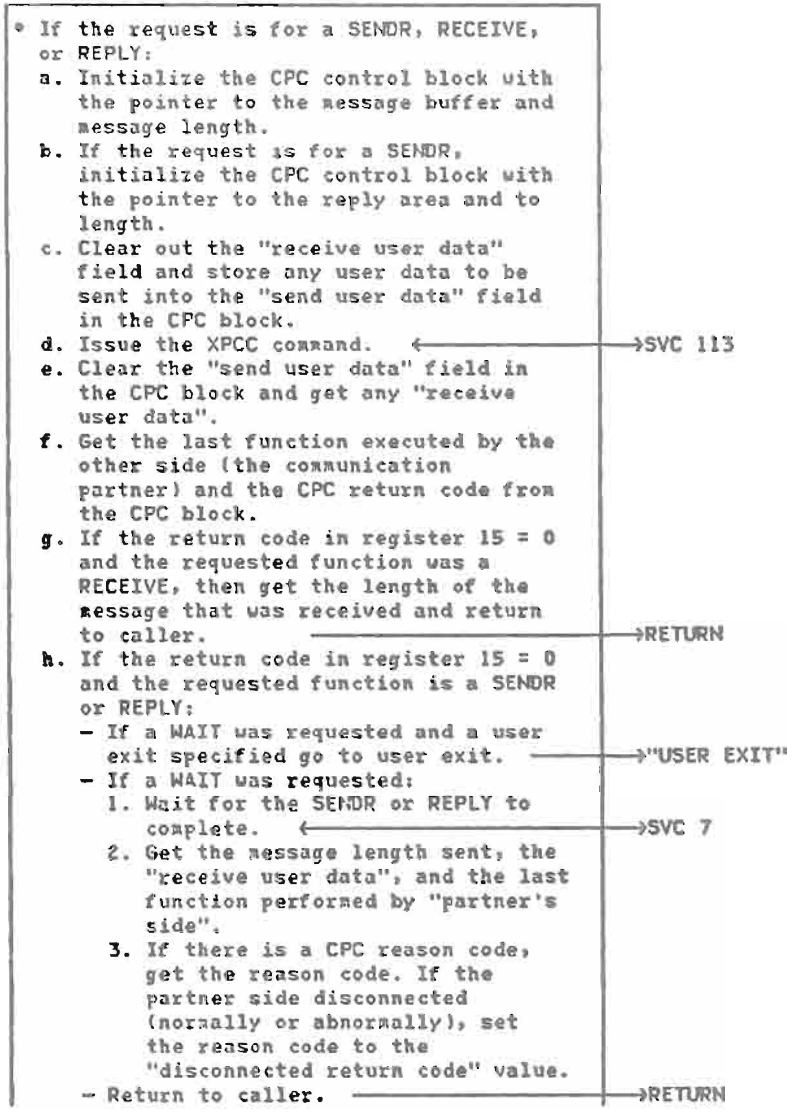
* If interrupt type is 'pending connect', then:
  a. If no pseudo-agent is available, SYSTERR.
  b. Use first available pseudo-agent.
  c. If shutdown is in progress, call communication manager to sever with user data 'SHUTDOWN'.
  d. If running in SVMM, call communication manager to sever with user data 'SVMMODE'.
  e. If dbname being connected to does not match dbname that is running, call communication manager to sever with user data 'WRONGDB'.
  f. If no SEVER or SYSTERR condition, then:
     - Call communication manager to accept connection.
     - Allocate pseudo-agent from available queue and put on in-use queue.
     - Return to caller.

* If interrupt type is 'pending message', then:
  a. Save interrupt information in VMCBLOCK.
  b. Post RECEIVE ECB.
  c. If not already connected to real agent, then:
     - If real agent is available, connect pseudo-agent to real agent.
     - If real agent is not available, put pseudo-agent on waiting queue.
  d. Return to caller.

* If interrupt type is 'severed', then:
  a. Save interrupt information in VMCBLOCK.
  b. Post RECEIVE ECB.
  c. Put translated sever code into VMCBLOCK.
  d. If pseudo-agent is not connected to real-agent, then:
     - If shutdown is in progress and this is last pseudo-agent, then:
       1. Connect pseudo-agent
       2. Return to caller.
     - Call communication manager to do sever.
     - Remove pseudo-agent from in-use queue

```
      and put it on available queue.
  e. Return to caller.

• If interrupt type is any other, SYSTERR.


Entry Point ARICINT1

• If interrupt type is 'completed
  connection', then:
  - Save interrupt information in VMCBLOCK.
  - Save starting and ending block numbers.
  - Save R/O or R/W status.
  - Post CONNECT ECB.

• If interrupt type is 'severed', then save
  interrupt information in VMCBLOCK.

• If interrupt type is 'completed message',
  then:
  - Save interrupt information in VMCBLOCK.
  - Post SEND ECB.

• If interrupt type is an other, SYSTERR
  and return to caller.
```

ARICCOM (SQL/DS Communication Manager) - VSE



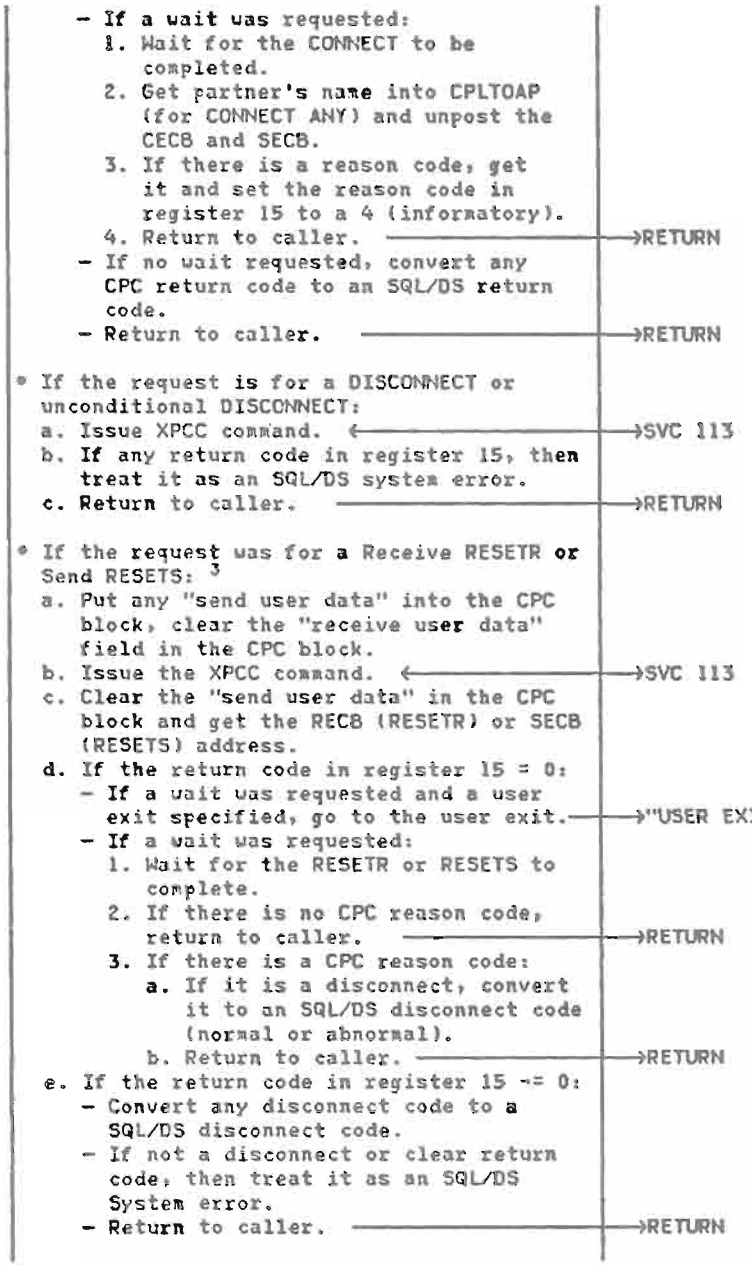
i. If the return code in register 15 ¬= 0:
   - If the other side disconnected, convert the disconnect codes to SQL/DS disconnect codes (normal or abnormal).
   - If the return code was not a disconnect or a clear, treat the return code as an SQL/DS system error.
   - Return to caller. ——→RETURN

* If the request is for a WAIT then wait for the ECB to be posted. ←——→SVC 7

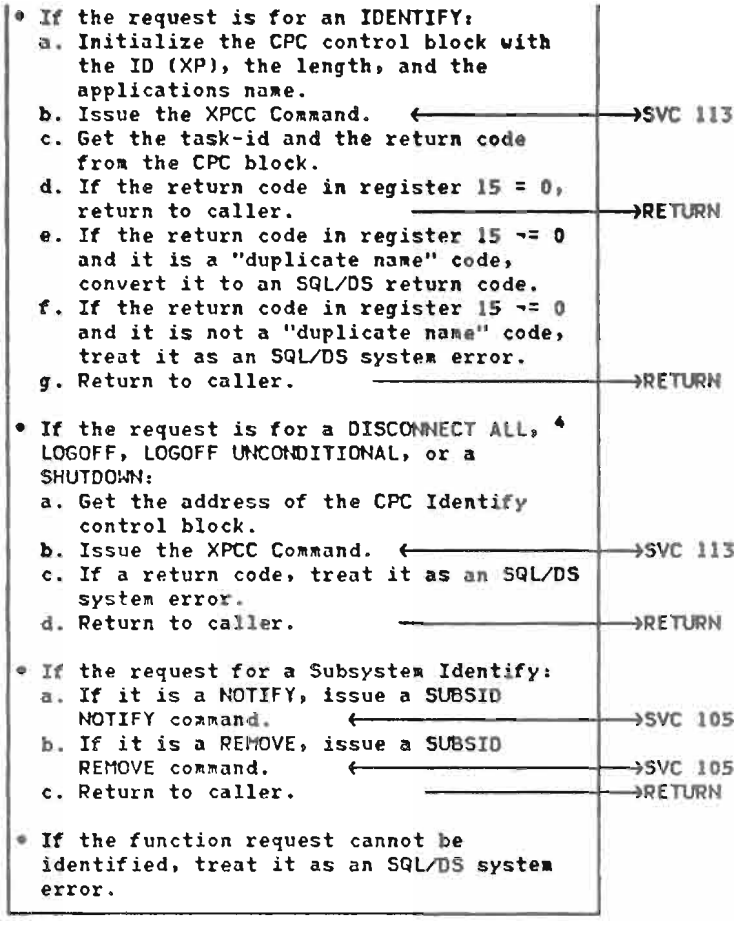
a. Unpost the ECB and get the message length, the user data, and the last function performed by partner's side from the CPC block.
b. If there was a reason code (in the CPC control block) and if partner's side disconnected, convert the disconnect codes to SQL/DS disconnect codes.
c. Return to caller. ——→RETURN

* If the request is for a CONNECT:

a. Initialize the CPC control block with the CPLTOAP name (or binary zeros if a CONNECT ANY request), the ID (XP) of the CPC control block, the CPC control block length, and the ID key from the IDENTIFY function.
b. Issue the XPCC command. ←——→SVC 113
c. If the return code in register 15 = 0: Get partner's name into CPLTOAP (for CONNECT ANY), unpost the CONNECT ECB (CECB) and the SEND ECB (SECB) and return to the caller. ——→RETURN
d. If the return code in register 15 ¬= 0, and it is a warning code (8) and the CPC return code is not a "quiesce" code, then treat it as if an SQL/DS system error occurred and return to caller. ——→RETURN
e. If the return code is an informational code (4):
   - If a WAIT was requested and a user exit specified, go to the user exit. ——→"USER EXIT"

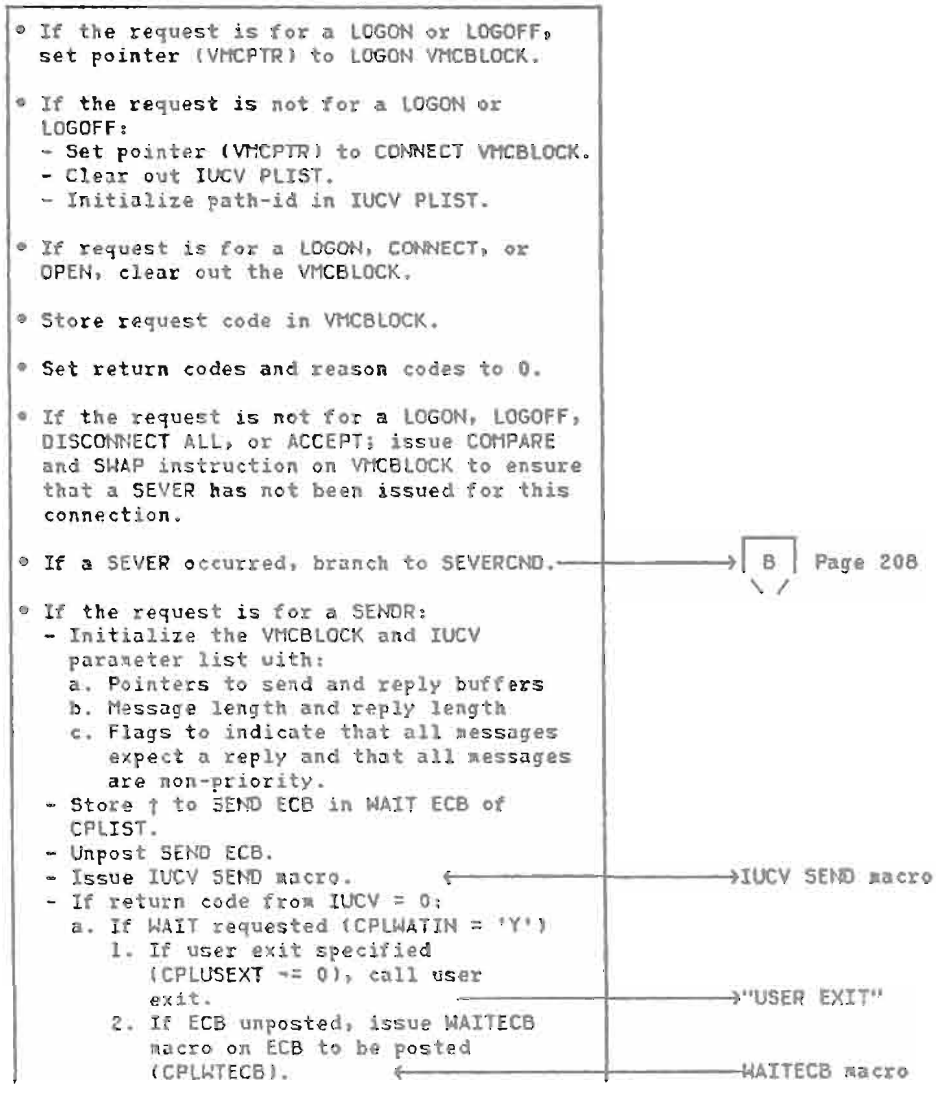


[3] The SQL/DS RESETR and RESETS functions are the XPCC PURGE and CLEAR functions respectively.



[4] LOGOFF is the XPCC TERMINATE function.
LOGOFF UNCONDITIONAL is the XPCC TERMPRGE function.
SHUTDOWN is the XPCC TERMQSCE function.

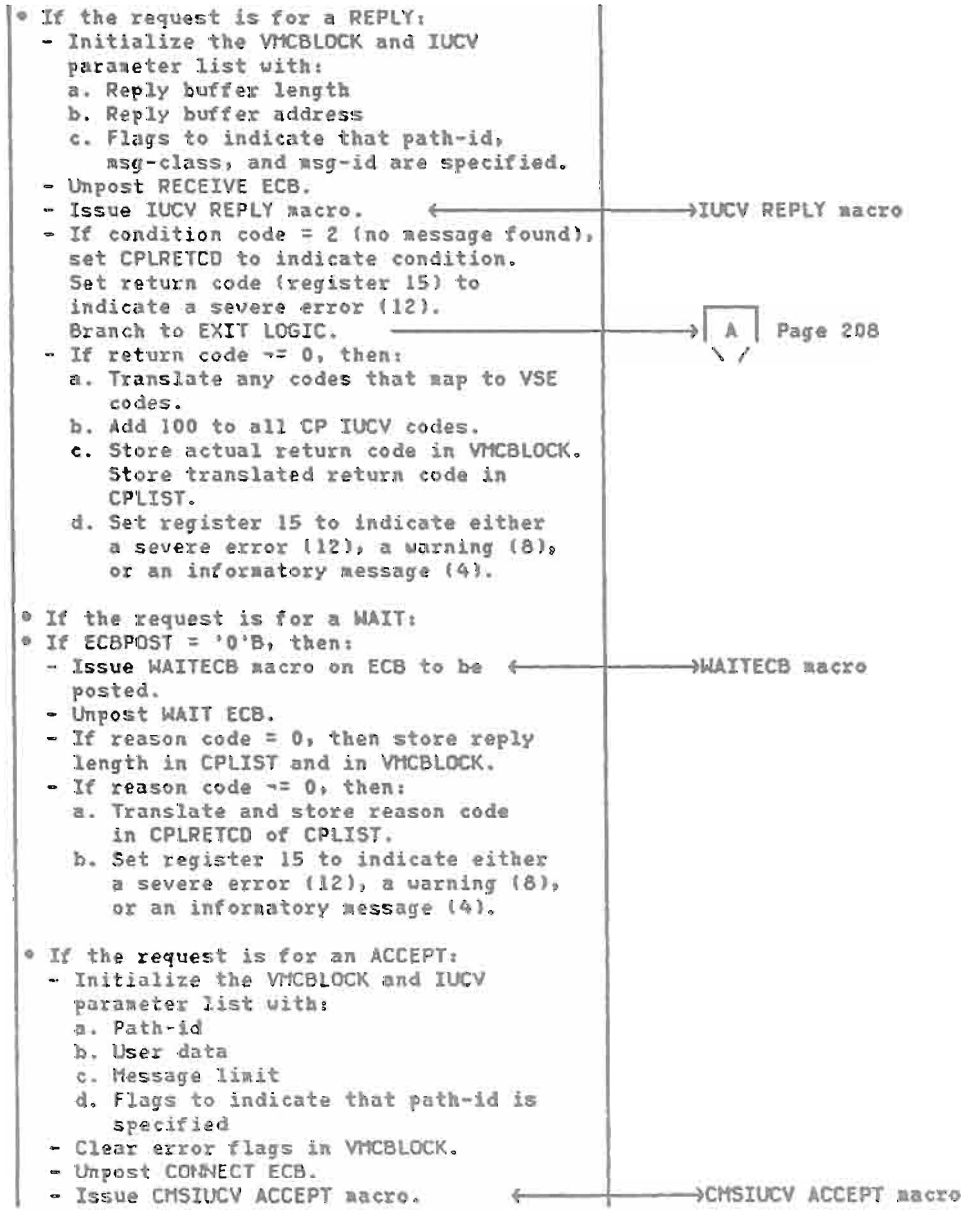
ARICCOM (SQL/DS Communication Manager) - VM



3. If return code ¬= 0, then:
   - Translate and store reason code (CPLRETCD) in CPLIST and in reason code field in VMCBLOCK.
   - Set register 15 accordingly.

- If asynchronous return code (reason code = 0:
  a. Store message ID (VMCMIDP) for IUCV function execution in VMCBLOCK.
  b. Store actual length of reply in VMCBLOCK and in CPLIST.
- If return code from IUCV ¬= 0, then:
  a. For any return codes > 1000 (CP IUCV codes), subtract 900.
  b. Translate any codes that map to VSE codes.
  c. Store actual return code in VMCRETCD. Store translated return code in CPLRETCD.
  d. Set register 15 to indicate either a severe error (12), a warning (8), or an informatory message (4).

• If the request is for a RECEIVE:
- Initialize the VMCBLOCK and IUCV parameter list with:
  a. Receive buffer address
  b. Receive buffer length
  c. Flag to indicate that path-id is specified.
- Unpost RECEIVE ECB.
- Issue IUCV RECEIVE macro. ←→ IUCV RECEIVE macro
- If condition code = 2 (no message found), set CPLRETCD to indicate condition. Set return code (register 15) to indicate a severe error (12). Branch to EXIT LOGIC. → A Page 208
- If return code ¬= 0, then:
  a. Translate any codes that map to VSE codes.
  b. Add 100 to all CP IUCV codes.
  c. Store actual return code in VMCBLOCK. Store translated return code in CPLIST.
  d. Set register 15 to indicate either a severe error (12), a warning (8), or an informatory message (4).


* If the request is for a REPLY:
- Initialize the VMCBLOCK and IUCV parameter list with:
a. Reply buffer length
b. Reply buffer address
c. Flags to indicate that path-id, msg-class, and msg-id are specified.
- Unpost RECEIVE ECB.
- Issue IUCV REPLY macro.
IUCV REPLY macro
- If condition code = 2 (no message found), set CPLRETCD to indicate condition. Set return code (register 15) to indicate a severe error (12). Branch to EXIT LOGIC.
A
Page 208
- If return code ¬= 0, then:
a. Translate any codes that map to VSE codes.
b. Add 100 to all CP IUCV codes.
c. Store actual return code in VMCBLOCK. Store translated return code in CPLIST.
d. Set register 15 to indicate either a severe error (12), a warning (8), or an informatory message (4).
* If the request is for a WAIT:
* If ECBPOST = '0'B, then:
- Issue WAITECB macro on ECB to be posted.
WAITECB macro
- Unpost WAIT ECB.
- If reason code = 0, then store reply length in CPLIST and in VMCBLOCK.
- If reason code ¬= 0, then:
a. Translate and store reason code in CPLRETCD of CPLIST.
b. Set register 15 to indicate either a severe error (12), a warning (8), or an informatory message (4).
* If the request is for an ACCEPT:
- Initialize the VMCBLOCK and IUCV parameter list with:
a. Path-id
b. User data
c. Message limit
d. Flags to indicate that path-id is specified
- Clear error flags in VMCBLOCK.
- Unpost CONNECT ECB.
- Issue CMSIUCV ACCEPT macro.
CMSIUCV ACCEPT macro

- If return code ¬=0, then:
  a. For any return codes > 1000 (CP IUCV codes), subtract 900.
  b. Translate any return codes that map to VSE codes.
  c. Store actual return code in VMCBLOCK. Store translated return code in CPLIST.
  d. Set register 15 to indicate either a severe error (12), a warning (8), or an informatory message (4).
- If return code = 0, set up CPLIST with the following passback information:
  a. Target user name
  b. User data
  c. Address of CONNECT ECB
  d. Address of wait ECB.

• If the request is for a CONNECT or OPEN:
- If request is for an OPEN:
  a. Initialize DISKID parameter list with DDNAME from CPLIST (CPLDDNAM).
  b. Store DDNAME in VMCBLOCK (VMCDDNAM).
  c. Call DISKID function via SVC 202. ←→ DISKID (SVC 202)
  d. If return code = 0:
     1. Set up exit address (address of External Interrupt Handler) in CMSIUCV PLIST.
     2. Store the virtual device address in CPLIST and in VMCBLOCK.
     3. Store the block size in CPLIST and in VMCBLOCK.
     4. Store the offset in CPLIST and in VMCBLOCK.
     5. Store the virtual address, block size, and offset in IPUSER for IUCV function execution.
     6. Store "*BLOCKIO" in CPLIST and in field for target application name (VMCTOAP) in VMCBLOCK.
     7. Set VMCFPDTA flag to indicate message will be in parameter list for block I/O.
  e. If return code ¬= 0, then:
     1. Store return codes in CPLIST and in VMCBLOCK.

2. Set return code in register 15 to indicate a severe error (12). Branch to EXIT LOGIC. → A Page 208

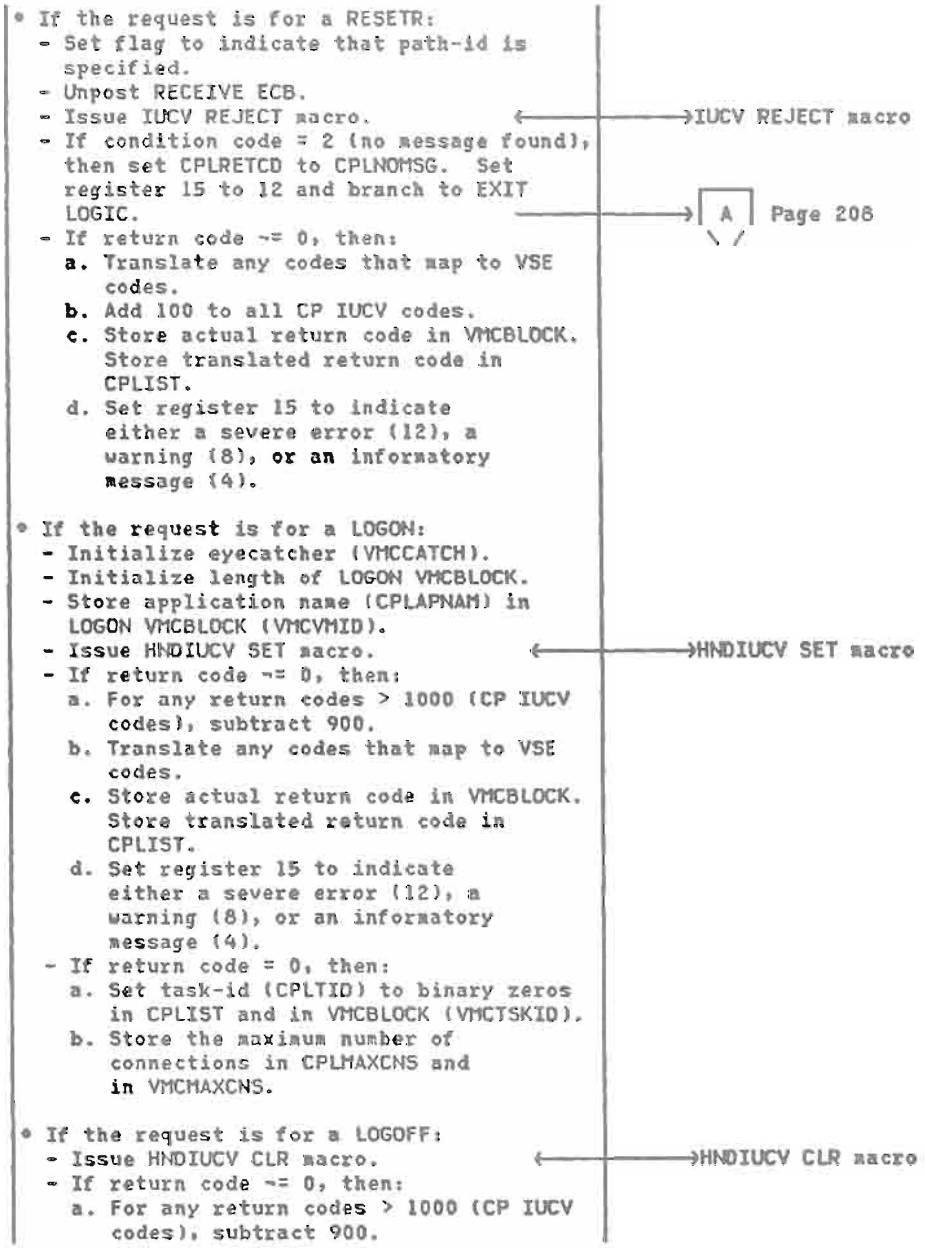
- If request is for a CONNECT:
  a. Initialize the VMCBLOCK and IUCV parameter list with:
     1. Partner user name
     2. User data
     3. Message limit
  b. Unpost CONNECT ECB.
- If request is for CONNECT or OPEN:
  a. Initialize the VMCBLOCK and IUCV parameter list with partner user name.
  b. Unpost CONNECT ECB.
  c. Issue CMSIUCV CONNECT macro. ↔ CMSIUCV CONNECT macro
  d. If return code ¬= 0, then:
     1. For any return codes > 1000 (CP IUCV codes), subtract 900.
     2. Translate any codes that map to VSE codes.
     3. Store actual return code in VMCBLOCK. Store translated return code in CPLIST.
     4. Set register 15 to indicate either a severe error (12), a warning (8), or an informatory message (4).
  e. If return code = 0, then:
     1. Store address of CONNECT ECB in WAIT ECB (CPLWTECB) in CPLIST.
     2. Store WAIT ECB in CONNECT ECB (CPLCECB) in CPLIST.
     3. Store address of SEND ECB in CPLSECB.
     4. Save path-id in VMCBLOCK for future calls.
     5. If WAIT requested:
        - If user exit specified, then call user exit. → "USER EXIT"
        - If WAIT ECB unposted, issue WAITECB on ECB to be posted.
        - Issue WAITECB on ECB to be posted. ↔ WAITECB macro
        - Set return code to 0.

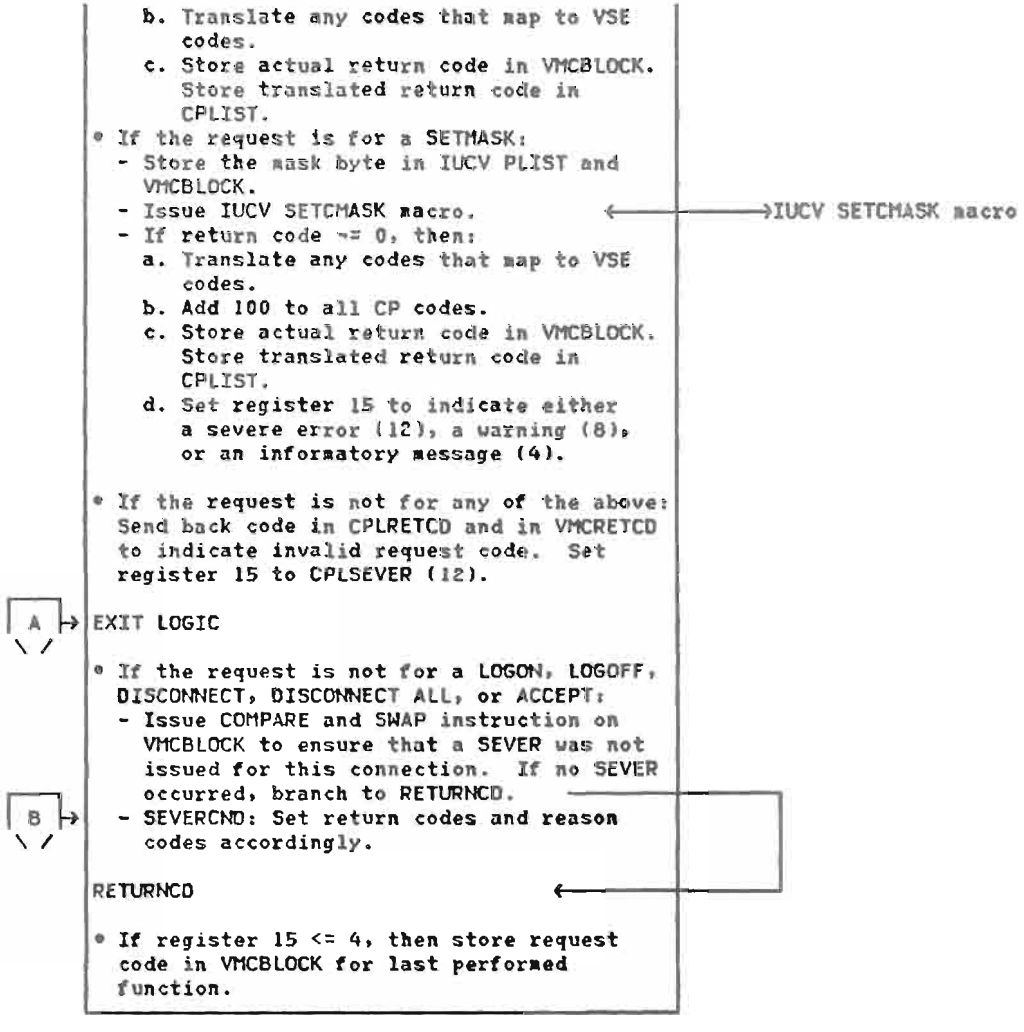
  f. If asynchronous return code (reason code) ¬= 0:
     1. If request is for OPEN:
        - Add 150 to all asynchronous return codes and pass back in CPLIST (CPLRETCD).
        - Set register 15 to CPLSEVER (12) to indicate a severe error.

2. If request is for CONNECT:
   - Pass back reason code in CPLIST (CPLRETCD).
   - Set register 15 to CPLWARN (8) to indicate a warning message.

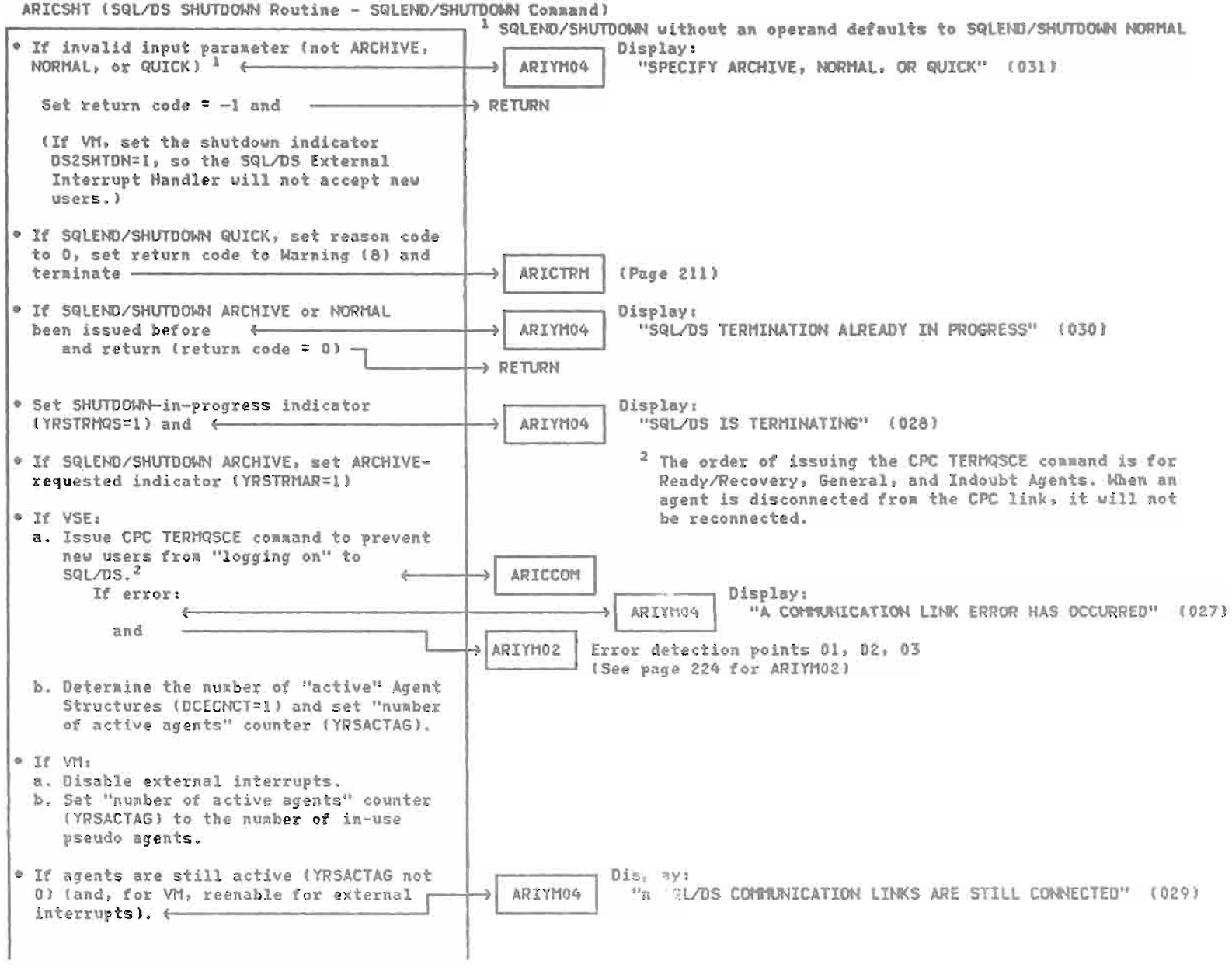
* If return code = 0 and request is for OPEN:
  - Store the number of blocks in the file in CPLIST (CPLNOBLK) and in VMCBLOCK (VMCNOBLK).
  - Set flag (VMCOPENF) to indicate the file is open.
  - If flag (VMCIOFLG) indicates file is read only:
    a. Set flag in CPLIST (CPLIOFLG) to indicate read only.
    b. Store return code to indicate read only in CPLIST (CPLRETCD) and in VMCBLOCK (VMCRETCD).
    c. Set register 15 to CPLINFO (4) to indicate an informatory message.
  - Clear wait bit in CONNECT ECB to indicate function has completed.

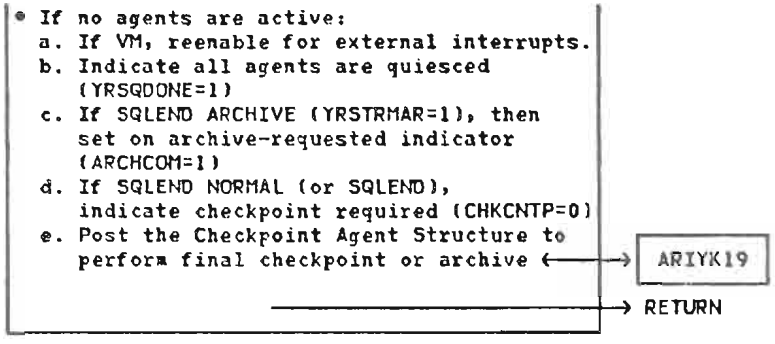
* If the request is for a DISCONNECT PURGE, DISCONNECT ALL or CLOSE:
  - If request is not for CLOSE, store user data in IUCV PLIST.
  - If request is for DISCONNECT ALL, then set code to disconnect all outstanding connections.
  - Issue CMSIUCV SEVER macro. ←→ CMSIUCV SEVER macro
  - If return code ¬= 0, then:
    a. For any return code > 1000 (CP IUCV codes), subtract 900.
    b. Translate any codes that map to VSE codes.
    c. Store actual return code in VMCBLOCK. Store translated return code in CPLIST.
    d. Set register 15 to indicate either a severe error (12), a warning (8), or an informatory message (4).
  - If return code = 0 and request is for CLOSE, then set open flag (VMCOPENF) to NO indicating that the file has been closed.


• If the request is for a RESETR:
- Set flag to indicate that path-id is specified.
- Unpost RECEIVE ECB.
- Issue IUCV REJECT macro.
IUCV REJECT macro
- If condition code = 2 (no message found), then set CPLRETCD to CPLNOMSG. Set register 15 to 12 and branch to EXIT LOGIC.
A
Page 208
- If return code ¬= 0, then:
a. Translate any codes that map to VSE codes.
b. Add 100 to all CP IUCV codes.
c. Store actual return code in VMCBLOCK. Store translated return code in CPLIST.
d. Set register 15 to indicate either a severe error (12), a warning (8), or an informatory message (4).
• If the request is for a LOGON:
- Initialize eyecatcher (VMCCATCH).
- Initialize length of LOGON VMCBLOCK.
- Store application name (CPLAPNAM) in LOGON VMCBLOCK (VMCVMID).
- Issue HNDIUCV SET macro.
HNDIUCV SET macro
- If return code ¬= 0, then:
a. For any return codes > 1000 (CP IUCV codes), subtract 900.
b. Translate any codes that map to VSE codes.
c. Store actual return code in VMCBLOCK. Store translated return code in CPLIST.
d. Set register 15 to indicate either a severe error (12), a warning (8), or an informatory message (4).
- If return code = 0, then:
a. Set task-id (CPLTID) to binary zeros in CPLIST and in VMCBLOCK (VMCTSKID).
b. Store the maximum number of connections in CPLMAXCNS and in VMCMAXCNS.
• If the request is for a LOGOFF:
- Issue HNDIUCV CLR macro.
HNDIUCV CLR macro
- If return code ¬= 0, then:
a. For any return codes > 1000 (CP IUCV codes), subtract 900.

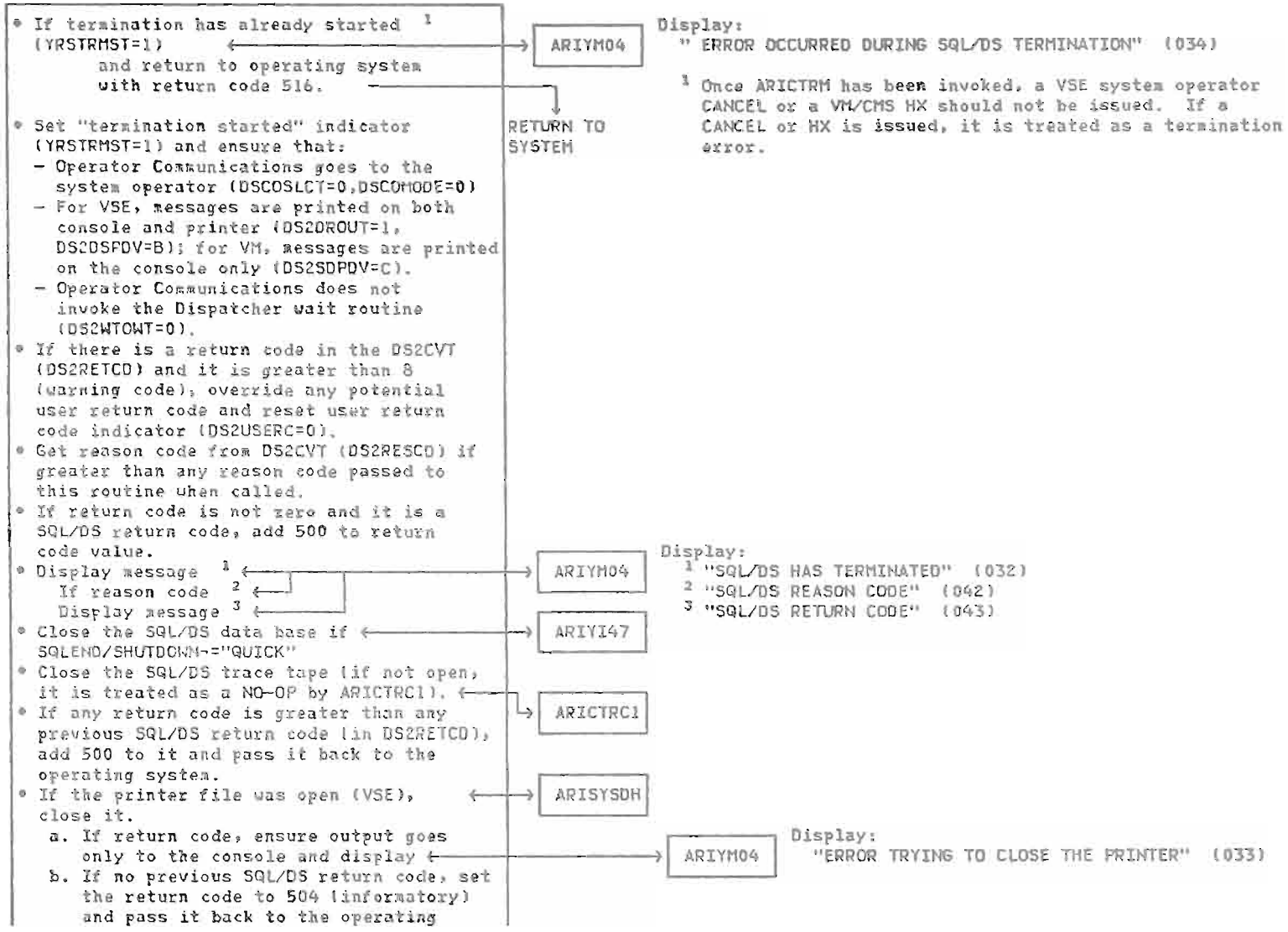


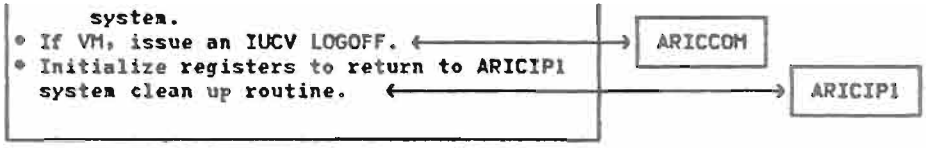
DSC TERMINATION





ARICTRM (SQL/DS Termination Routine)





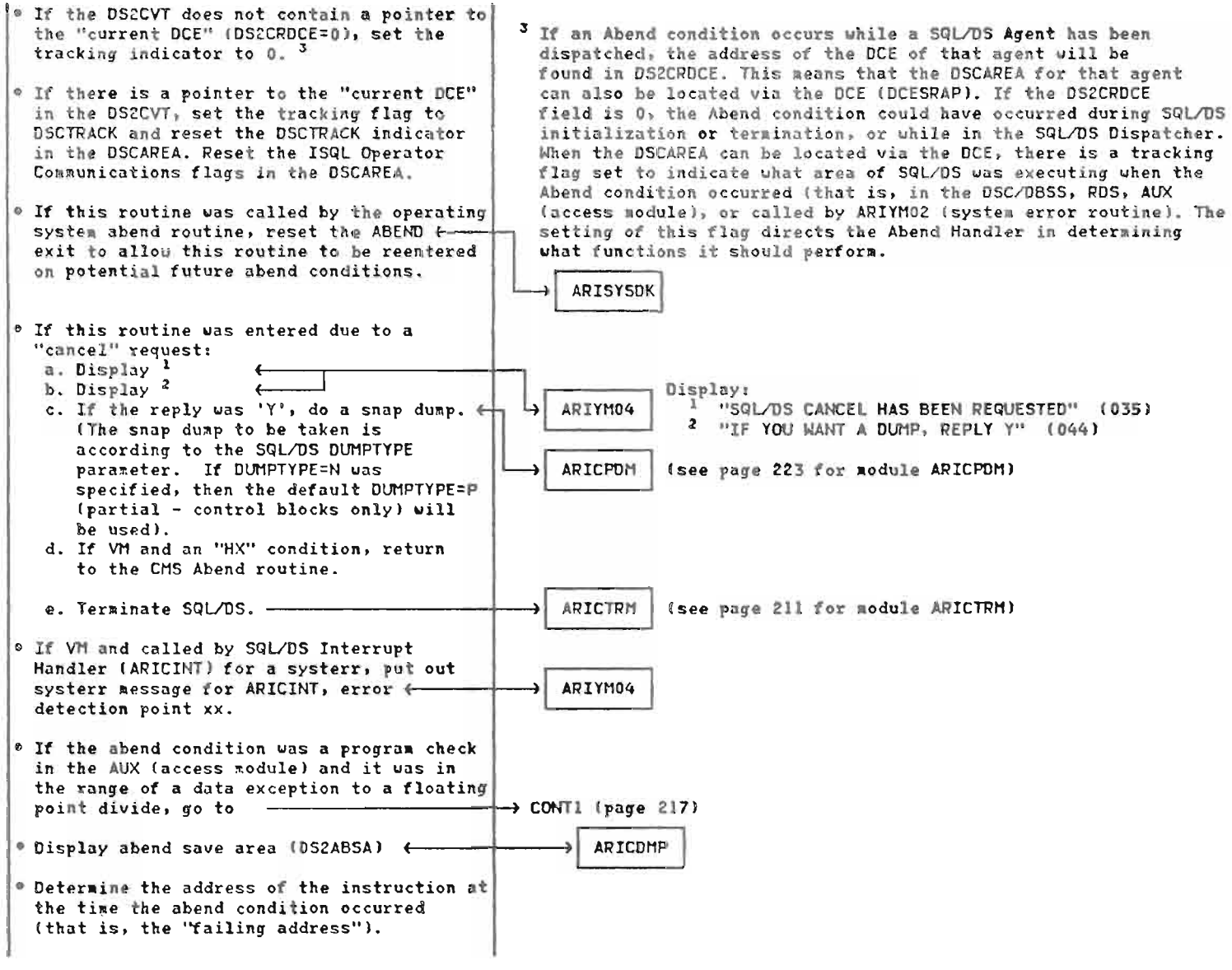
DSC Termination (continued)

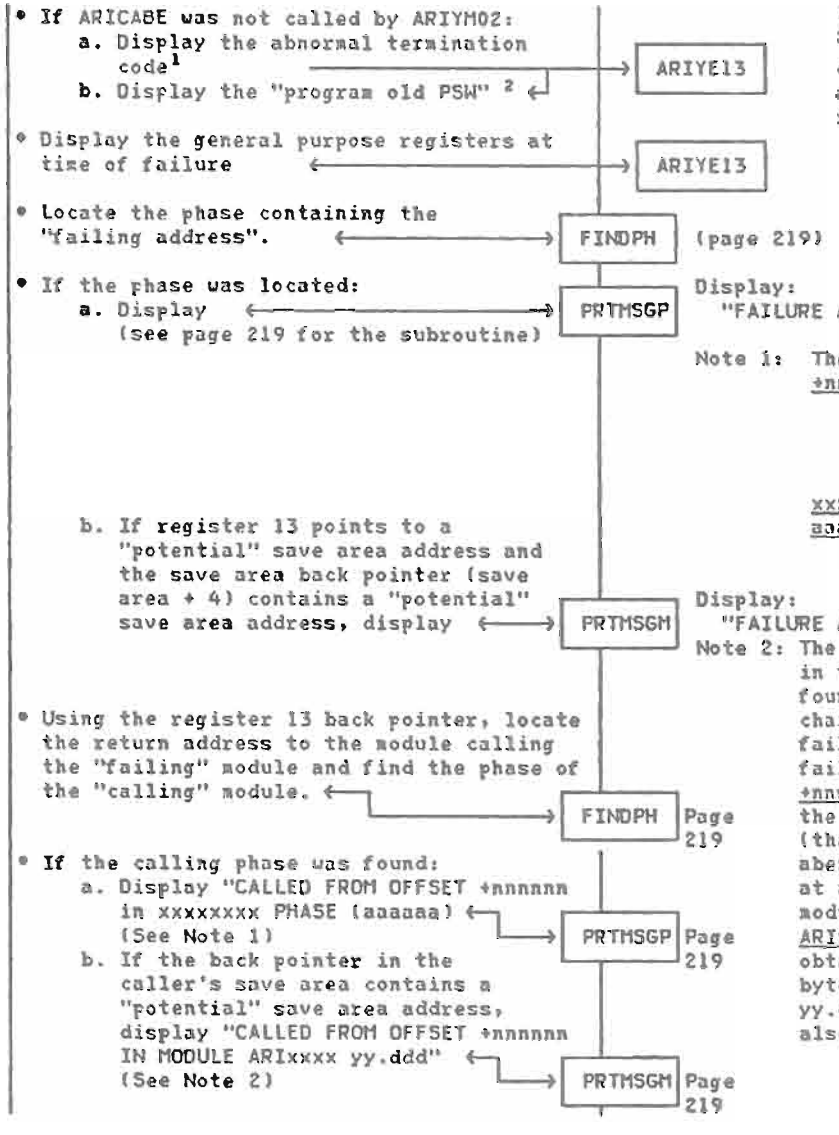
ARICABE (SQL/DS Abend Handler)

```
* * * * * * * * * * * * * * * * * * * * *
 For VM, "PHASE" ("phase") is translated
 to "PROGRAM" ("program").
* * * * * * * * * * * * * * * * * * * * *
```

- Set the pointers to the DS2CVT and the prototype YTABLE1.
- Ensure messages are printed according to the DSPLYDEV parameter (DS2DROUT=1) and the Dispatcher wait routine is not called by Operator Communications (DS2WTOWT=0).
- If the routine was called by the operating system abend routine, set the reason code to the abnormal termination code (if VSE, passed by the operating system in register 0; if VM, obtained from the abend PSW interrupt code).

For VSE, register 0 contains the abnormal termination code when ARICABE is called by the operating system (it contains 0 when called by ARIYM02). The abnormal termination codes are documented in VSE/Advanced Functions Macro Reference, SC24-5211, under the discussion of the STXIT AB macro instruction and in SQL/DS Messages and Codes (see the VSE manual for further information). Register 1 points to the start of the abend save area (in the DS2CVT, field DS2ABSA).
For VM, register 1 points to the start of the CMS Abend save area (DMSABW CSECT in DMSNUC). The abnormal termination code is obtained from the abend PSW interrupt code. These codes are documented in VM/SP System Messages and Codes, SC19-6204, under the discussion of CMS abend codes. The abend PSW and registers at the time of the abend are moved into DS2ABSA.
Preceding the DS2ABSA Abend save area is a pointer to the start of DS2CVT. In addition, the Abend process is driven off the prototype YTABLE1 established by, and used during, the SQL/DS initialization process.





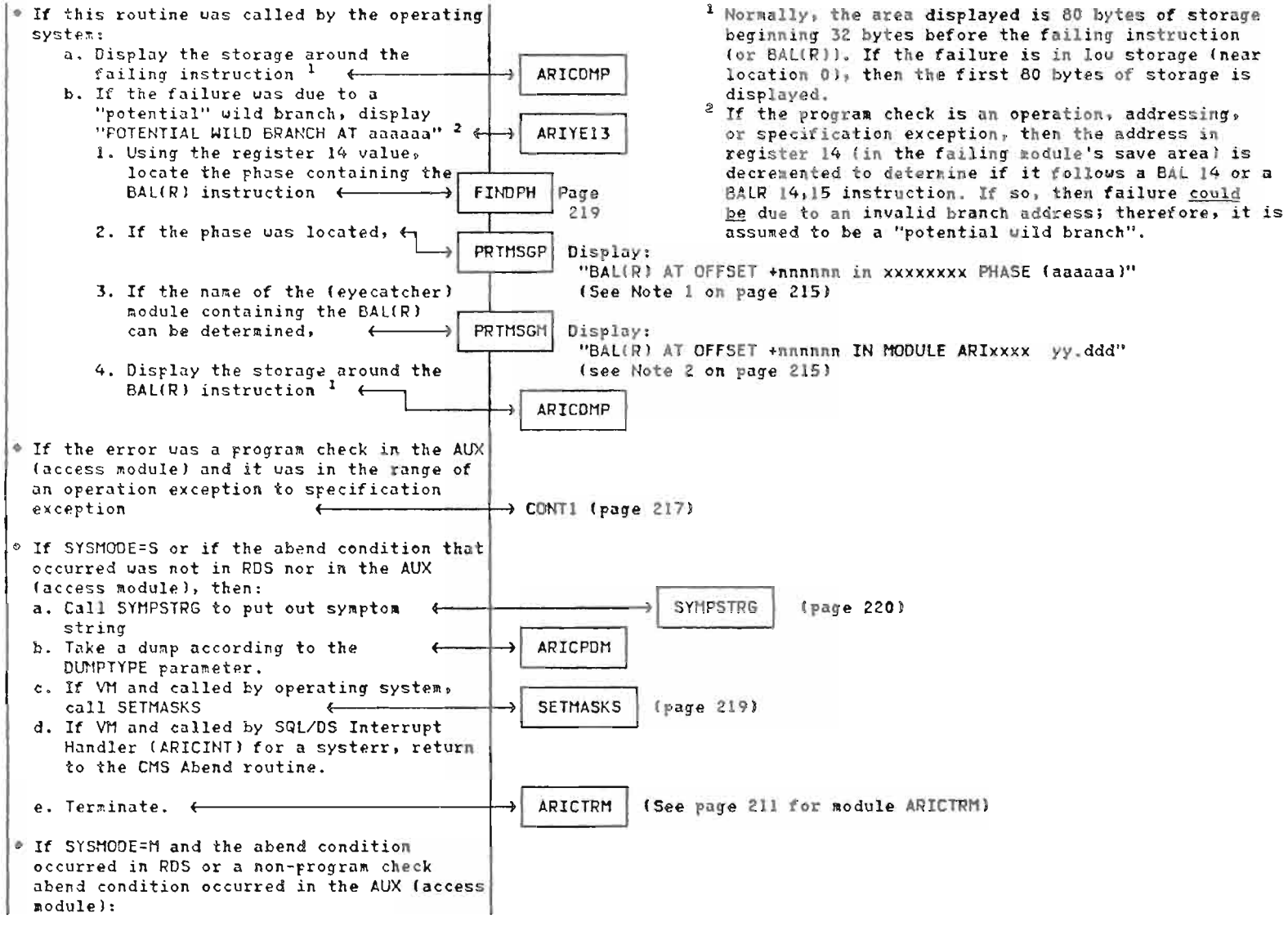
[1] "ABTERM CODE ccc AT aaaaaa"
[2] "PROGRAM OLD PSW IS: xxxxxxxx xxxxxxxx"
ccc - is the abnormal termination code
aaaaaa - is the failing address
xxxxxxxx - is the PSW separated into two 4-byte sections

Display:
"FAILURE AT OFFSET +nnnnnn in xxxxxxxx PHASE (aaaaaa)"

Note 1: The list of "known" phase addresses is located in the DS2CVT.
+nnnnnn is the hexadecimal offset from the start of the phase (the address derived from the PSW in the abend save area). This value can be added to the phase starting address obtained from the linkage-editor map to assist in in determining the failing module.
xxxxxxxx is the name of the phase
aaaaaa is the starting address of where the phase resides in storage.

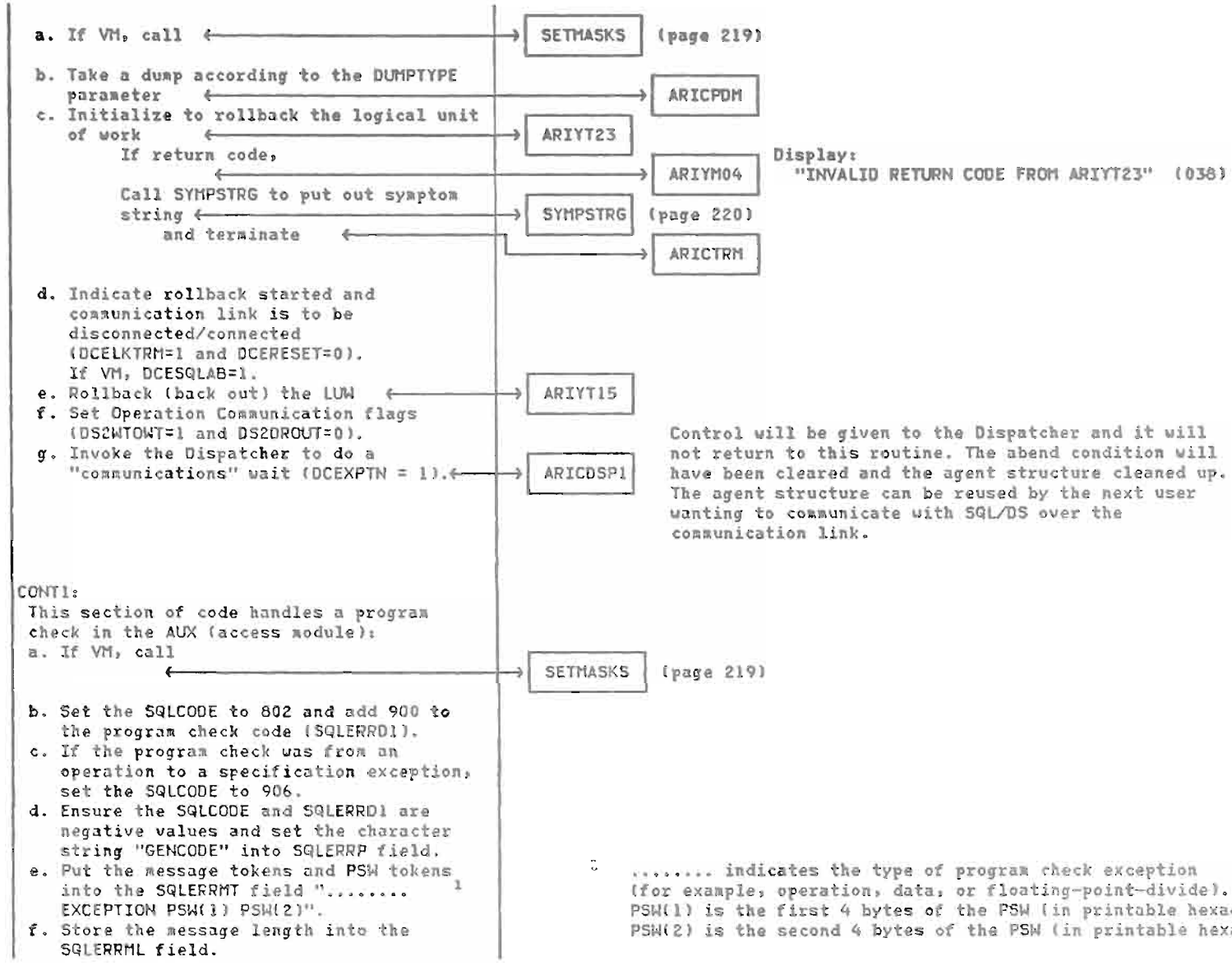
Display:
"FAILURE AT OFFSET +nnnnnn IN MODULE ARIxxxx yy.ddd"

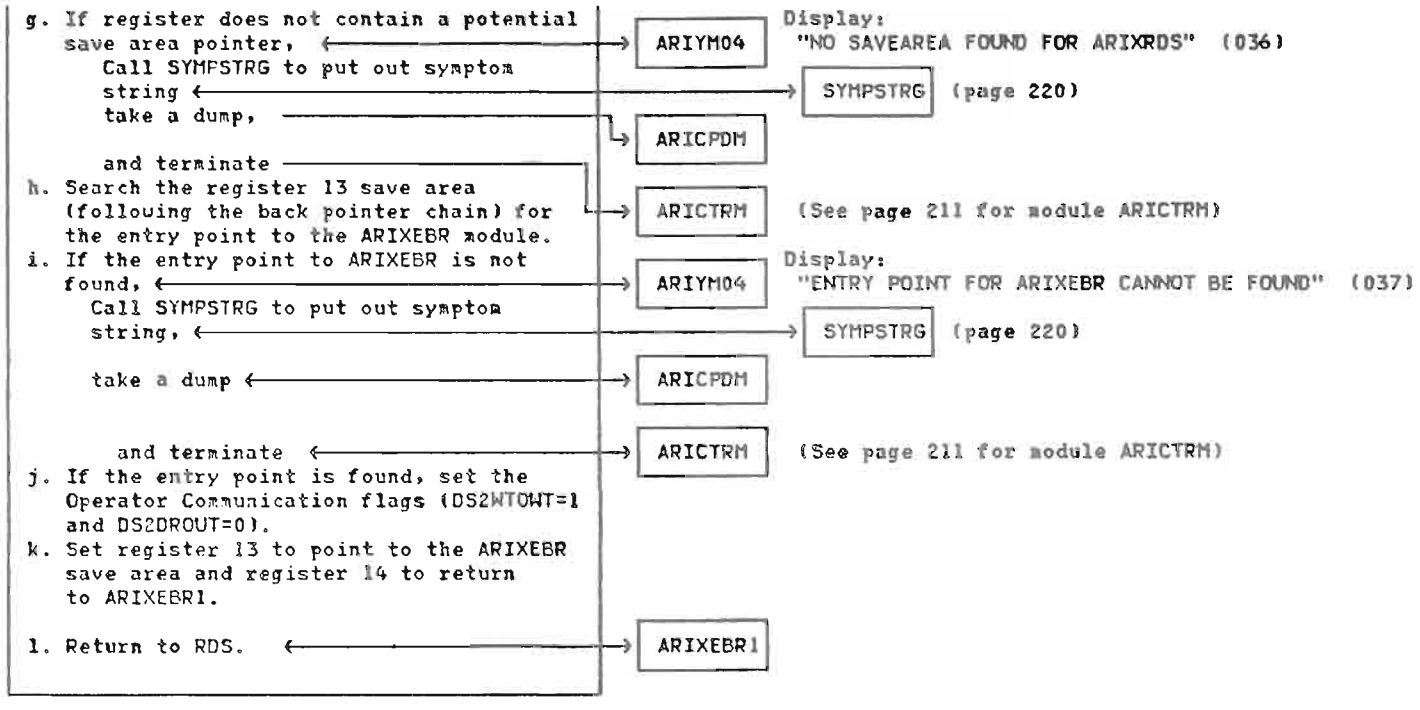
Note 2: The module name is found using the "eyecatcher" in the module where the failure occurred. It is found by following the register 13 save area back chain pointer (register 15 of the caller of the failing module points to the entry point of the failing module).
+nnnnnn is a hexadecimal value, the offset from the start of the module to the "failing address" (that is, the address derived from the PSW in the abend save area). This value can be used when looking at an assembler source listing of the referenced module (ARIxxxx).
ARIxxxx is the name of the failing module. This is obtained from the module "eyecatcher" five or seven bytes past the module entry point.
yy.ddd is the compilation date of the module. It is also obtained from the module "eyecatcher" field.



[1] Normally, the area displayed is 80 bytes of storage beginning 32 bytes before the failing instruction (or BAL(R)). If the failure is in low storage (near location 0), then the first 80 bytes of storage is displayed.

[2] If the program check is an operation, addressing, or specification exception, then the address in register 14 (in the failing module's save area) is decremented to determine if it follows a BAL 14 or a BALR 14,15 instruction. If so, then failure could be due to an invalid branch address; therefore, it is assumed to be a "potential wild branch".


a. If VM, call
SETMASKS
(page 219)
b. Take a dump according to the DUMPTYPE parameter
ARICPDM
c. Initialize to rollback the logical unit of work
ARIYT23
If return code,
ARIYM04
Display:
"INVALID RETURN CODE FROM ARIYT23" (038)
Call SYMPSTRG to put out symptom string
SYMPSTRG
(page 220)
and terminate
ARICTRM
d. Indicate rollback started and communication link is to be disconnected/connected (DCELKTRM=1 and DCERESET=0).
If VM, DCESQLAB=1.
e. Rollback (back out) the LUW
ARIYT15
f. Set Operation Communication flags (DS2WTOWT=1 and DS2OROUT=0).
g. Invoke the Dispatcher to do a "communications" wait (DCEXPTN = 1).
ARICDSP1
Control will be given to the Dispatcher and it will not return to this routine. The abend condition will have been cleared and the agent structure cleaned up. The agent structure can be reused by the next user wanting to communicate with SQL/DS over the communication link.
CONT1:
This section of code handles a program check in the AUX (access module):
a. If VM, call
SETMASKS
(page 219)
b. Set the SQLCODE to 802 and add 900 to the program check code (SQLERRD1).
c. If the program check was from an operation to a specification exception, set the SQLCODE to 906.
d. Ensure the SQLCODE and SQLERRD1 are negative values and set the character string "GENCODE" into SQLERRP field.
e. Put the message tokens and PSW tokens into the SQLERRMT field "..... EXCEPTION PSW(1) PSW(2)".
1
f. Store the message length into the SQLERRML field.
..... indicates the type of program check exception (for example, operation, data, or floating-point-divide).
PSW(1) is the first 4 bytes of the PSW (in printable hexadecimal).
PSW(2) is the second 4 bytes of the PSW (in printable hexadecimal).



DSC Termination (continued)

ARICABE Subroutines

PRTMSGP

- Convert the phase offset (+nnnnnn) and phase address (aaaaaa) to printable format and create the message:
  "xxxxxx AT OFFSET +nnnnnn in xxxxxxxx PHASE (aaaaaa)" [1]
- Display the message and return. → ARIYE13

[1] xxxxxx - is one of the following character strings: "FAILURE", "CALLED", or "BAL(R)".
nnnnnn - is the offset (in hexadecimal) from the start of the phase or module. Note: It is possible that if an error occurs while in the "prolog" code (that is, before it has established the pointer to the save area in register 13), the offset may result in a negative value based on the entry point in the calling module.
xxxxxxxx - is the name of the phase (or module).
yy.ddd - is the year and the day of the year the module was compiled.

PRTMSGM

- If the module contains an eyecatcher and it can be found: [2]
  - Convert the module offset (+nnnnnn) to printable format and create the message: "xxxxxx AT OFFSET +nnnnnn IN ARIxxxx yy.ddd" [1]
  - Display the message and return. → ARIYE13

[2] The start of each module should have one of the following code strings:

A.

```
USING *, 15
B  @PROLOG
DC AL1(16)
DC CL16'modname   yy.ddd'
     or
DC CL24'modname   yy.ddd PFTID'
@PROLOG STM
```

B.

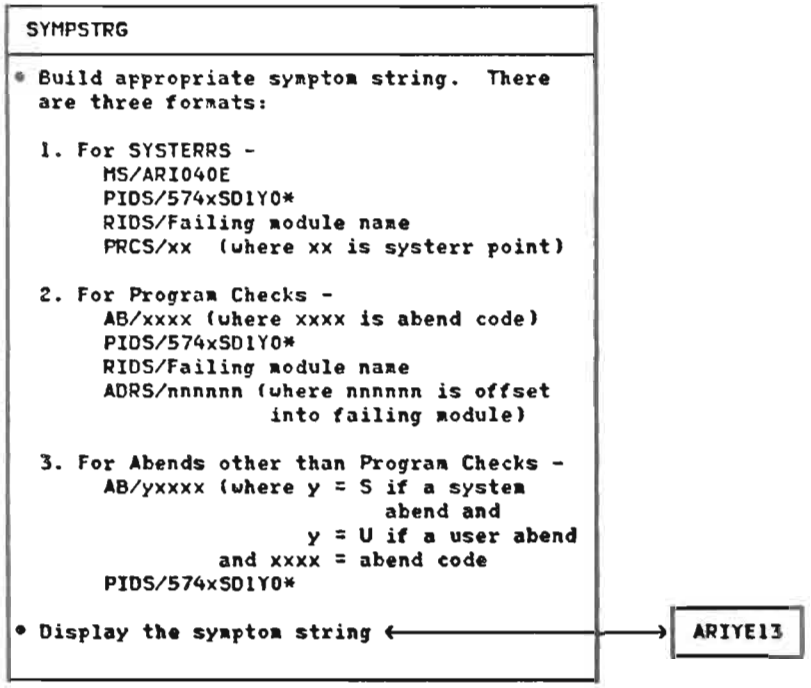
```
BALR 15,0
USING *, 15
B  @PROLOG
DC AL1(16)
DC CL16'modname   yy.ddd'
     or
DC CL24'modname   yy.ddd PFTID'
@PROLOG STM
```

FINDPH

- If the search address falls within the range of one of the phase starting and ending addresses in the DSCVT, determine the offset into the phase by subtracting the address from the start of the phase.
- Return

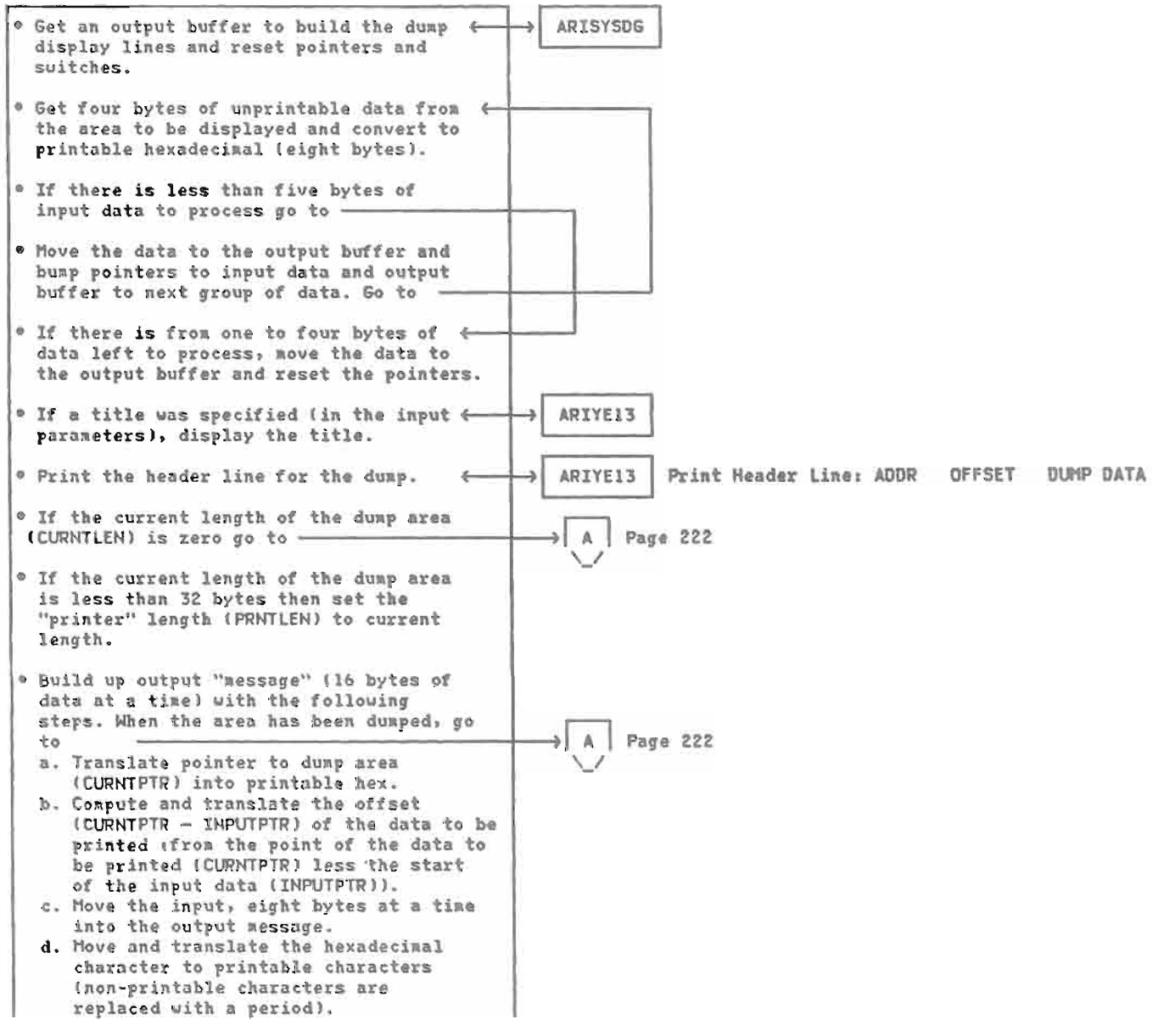
SETMASKS (VM/SP)

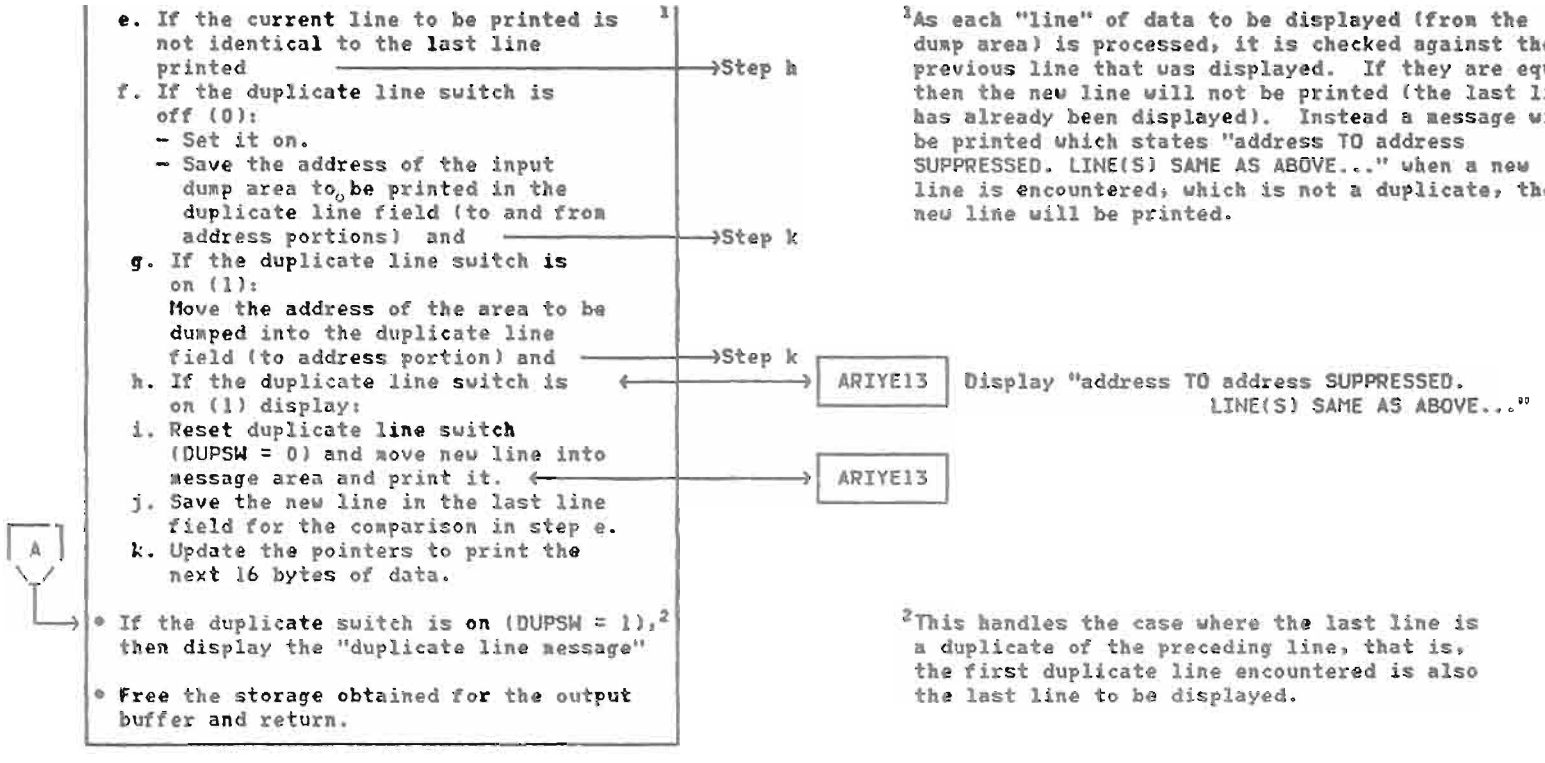
- Change the CMS protect key to the user key - that is, key '8'. (CMS Abend Handler gives control to ARICABE in the nucleus protect key - that is, key '0'.)
- Set the system mask enabled for external interrupts.
- Set the program mask.



* 574xSD1Y0 is SQL/DS Program Number where: x = 6 for VSE
x = 8 for VM

ARICDMP (SQL/DS DUMP Routine (1))

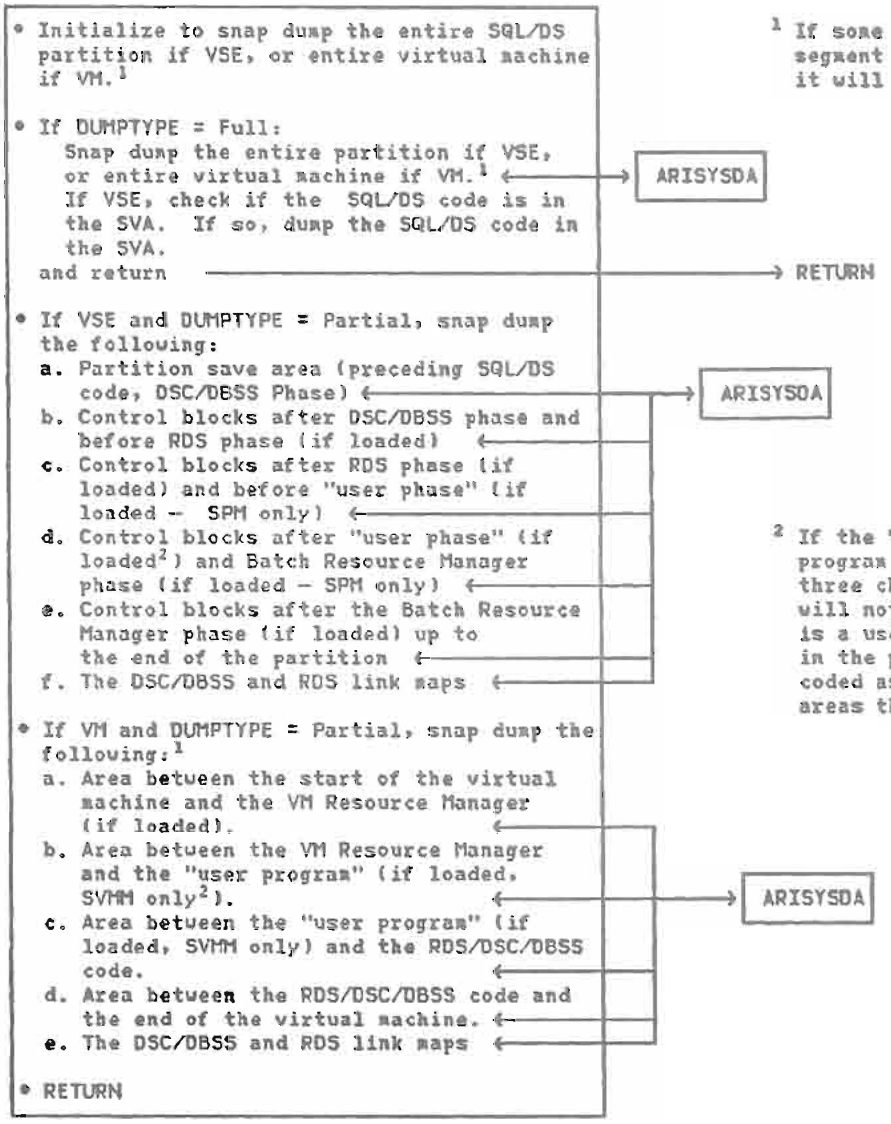




[1] As each "line" of data to be displayed (from the dump area) is processed, it is checked against the previous line that was displayed. If they are equal, then the new line will not be printed (the last line has already been displayed). Instead a message will be printed which states "address TO address SUPPRESSED. LINE(S) SAME AS ABOVE..." when a new line is encountered, which is not a duplicate, the new line will be printed.

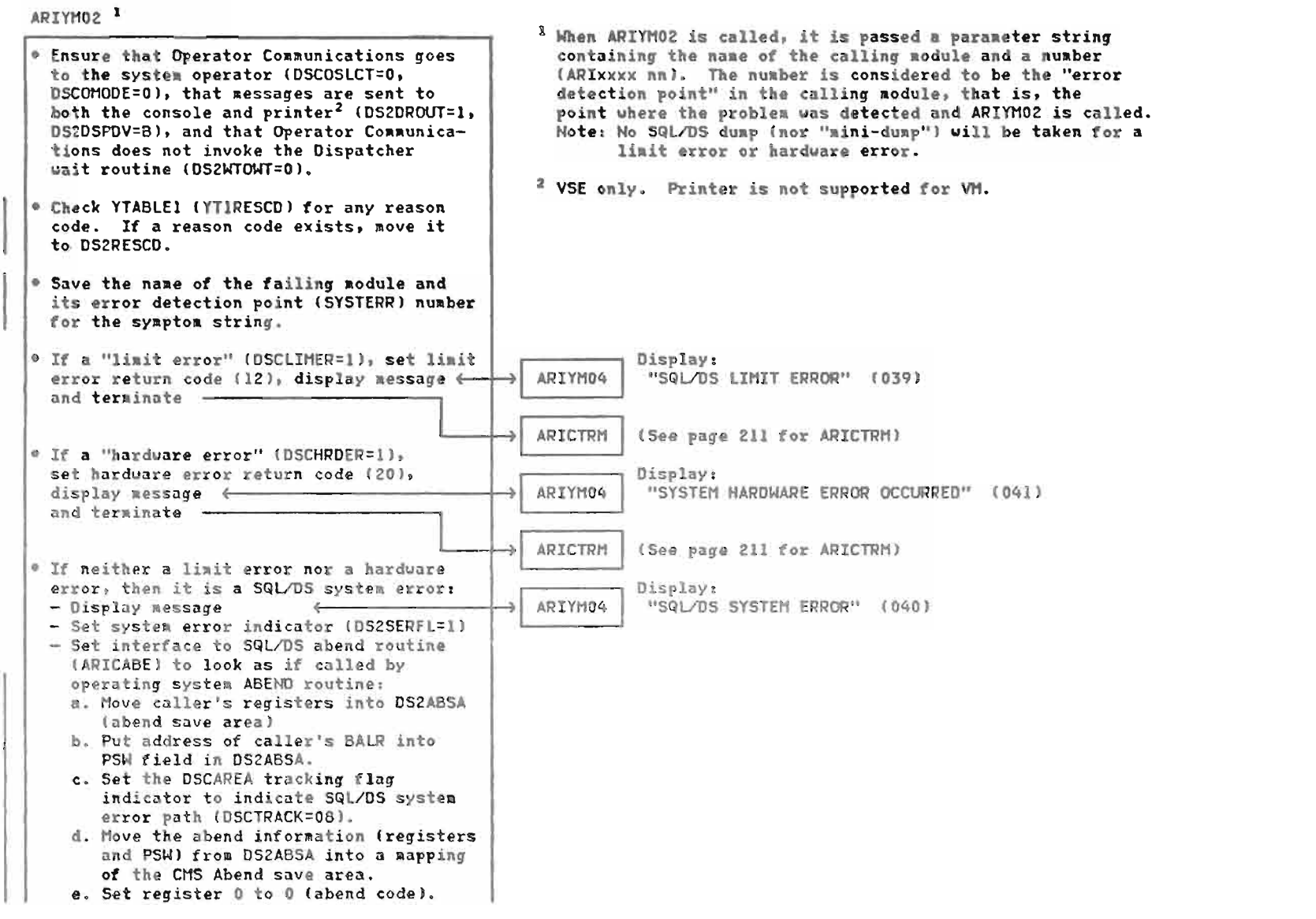
[2] This handles the case where the last line is a duplicate of the preceding line, that is, the first duplicate line encountered is also the last line to be displayed.

ARICPDM (SQL/DS Snap Dump Routine)



[1] If some of the code is in the discontiguous saved segment area (beyond the end of the virtual machine) it will also be dumped.

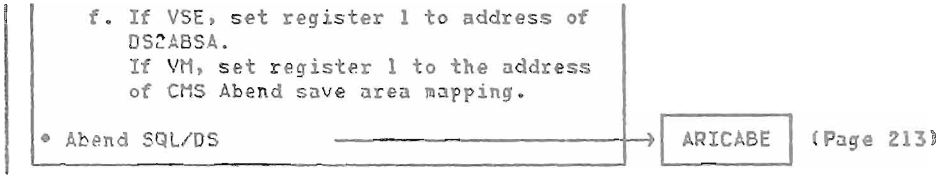
[2] If the "user phase" or "user program" is an SQL/DS program such as DBSU, PREP, etc., (where the first three characters of the phase name are "ARI") it will not be included in the dump. If the "user phase" is a user application program then it will be included in the partial dump. If the "user phase" may not be coded as reentrant, and therefore may have vital data areas that need to be displayed.



1 When ARIYM02 is called, it is passed a parameter string containing the name of the calling module and a number (ARIxxxx nn). The number is considered to be the "error detection point" in the calling module, that is, the point where the problem was detected and ARIYM02 is called.
Note: No SQL/DS dump (nor "mini-dump") will be taken for a limit error or hardware error.

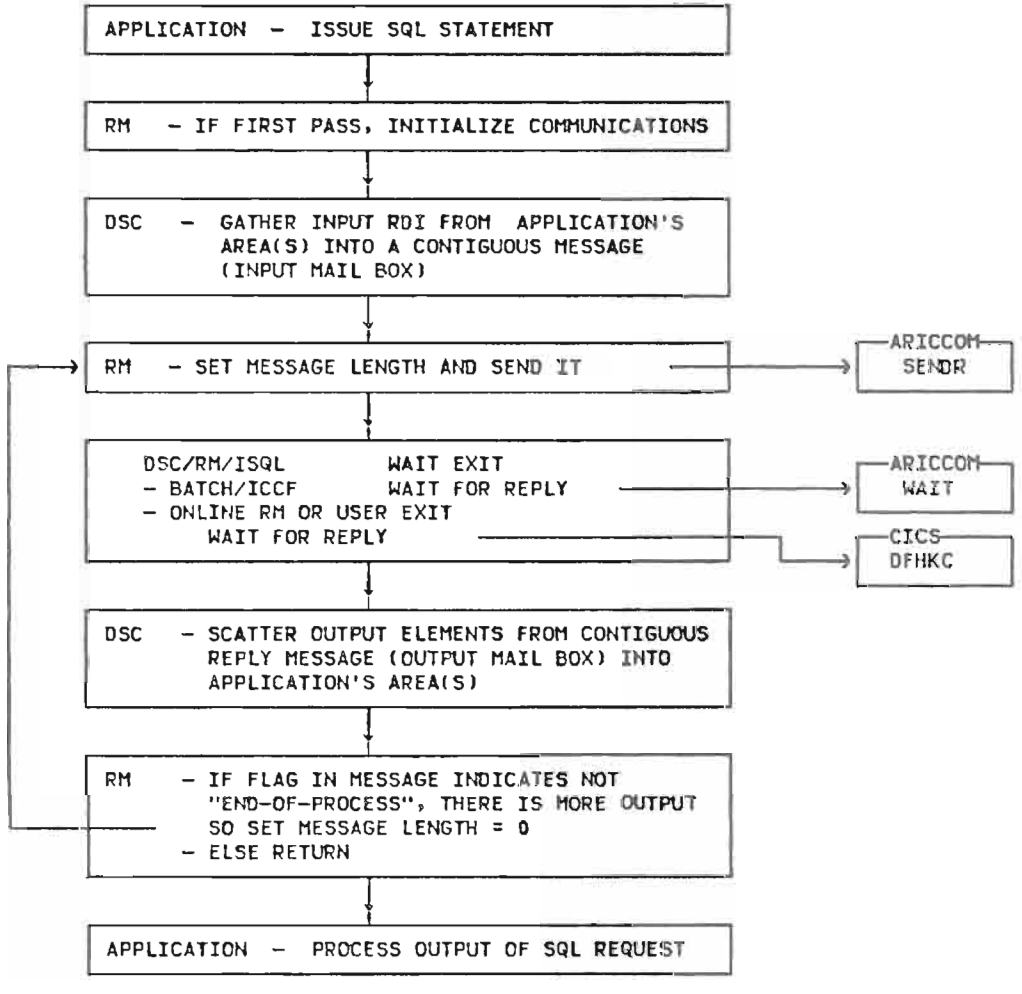
2 VSE only. Printer is not supported for VM.

DSC Termination (continued)



The VM Resource Manager description begins on page 260.
The VSE/Advanced Functions Resource Manager description begins on page 226.

RESOURCE MANAGER (RM) FLOW (VSE/ADVANCED FUNCTIONS)

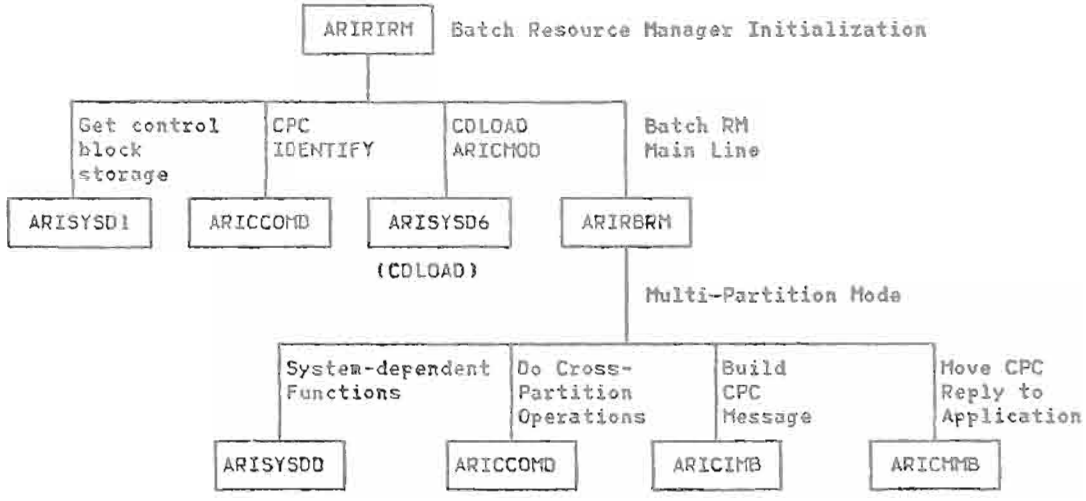


BATCH RESOURCE MANAGER

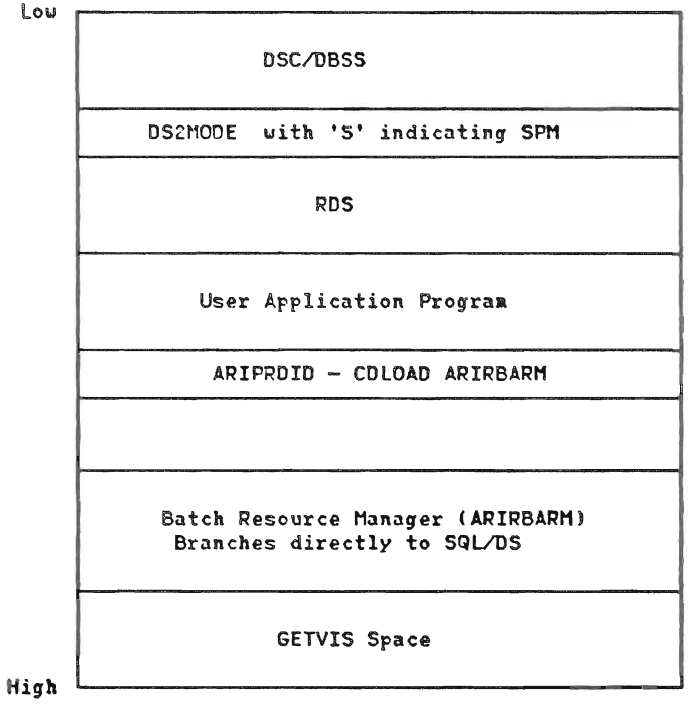
The Batch Resource Manager phase is ARIRBARM. It includes these modules:

- ARIRIRM - Initialization
- ARIRBRM - Mainline function
- ARICCOMD - Cross-partition communication
- ARICIMB - Cross-partition input mailbox build
- ARICMMB - Move cross-partition reply (output mailbox) to application areas
- ARISYSDD - System-dependent routines
  - Entry Points: ARISYSD1 - GETVIS
    ARISYSD2 - FREEVIS
    ARISYSD6 - CDLOAD
    ARISYSD7 - ABEND
    ARISYSDA - PDUMP
- ARICGSE - Get stack extension
- ARICFSE - Free stack extension
- ARIRB13 - Resource Manager WTO function
- ARIRB51 - Resource Manager WTOR function
- ARIYM04 - First-level message module

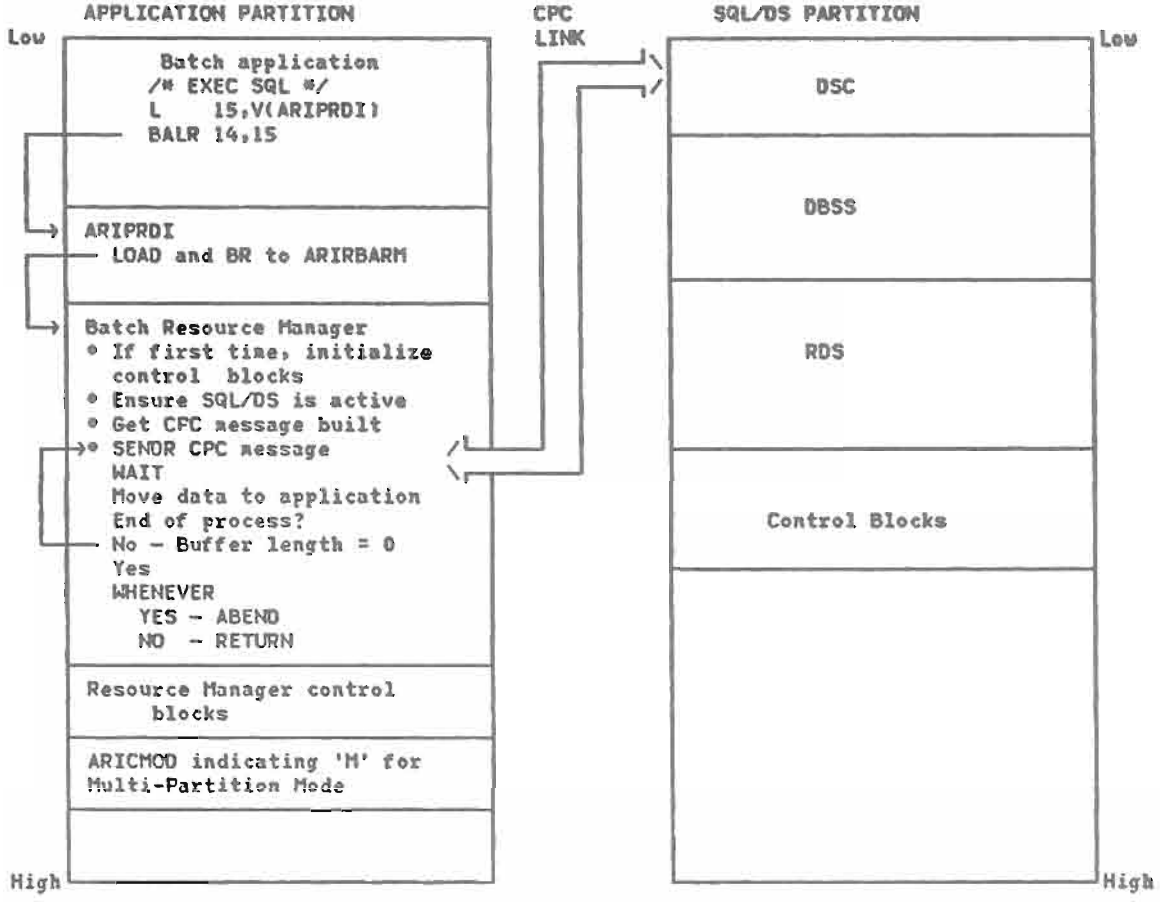
Batch Resource Manager Execution Overview



Batch Resource Manager Single-Partition Mode Environment



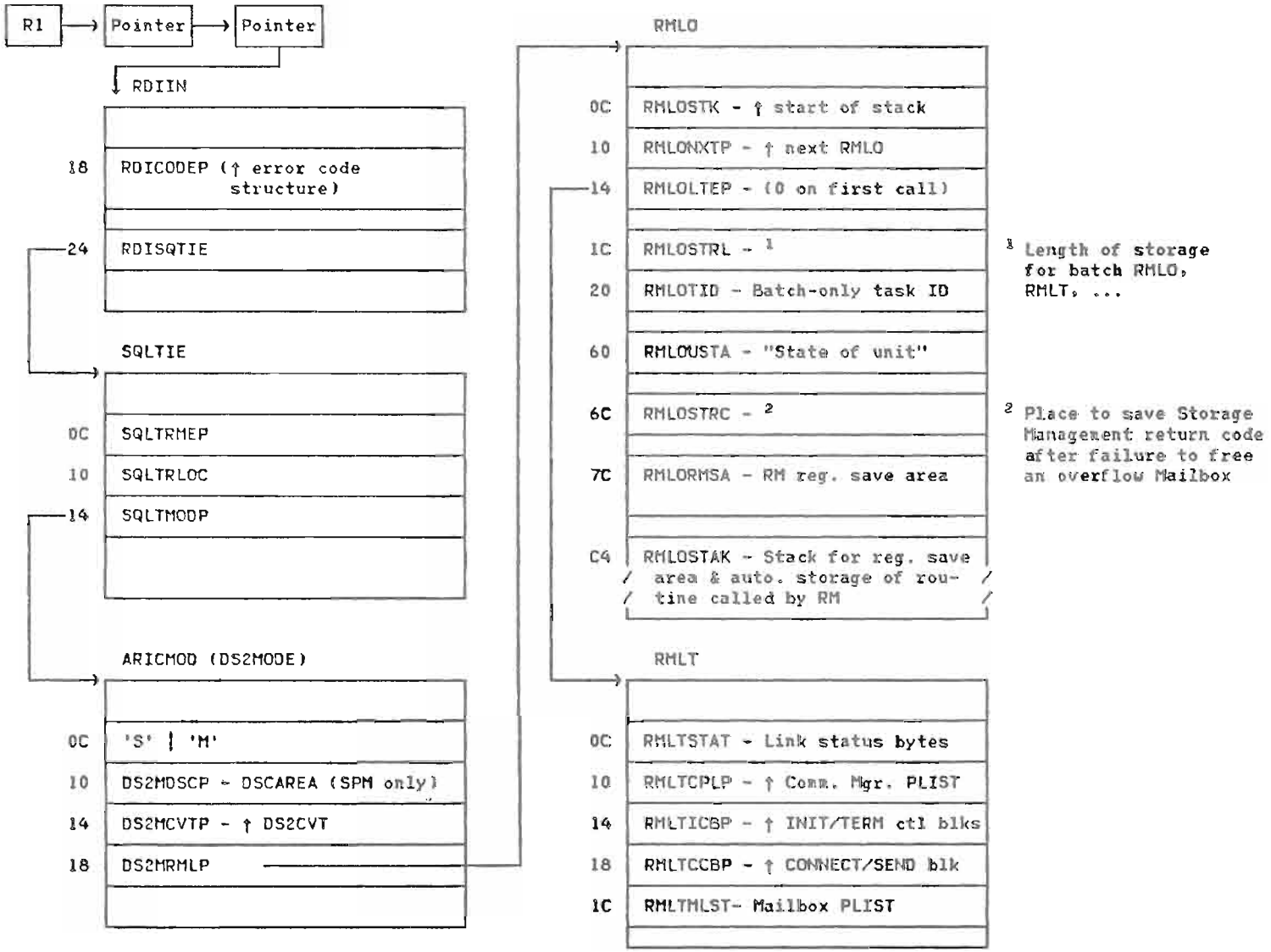
Batch Resource Manager Multi-Partition Mode Environment



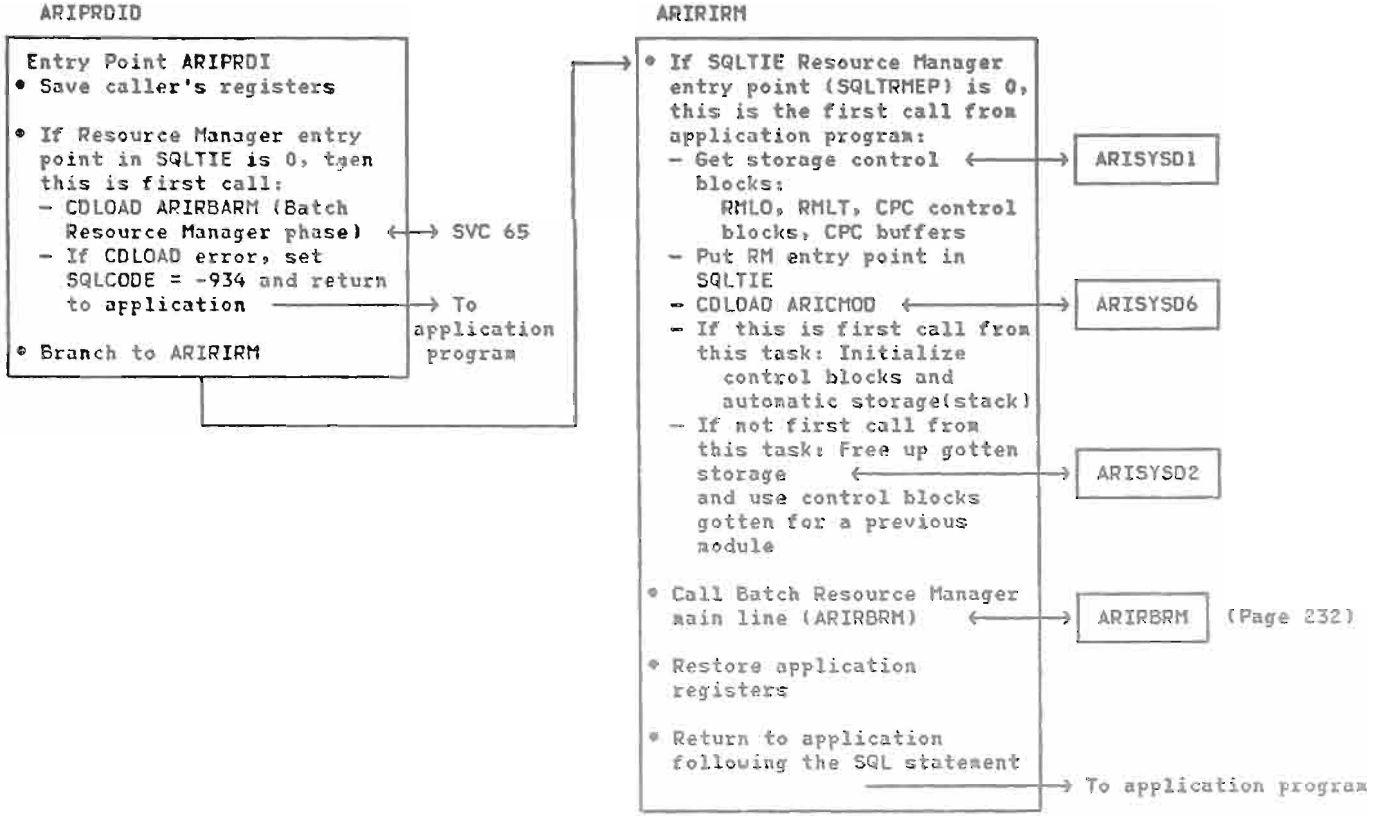
Batch Resource Manager Assumptions

The Batch Resource Manager:
- Assumes a 1-task/1-link relationship.
- Assumes mode transparency - Single- vs Multi-Partition Modes

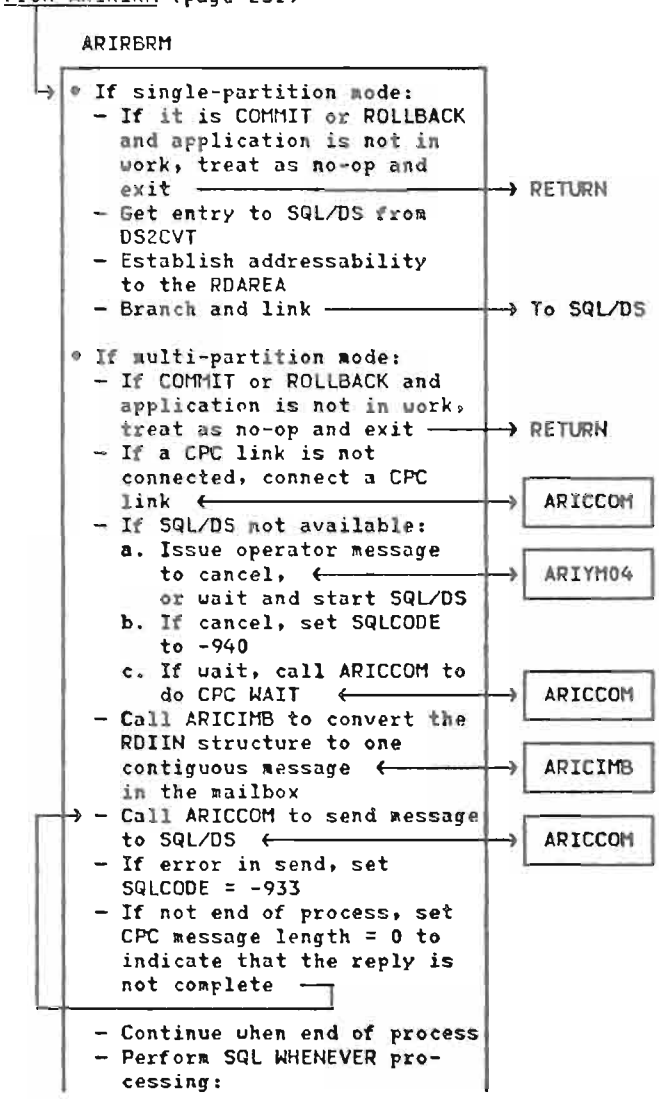
Note: Offsets are in hexadecimal. The RMLO, RMLT RDIIN, and DS2MODE are fully described in the Data Areas section of SQL/DS Logic, Volume 2.

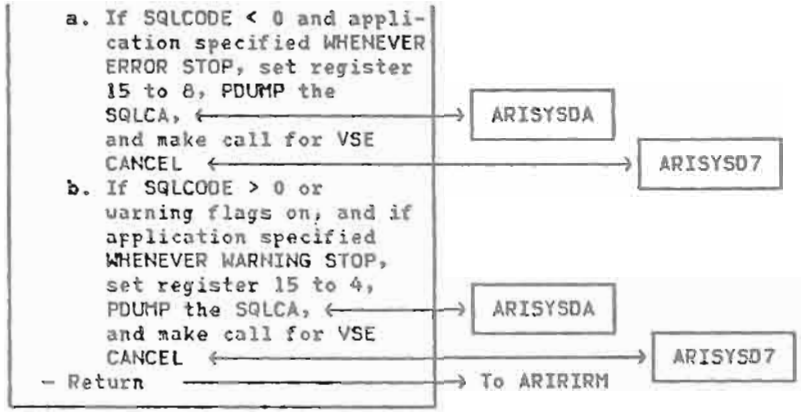


Batch Resource Manager Execution

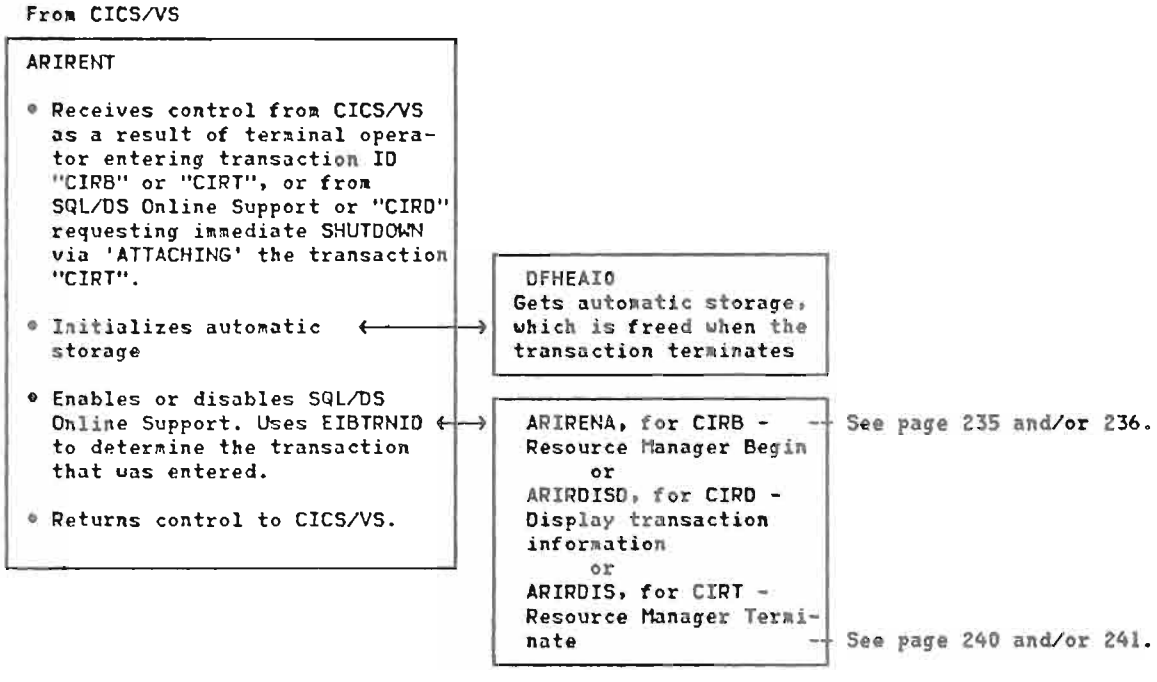


From ARIRIRM (page 231)

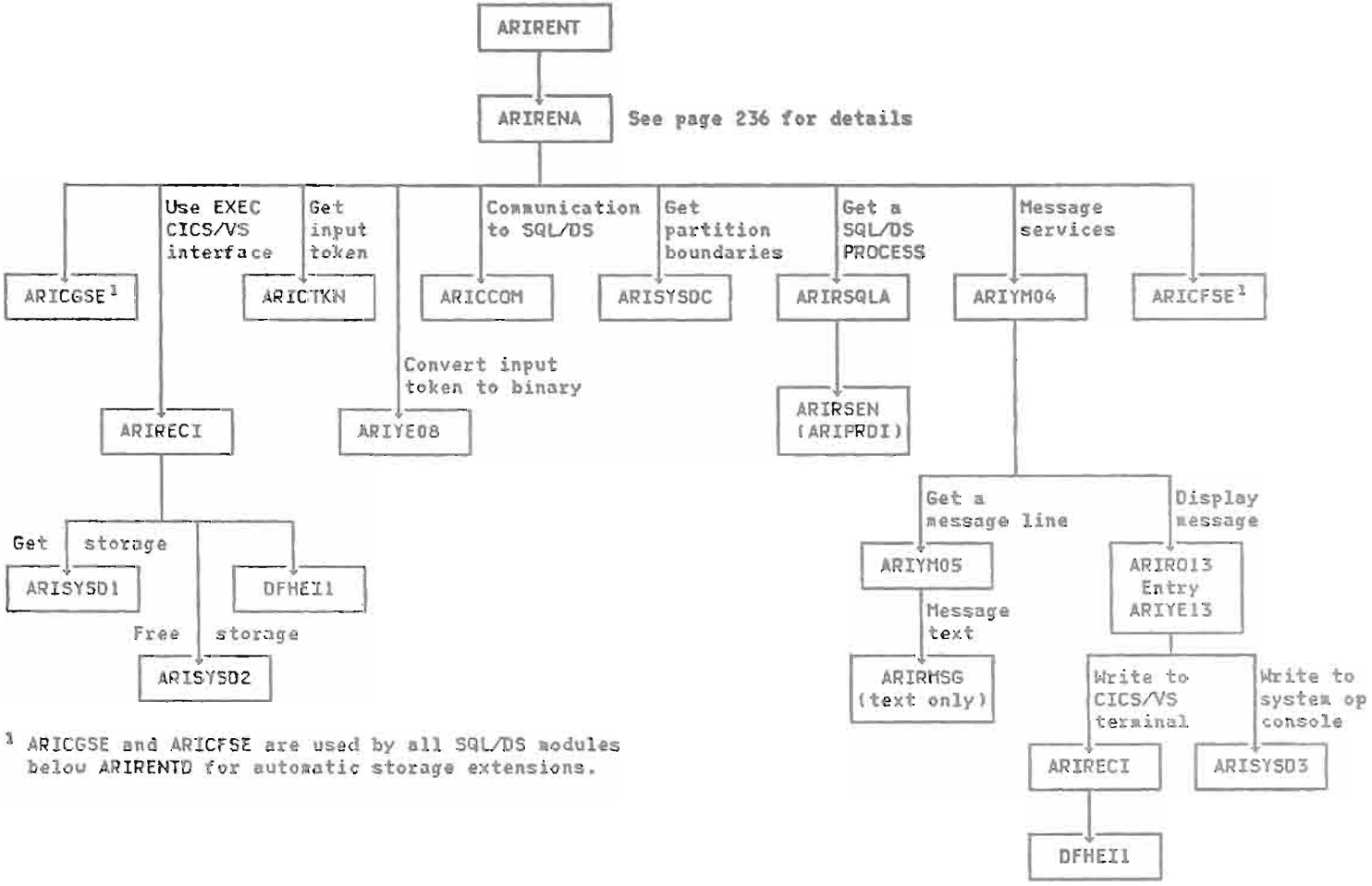



a. If SQLCODE < 0 and application specified WHENEVER ERROR STOP, set register 15 to 8, PDUMP the SQLCA,
ARISYSDA
and make call for VSE CANCEL
ARISYSD7
b. If SQLCODE > 0 or warning flags on, and if application specified WHENEVER WARNING STOP, set register 15 to 4, PDUMP the SQLCA,
ARISYSDA
and make call for VSE CANCEL
ARISYSD7
- Return
To ARIRIRM

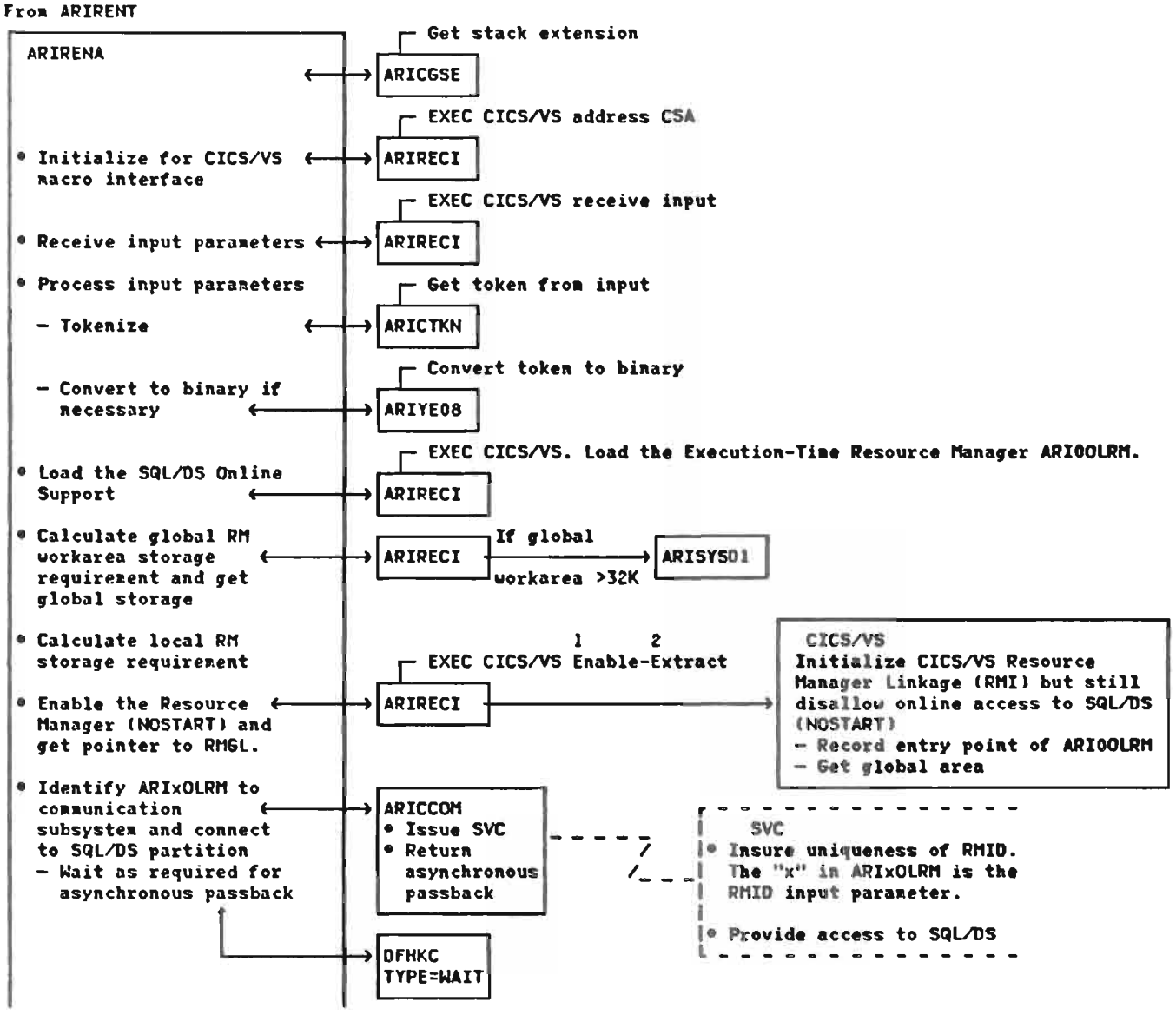
Online Resource Manager - Initialization/Termination Control Overview

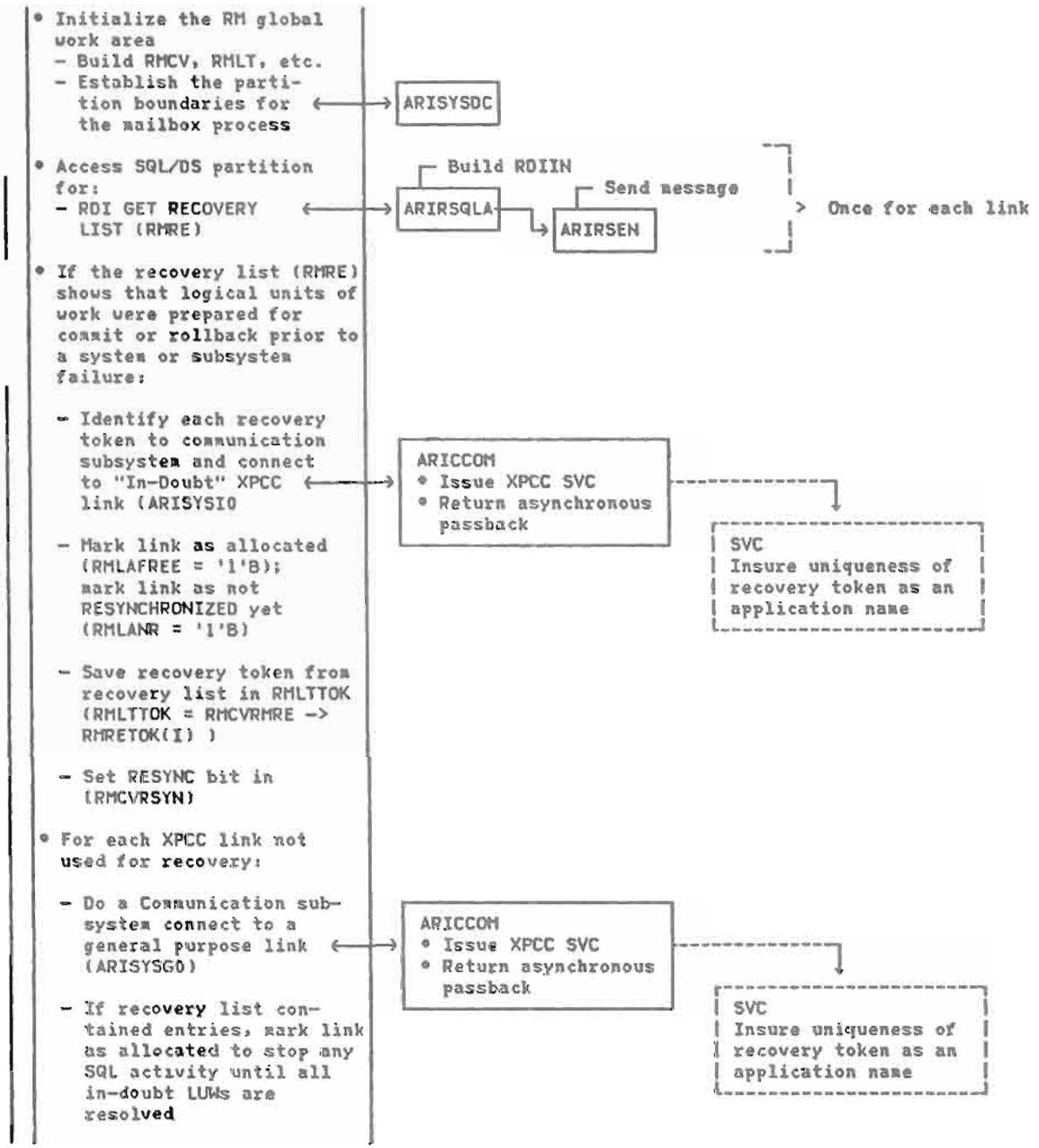


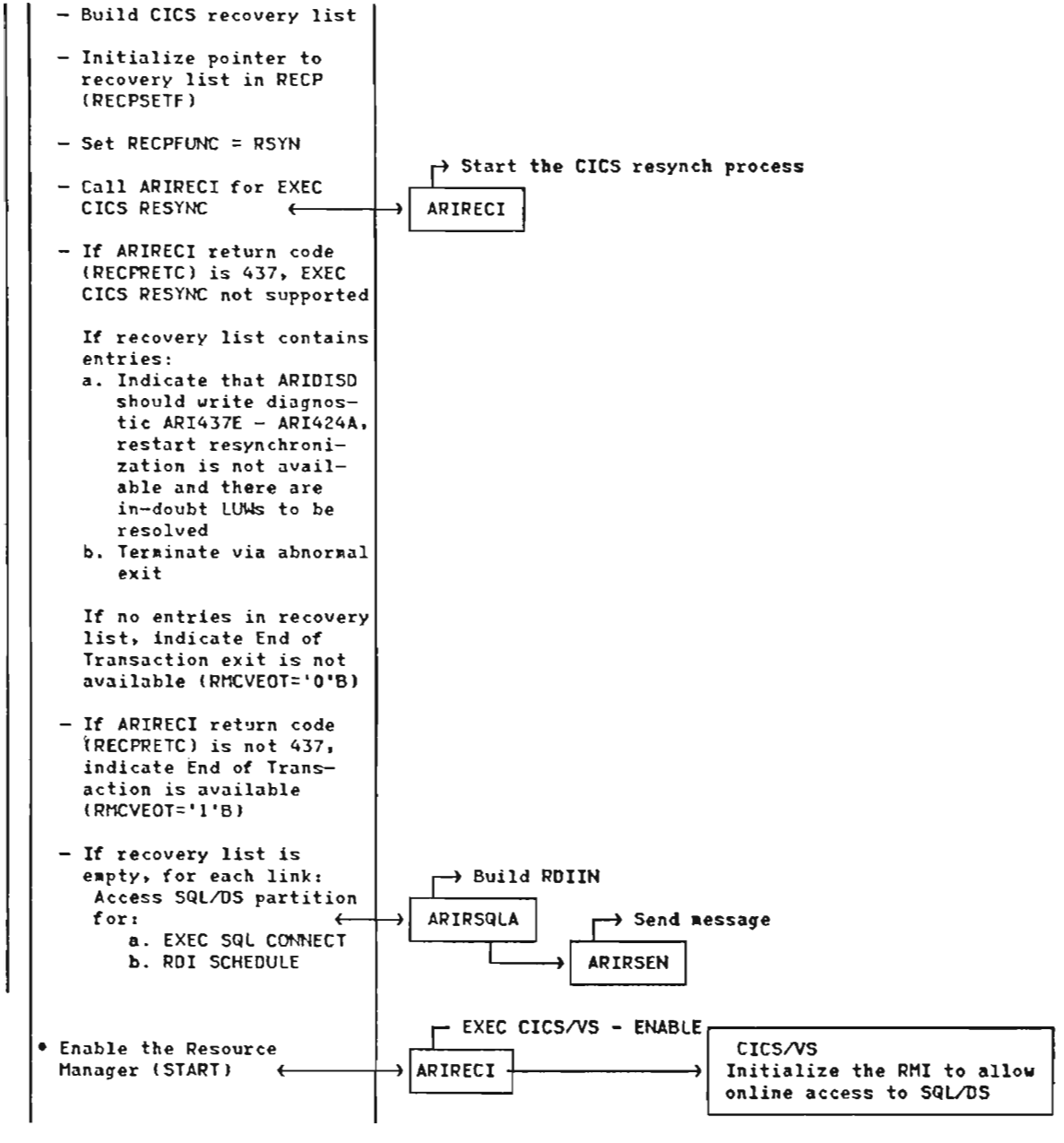
Online Resource Manager - Initialization Overview

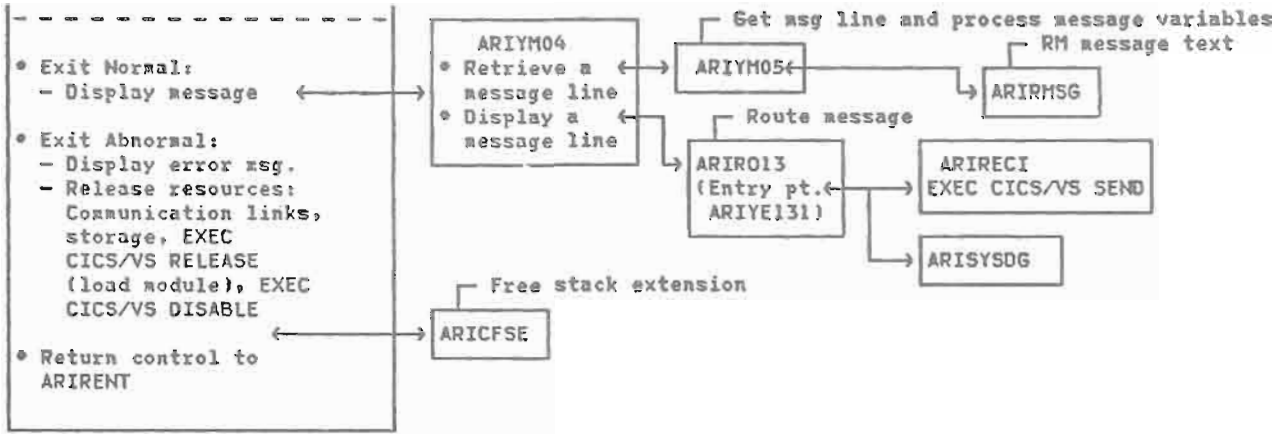


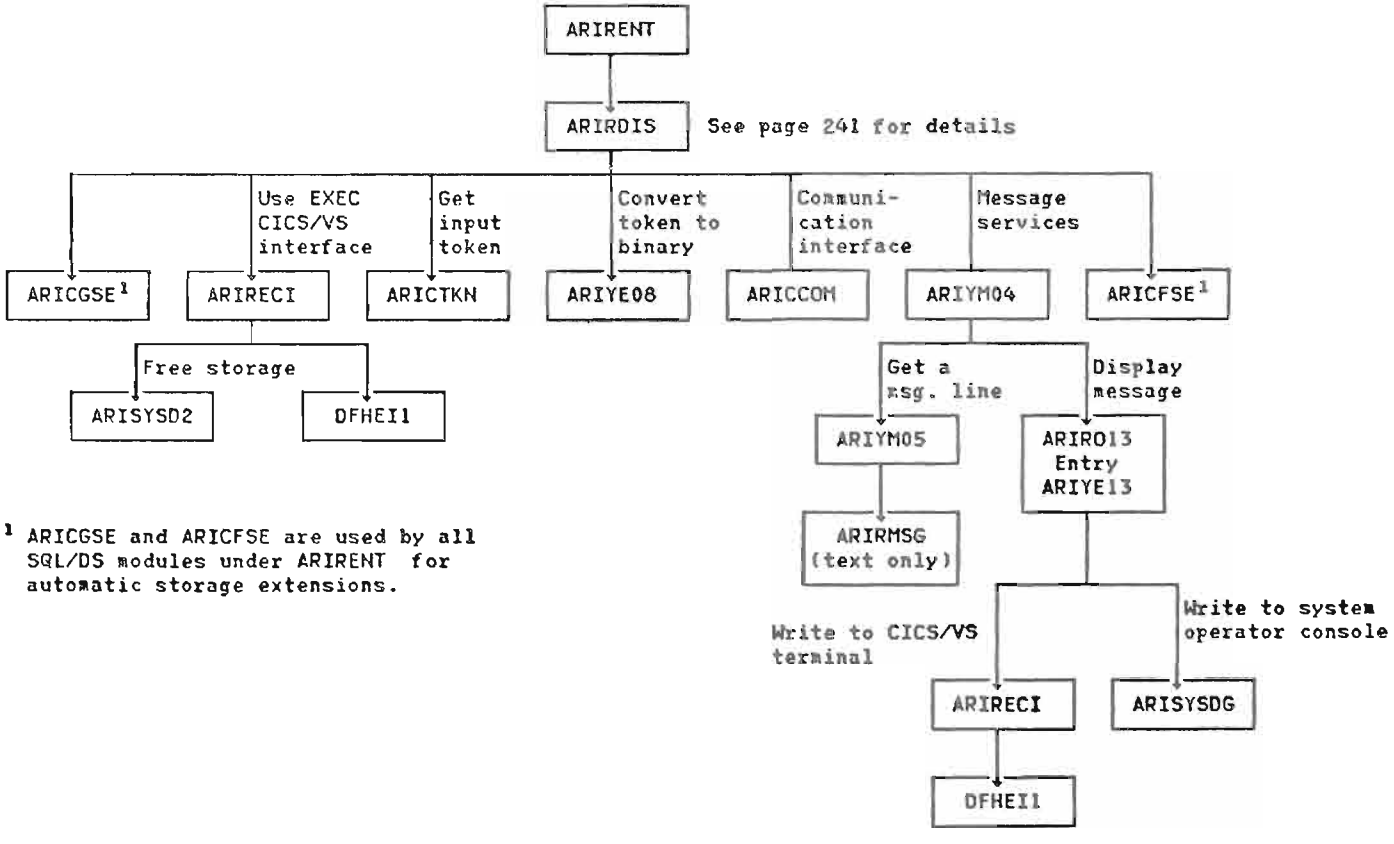
[1] ARICGSE and ARICFSE are used by all SQL/DS modules below ARIRENTD for automatic storage extensions.




• Initialize the RM global work area
- Build RMCV, RMLT, etc.
- Establish the partition boundaries for the mailbox process
ARISYSDC
• Access SQL/DS partition for:
- RDI GET RECOVERY LIST (RMRE)
Build RDIIN
Send message
ARIRSQLA
ARIRSEN
> Once for each link
• If the recovery list (RMRE) shows that logical units of work were prepared for commit or rollback prior to a system or subsystem failure:
- Identify each recovery token to communication subsystem and connect to "In-Doubt" XPCC link (ARISYSI0
ARICCOM
• Issue XPCC SVC
• Return asynchronous passback
SVC
Insure uniqueness of recovery token as an application name
- Mark link as allocated (RMLAFREE = '1'B); mark link as not RESYNCHRONIZED yet (RMLANR = '1'B)
- Save recovery token from recovery list in RMLTTOK (RMLTTOK = RMCVRMRE -> RMRETOK(I))
- Set RESYNC bit in (RMCVRSYN)
• For each XPCC link not used for recovery:
- Do a Communication subsystem connect to a general purpose link (ARISYSG0)
ARICCOM
• Issue XPCC SVC
• Return asynchronous passback
SVC
Insure uniqueness of recovery token as an application name
- If recovery list contained entries, mark link as allocated to stop any SQL activity until all in-doubt LUWs are resolved

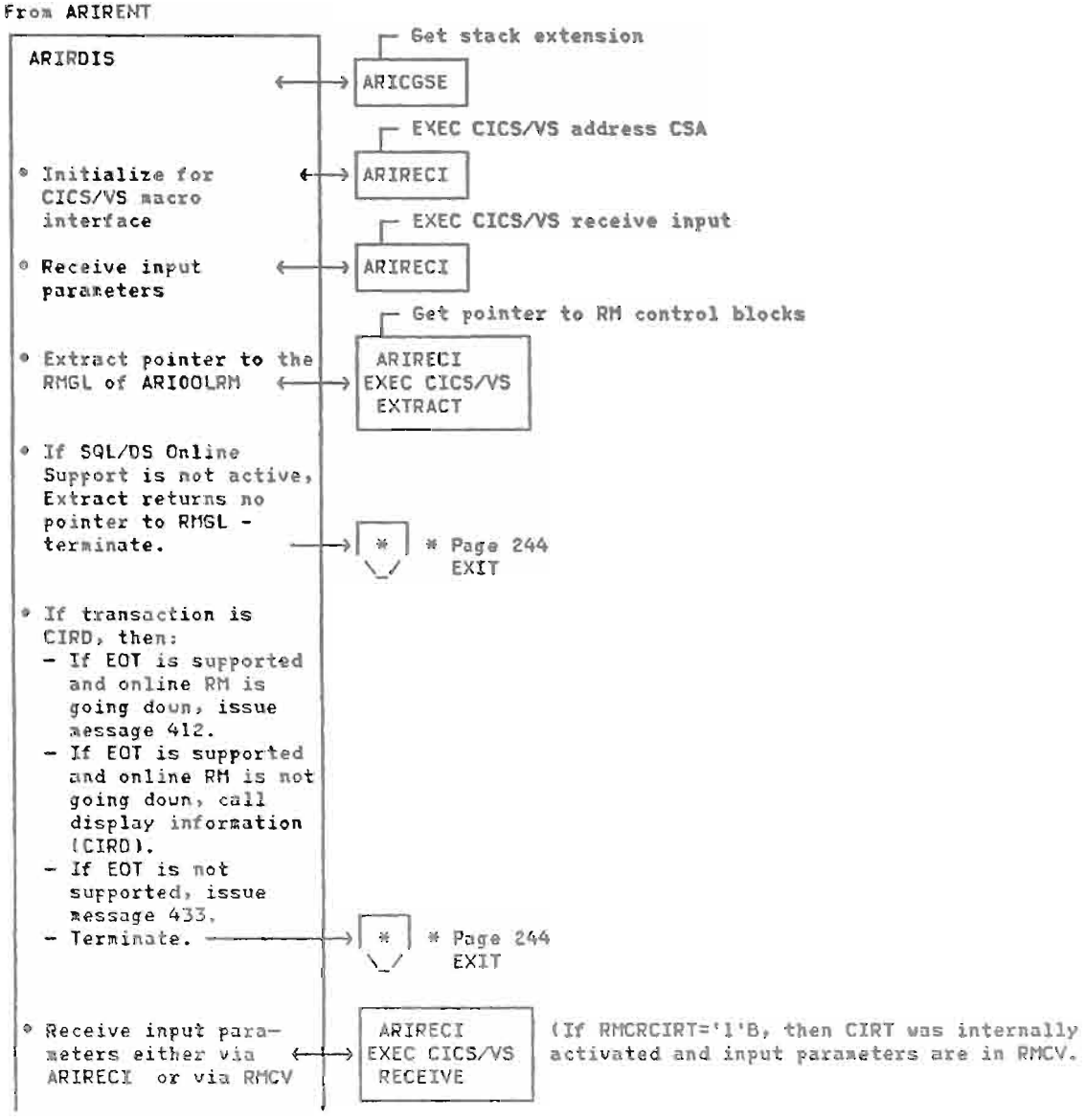


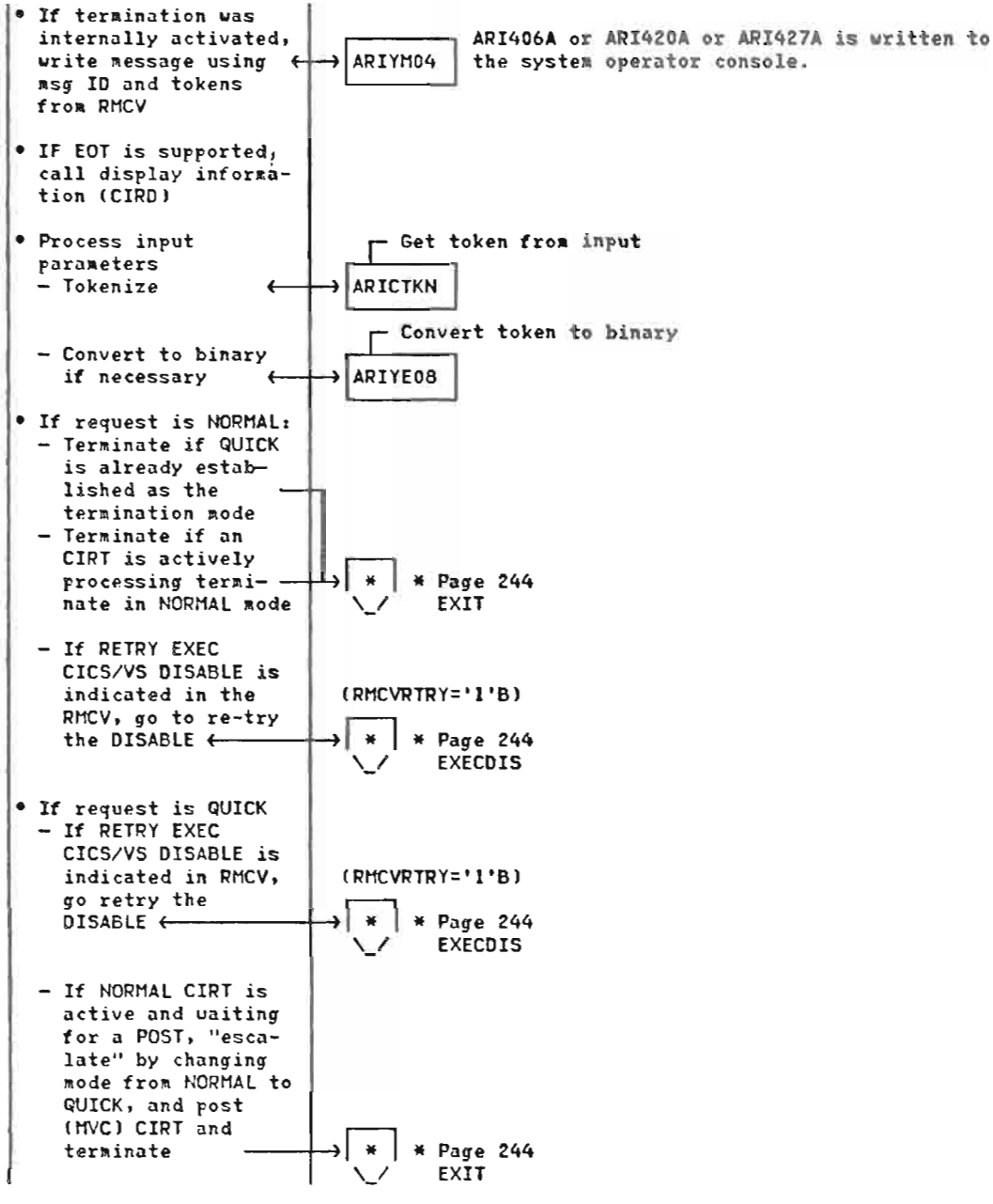

• Exit Normal:
– Display message
• Exit Abnormal:
– Display error msg.
– Release resources:
Communication links,
storage, EXEC
CICS/VS RELEASE
(load module), EXEC
CICS/VS DISABLE
• Return control to
ARIRENT
ARIYM04
• Retrieve a message line
• Display a message line
Get msg line and process message variables
ARIYM05
RM message text
ARIRMSG
Route message
ARIRO13
(Entry pt. ARIYE131)
ARIRECI
EXEC CICS/VS SEND
ARISYSDG
Free stack extension
ARICFSE

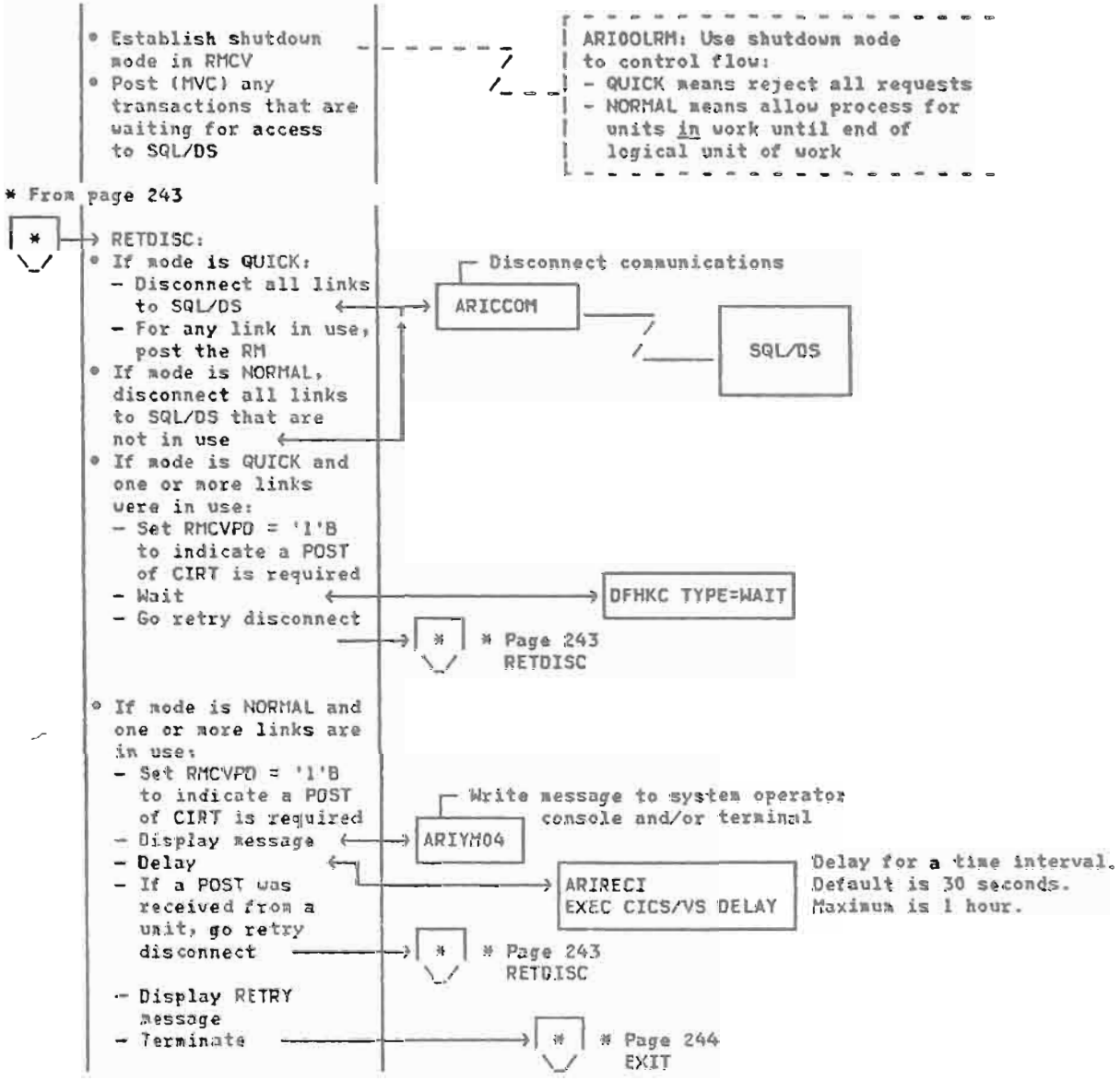


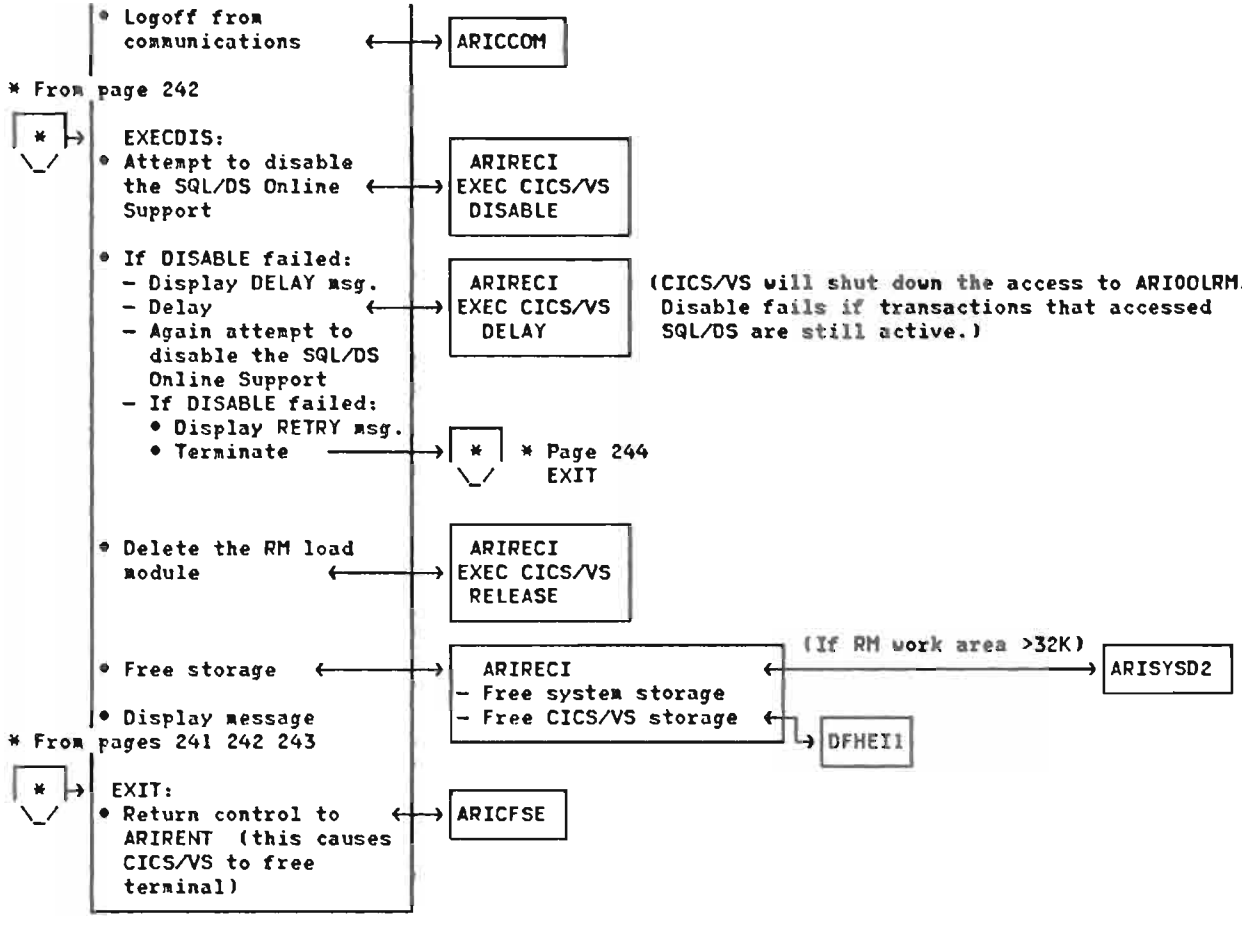
[1] ARICGSE and ARICFSE are used by all SQL/DS modules under ARIRENT for automatic storage extensions.

Online Resource Manager - Termination Detail

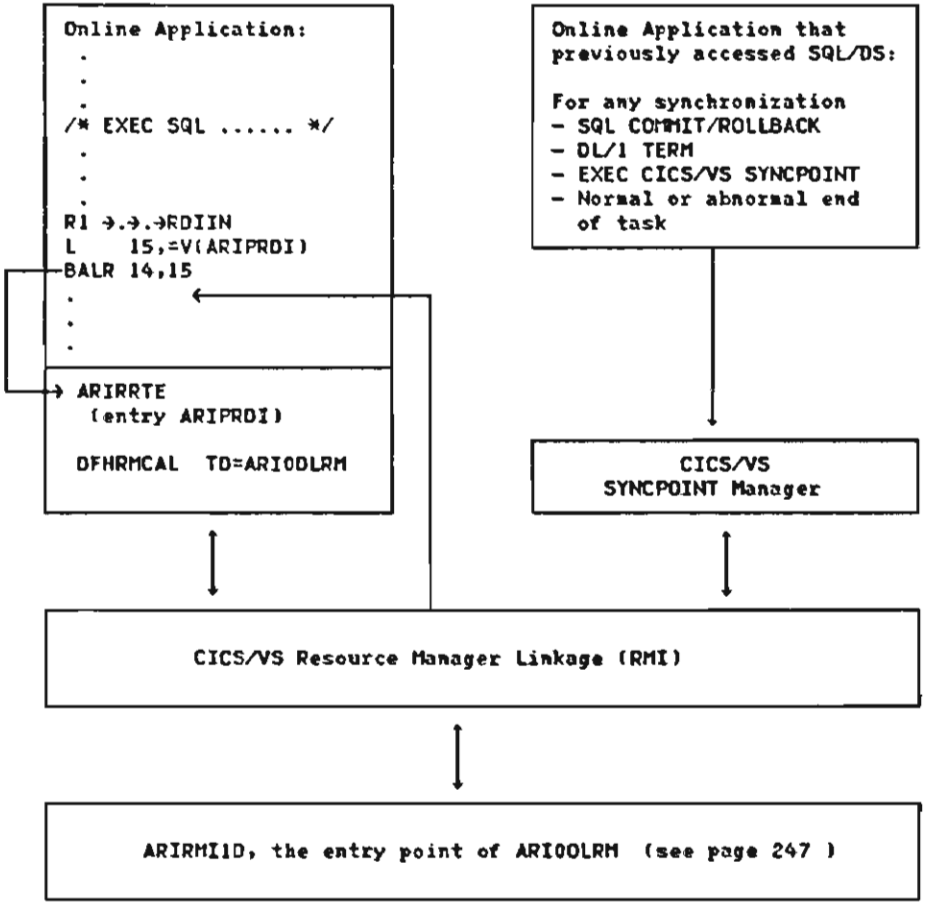




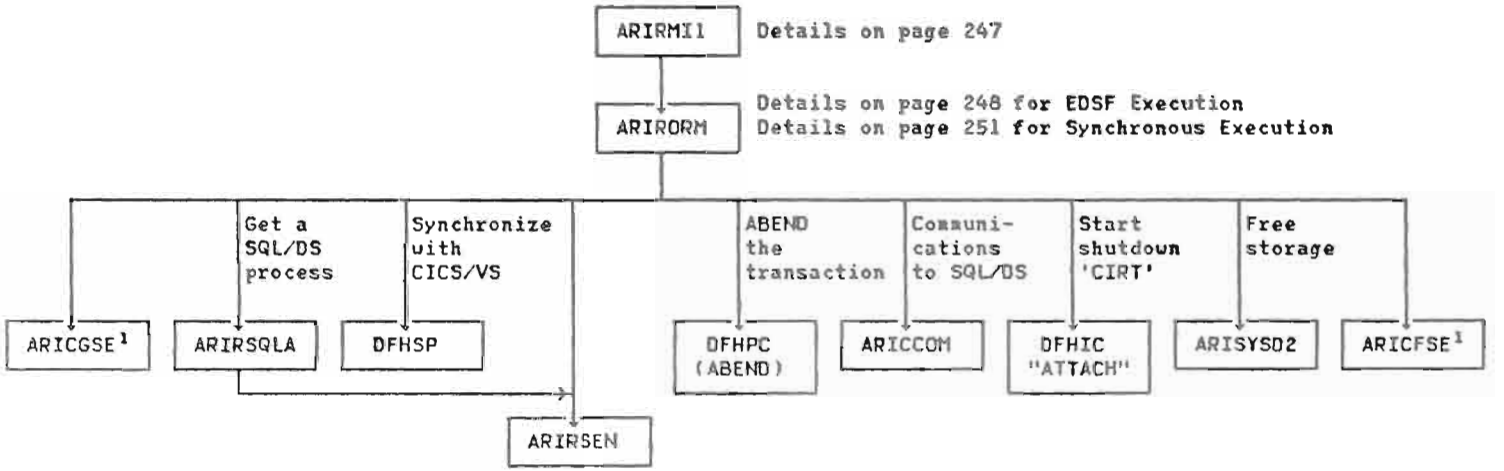

• Establish shutdown mode in RMCV
• Post (MVC) any transactions that are waiting for access to SQL/DS
ARI0OLRM: Use shutdown mode to control flow:
- QUICK means reject all requests
- NORMAL means allow process for units in work until end of logical unit of work
* From page 243
*
RETDISC:
• If mode is QUICK:
- Disconnect all links to SQL/DS
- For any link in use, post the RM
• If mode is NORMAL, disconnect all links to SQL/DS that are not in use
Disconnect communications
ARICCOM
SQL/DS
• If mode is QUICK and one or more links were in use:
- Set RMCVPO = '1'B to indicate a POST of CIRT is required
- Wait
- Go retry disconnect
DFHKC TYPE=WAIT
* Page 243 RETDISC
• If mode is NORMAL and one or more links are in use:
- Set RMCVPO = '1'B to indicate a POST of CIRT is required
- Display message
- Delay
- If a POST was received from a unit, go retry disconnect
Write message to system operator console and/or terminal
ARIYM04
ARIRECI EXEC CICS/VS DELAY
Delay for a time interval. Default is 30 seconds. Maximum is 1 hour.
* Page 243 RETDISC
- Display RETRY message
- Terminate
* Page 244 EXIT



Online Resource Manager - Invocation of Execution

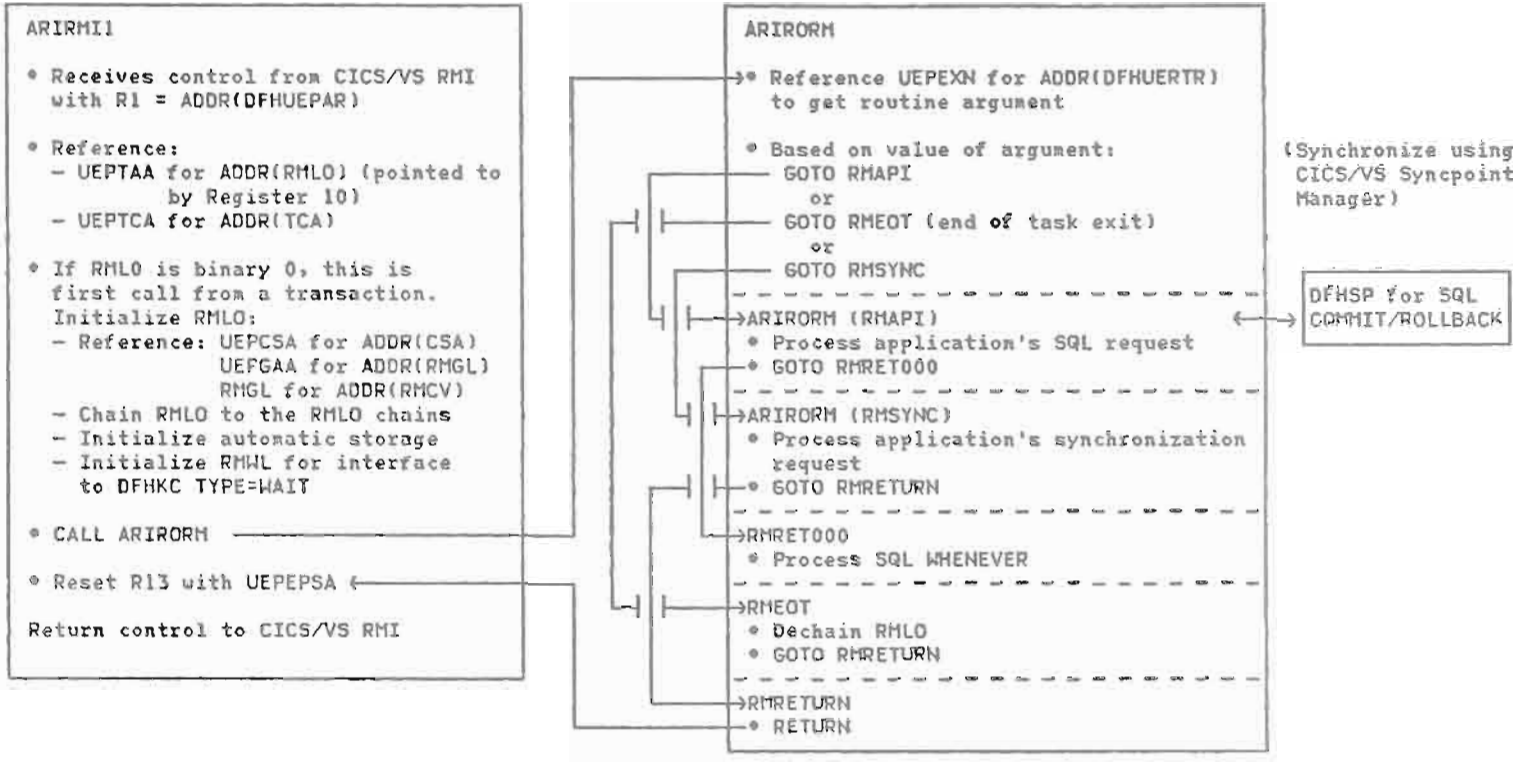


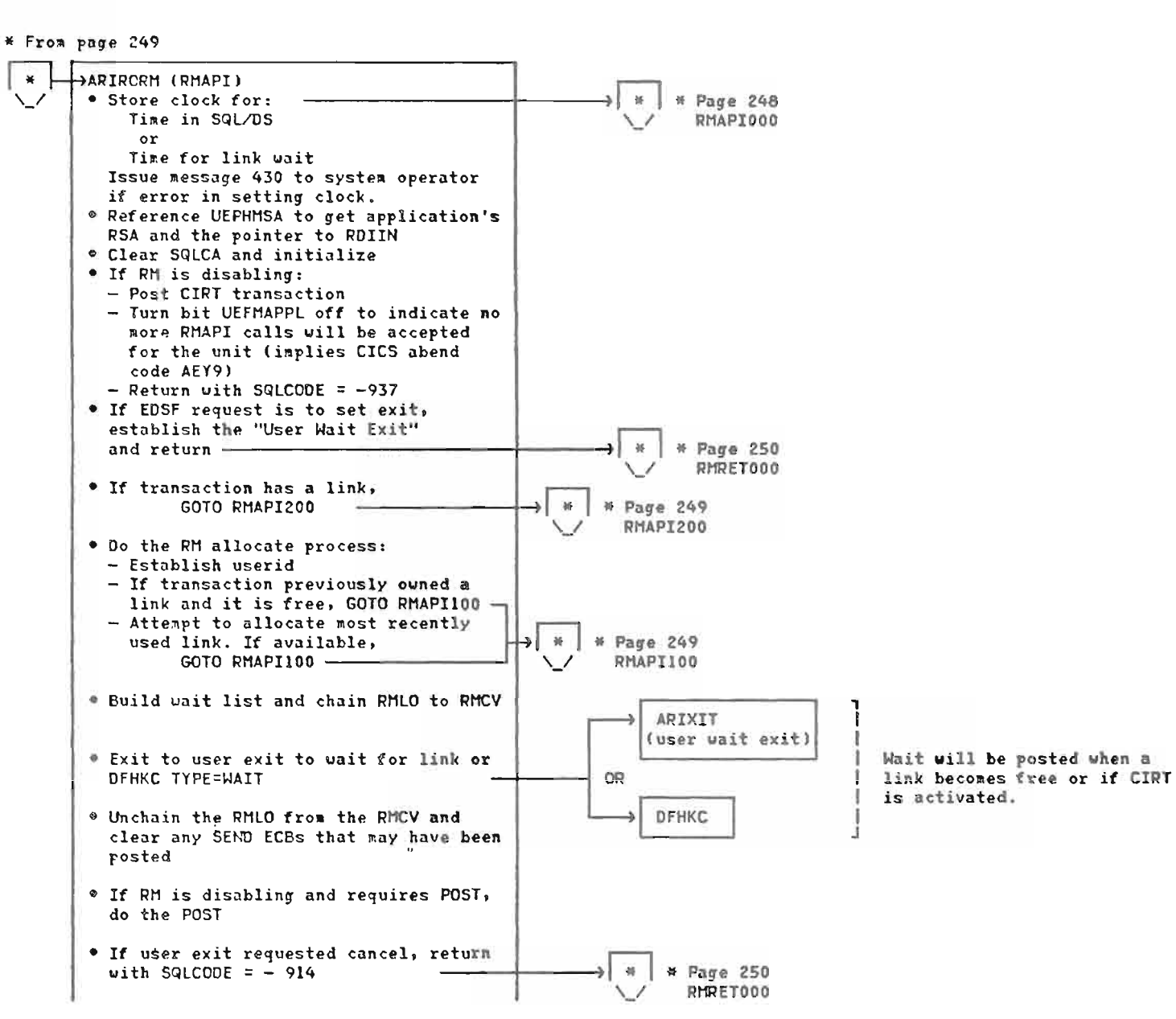
Online Resource Manager — Execution Overview

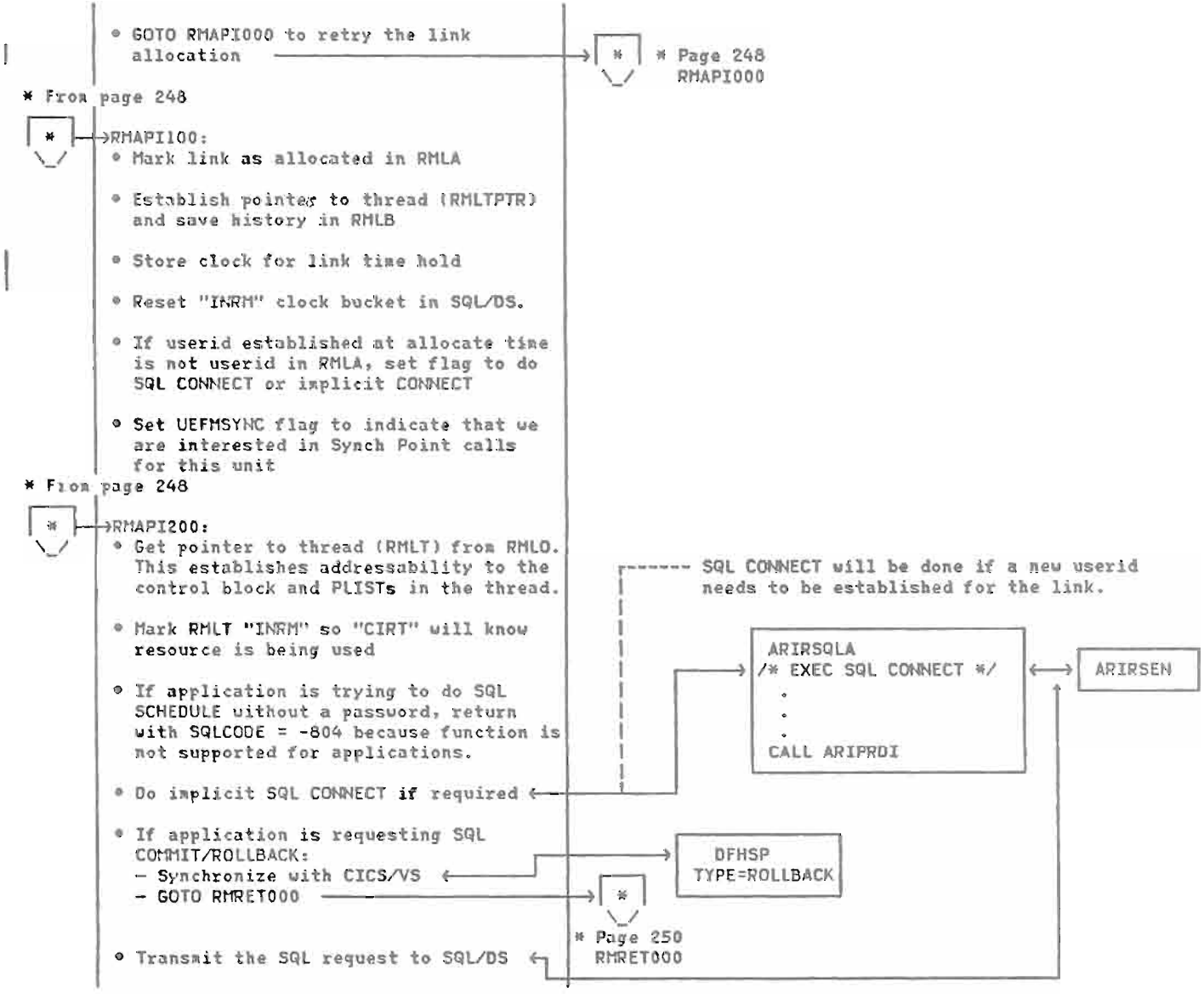


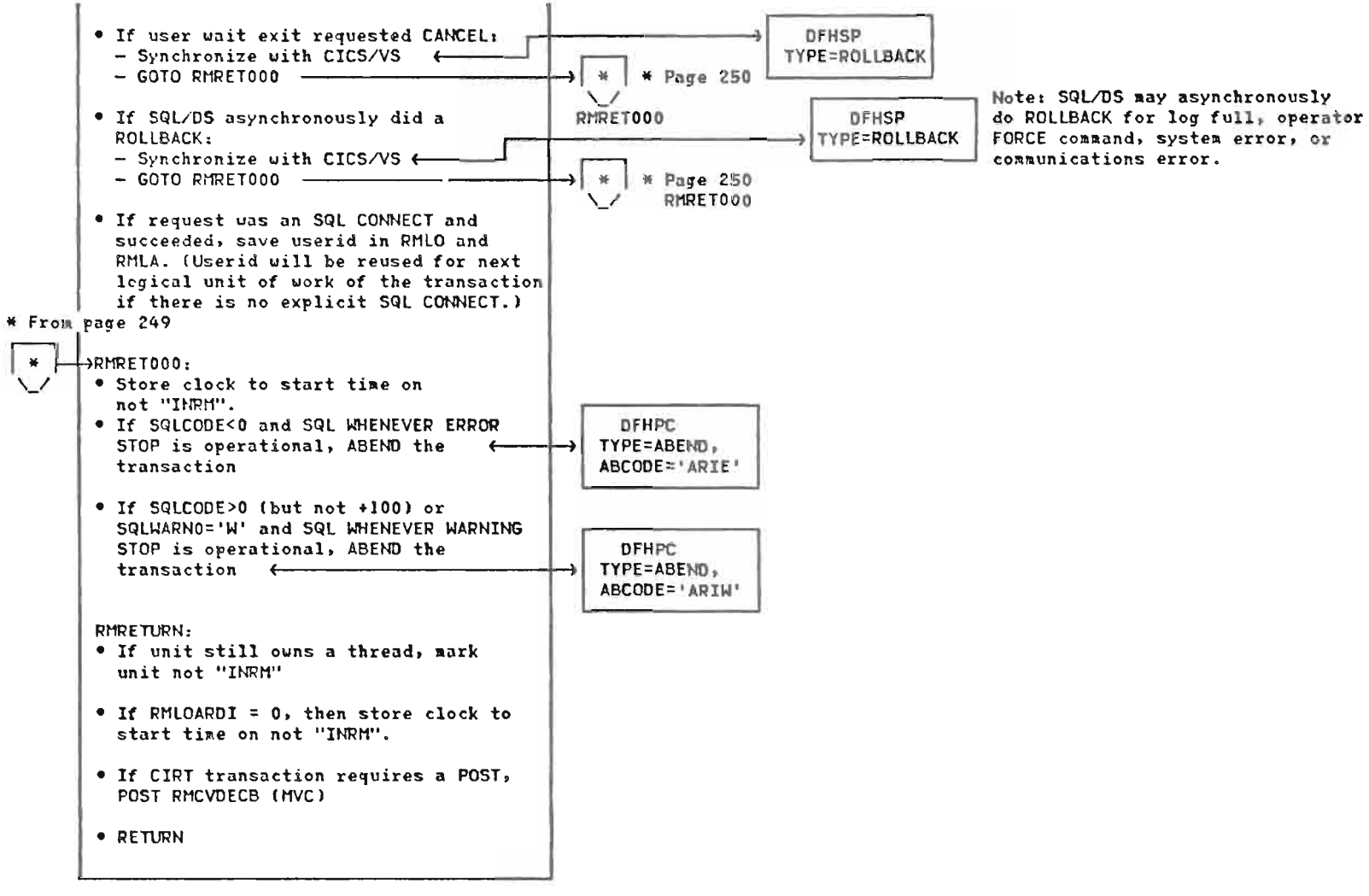
[1] ARICGSE and ARICFSE are used by all SQL/DS modules below ARIRMIID for automatic storage extensions.

Online Resource Manager - Execution Control

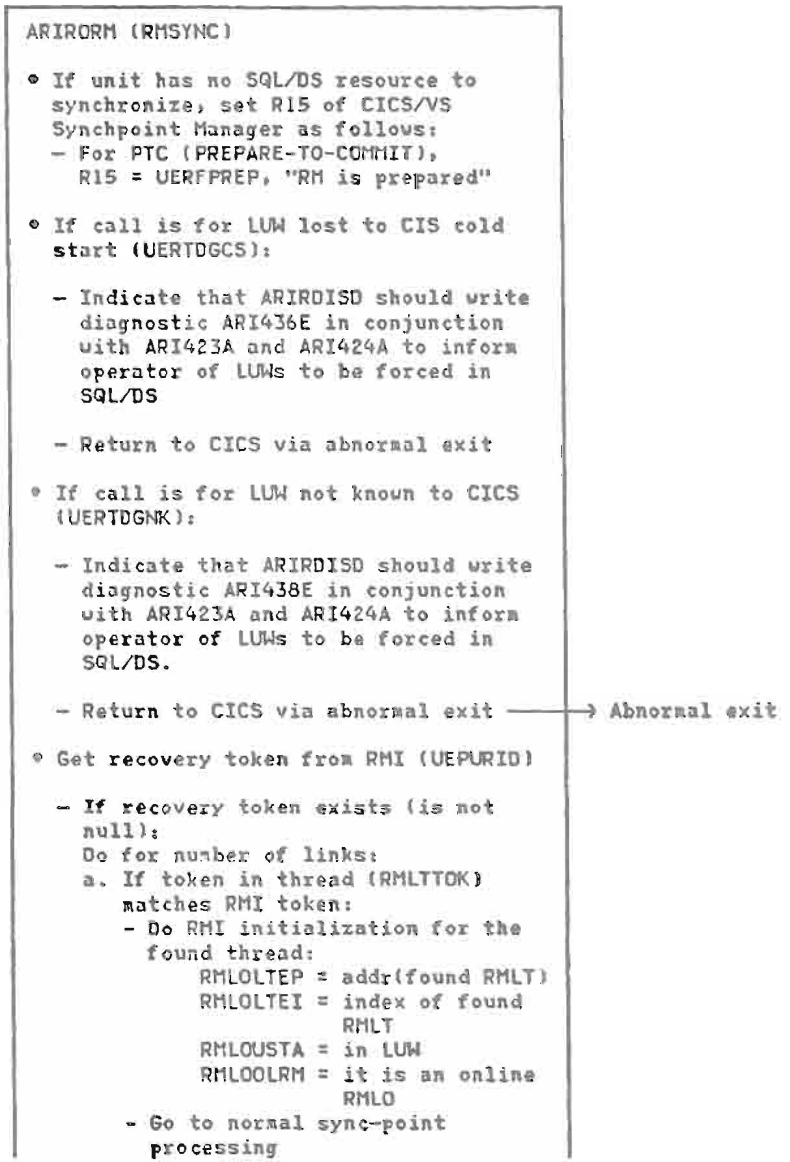


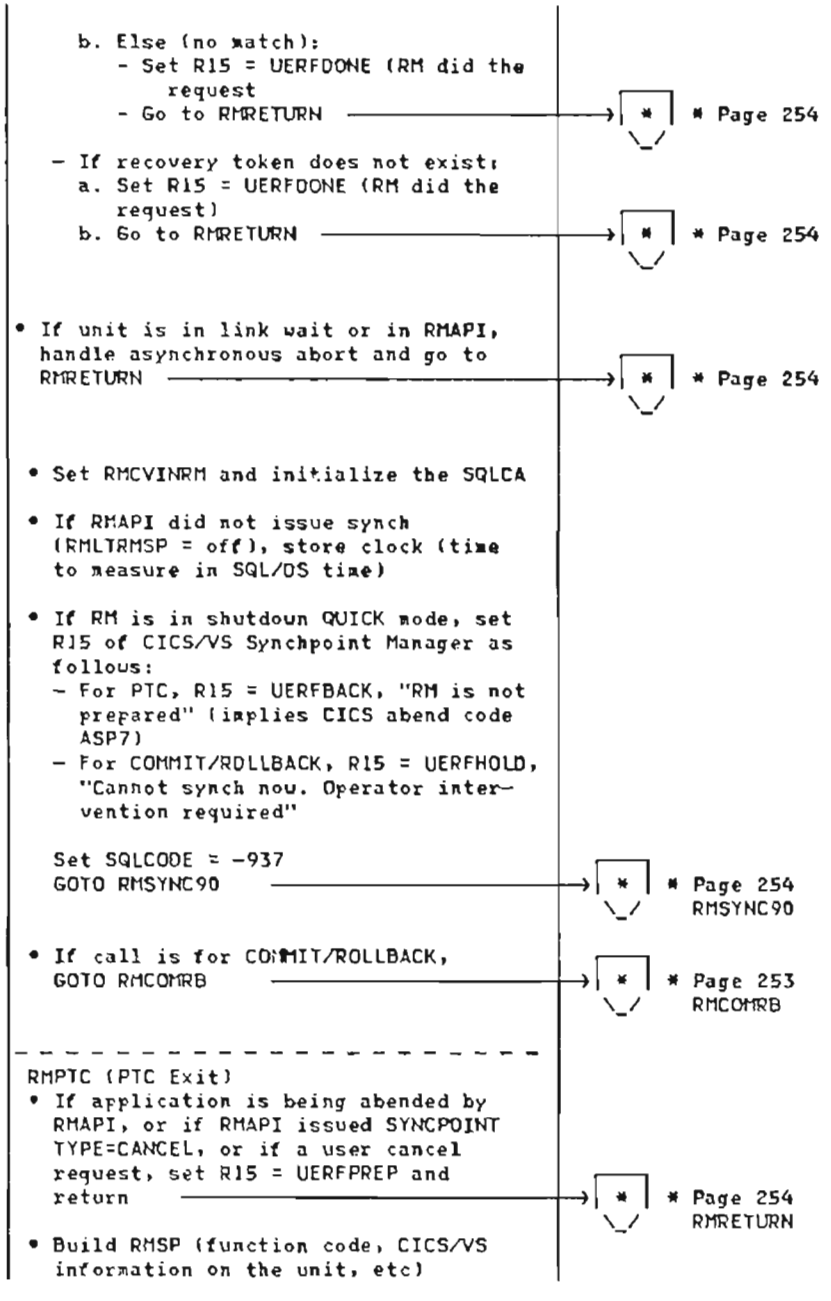


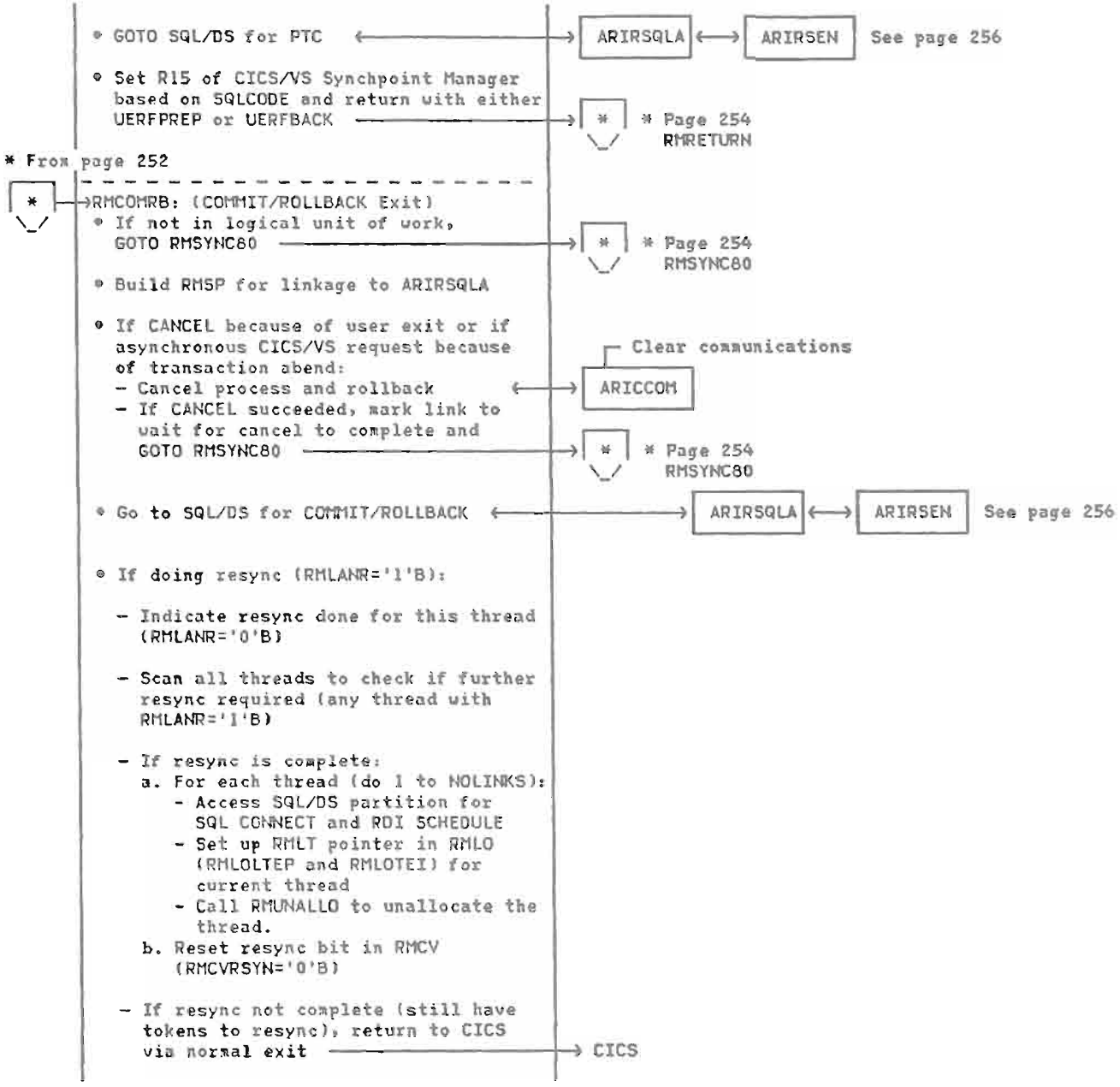

• GOTO RMAPI000 to retry the link allocation
* Page 248 RMAPI000
* From page 248
RMAPI100:
• Mark link as allocated in RMLA
• Establish pointer to thread (RMLTPTR) and save history in RMLB
• Store clock for link time hold
• Reset "INRM" clock bucket in SQL/DS.
• If userid established at allocate time is not userid in RMLA, set flag to do SQL CONNECT or implicit CONNECT
• Set UEFMSYNC flag to indicate that we are interested in Synch Point calls for this unit
* From page 248
RMAPI200:
• Get pointer to thread (RMLT) from RMLO. This establishes addressability to the control block and PLISTs in the thread.
• Mark RMLT "INRM" so "CIRT" will know resource is being used
• If application is trying to do SQL SCHEDULE without a password, return with SQLCODE = -804 because function is not supported for applications.
• Do implicit SQL CONNECT if required
• If application is requesting SQL COMMIT/ROLLBACK:
– Synchronize with CICS/VS
– GOTO RMRET000
* Page 250 RMRET000
• Transmit the SQL request to SQL/DS
------ SQL CONNECT will be done if a new userid needs to be established for the link.
ARIRSQLA
/* EXEC SQL CONNECT */
.
.
.
CALL ARIPRDI
ARIRSEN
DFHSP TYPE=ROLLBACK

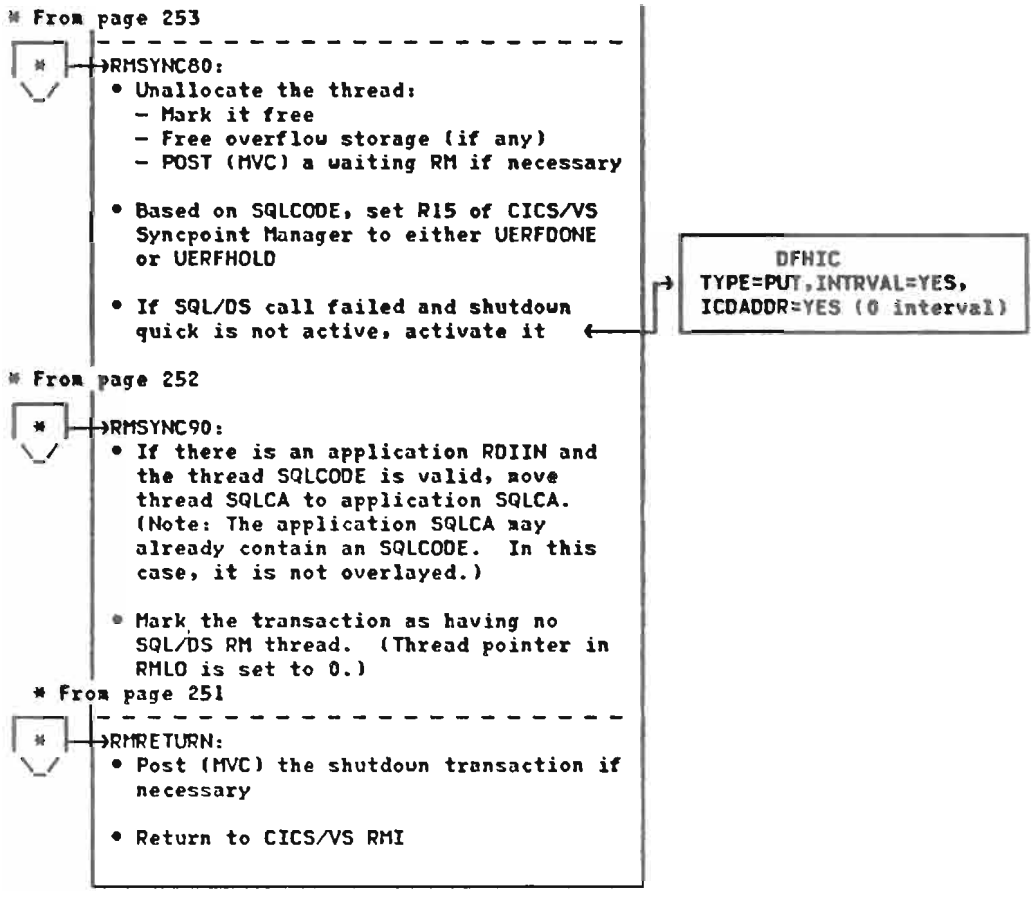


Online Resource Manager - Synchronization Execution



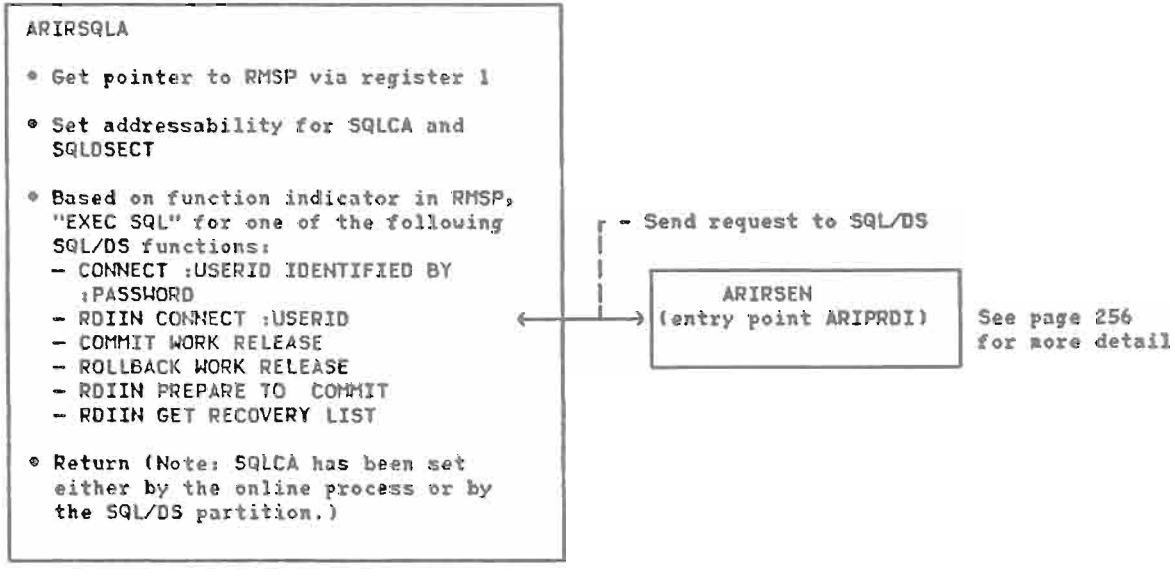


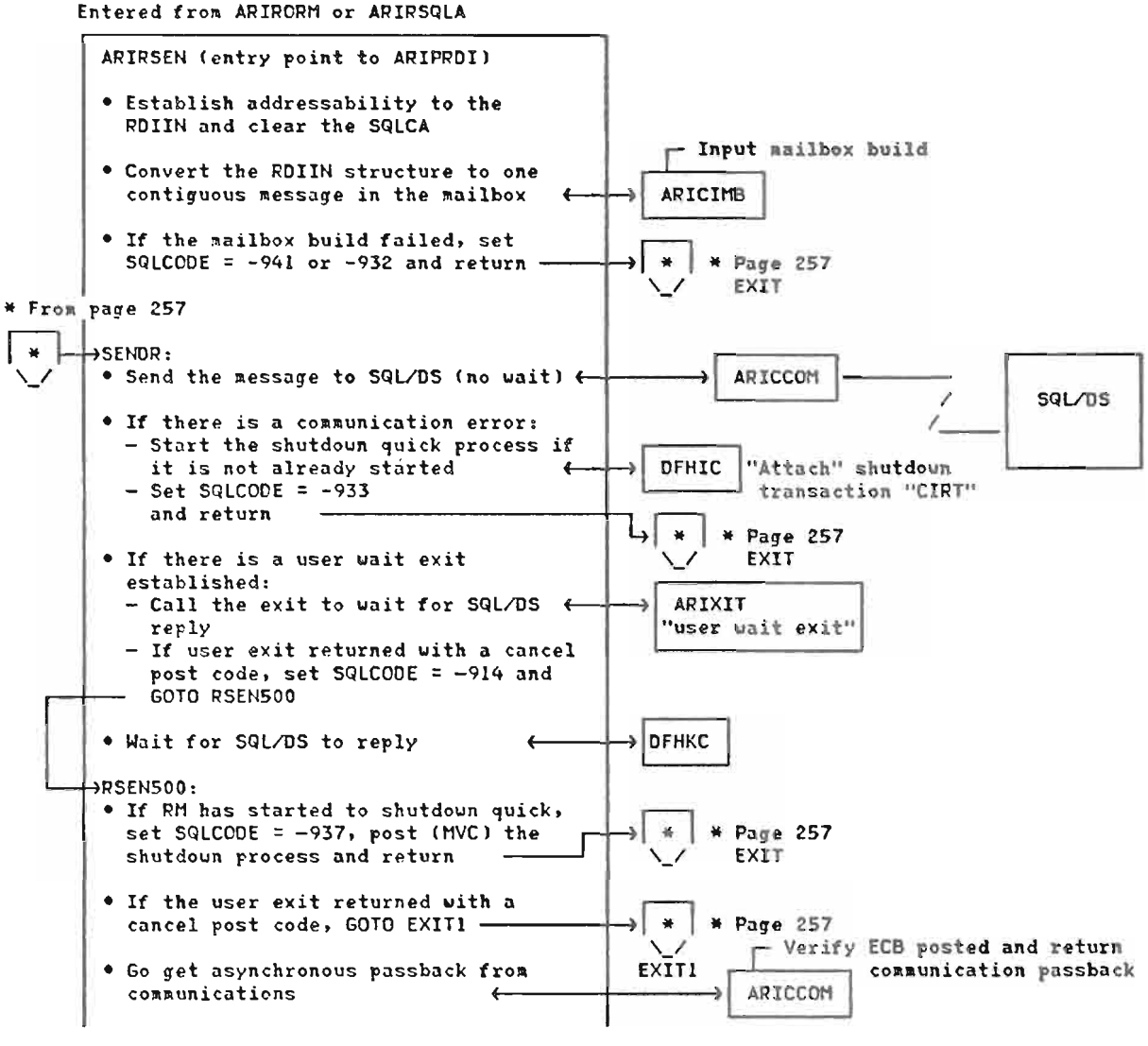

* GOTO SQL/DS for PTC
ARIRSQLA
ARIRSEN
See page 256
* Set R15 of CICS/VS Synchpoint Manager based on SQLCODE and return with either UERFPREP or UERFBACK
* Page 254 RMRETURN
* From page 252
RMCOMRB: (COMMIT/ROLLBACK Exit)
* If not in logical unit of work, GOTO RMSYNC80
* Page 254 RMSYNC80
* Build RMSP for linkage to ARIRSQLA
* If CANCEL because of user exit or if asynchronous CICS/VS request because of transaction abend:
Clear communications
ARICCOM
- Cancel process and rollback
- If CANCEL succeeded, mark link to wait for cancel to complete and GOTO RMSYNC80
* Page 254 RMSYNC80
* Go to SQL/DS for COMMIT/ROLLBACK
ARIRSQLA
ARIRSEN
See page 256
* If doing resync (RMLANR='1'B):
- Indicate resync done for this thread (RMLANR='0'B)
- Scan all threads to check if further resync required (any thread with RMLANR='1'B)
- If resync is complete:
a. For each thread (do 1 to NOLINKS):
- Access SQL/DS partition for SQL CONNECT and RDI SCHEDULE
- Set up RMLT pointer in RMLO (RMLOLTEP and RMLOTEI) for current thread
- Call RMUNALLO to unallocate the thread.
b. Reset resync bit in RMCV (RMCVRSYN='0'B)
- If resync not complete (still have tokens to resync), return to CICS via normal exit
CICS

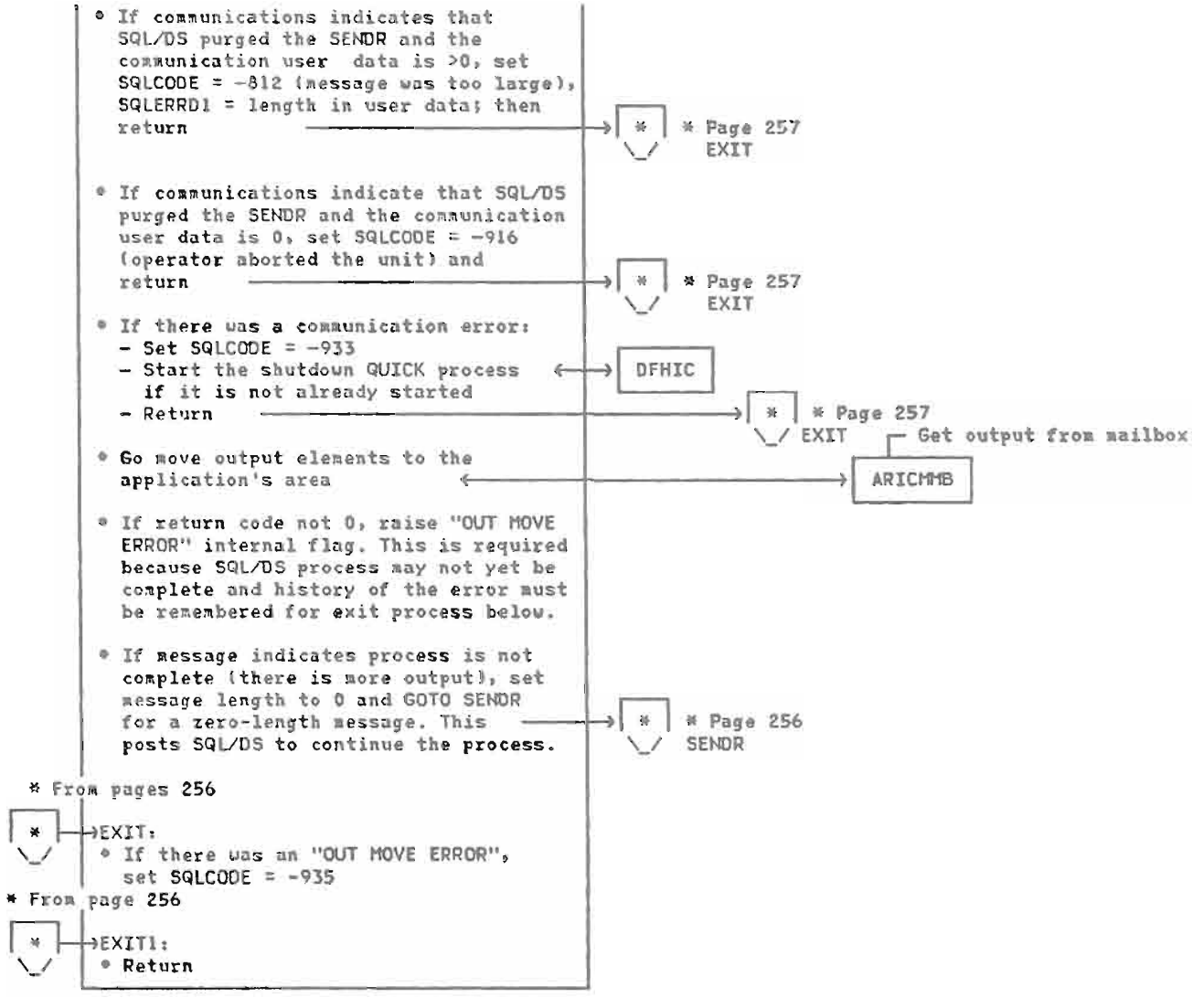


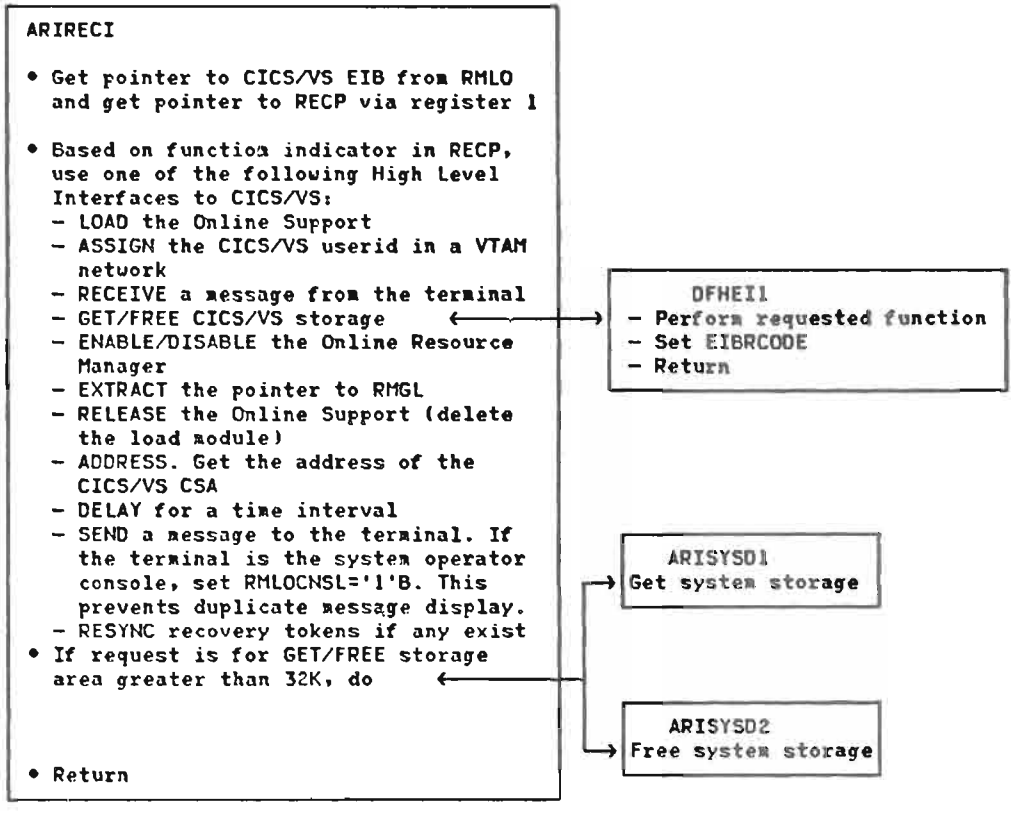
Online Resource Manager - Miscellaneous Functions

Online Resource Manager - Execute SQL Linkage

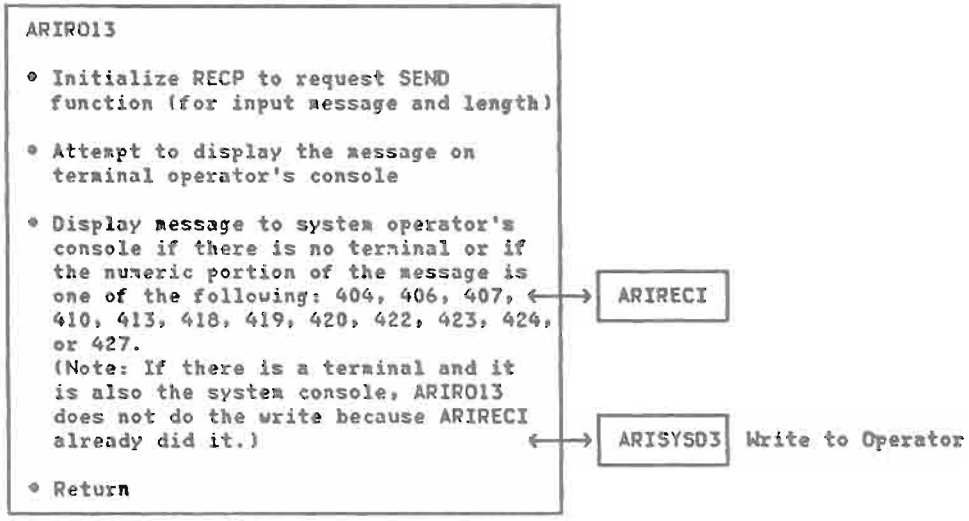





• If communications indicates that SQL/DS purged the SENDR and the communication user data is >0, set SQLCODE = -812 (message was too large), SQLERRD1 = length in user data; then return
* Page 257 EXIT
• If communications indicate that SQL/DS purged the SENDR and the communication user data is 0, set SQLCODE = -916 (operator aborted the unit) and return
* Page 257 EXIT
• If there was a communication error:
- Set SQLCODE = -933
- Start the shutdown QUICK process if it is not already started
- Return
DFHIC
* Page 257 EXIT
Get output from mailbox
• Go move output elements to the application's area
ARICMMB
• If return code not 0, raise "OUT MOVE ERROR" internal flag. This is required because SQL/DS process may not yet be complete and history of the error must be remembered for exit process below.
• If message indicates process is not complete (there is more output), set message length to 0 and GOTO SENDR for a zero-length message. This posts SQL/DS to continue the process.
* Page 256 SENDR
* From pages 256
EXIT:
• If there was an "OUT MOVE ERROR", set SQLCODE = -935
* From page 256
EXIT1:
• Return



Online Resource Manager - Route Message



Online Resource Manager Data Areas

The Resource Manager data areas are the RECP, REIB, RMAR, RMCV, RMGL, RMLA, RMLO, RMLT, RMMR, RMRE, RMSP, and RMWL.

See the "Data Areas" section of SQL/DS Logic, Volume 2, for data areas details.

VM RESOURCE MANAGER

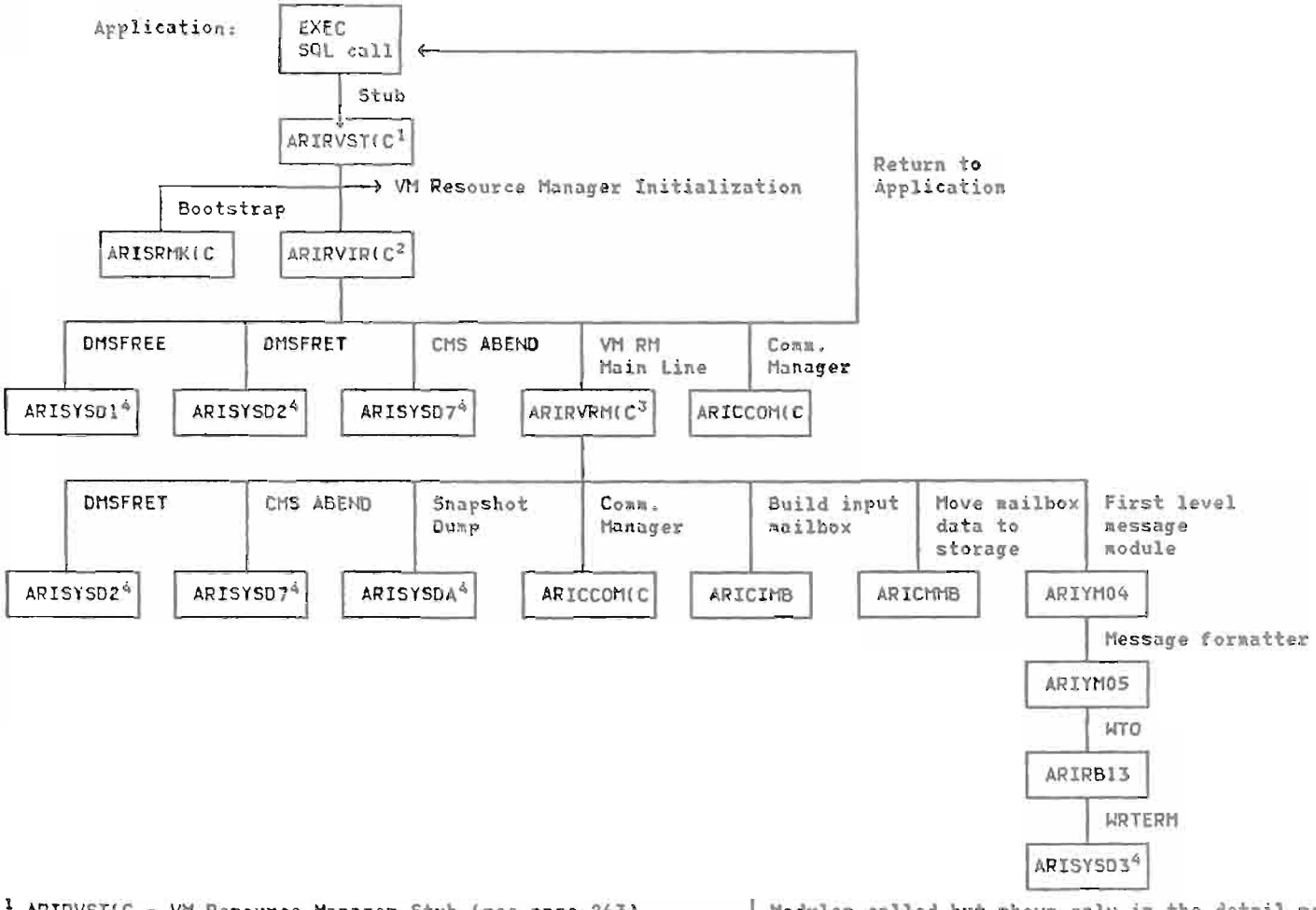
The VM Resource Manager phase is ARIRVMRM. It includes these modules:

ARIRVIR(C - Top Resource Manager module (does initialization)
ARIRVRM(C - Main line VM Resource Manager module
ARIRCL1(C - Cancel exit top module (invoked by CMS on behalf of an operator-initiated immediate command)
Cancel support exit entry point
ARIRCL2(C - Cancel exit second level module
Cancel support exit main line
ARIRTL1(C - Implicit COMMIT/ROLLBACK exit top module (invoked by CMS on normal or abnormal end of command)
Implicit COMMIT/ROLLBACK exit entry point
ARIRTL2(C - Implicit COMMIT/ROLLBACK second level module
Implicit COMMIT/ROLLBACK exit main line
ARIRINT(C - Resource Manager IUCV external interrupt handler
ARISYSDC - VM system-dependent routines
ARICCOM(C - IUCV Communications manager
ARICIMB - Build input mailbox (IUCV message)
ARICMMB - Move mailbox data to application
ARIYM04 - First level message module
ARIYM05 - Second level message module
ARIRB13 - Call to system-dependent write-to-operator function
ARICGSE - Get stack extension
ARICFSE - Free stack extension

The VM Resource Manager subcomponent includes these modules:

ARISRMK(C - VM Resource Manager bootstrap routine
ARIRVST(C - VM Resource Manager stub (a copy of this module is linked with each application)

VM Resource Manager Execution Overview



[1] ARIRVST(C - VM Resource Manager Stub (see page 263)
[2] ARIRVIR(C - Initialization routine (see page 264)
[3] ARIRVRM(C - Main-line routine (see page 265)
[4] ARISYSD(C - VM system-dependent routines (see descriptions in Volume 2, Section 3)

Modules called but not shown in this section are:
ARICFSE - Free stack extension
ARICGSE - Get stack extension

Modules called but shown only in the detail section are:
ARIRINT(C - IUCV external interrupt handler
ARIRCL1(C - Cancel exit top level module (see page 267)
ARIRCL2(C - Cancel exit second level module (see page 268)
ARIRTL1(C - Implicit COMMIT/ROLLBACK exit top level module (see page 269)
ARIRTL2(C - Implicit COMMIT/ROLLBACK exit second level module (see page 270)

ARIRINTC (Resource Manager External Interrupt Handler)

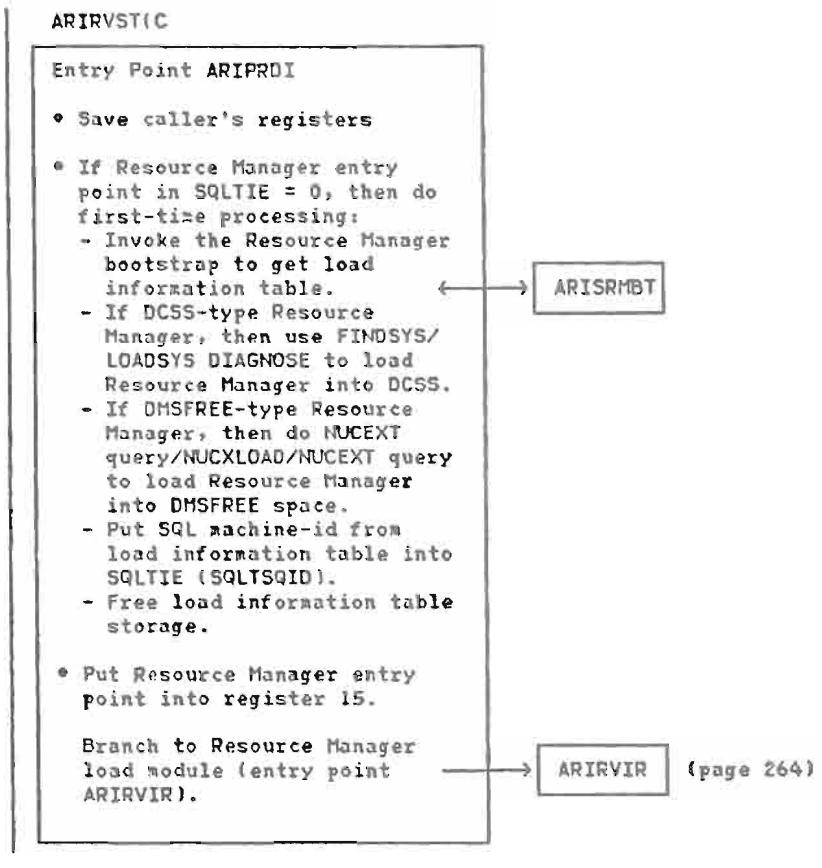
Entry Point - ARIRINT

- If interrupt type is 'completed connection', post the CONNECT ECB.

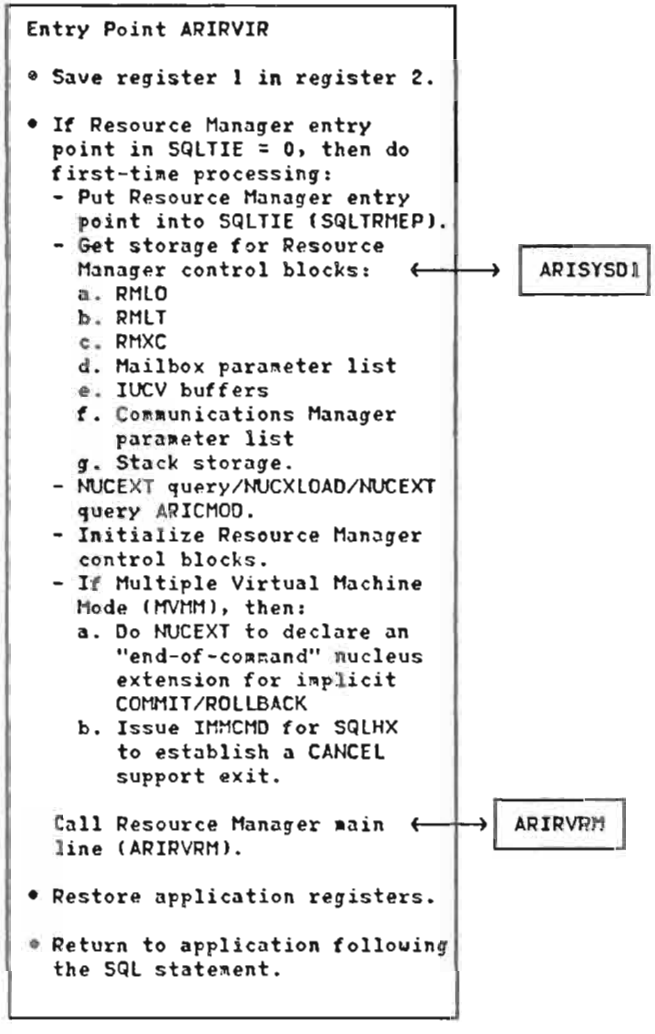
- If interrupt type is 'pending reply', then:
  - Post SEND ECB.
  - If audit field is zero, save length of reply.
  - If audit field is non-zero, translate and save audit information.
- If interrupt type is 'severed', then:
  - Set reason code and sever code to 'abnormal sever'.
  - If last function was CONNECT,
    a. If user data is 'SHUTDOWN' 'SVMMODE', or 'WRONGDB':
       1. Post CONNECT ECB
       2. Set reason code according to user data.
  - If last function was not CONNECT, post SEND ECB.
  - If reason code is still set to 'abnormal sever', set VMCLOCK=1.

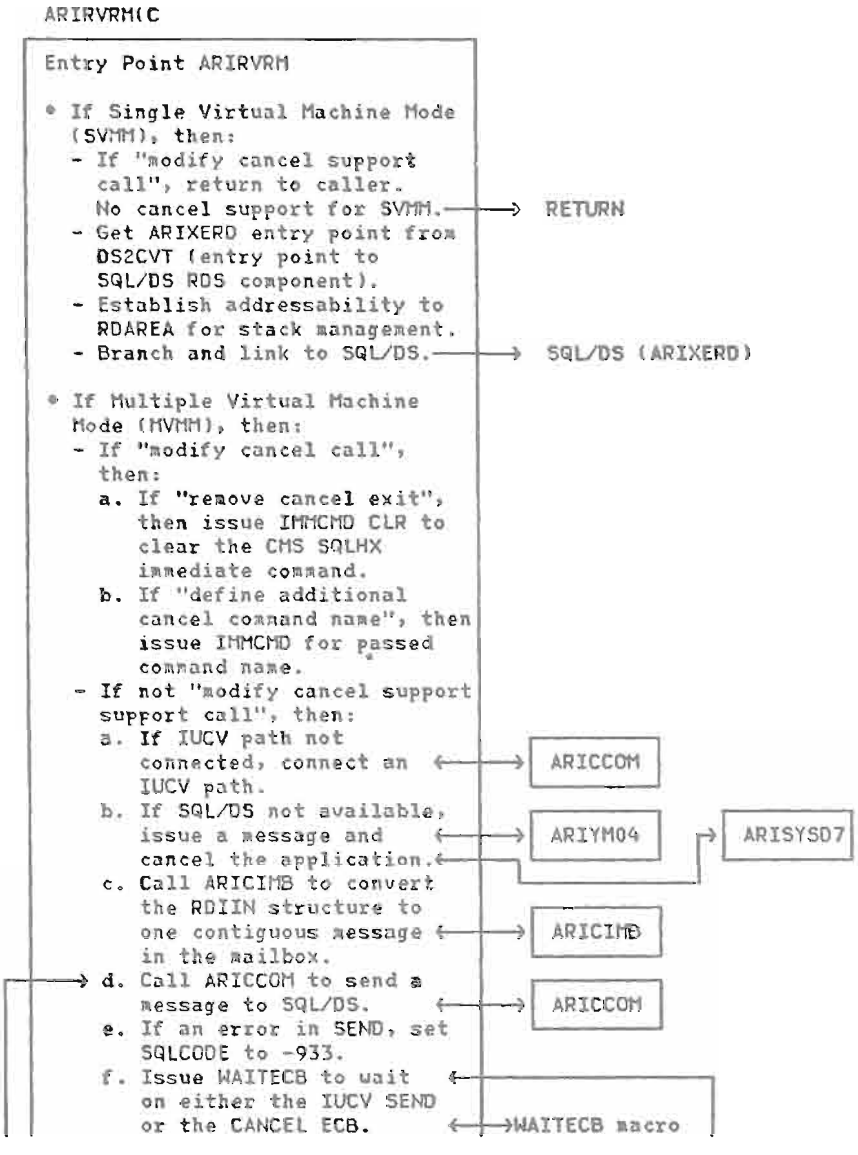
- If interrupt type is any other, then:
  - Post SEND ECB.
  - Set reason code 'invalid interrupt'.

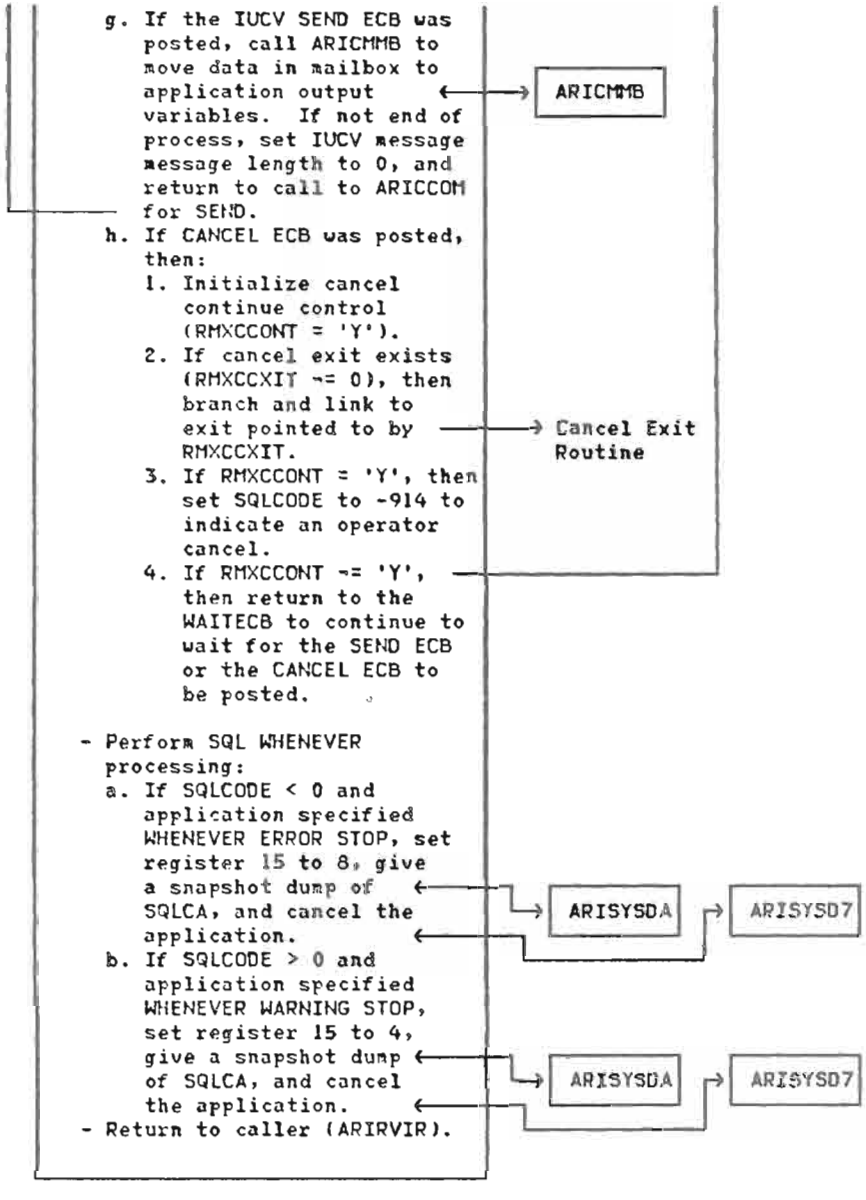
- Return to caller with R15 = 0.


ARIRVST(C
Entry Point ARIPRDI
• Save caller's registers
• If Resource Manager entry point in SQLTIE = 0, then do first-time processing:
- Invoke the Resource Manager bootstrap to get load information table.
- If DCSS-type Resource Manager, then use FINDSYS/ LOADSYS DIAGNOSE to load Resource Manager into DCSS.
- If DMSFREE-type Resource Manager, then do NUCEXT query/NUCXLOAD/NUCEXT query to load Resource Manager into DMSFREE space.
- Put SQL machine-id from load information table into SQLTIE (SQLTSQID).
- Free load information table storage.
• Put Resource Manager entry point into register 15.
Branch to Resource Manager load module (entry point ARIRVIR).
ARISRMBT
ARIRVIR
(page 264)

ARIRVIR(C




ARIRVRM(C
Entry Point ARIRVRM
* If Single Virtual Machine Mode (SVMM), then:
- If "modify cancel support call", return to caller. No cancel support for SVMM.
RETURN
- Get ARIXERD entry point from DS2CVT (entry point to SQL/DS RDS component).
- Establish addressability to RDAREA for stack management.
- Branch and link to SQL/DS.
SQL/DS (ARIXERD)
* If Multiple Virtual Machine Mode (MVMM), then:
- If "modify cancel call", then:
a. If "remove cancel exit", then issue IMMCMD CLR to clear the CMS SQLHX immediate command.
b. If "define additional cancel command name", then issue IMMCMD for passed command name.
- If not "modify cancel support call", then:
a. If IUCV path not connected, connect an IUCV path.
ARICCOM
b. If SQL/DS not available, issue a message and cancel the application.
ARIYM04
ARISYSD7
c. Call ARICIMB to convert the RDIIN structure to one contiguous message in the mailbox.
ARICIMB
d. Call ARICCOM to send a message to SQL/DS.
ARICCOM
e. If an error in SEND, set SQLCODE to -933.
f. Issue WAITECB to wait on either the IUCV SEND or the CANCEL ECB.
WAITECB macro



ARIRCL1(C

Entry Point ARIRCL1

* Using CMS IMMBLOK, get address of ARICMOD; using ARICMOD, get address of RMLO.

* If RMLO eyecatcher is initialized, then:
  - Using Test and Set logic, check cancel exit control byte (RMLOINCX). If byte indicates reentry to cancel exit, return immediately to CMS.
  - Save caller's register 13 in RMLOCA13.
  - Set register 13 to point to cancel exit save area (RMLOCNSA).
  - Call ARIRCL2 to complete the cancel exit processing. ←→ ARIRCL2 (page 268)
  - Reset register 13 to caller's save area.

* Return to CMS.

ARIRCL2(C

Entry Point ARIRCL2

- Using RMLO, get the address of RMXC (RMLORMXP).
- If RMXCXC indicates waiting on an SQL request, post the CANCEL ECB to cause this SQL request to be canceled by the Resource Manager cancel support.
- If RMXCXC indicates waiting on COMMIT or ROLLBACK, set the RMLOCNSY flag on to let the Resource Manager support know a cancel request was attempted on a COMMIT or ROLLBACK.
- If not waiting on an SQL request at all, set RMXCXC to RMXCCAN (-2) letting an application know that a CANCEL has been requested and post the CANCEL ECB. This will cause the next SQL request to be canceled.
- Return to ARIRCL1.

ARIRTL1(C

Entry Point ARIRTL1

- Using CMS SCBLOCK, get address of ARICMOD; using ARICMOD, get address of RMLO.

- If RMLO eyecatcher is initialized, then:
  - Using Test and Set logic, check termination exit control byte (RMLOINTM). If byte indicates reentry to exit, immediately return to CMS.
  - Save caller's register 13 in RMLOCA13.
  - Set register 13 to point to Resource Manager save area (RMLORMSA).
  - Call ARIRTL2 to complete the cancel exit processing. ←→ ARIRTL2 (page 270)
  - Reset register 13 to caller's save area.
  - Clear storage acquired by the Resource Manager, getting length from RMLO (RMLOSTRL).
  - Free storage acquired by the Resource Manager.

- Return to CMS.

ARIRTL2(C

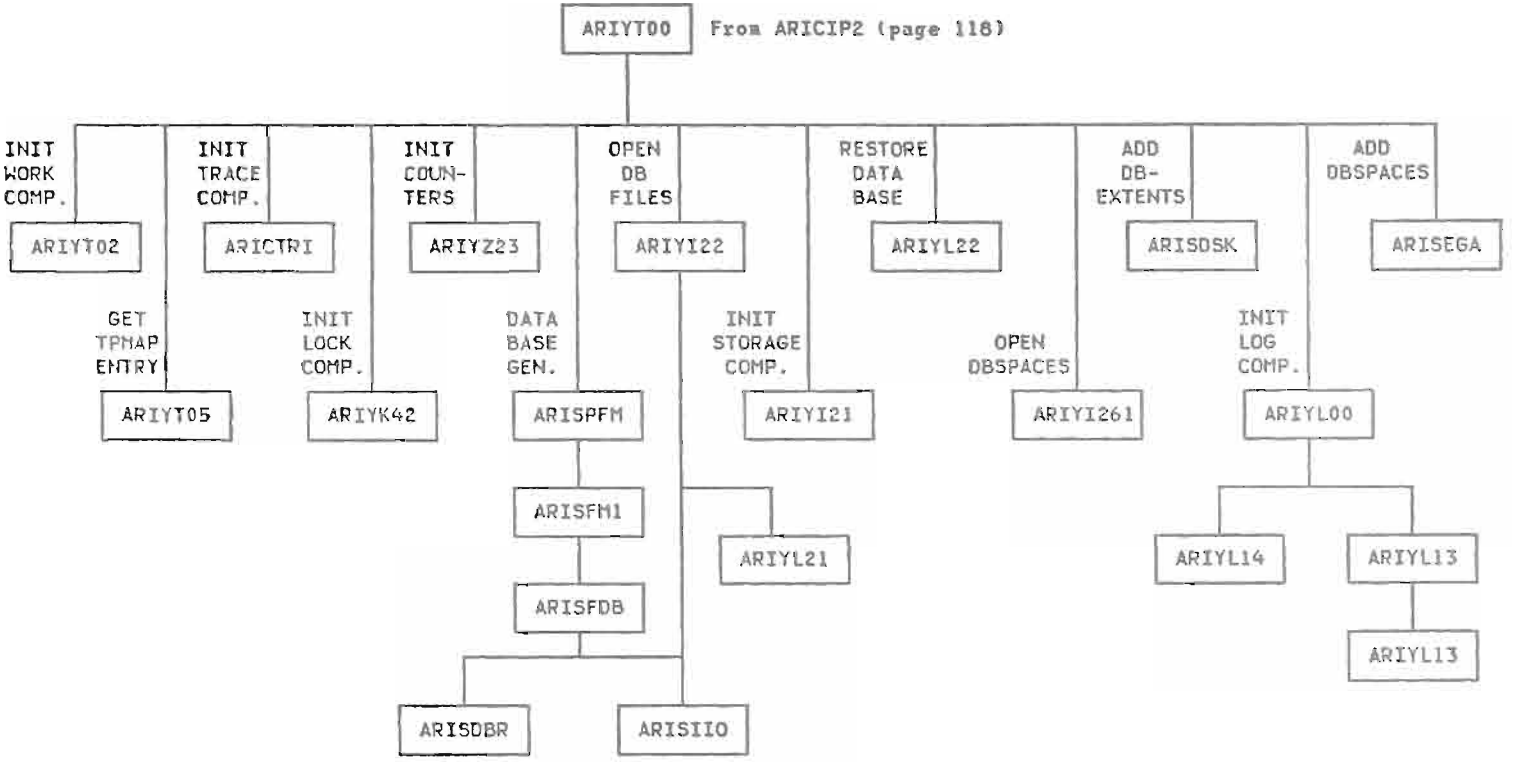
Entry Point ARIRTL2

- Using register 1, establish addressability to the CMS parameter list.
- If the parameter list indicates a normal end of command, put 'COMMIT ' in the SEVER parameter list user data fields to indicate a COMMIT SEVER to the SQL/DS machine.
- If the parameter list indicates an abnormal end of command, put binary zeros in the SEVER parameter list user data field to indicate a ROLLBACK SEVER to the SQL/DS machine.
- Call ARICCOM to request a SEVER. ⟷ ARICCOM
- Call ARICCOM to request a LOGOFF. ⟷ ARICCOM
- Issue a NUCXDROP for the end of command NUCEXT ARIRTL1.
- Issue NUCEXT query to determine if the Resource Manager was loaded as a nucleus extension.
- If Resource Manager is a nucleus extension, issue an ATTN to schedule a NUCXDROP of the Resource Manager on return to CMS.
- Return to ARIRTL1.

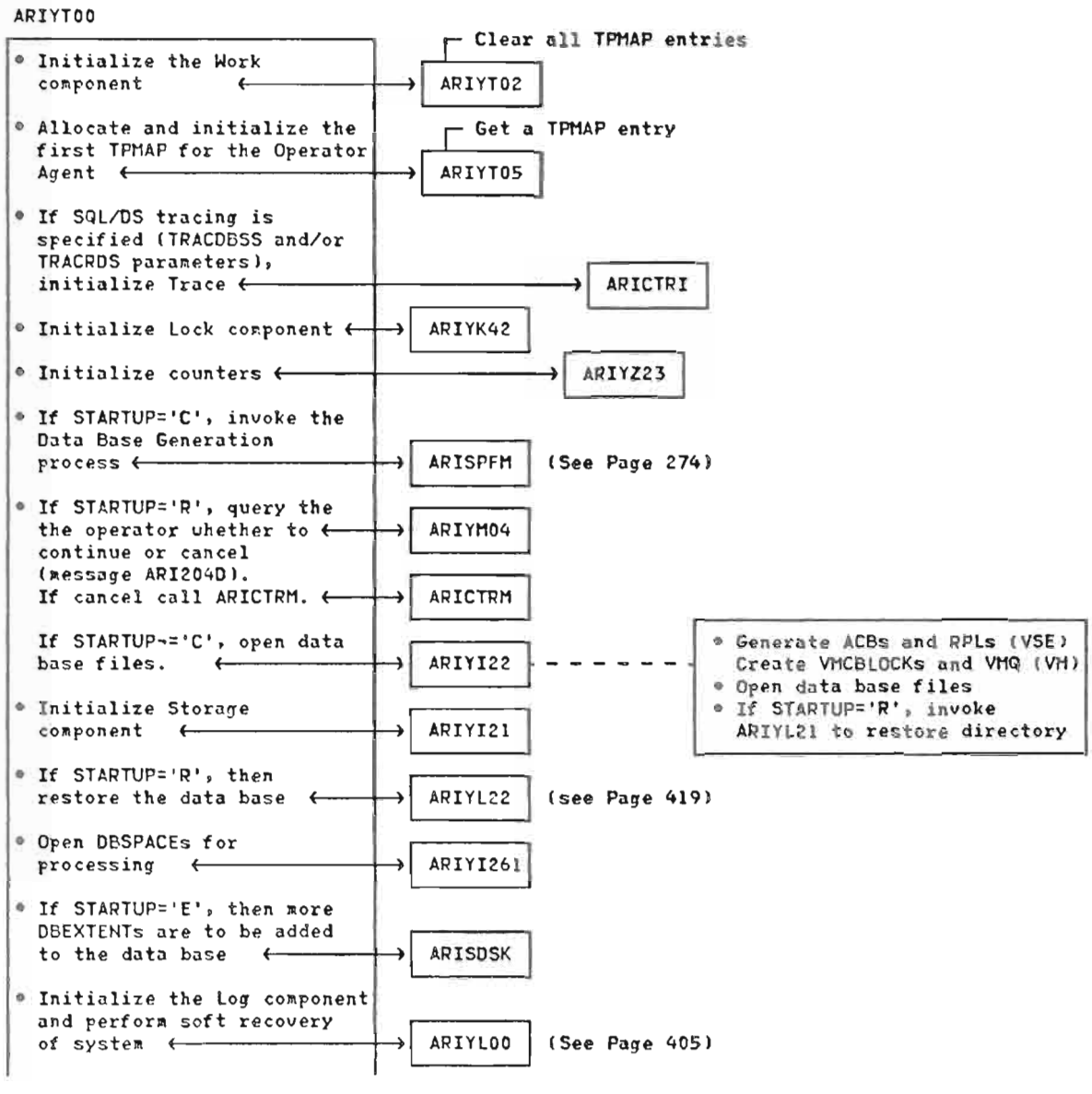
DBSS (DATA BASE STORAGE SYSTEM)

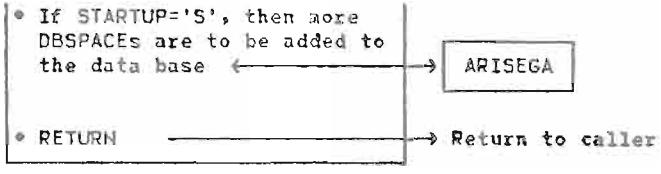
DBSS INITIALIZATION

DBSS INITIALIZATION - GENERAL FLOW

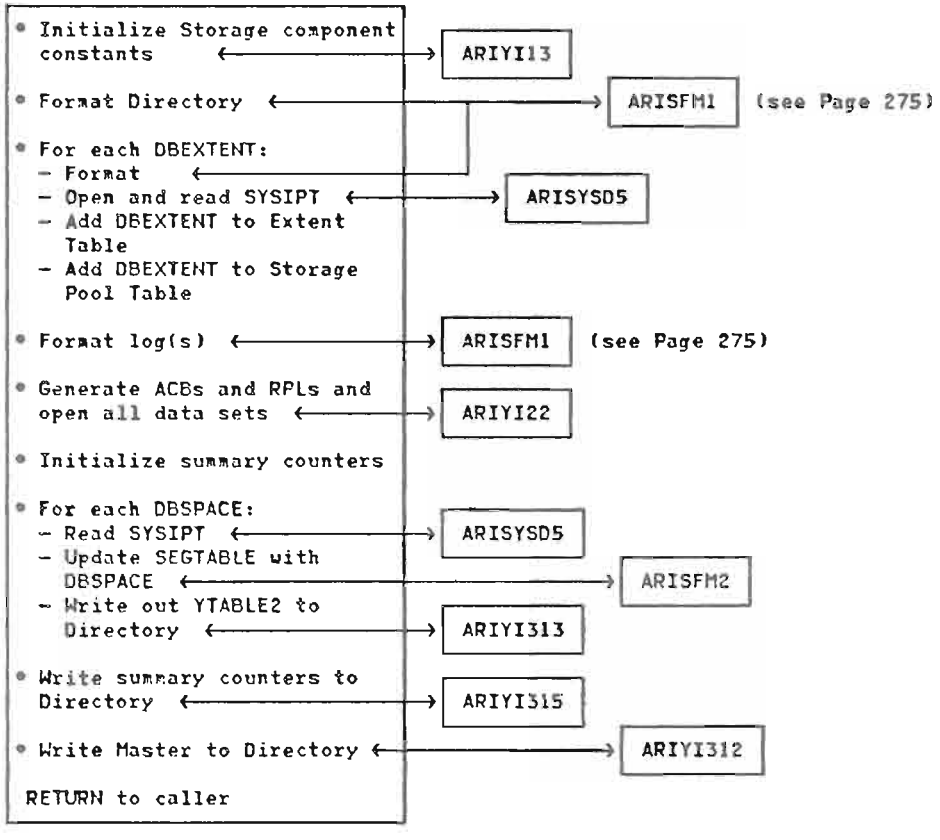


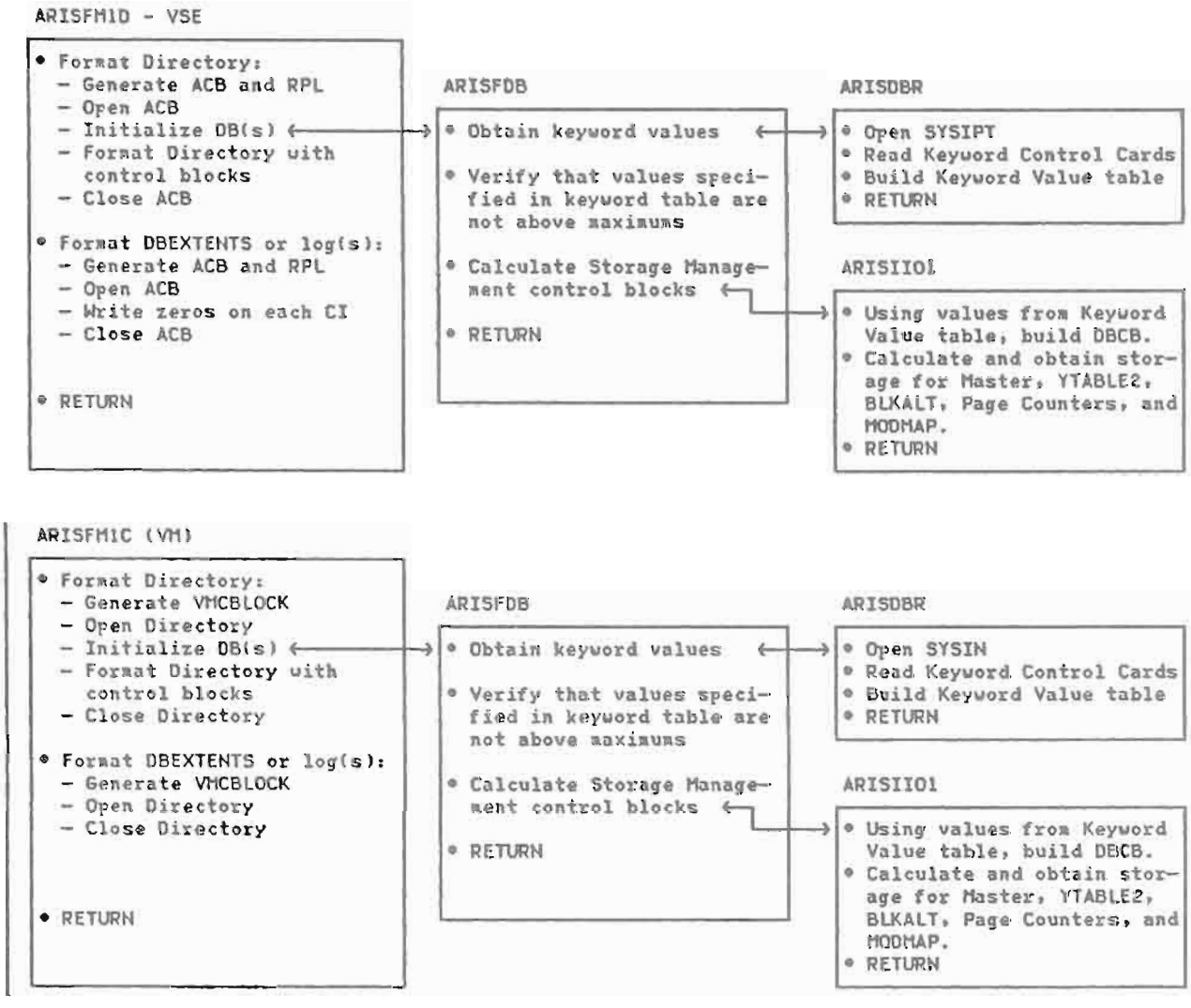
DBSS INITIALIZATION - DETAIL FLOW





(From ARIYT00, Page 272)
ARISPFM (Data Base Generation)




ARISFMID - VSE
• Format Directory:
– Generate ACB and RPL
– Open ACB
– Initialize DB(s)
– Format Directory with control blocks
– Close ACB
• Format DBEXTENTS or log(s):
– Generate ACB and RPL
– Open ACB
– Write zeros on each CI
– Close ACB
• RETURN
ARISFDB
• Obtain keyword values
• Verify that values specified in keyword table are not above maximums
• Calculate Storage Management control blocks
• RETURN
ARISDBR
• Open SYSIPT
• Read Keyword Control Cards
• Build Keyword Value table
• RETURN
ARISIIOI
• Using values from Keyword Value table, build DBCB.
• Calculate and obtain storage for Master, YTABLE2, BLKALT, Page Counters, and MODMAP.
• RETURN
ARISFMIC (VM)
• Format Directory:
– Generate VMCBLOCK
– Open Directory
– Initialize DB(s)
– Format Directory with control blocks
– Close Directory
• Format DBEXTENTS or log(s):
– Generate VMCBLOCK
– Open Directory
– Close Directory
• RETURN
ARISFDB
• Obtain keyword values
• Verify that values specified in keyword table are not above maximums
• Calculate Storage Management control blocks
• RETURN
ARISDBR
• Open SYSIN
• Read Keyword Control Cards
• Build Keyword Value table
• RETURN
ARISIIOI
• Using values from Keyword Value table, build DBCB.
• Calculate and obtain storage for Master, YTABLE2, BLKALT, Page Counters, and MODMAP.
• RETURN

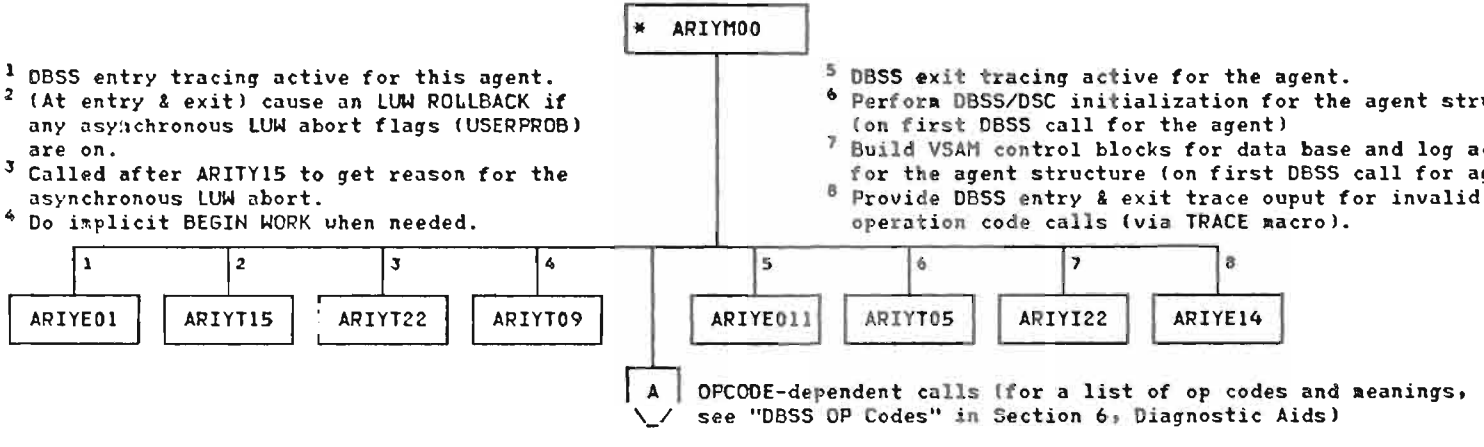
DATA BASE STORAGE SYSTEM LINKAGE (DBSI)

Module ARIYM00, the Data Base Storage System Linkage (DBSI), is the linkage between RDS and DBSS. It is the DBSS entry point for executing DBSS operation codes. It determines what module entry point should be called to execute the operation code and calls that entry point, passing the pointer to the operation parameter structure (PARAMS). On entry and on exit, tests are made for DBSS entry and DBSS exit tracing, and if that trace function is active, ARIYM00 makes a call to perform the trace function. Also, on entry and exit, ARIYM00 tests to determine if the current DBSS logical unit of work has been flagged for asynchronous termination. If so, ARIYM00 calls DBSS routines to rollback the logical unit of work and to obtain the cause (for return code) and bypasses operation code execution.

For each operation code call, ARIYM00 tests to determine if a logical unit of work needs to be created for the operation (and possible subsequent operations). If so, it calls the DBSS BEGIN WORK function (implicit BEGIN work). ARIYM00 ensures that DBSS has been initialized for the agent before any operations other than BEGIN DBSS are executed. If called with an invalid operation code, this is detected and an error return code is set (and appropriate trace output if DBSS entry and/or DBSS exit tracing is active) and returned to the caller. For all calls, a DBSS return code is returned in the RTNCODE parameter and in register 15.

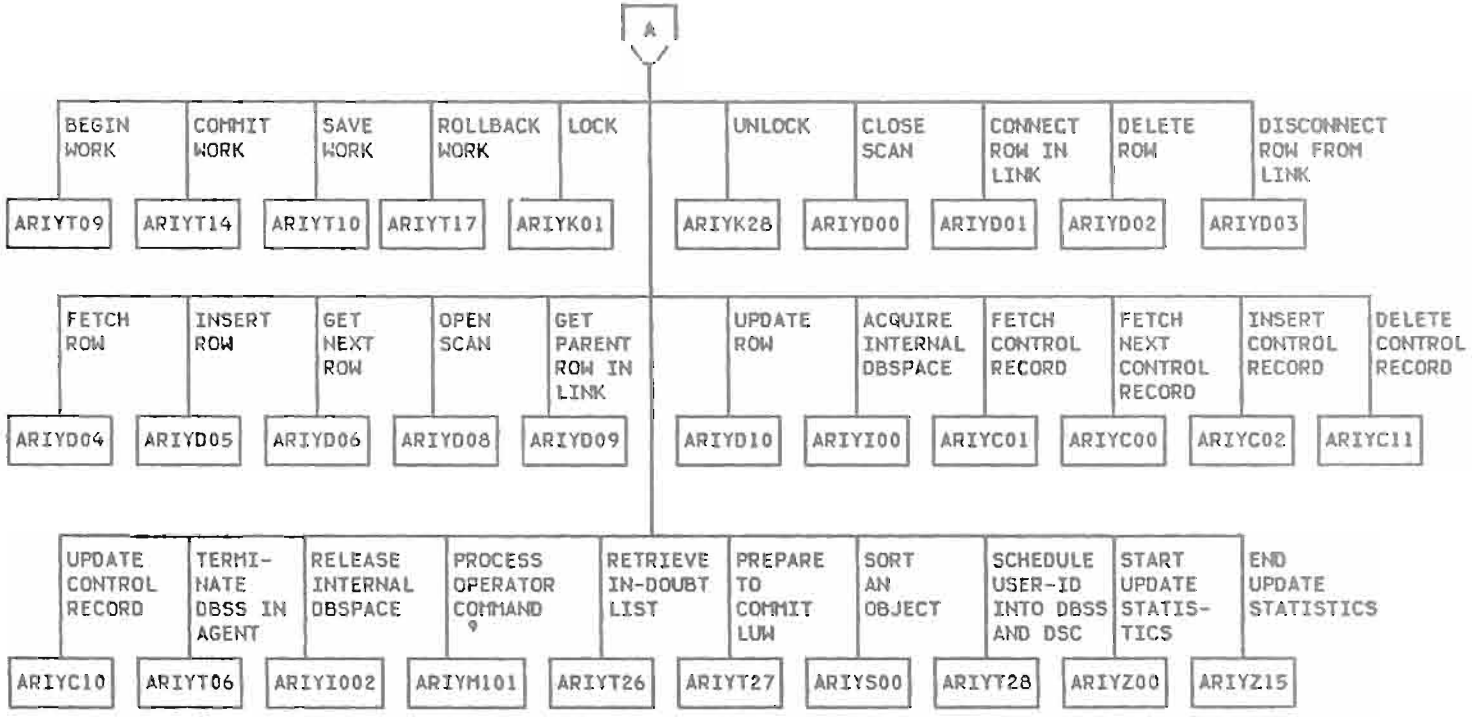
The following diagram shows a general view of the calls made via ARIYM00 and for what reasons ARIYM00 was called.

* From: ARICCRA - BEGIN DBSS calls during create agent function of SQL/DS initialization.
ARIGCAT - Catalog generation function of data base generation (to create/initialize SQL/DS system catalogs).
ARISEGB - Add DBSPACE (does catalog updates).
ARIXEDB - RDS linkage to DBSS (all DBSS OPCODE calls except END DBSS).
ARIXERD - Issues END DBSS calls (for agent cleanup).



Subcomponents and page reference:

ARIYI.. modules belong to the Storage (I/O) subcomponent.
See page 401.
ARIYT.. modules belong to the Work subcomponent. See page 386.
ARIYE01 and ARIYE14 modules are described under DSC Trace Services. See page 188.



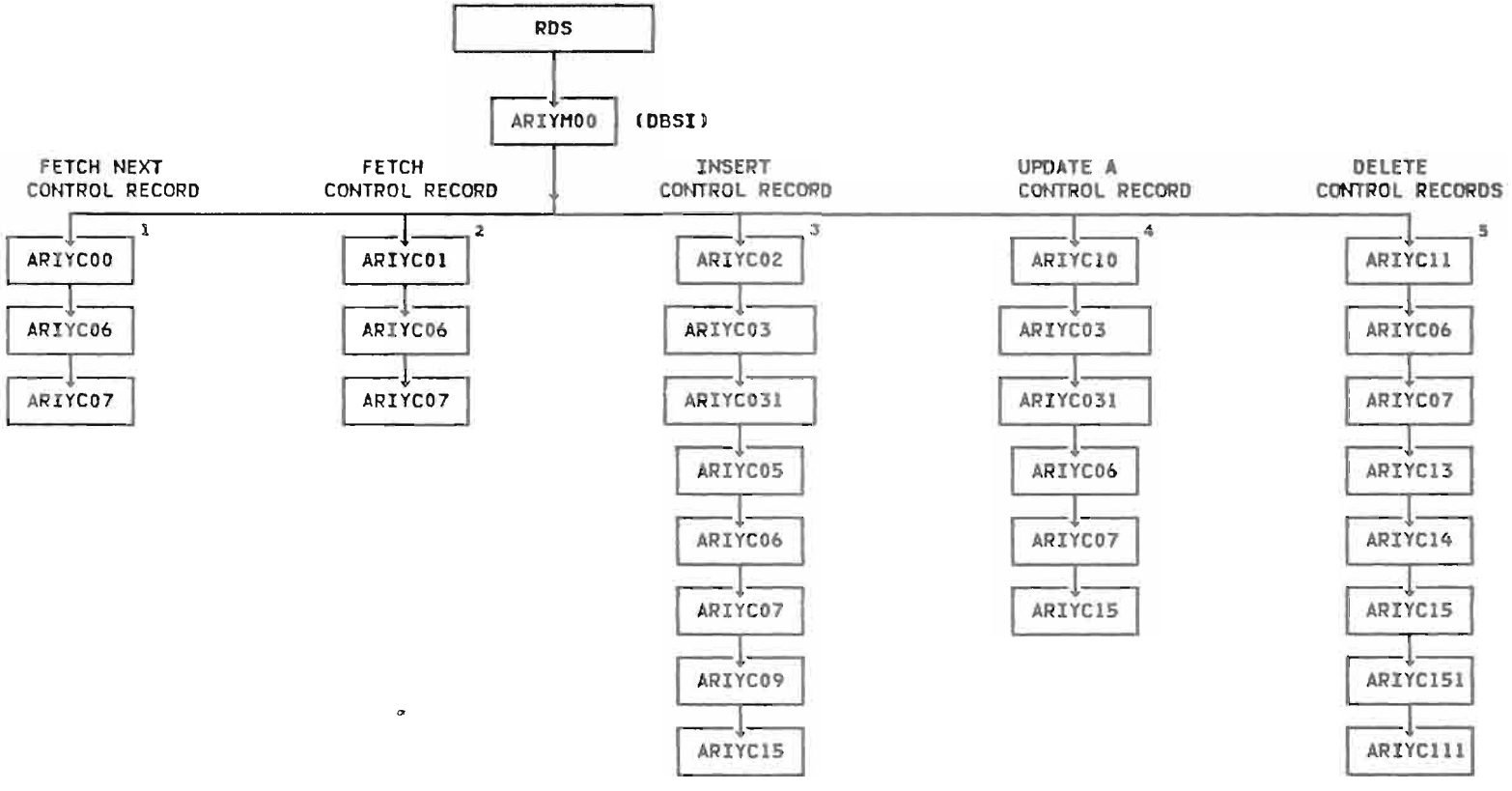
9 This module is described under DSC Operator Services, page 144.

Subcomponents and page reference:

ARIYC.. modules belong to the Data Control subcomponent. See page 278.
ARIYD.. modules belong to the Data Manipulation subcomponent. See page 306.
ARIYI.. modules belong to the Storage subcomponent. See page 401.
ARIYK.. modules belong to the Lock subcomponent. See page 420.
ARIYS.. modules belong to the Sort subcomponent. See page 344.
ARIYT.. modules belong to the Work subcomponent. See page 386.
ARIYZ.. modules belong to the Update Statistics/COUNTER command subcomponent. See page 385.

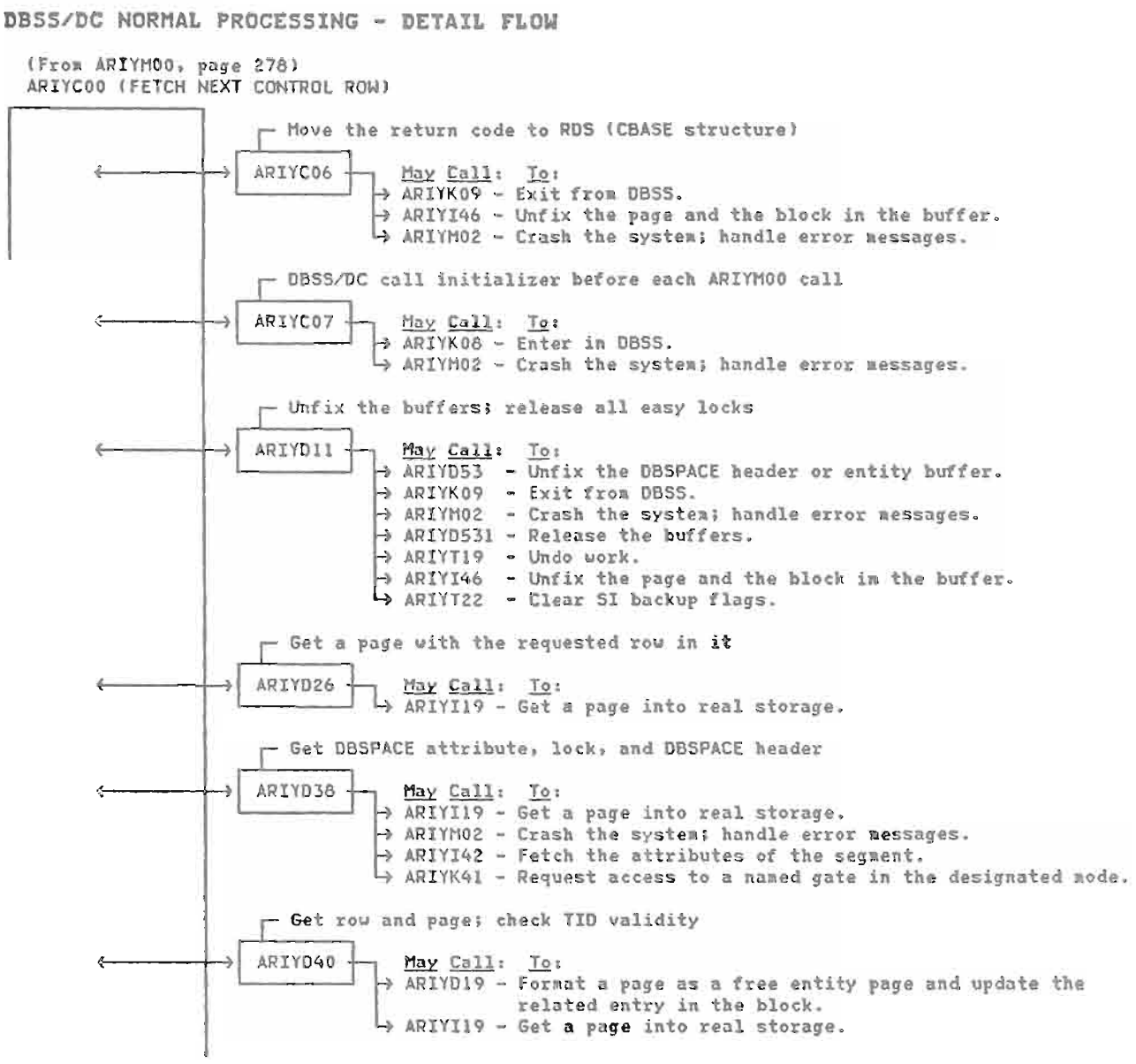
DBSS/DC NORMAL PROCESSING - GENERAL FLOW (MAJOR DC MODULES)

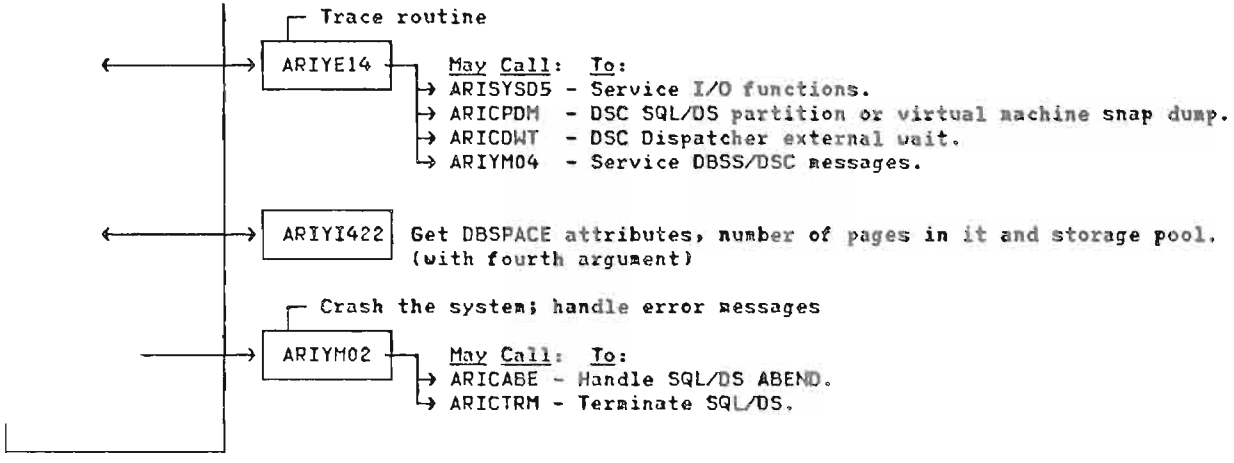
In this diagram, only the DBSS/DC major modules are shown. See the following detail diagrams for other modules being called.



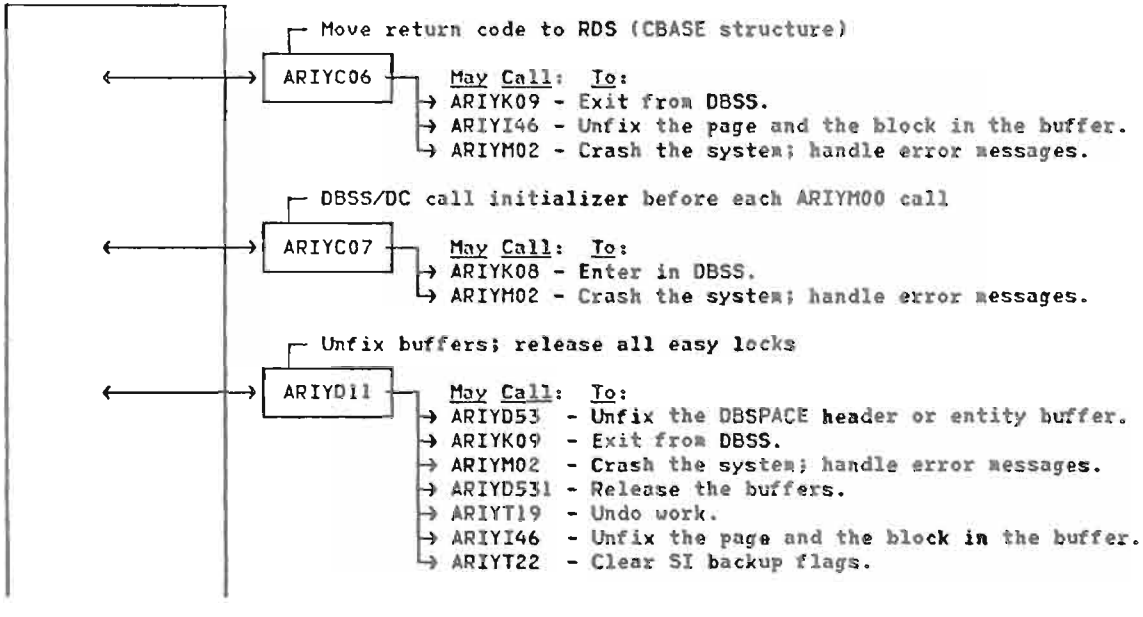
[1] See page 279 for detail flow of FETCH NEXT CONTROL ROW.
[2] See page 280 for detail flow of FETCH CONTROL ROW.
[3] See page 282 for detail flow of INSERT CONTROL ROW.
[4] See page 286 for detail flow of UPDATE A CONTROL ROW.
[5] See page 288 for detail flow of DELETE CONTROL ROWS.

DBSS/DC NORMAL PROCESSING - DETAIL FLOW

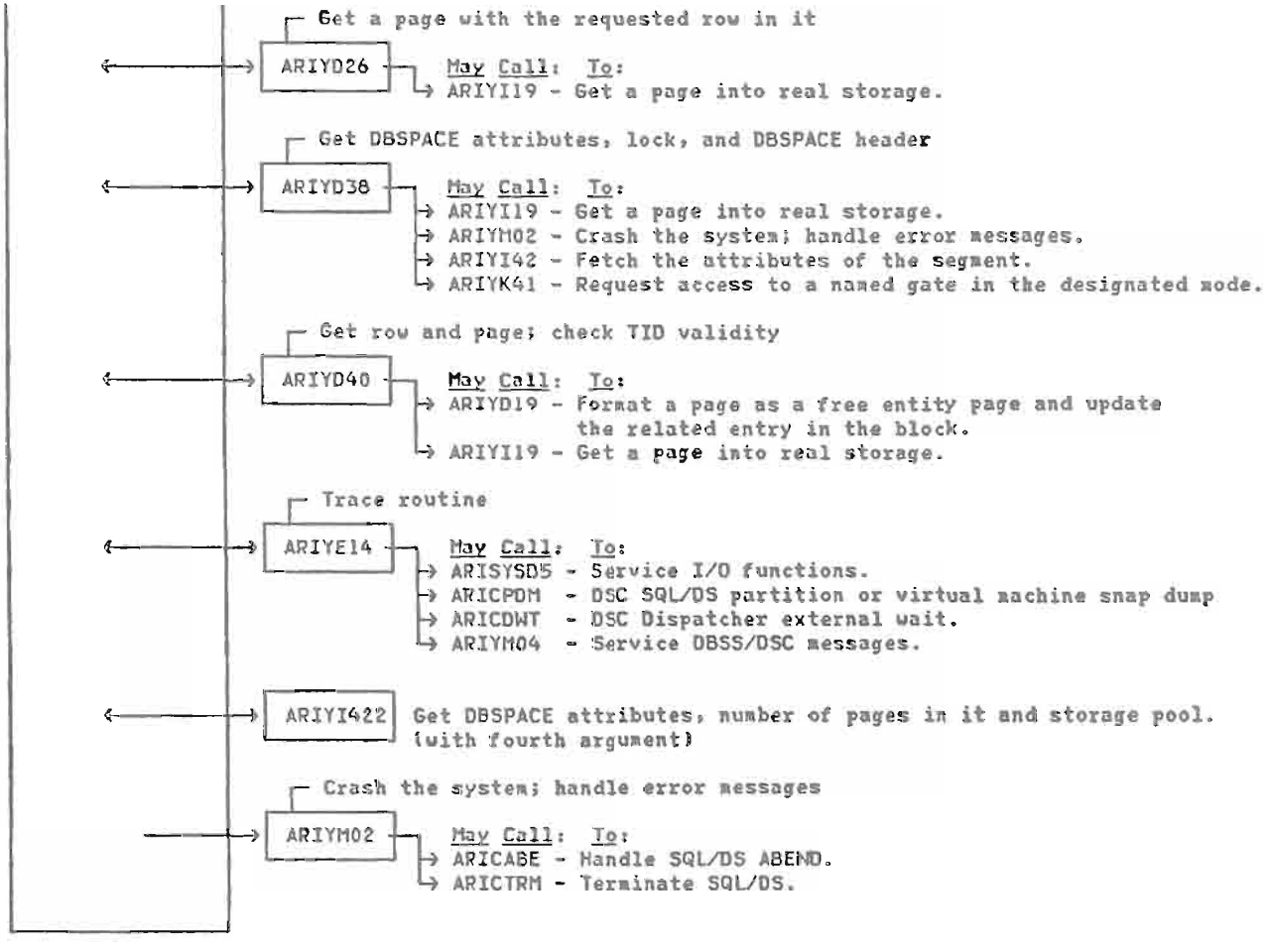




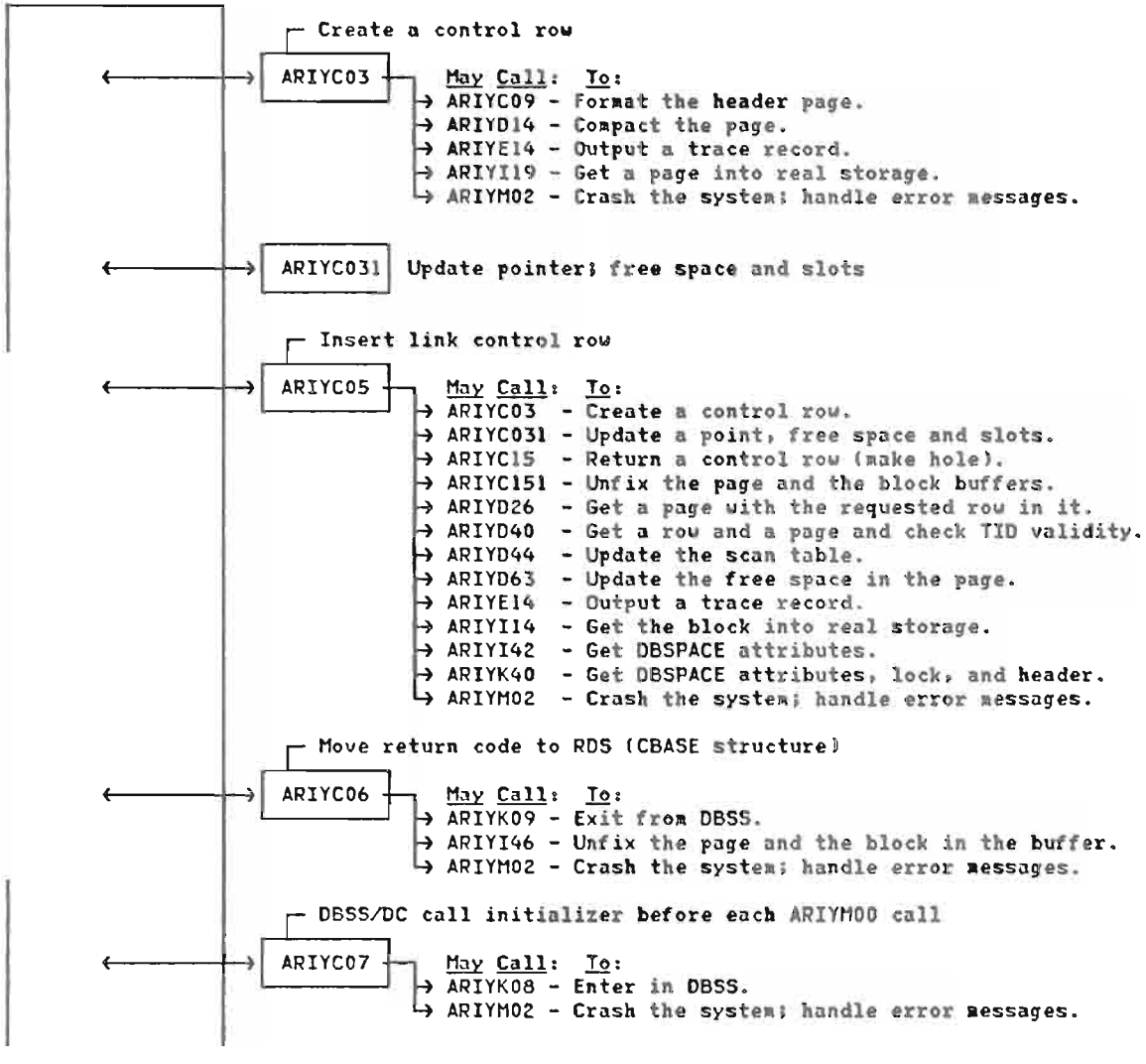
(From ARIYM00, page 278)
ARIYC01 (FETCH CONTROL ROW)

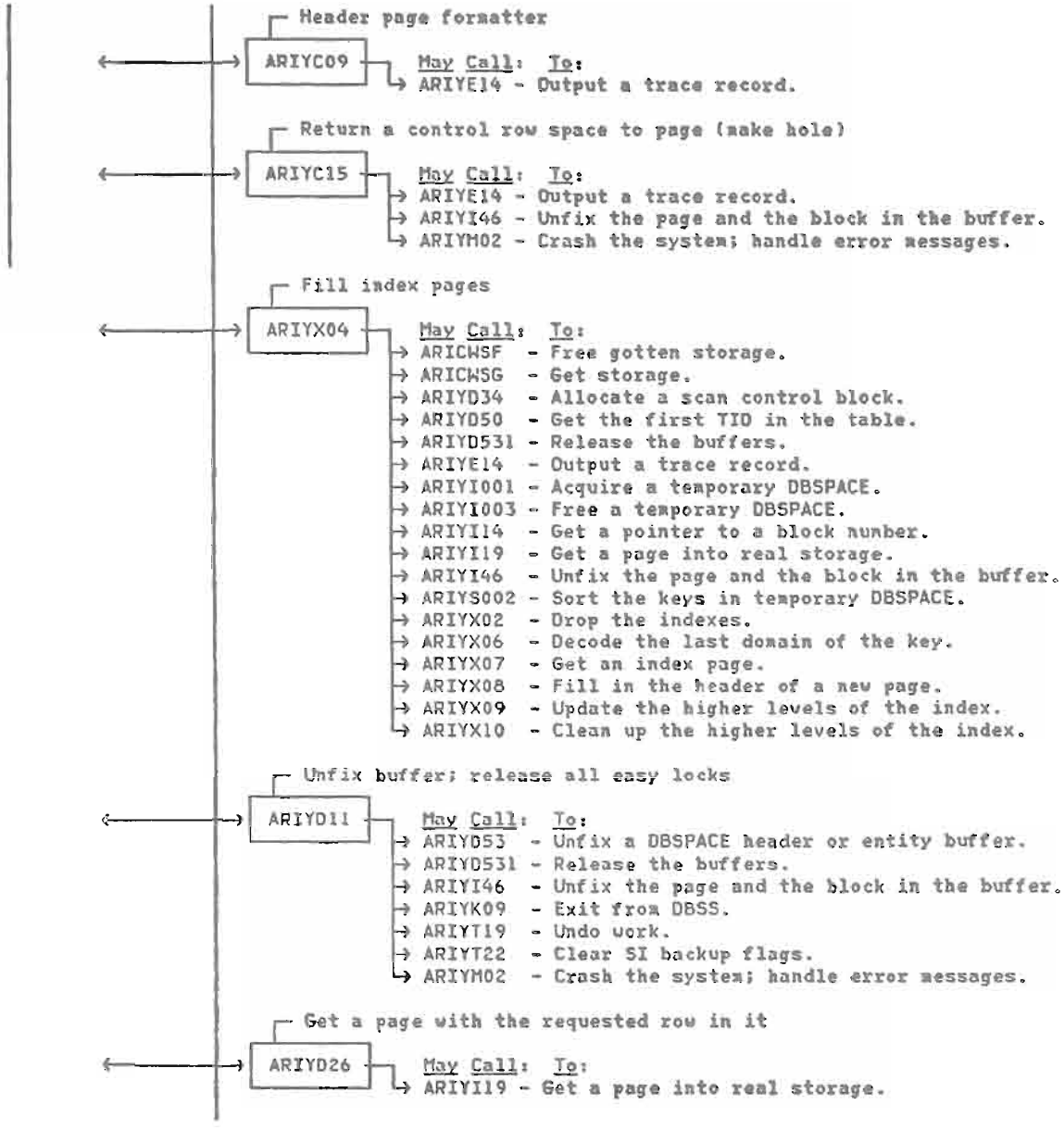


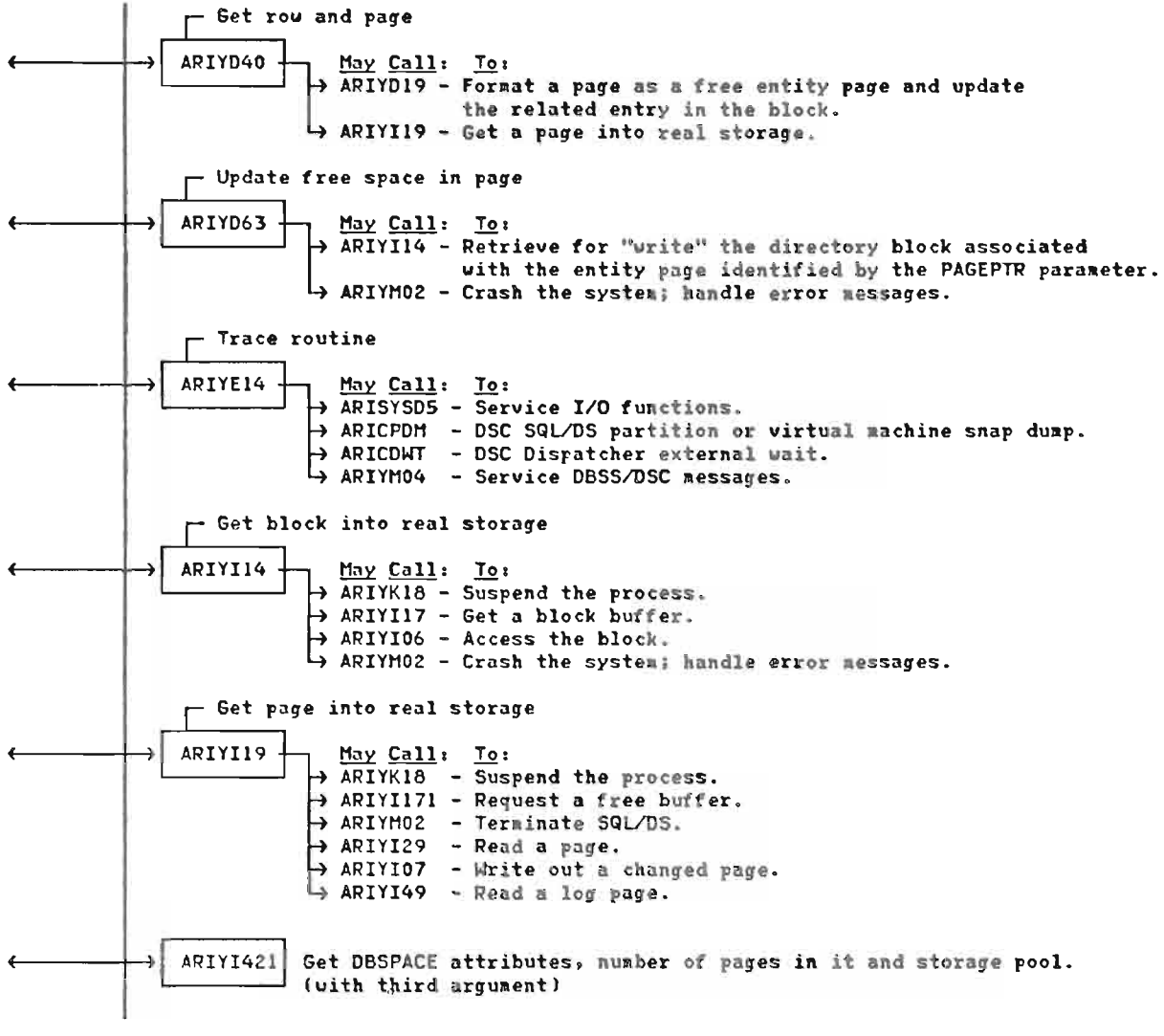
DBSS/DC (continued)

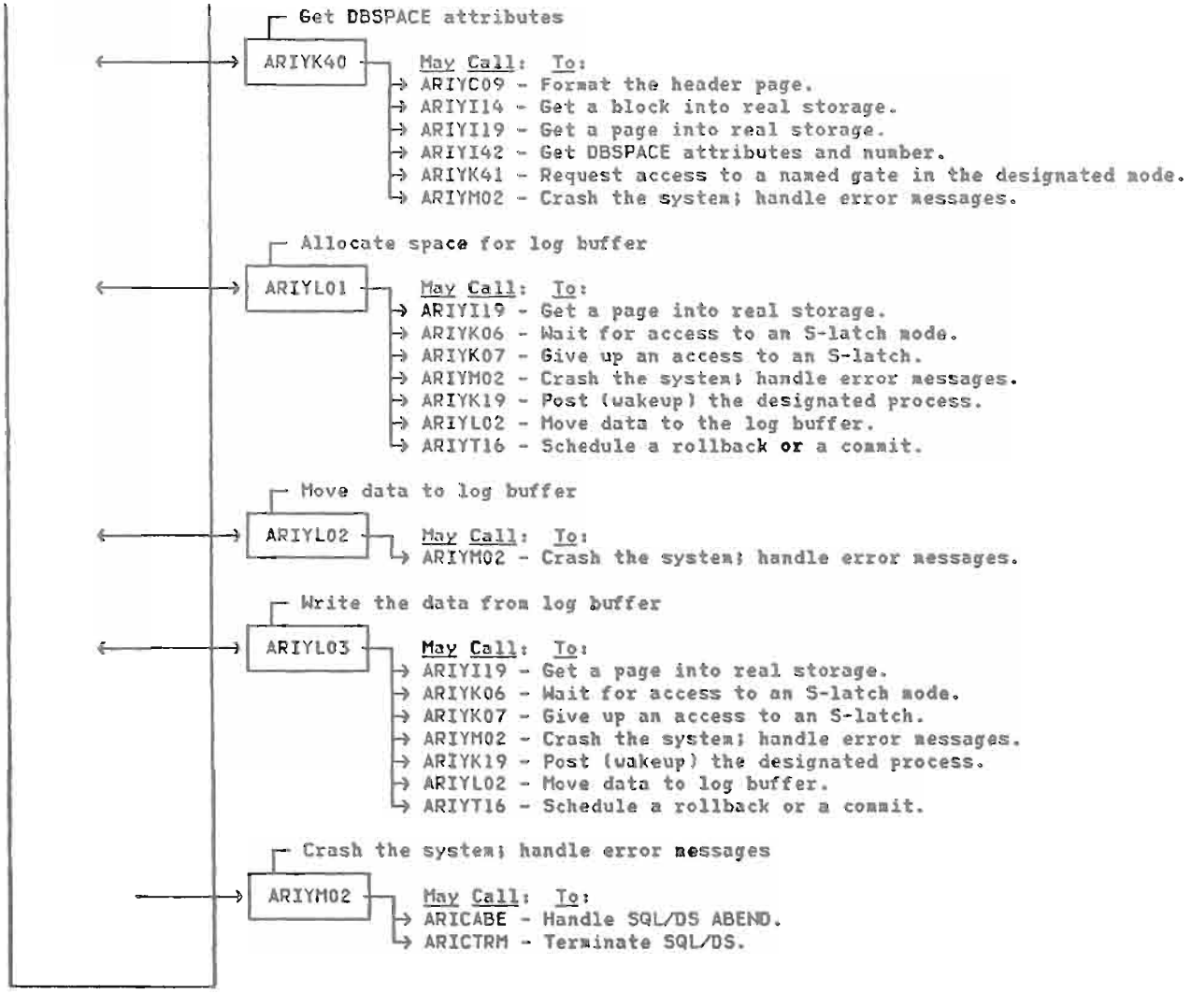


(From ARIYM00, page 278)
ARIYC02 (INSERT CONTROL ROW)

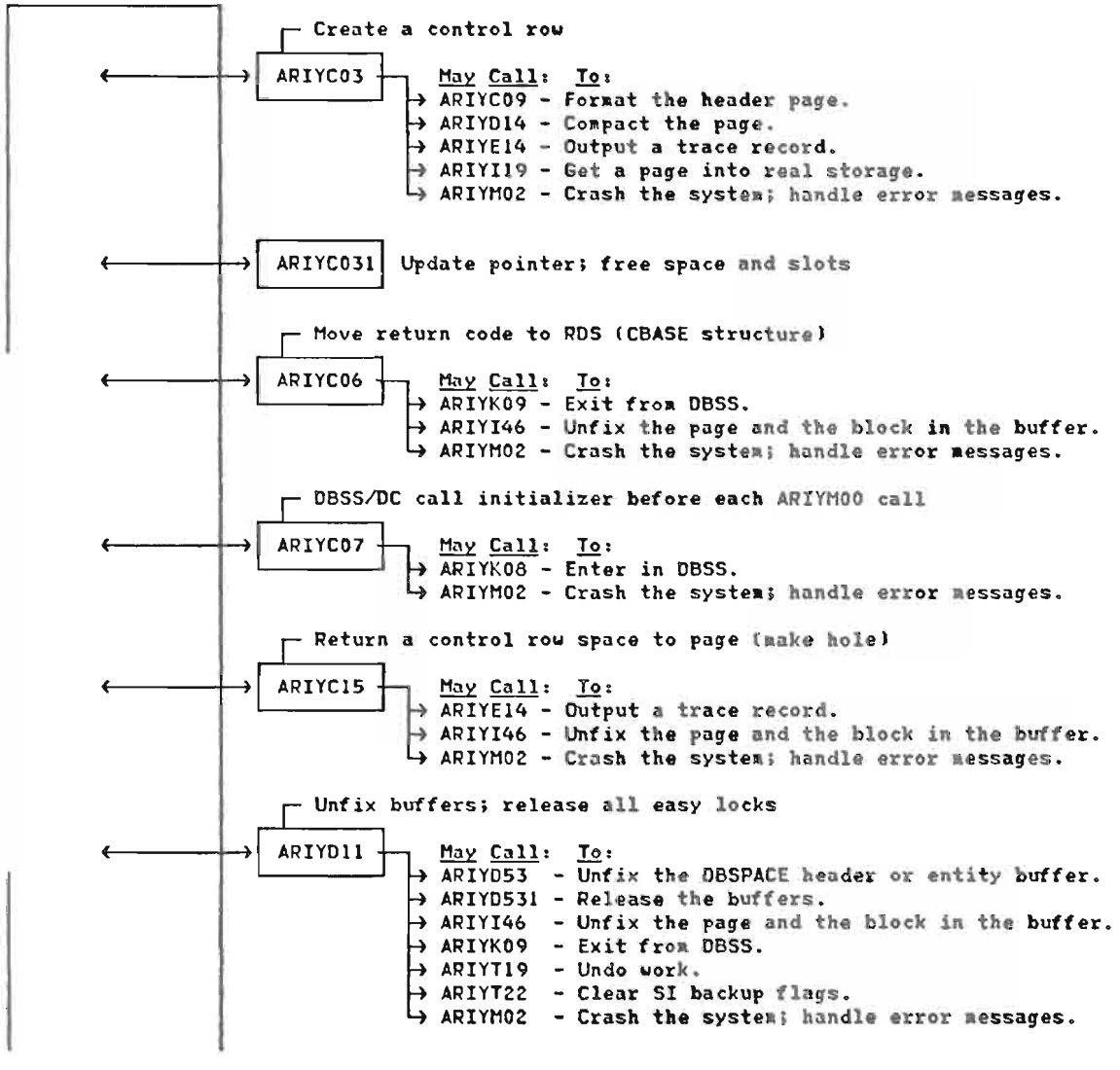



Header page formatter
ARIYC09
May Call: To:
ARIYE14 - Output a trace record.
Return a control row space to page (make hole)
ARIYC15
May Call: To:
ARIYE14 - Output a trace record.
ARIYI46 - Unfix the page and the block in the buffer.
ARIYM02 - Crash the system; handle error messages.
Fill index pages
ARIYX04
May Call: To:
ARICWSF - Free gotten storage.
ARICWSG - Get storage.
ARIYD34 - Allocate a scan control block.
ARIYD50 - Get the first TID in the table.
ARIYD531 - Release the buffers.
ARIYE14 - Output a trace record.
ARIYI001 - Acquire a temporary DBSPACE.
ARIYI003 - Free a temporary DBSPACE.
ARIYI14 - Get a pointer to a block number.
ARIYI19 - Get a page into real storage.
ARIYI46 - Unfix the page and the block in the buffer.
ARIYS002 - Sort the keys in temporary DBSPACE.
ARIYX02 - Drop the indexes.
ARIYX06 - Decode the last domain of the key.
ARIYX07 - Get an index page.
ARIYX08 - Fill in the header of a new page.
ARIYX09 - Update the higher levels of the index.
ARIYX10 - Clean up the higher levels of the index.
Unfix buffer; release all easy locks
ARIYD11
May Call: To:
ARIYD53 - Unfix a DBSPACE header or entity buffer.
ARIYD531 - Release the buffers.
ARIYI46 - Unfix the page and the block in the buffer.
ARIYK09 - Exit from DBSS.
ARIYT19 - Undo work.
ARIYT22 - Clear SI backup flags.
ARIYM02 - Crash the system; handle error messages.
Get a page with the requested row in it
ARIYD26
May Call: To:
ARIYI19 - Get a page into real storage.

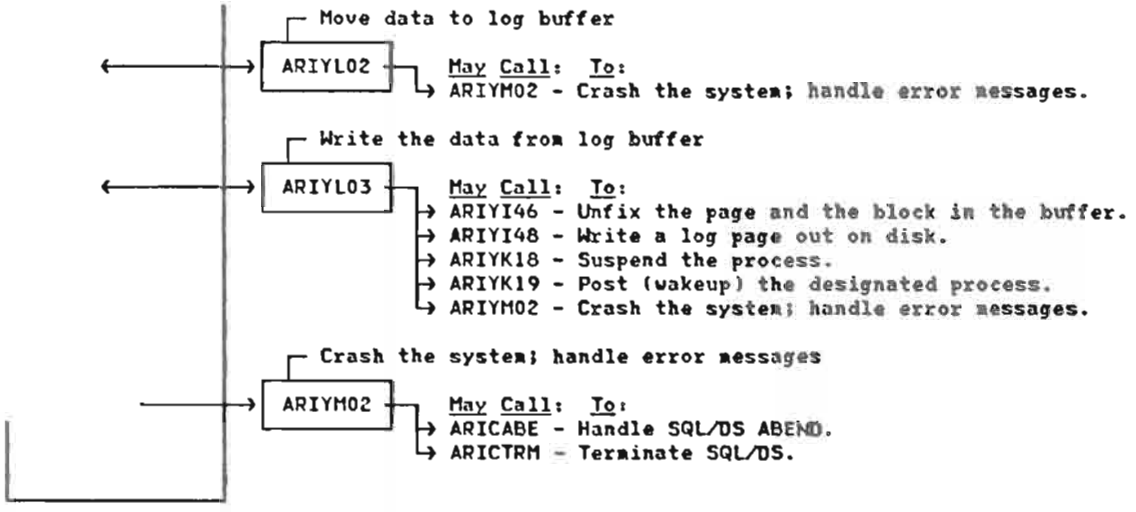



Get DBSPACE attributes
ARIYK40
May Call: To:
ARIYC09 - Format the header page.
ARIYI14 - Get a block into real storage.
ARIYI19 - Get a page into real storage.
ARIYI42 - Get DBSPACE attributes and number.
ARIYK41 - Request access to a named gate in the designated mode.
ARIYM02 - Crash the system; handle error messages.
Allocate space for log buffer
ARIYL01
May Call: To:
ARIYI19 - Get a page into real storage.
ARIYK06 - Wait for access to an S-latch mode.
ARIYK07 - Give up an access to an S-latch.
ARIYM02 - Crash the system; handle error messages.
ARIYK19 - Post (wakeup) the designated process.
ARIYL02 - Move data to the log buffer.
ARIYT16 - Schedule a rollback or a commit.
Move data to log buffer
ARIYL02
May Call: To:
ARIYM02 - Crash the system; handle error messages.
Write the data from log buffer
ARIYL03
May Call: To:
ARIYI19 - Get a page into real storage.
ARIYK06 - Wait for access to an S-latch mode.
ARIYK07 - Give up an access to an S-latch.
ARIYM02 - Crash the system; handle error messages.
ARIYK19 - Post (wakeup) the designated process.
ARIYL02 - Move data to log buffer.
ARIYT16 - Schedule a rollback or a commit.
Crash the system; handle error messages
ARIYM02
May Call: To:
ARICABE - Handle SQL/DS ABEND.
ARICTRM - Terminate SQL/DS.

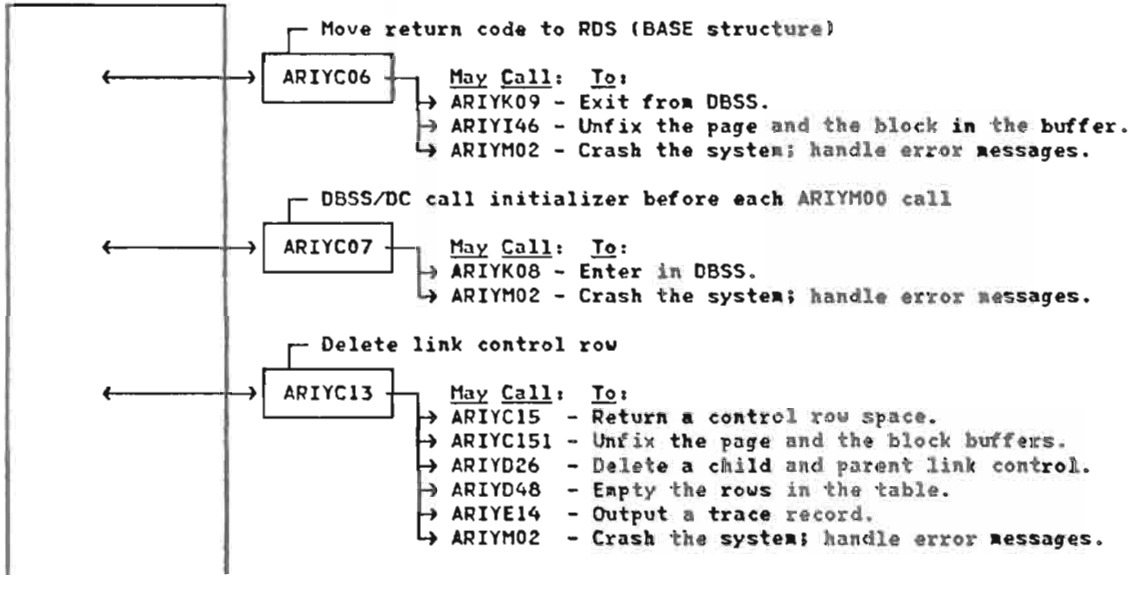
(From ARIYM00, page 278)
ARIYC10 (UPDATE A CONTROL ROW)



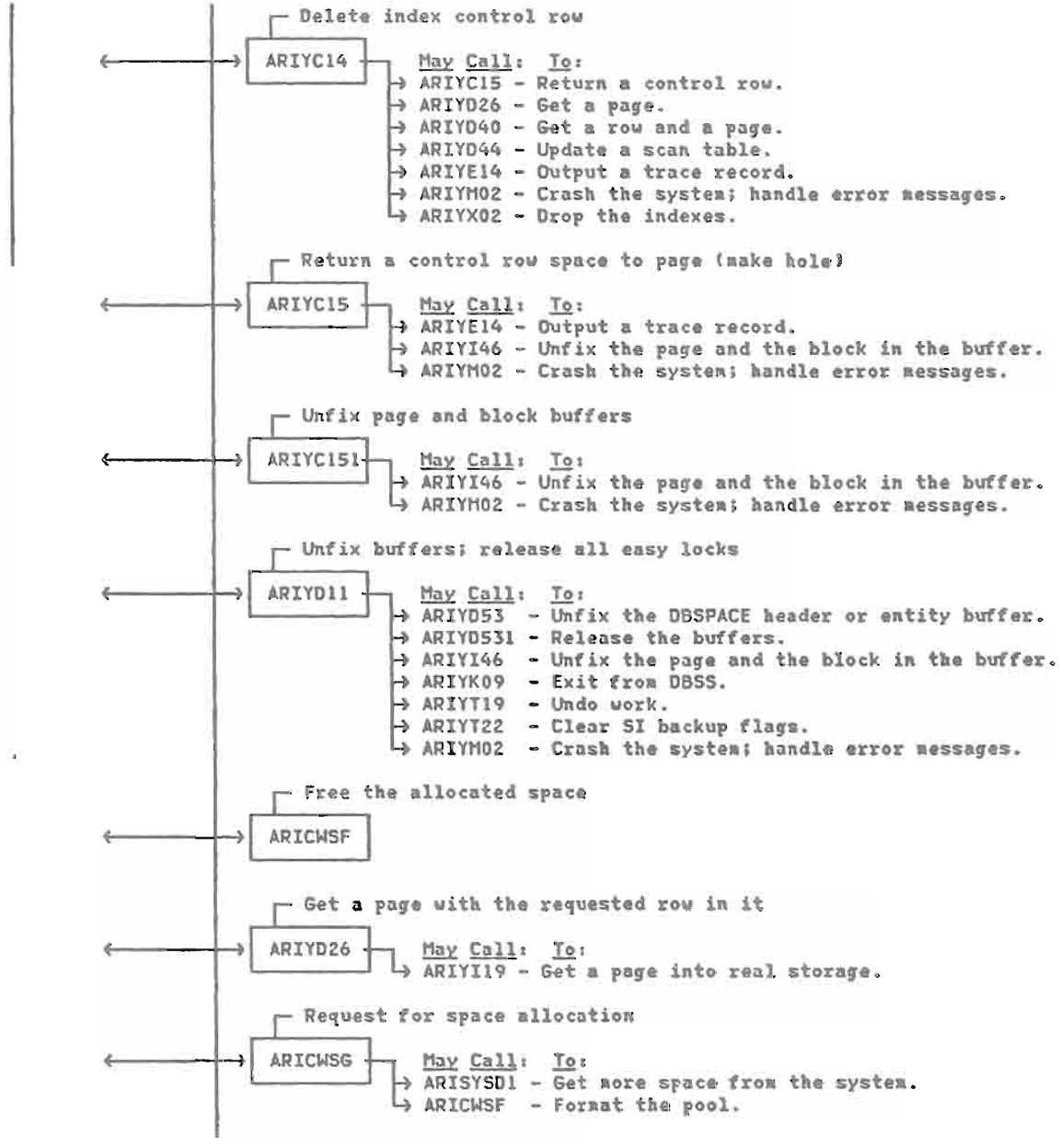
DBSS/DC (continued)

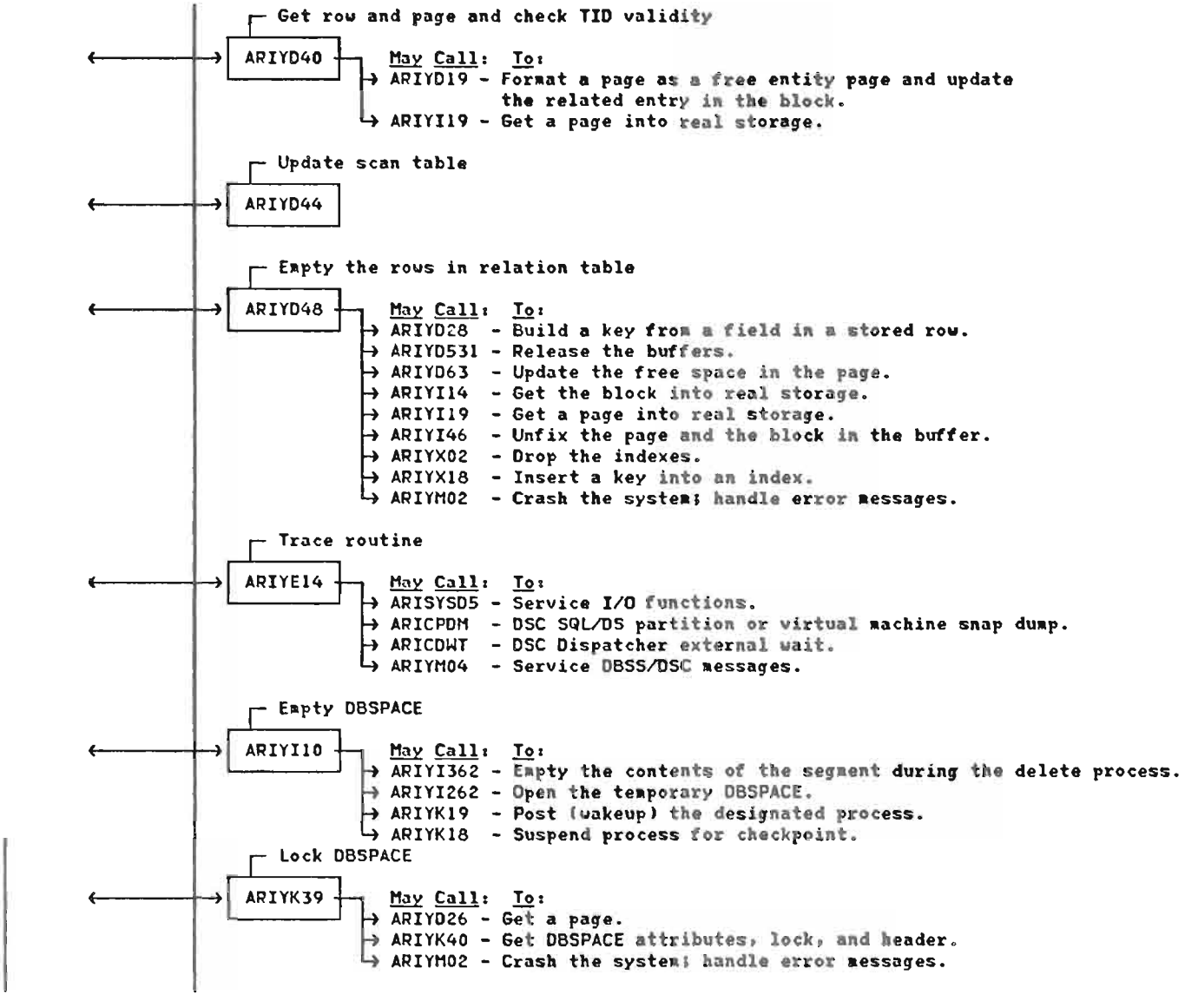


(From ARIYM00, page 278)
ARIYC11 (DELETE CONTROL ROWS)

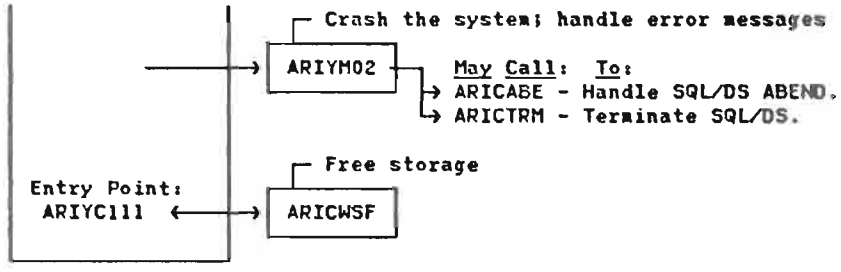


DBSS/DC (continued)



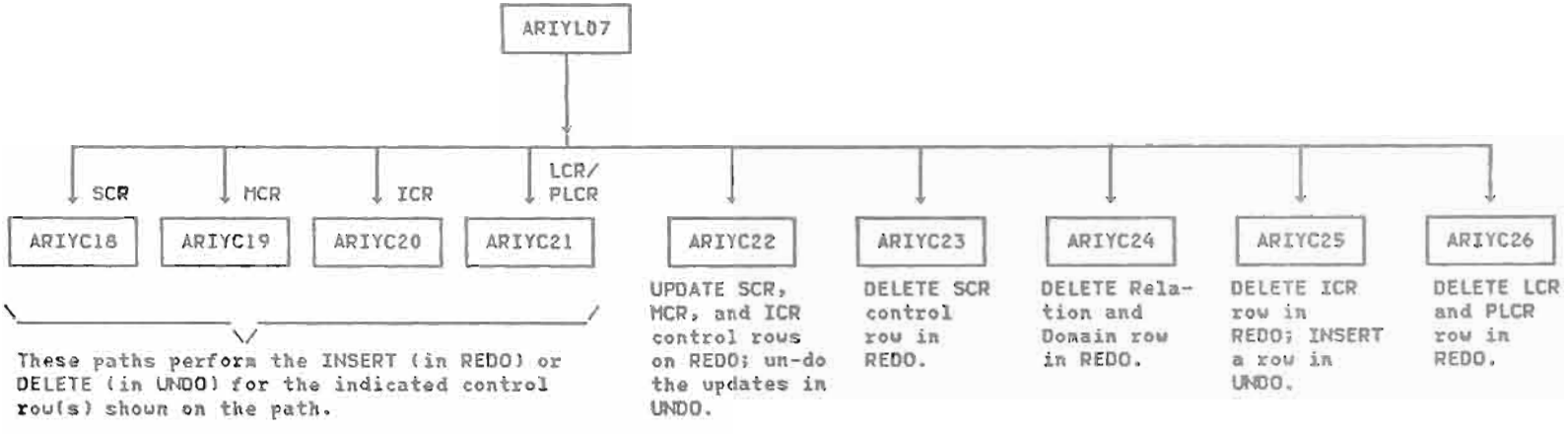



Get DBSPACE attributes, lock, and header
ARIYK40
May Call: To:
ARIYC09 - Format the header page.
ARIYI14 - Get a block into real storage.
ARIYI19 - Get a page into real storage.
ARIYI42 - Get DBSPACE attributes and number.
ARIYK41 - Request access to a named gate in the designated mode.
ARIYM02 - Crash the system; handle error messages.
Get space for log buffer
ARIYL01
May Call: To:
ARIYI19 - Get a page into real storage.
ARIYK06 - Wait for access to an S-latch mode.
ARIYK07 - Give up an access to an S-latch.
ARIYM02 - Crash the system; handle error messages.
ARIYK19 - Post (wakeup) the designated process.
ARIYL02 - Move data to the log buffer.
ARIYT16 - Schedule a rollback or a commit.
Move data to log buffer
ARIYL02
May Call: To:
ARIYM02 - Crash the system; handle error messages.
Write the data from log buffer
ARIYL03
May Call: To:
ARIYI46 - Unfix the page and the block in the buffer.
ARIYI48 - Write a log page out on disk.
ARIYK18 - Suspend the process.
ARIYK19 - Post (wakeup) the designated process.
ARIYM02 - Crash the system; handle error messages.
Force end of transaction
ARIYT141
May Call: To:
ARIYI05 - Release internal DBSPACEs.
ARIYK08 - Enter in DBSS.
ARIYK09 - Exit from DBSS.
ARIYL01 - Allocate space for the log buffer.
ARIYL02 - Move data to the log buffer.
ARIYL03 - Write the data from the log buffer.
ARIYL08 - Perform system checkpoints.
ARIYM02 - Crash the system; handle error messages.
ARIYT19 - Undo work.
ARIYT22 - Clear SI backup flags.



DBSS/DC RECOVERY PROCESSING - GENERAL FLOW (MAJOR DC MODULES)

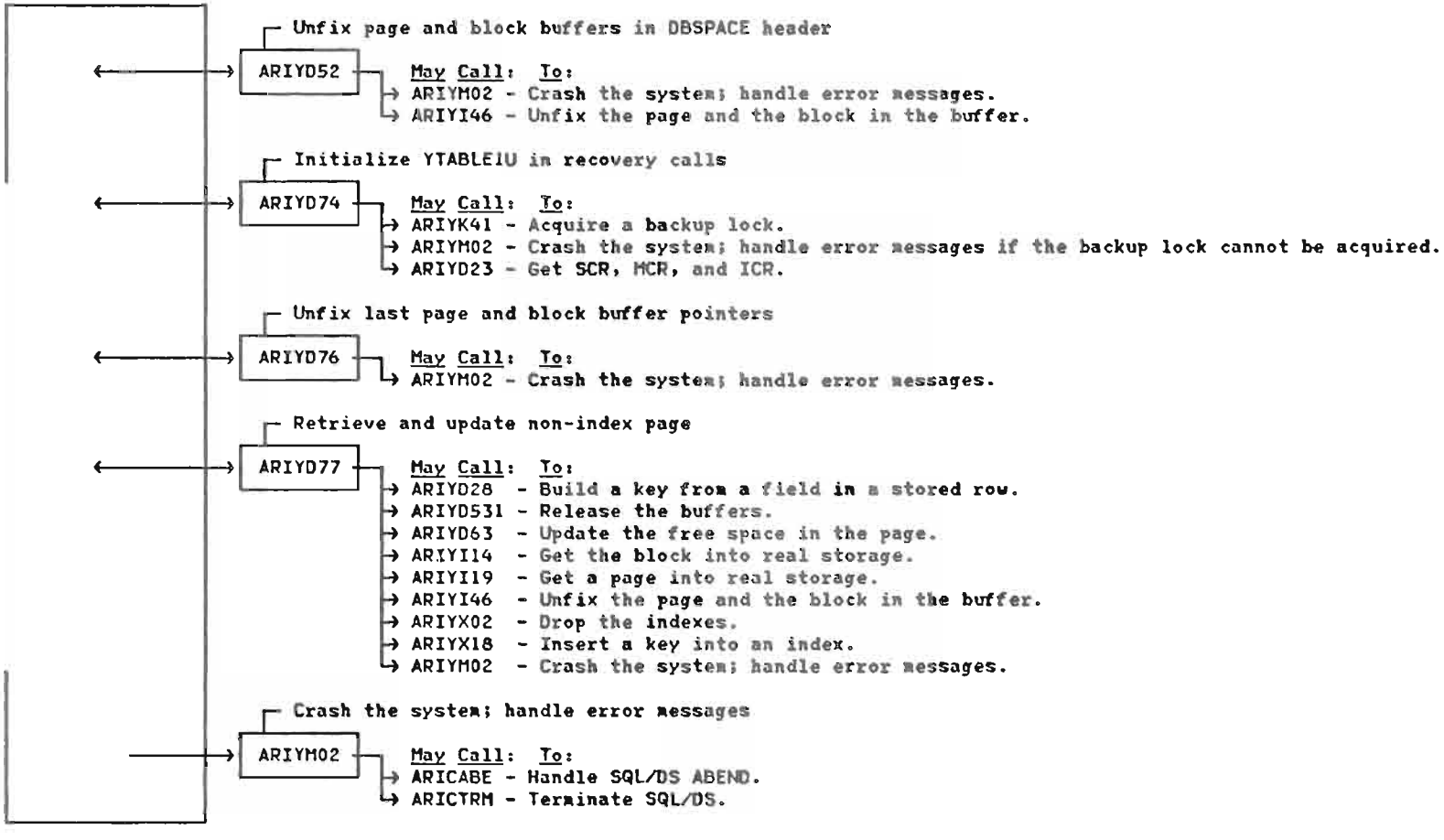
In this diagram, only the DBSS/DC major modules are shown. See the following detail diagrams for other modules being called.



See the indicated pages for detail flow for these modules:
ARIYC18 - 294; ARIYC19 - 295; ARIYC20 - 296;
ARIYC21 - 295; ARIYC22 - 294; ARIYC23 - 298;
ARIYC24 - 300; ARIYC25 - 302; ARIYC26 - 304.

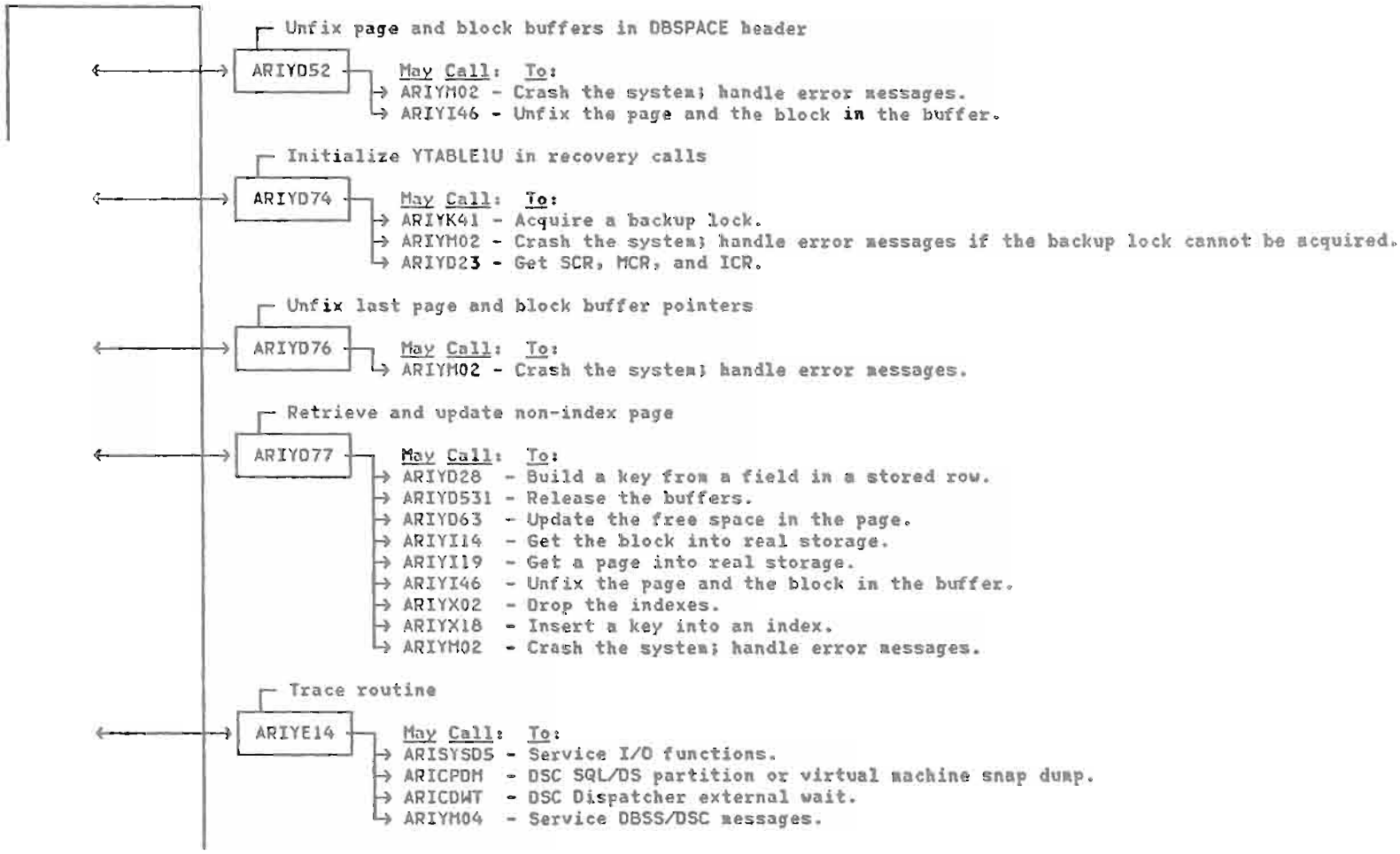
DBSS/DC RECOVERY PROCESSING - DETAIL FLOW

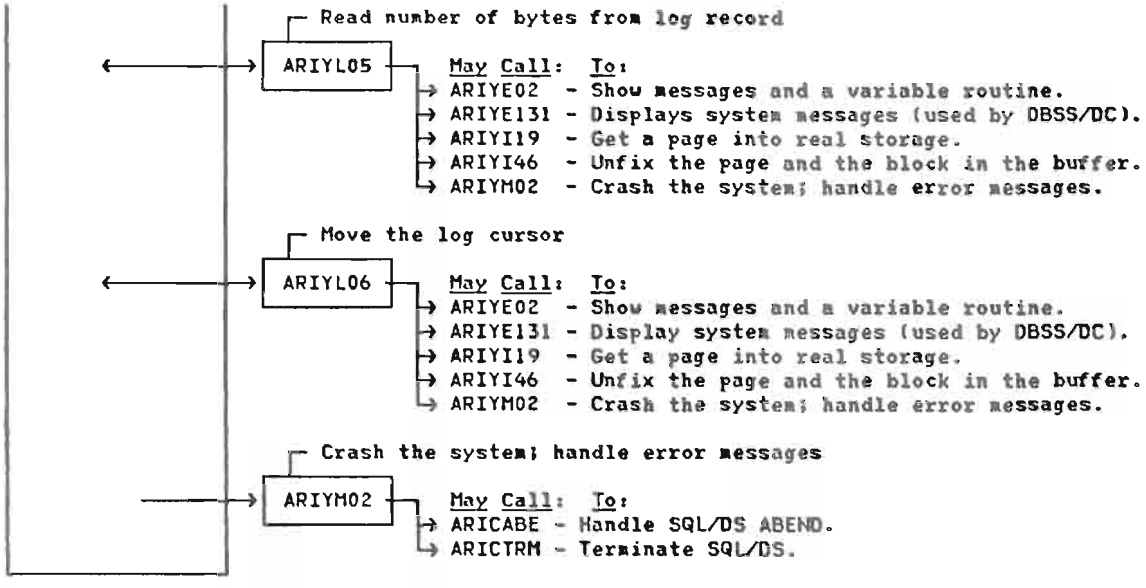
(From ARIYL07, page 293)
ARIYC18 (INSERT (in re-do) or DELETE (in un-do) SCR control row
ARIYC22 (UPDATE SCR, MCR, and ICR control rows on REDO; un-do the
updates in UNDO)
Note: The paths for these functions are essentially the same.



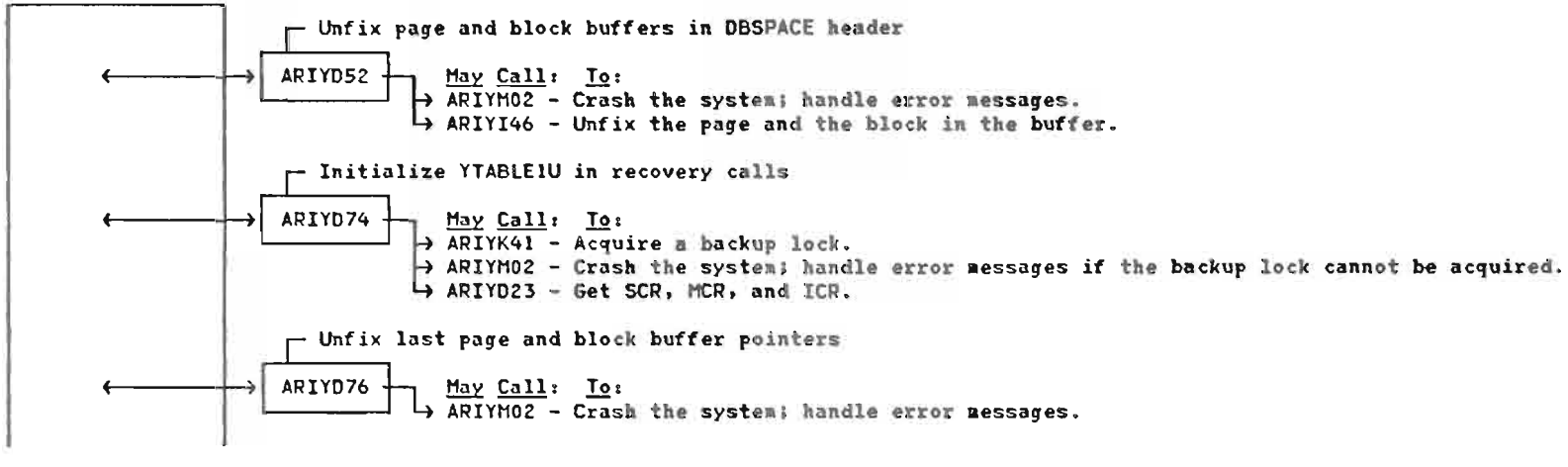
DBSS/DC (continued)

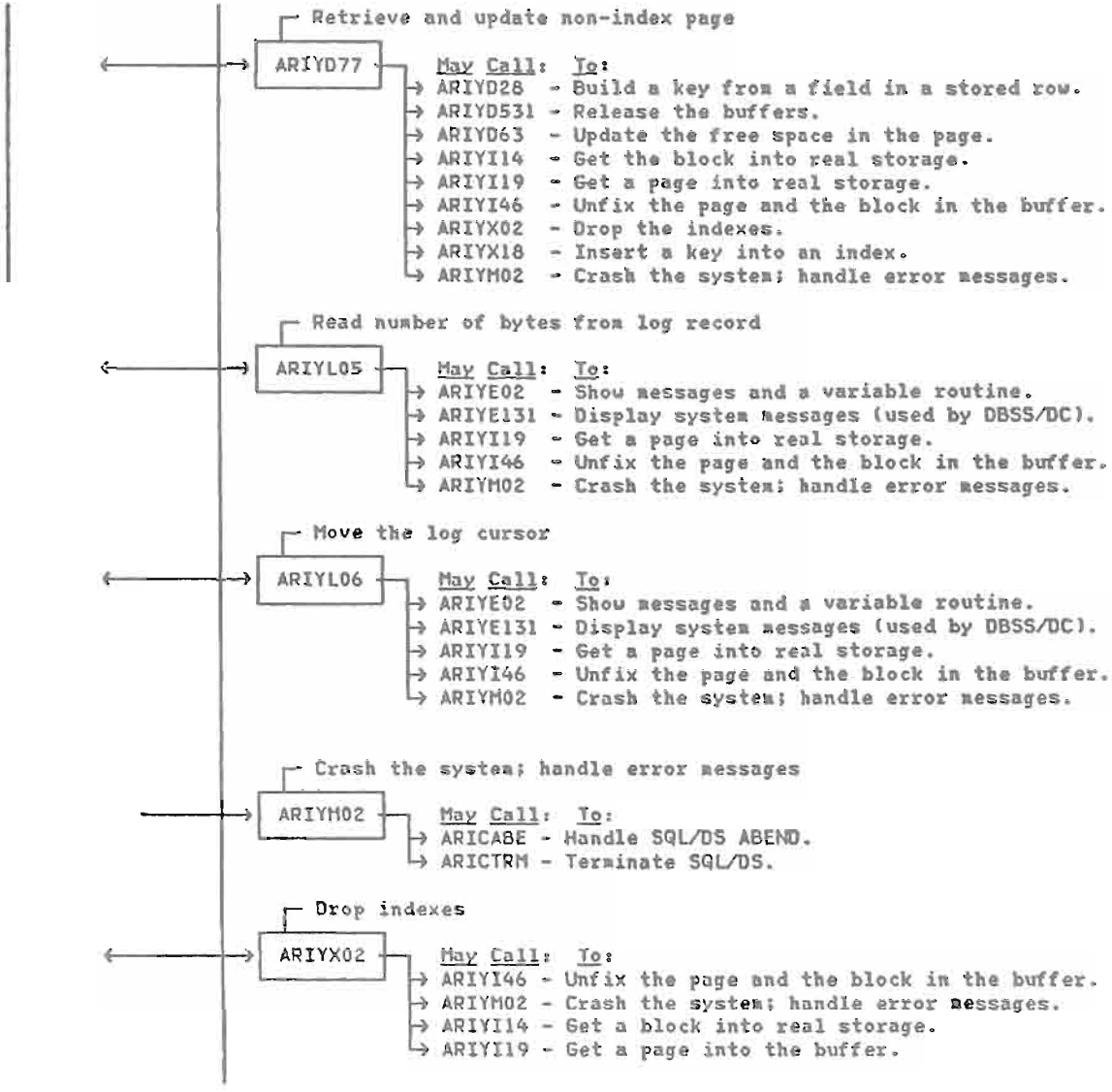
(From ARIYL07, page 293)
ARIYC19 (INSERT (in re-do) or DELETE (in un-do) MCR control row
ARIYC21 (INSERT (in re-do) or DELETE (in un-do) LCR/PLCR control row
Note: The paths for these functions are essentially the same.

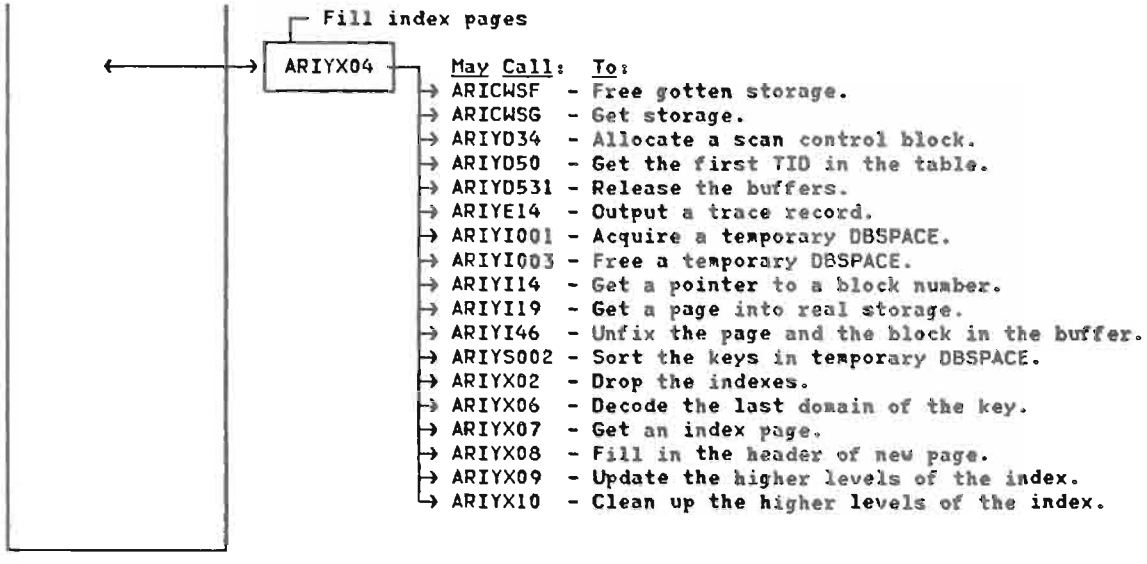




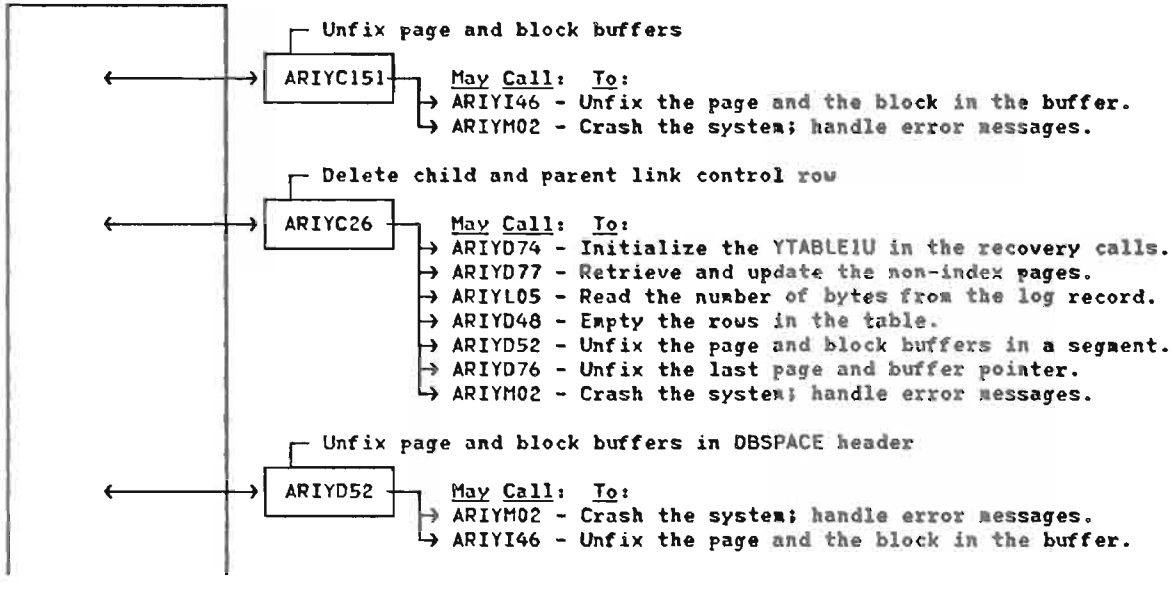
(From ARIYL07, page 293)
ARIYC20 (INSERT (in re-do) or DELETE (in un-do) ICR control row

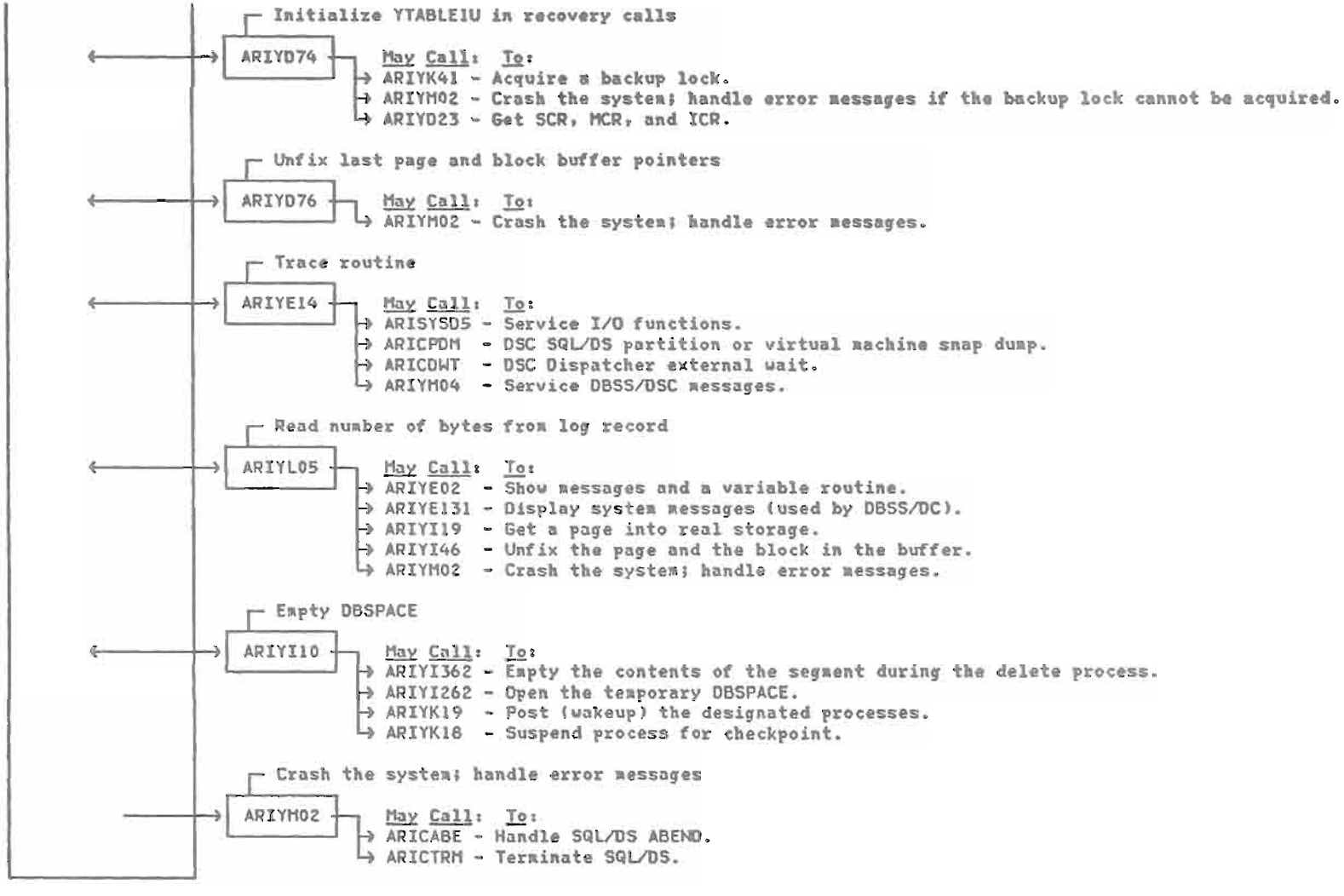



Retrieve and update non-index page
ARIYD77
May Call: To:
ARIYD28 - Build a key from a field in a stored row.
ARIYD531 - Release the buffers.
ARIYD63 - Update the free space in the page.
ARIYI14 - Get the block into real storage.
ARIYI19 - Get a page into real storage.
ARIYI46 - Unfix the page and the block in the buffer.
ARIYX02 - Drop the indexes.
ARIYX18 - Insert a key into an index.
ARIYM02 - Crash the system; handle error messages.
Read number of bytes from log record
ARIYL05
May Call: To:
ARIYE02 - Show messages and a variable routine.
ARIYE131 - Display system messages (used by DBSS/DC).
ARIYI19 - Get a page into real storage.
ARIYI46 - Unfix the page and the block in the buffer.
ARIYM02 - Crash the system; handle error messages.
Move the log cursor
ARIYL06
May Call: To:
ARIYE02 - Show messages and a variable routine.
ARIYE131 - Display system messages (used by DBSS/DC).
ARIYI19 - Get a page into real storage.
ARIYI46 - Unfix the page and the block in the buffer.
ARIYM02 - Crash the system; handle error messages.
Crash the system; handle error messages
ARIYM02
May Call: To:
ARICABE - Handle SQL/DS ABEND.
ARICTRM - Terminate SQL/DS.
Drop indexes
ARIYX02
May Call: To:
ARIYI46 - Unfix the page and the block in the buffer.
ARIYM02 - Crash the system; handle error messages.
ARIYI14 - Get a block into real storage.
ARIYI19 - Get a page into the buffer.

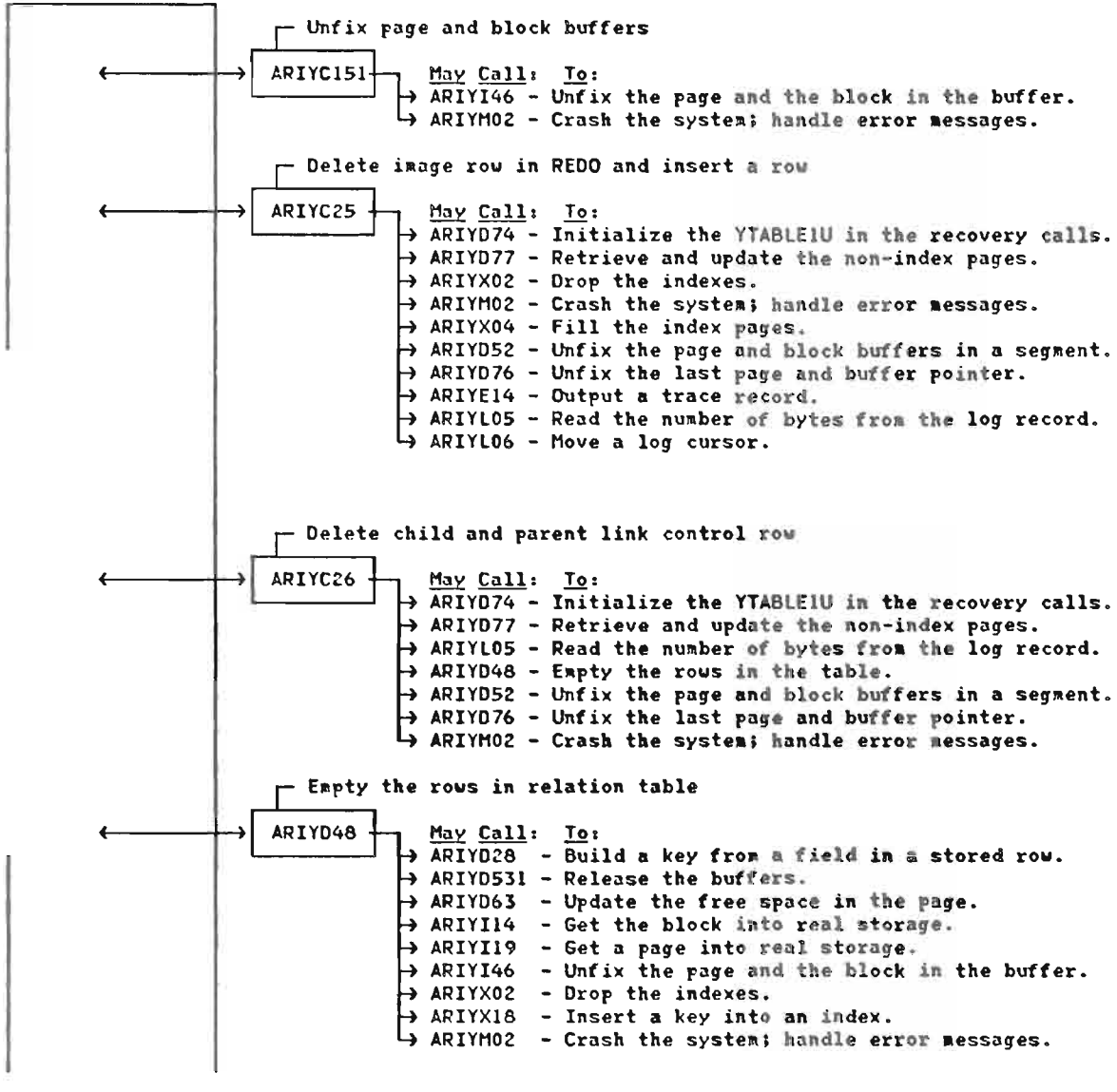


(From ARIYL07, page 293)
ARIYC23 (DELETE SCR control row in REDO)



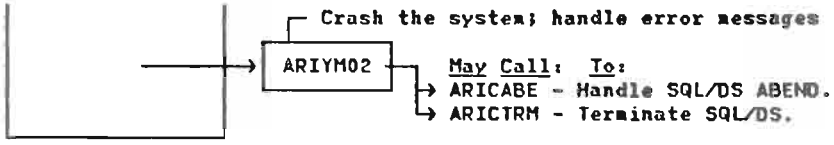

Initialize YTABLE1U in recovery calls
ARIYD74
May Call: To:
ARIYK41 - Acquire a backup lock.
ARIYM02 - Crash the system; handle error messages if the backup lock cannot be acquired.
ARIYD23 - Get SCR, MCR, and ICR.
Unfix last page and block buffer pointers
ARIYD76
May Call: To:
ARIYM02 - Crash the system; handle error messages.
Trace routine
ARIYE14
May Call: To:
ARISYSD5 - Service I/O functions.
ARICPDM - DSC SQL/DS partition or virtual machine snap dump.
ARICDWT - DSC Dispatcher external wait.
ARIYM04 - Service DBSS/DSC messages.
Read number of bytes from log record
ARIYL05
May Call: To:
ARIYE02 - Show messages and a variable routine.
ARIYE131 - Display system messages (used by DBSS/DC).
ARIYI19 - Get a page into real storage.
ARIYI46 - Unfix the page and the block in the buffer.
ARIYM02 - Crash the system; handle error messages.
Empty DBSPACE
ARIYI10
May Call: To:
ARIYI362 - Empty the contents of the segment during the delete process.
ARIYI262 - Open the temporary DBSPACE.
ARIYK19 - Post (wakeup) the designated processes.
ARIYK18 - Suspend process for checkpoint.
Crash the system; handle error messages
ARIYM02
May Call: To:
ARICABE - Handle SQL/DS ABEND.
ARICTRM - Terminate SQL/DS.

(From ARIYL07, page 293)
ARIYC24 (Delete relation and domain row in REDO)

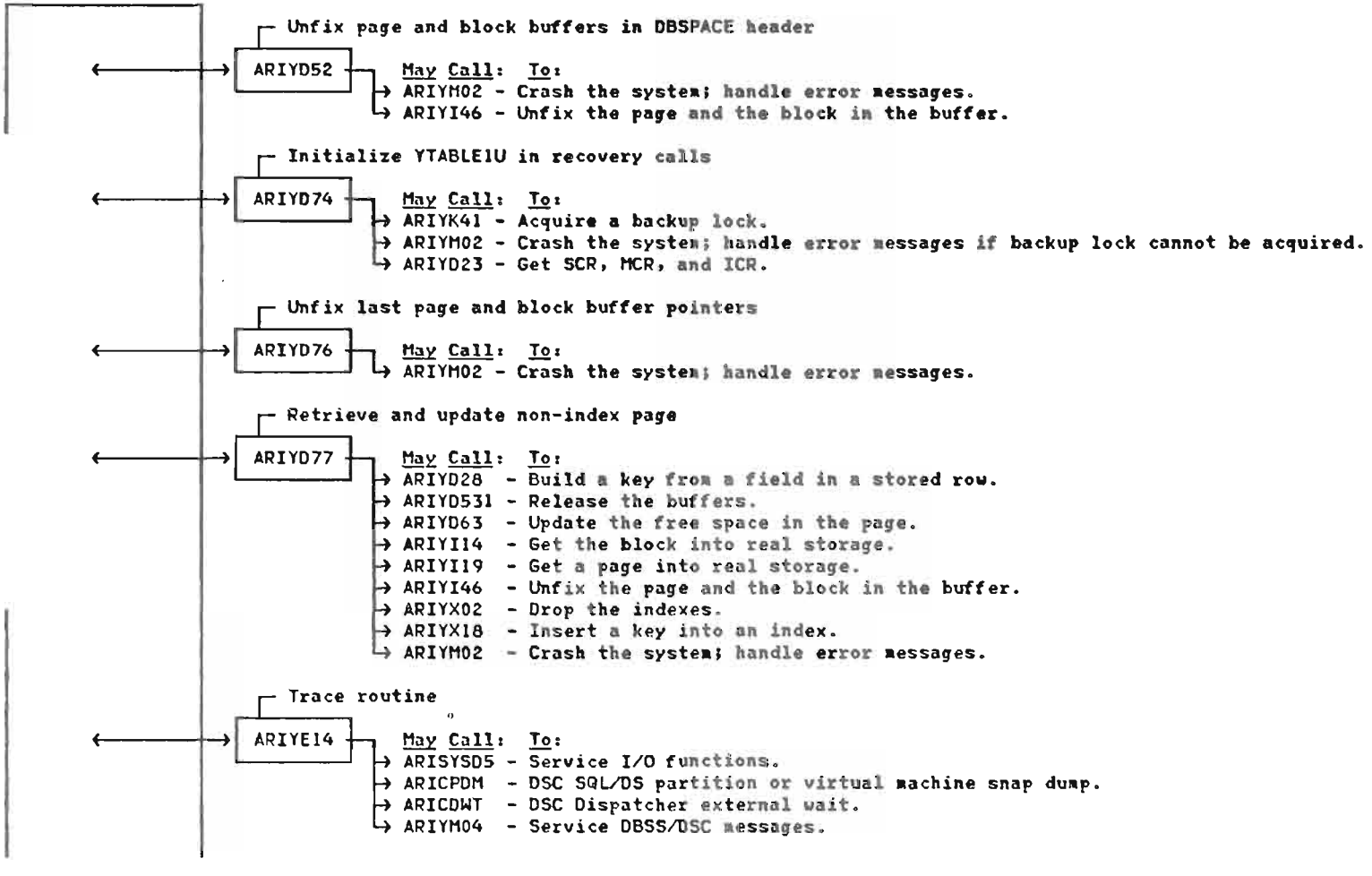


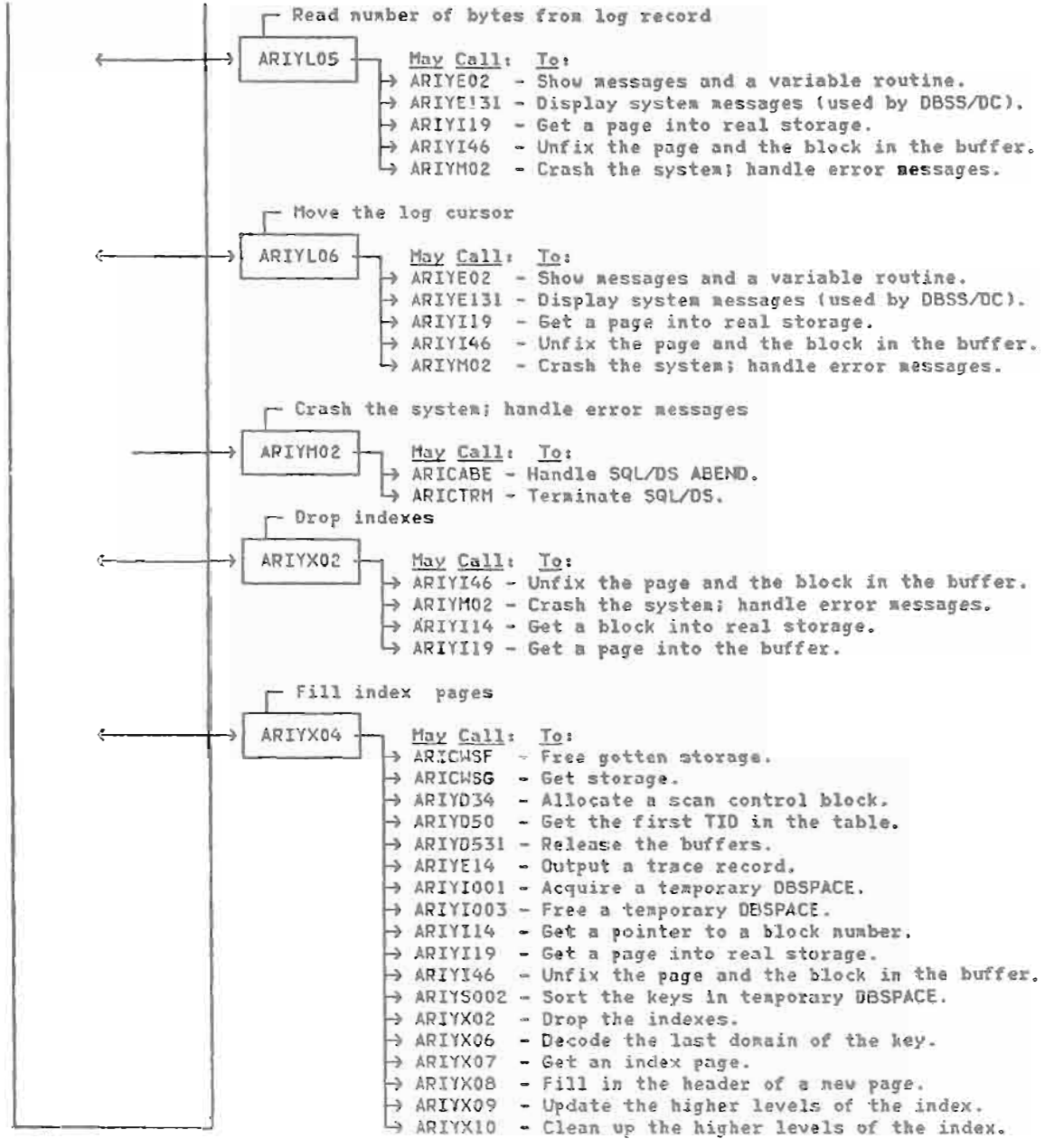

Unfix page and block buffers in DBSPACE header
ARIYD52
May Call: To:
ARIYM02 - Crash the system; handle error messages.
ARIYI46 - Unfix the page and the block in the buffer.
Initialize YTABLE1U in recovery calls
ARIYD74
May Call: To:
ARIYK41 - Acquire a backup lock.
ARIYM02 - Crash the system; handle error messages if backup lock cannot be acquired.
ARIYD23 - Get SCR, MCR, and ICR.
Unfix last page and block buffer pointers
ARIYD76
May Call: To:
ARIYM02 - Crash the system; handle error messages.
Retrieve and update non-index page
ARIYD77
May Call: To:
ARIYD28 - Build a key from a field in a stored row.
ARIYD531 - Release the buffers.
ARIYD63 - Update the free space in the page.
ARIYI14 - Get the block into real storage.
ARIYI19 - Get a page into real storage.
ARIYI46 - Unfix the page and the block in the buffer.
ARIYX02 - Drop the indexes.
ARIYX18 - Insert a key into an index.
ARIYM02 - Crash the system; handle error messages.
Trace routine
ARIYE14
May Call: To:
ARISYSD5 - Service I/O functions.
ARICPDM - DSC SQL/DS partition or virtual machine snap dump.
ARICDWT - DSC Dispatcher external wait.
ARIYM04 - Service DBSS/DSC messages.
Read number of bytes from log record
ARIYL05
May Call: To:
ARIYE02 - Show messages and a variable routine.
ARIYE131 - Display system messages (used by DBSS/DC).
ARIYI19 - Get a page into real storage.
ARIYI46 - Unfix the page and the block in the buffer.
ARIYM02 - Crash the system; handle error messages.

DBSS/DC (continued)

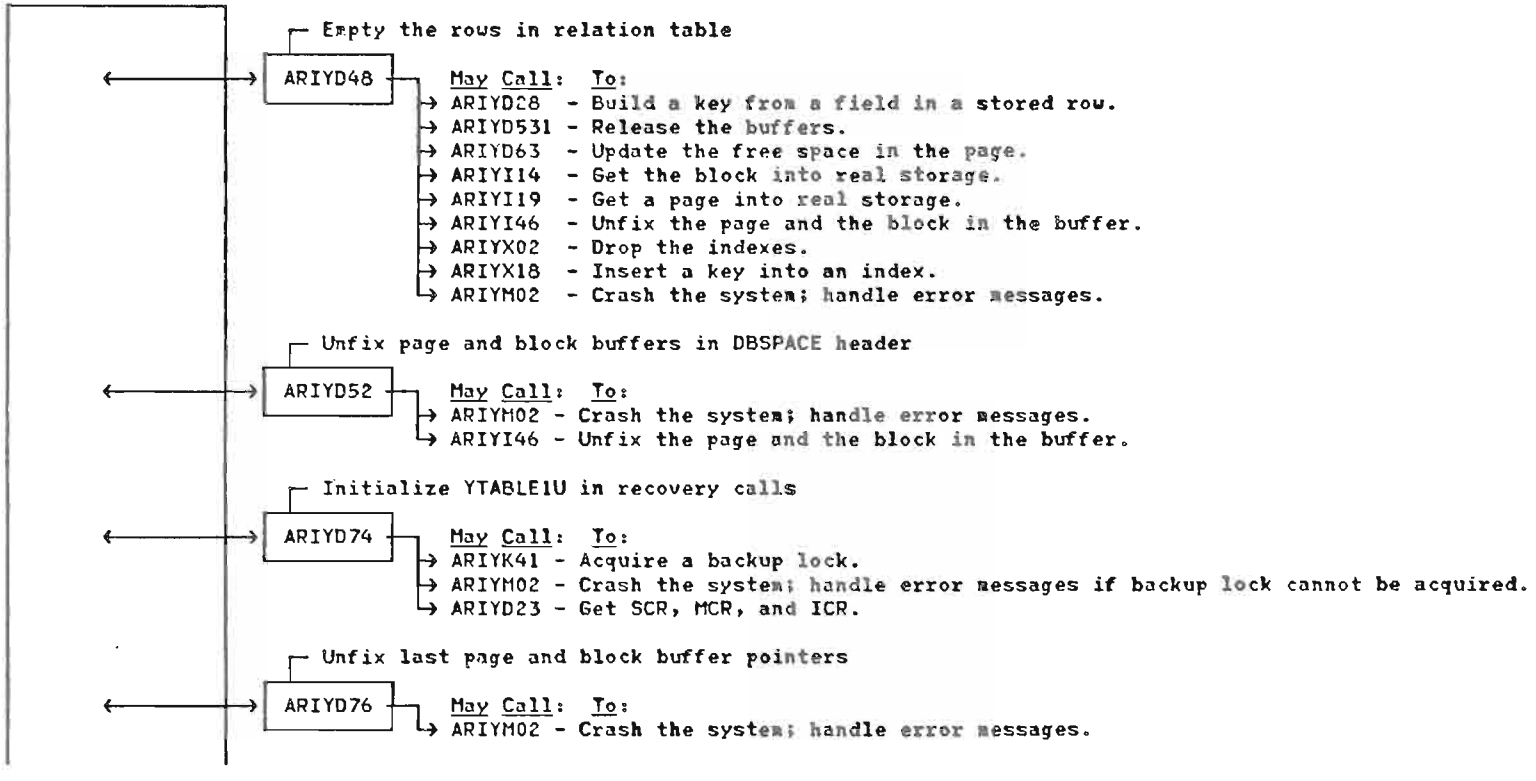


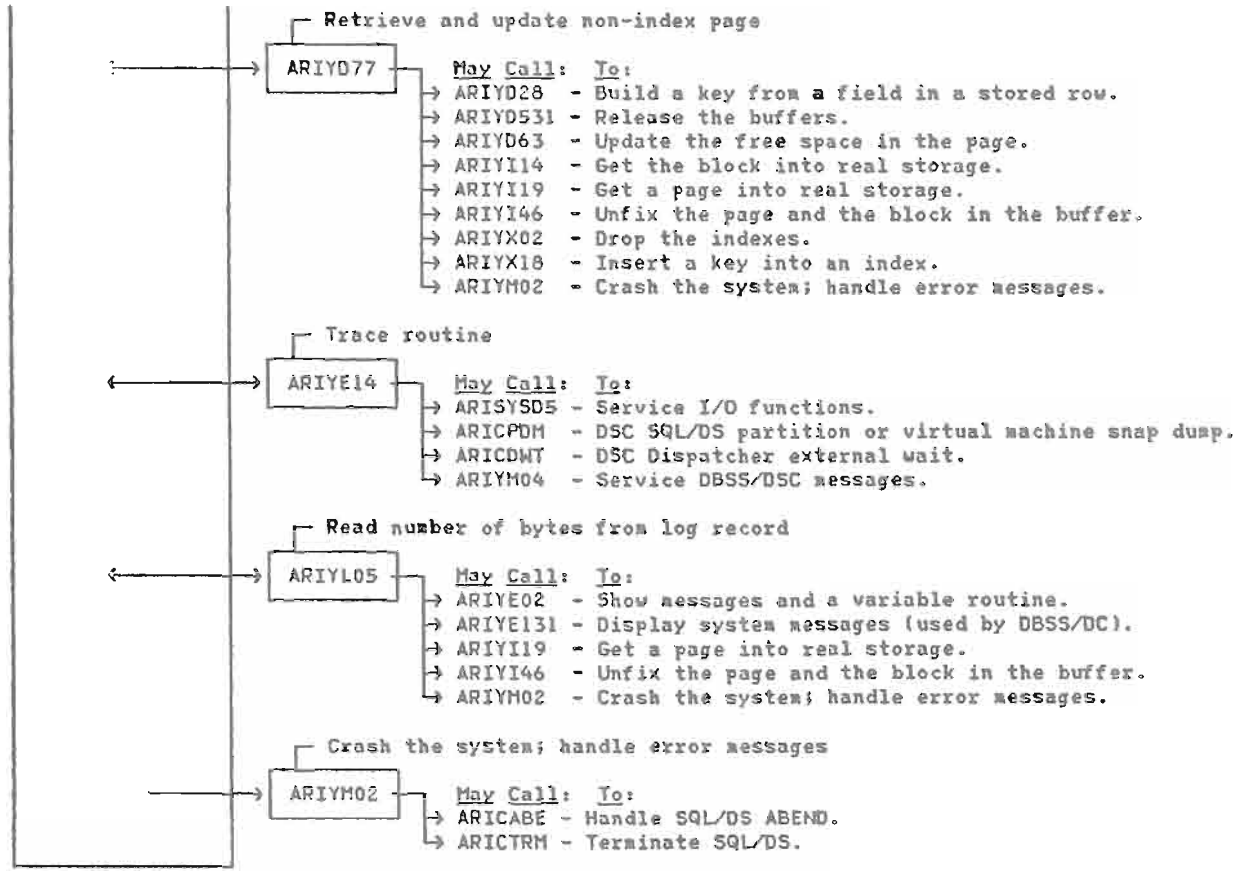
(From ARIYL07, page 293)
ARIYC25 (Delete ICR row in REDO; insert a row in UNDO)





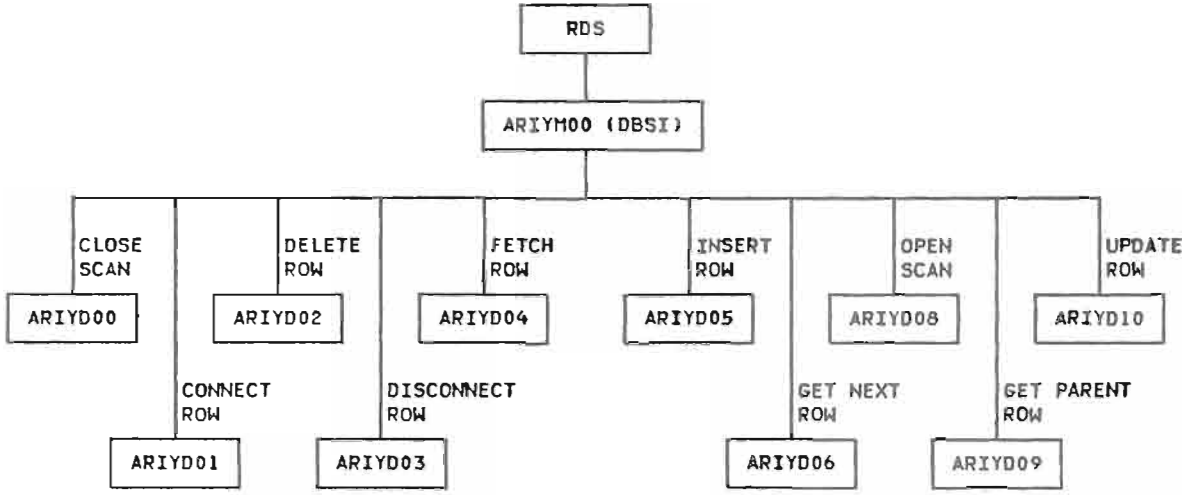
(From ARIYL07, page 293)
ARIYC26 (Delete LCR and PLCR row in REDO)




Retrieve and update non-index page
ARIYD77
May Call: To:
ARIYD28 - Build a key from a field in a stored row.
ARIYD531 - Release the buffers.
ARIYD63 - Update the free space in the page.
ARIYI14 - Get the block into real storage.
ARIYI19 - Get a page into real storage.
ARIYI46 - Unfix the page and the block in the buffer.
ARIYX02 - Drop the indexes.
ARIYX18 - Insert a key into an index.
ARIYM02 - Crash the system; handle error messages.
Trace routine
ARIYE14
May Call: To:
ARISYSD5 - Service I/O functions.
ARICPDM - DSC SQL/DS partition or virtual machine snap dump.
ARICDWT - DSC Dispatcher external wait.
ARIYM04 - Service DBSS/DSC messages.
Read number of bytes from log record
ARIYL05
May Call: To:
ARIYE02 - Show messages and a variable routine.
ARIYE131 - Display system messages (used by DBSS/DC).
ARIYI19 - Get a page into real storage.
ARIYI46 - Unfix the page and the block in the buffer.
ARIYM02 - Crash the system; handle error messages.
Crash the system; handle error messages
ARIYM02
May Call: To:
ARICABE - Handle SQL/DS ABEND.
ARICTRM - Terminate SQL/DS.

DBSS/DM NORMAL PROCESSING - GENERAL FLOW (TOP DM MODULES)

In this diagram, only the DBSS/DM normal procesing top modules are shown. See the following detail diagrams for other modules being called.

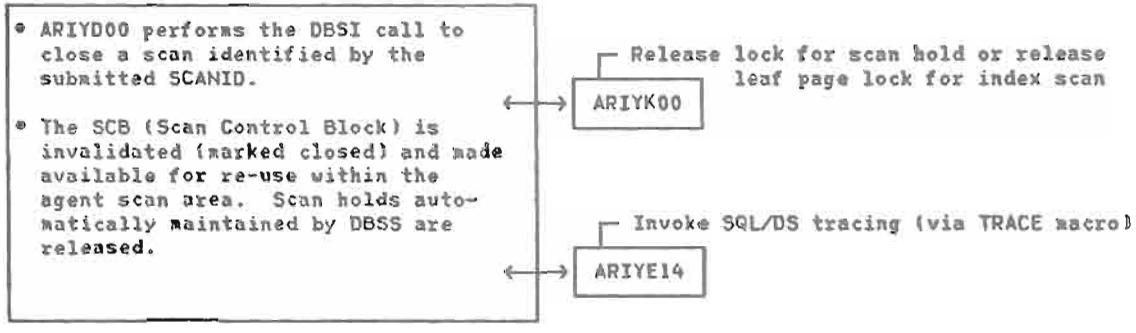


For:

- CLOSE SCAN (ARIYD00), see page 307.
- CONNECT ROW (ARIYD01), see page 308.
- DELETE ROW (ARIYD02), see page 310.
- DISCONNECT ROW (ARIYD03), see page 313.
- FETCH ROW (ARIYD04), see page 315.
- INSERT ROW (ARIYD05), see page 317.
- GET NEXT ROW (ARIYD06), see page 321.
- OPEN SCAN (ARIYD08), see page 326.
- GET PARENT ROW (ARIYD09), see page 332.
- UPDATE ROW (ARIYD10), see page 334.

CLOSE SCAN

* Called from ARIYM00
  ARIYD00 (CLOSE SCAN)



CONNECT Child ROW into a Link

* Called from ARIYM00
ARIYD01 (CONNECT ROW)

- ARIYD01 performs the DBSI call to connect child row T1 before or after another row T2 in a link.

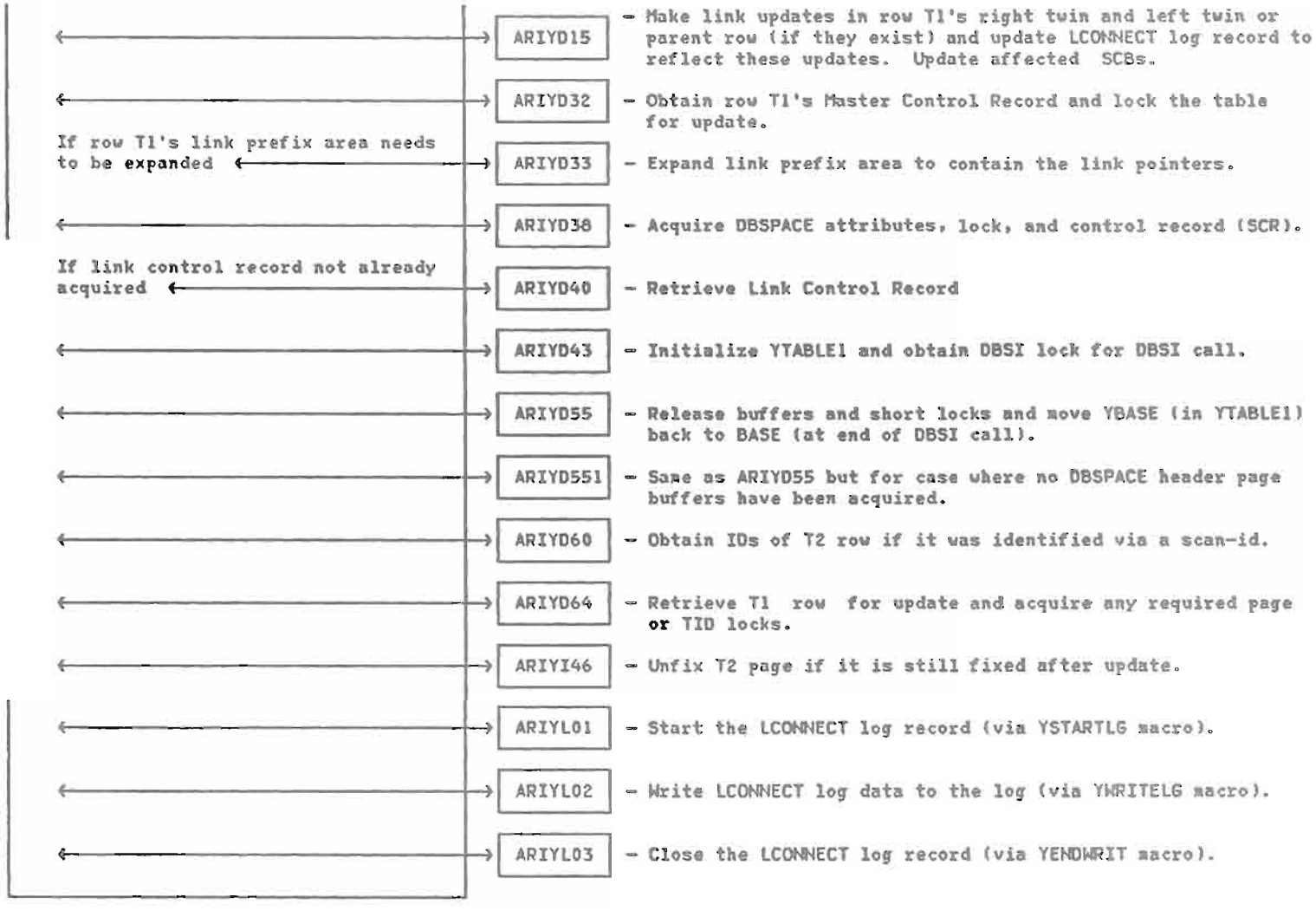
  The link is identified by providing DBSPACE and link-id (LID). The row T1 is identified by providing RID and legal TID. Row T2, which in a binary link may be a child or parent, is identified by providing:
  - The ID of active and positioned 'ON' scan in SCANID, or
  - SCANID=0, PSEGMENT, PRID and legal PTID. Note that these variables may be used (unlike in other calls) to identify a child.

  QUALF must be set to 'B' (CONNECT BEFORE) or to 'A' (CONNECT AFTER). 'B' is illegal if T2 identifies a parent. T1 cannot be a row already connected in the used link. If T2 is a child in a binary link, it must already have been connected in the link itself.

  CONNECT will change to on 'T1' the state of a scan that:
  - Was opened by the LUW issuing CONNECT, and
  - Is a link-scan on the link identified by LID, and
  - Is positioned between T2 and the previous row (or at 'TOF' if T2 is the first row) and QUALF='B', or is positioned between T2 and the following row (or at 'EOF' if T2 is the last row) and QUALF='A'.

- The following modules may be called (not necessarily in the order shown):

  If DBSI-call backup is occurring ⟷ ARIYD11
  - Unfix DBSI-call held buffers and release acquired DBSI locks.
  - Initiate LUW rollback if required.
  - Determine if the DBSI call should be retried.


ARIYD15
- Make link updates in row T1's right twin and left twin or parent row (if they exist) and update LCONNECT log record to reflect these updates. Update affected SCBs.
ARIYD32
- Obtain row T1's Master Control Record and lock the table for update.
If row T1's link prefix area needs to be expanded
ARIYD33
- Expand link prefix area to contain the link pointers.
ARIYD38
- Acquire DBSPACE attributes, lock, and control record (SCR).
If link control record not already acquired
ARIYD40
- Retrieve Link Control Record
ARIYD43
- Initialize YTABLE1 and obtain DBSI lock for DBSI call.
ARIYD55
- Release buffers and short locks and move YBASE (in YTABLE1) back to BASE (at end of DBSI call).
ARIYD551
- Same as ARIYD55 but for case where no DBSPACE header page buffers have been acquired.
ARIYD60
- Obtain IDs of T2 row if it was identified via a scan-id.
ARIYD64
- Retrieve T1 row for update and acquire any required page or TID locks.
ARIYI46
- Unfix T2 page if it is still fixed after update.
ARIYL01
- Start the LCONNECT log record (via YSTARTLG macro).
ARIYL02
- Write LCONNECT log data to the log (via YWRITELG macro).
ARIYL03
- Close the LCONNECT log record (via YENDWRIT macro).

DELETE ROW from a Table

* Called from ARIYM00
  ARIYD02 (DELETE ROW)

• ARIYD02 performs the DBSI call to delete a row from a table and, optionally, disconnect it from a link. The row to be deleted can be identified by providing:
  - The ID of an active and positioned 'ON' scan in SCANID, or
  - SCANID=0, DBSPACE, RID and legal TID, or
  - SCANID=0, DBSPACE, RID, TID=0, IID, ICOMP, and a key value in KDOMAINS (no key value if ICOMP=FIRST).

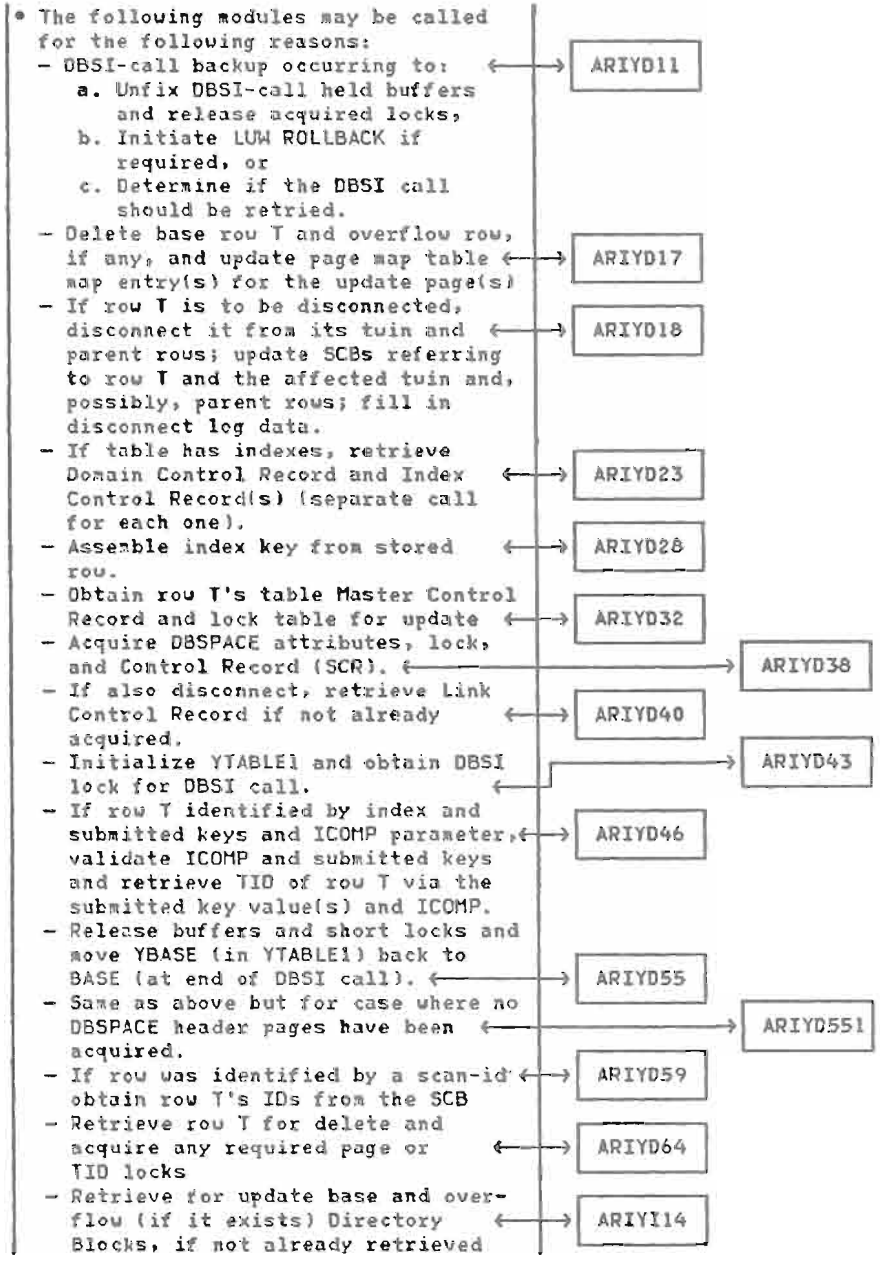
  DBSPACE, table-id, and TID of the deleted row are returned in SEGMENT, RID, and TID (fields in BASE).

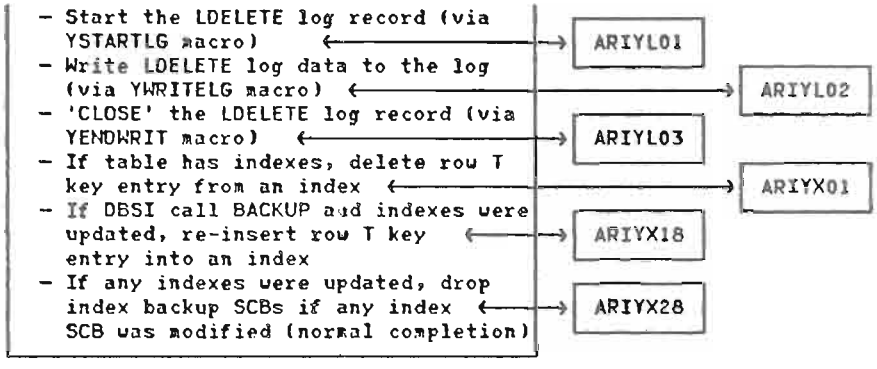
  The delete operation is not accepted if the row to be deleted is a parent of some child in a link. If the row is a child in a link, the delete operation is not accepted unless the link ID is specified in LID. The protocols for possible link scan update as in CONNECT. Also, ARIYD02 sets PSEGMENT, PRID, and PTID as is done in ARIYD03 (DISCONNECT).

  All scans opened by the LUW issuing DELETE T and positioned 'on' or adjacent to T are repositioned as follows:
  - At 'TOF' if T is the first row of the scan-set,
  - At 'EOF' if T is the last row of the scan-set,
  - 'BETWEEN' T1 and T2 if T is between T1 and T2 in the scan-set,

  The scan is set to empty if T was the only row of the scan-set.

DBSS Data Manipulation (continued)





DISCONNECT Child ROW from a Link

* Called from ARIYM00
ARIYD03 (DISCONNECT ROW)

The link is identified by providing DBSPACE and link-id. The row to be disconnected can be identified by providing:
1. The ID of an active and positioned 'ON' scan in SCANID, or
2. SCANID=0, RID, and legal TID.

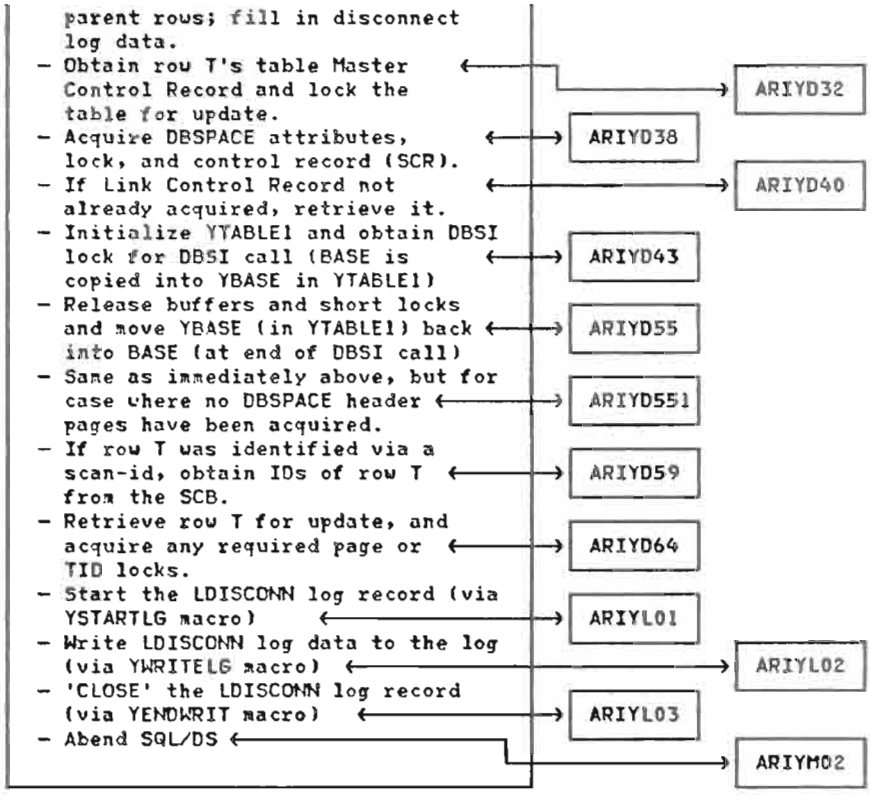
The DBSPACE, table-id, and TID of the row are returned in SEGMENT, RID, and TID (in BASE). For binary links, the DBSPACE, table-id, and TID of the parent are returned in PSEGMENT, PRID, and PTID (in BASE). For unary links, 0's (zeros) are returned in PSEGMENT, PRID, and PTID.

ARIYD03 will change the state of a scan that:

1. Was opened by the LUW issuing the DISCONNECT call, and
2. is a link-scan on the link identified by LID, and
3. is positioned 'on' or adjacent to T.

The new SCB state will be 'TOF' if T was the first row, 'EOF' if T was the last row, 'BETWEEN' T1 and T2 if T was connected between T1 and T2. The scan will be closed if T was the only row in the link set.

* The following modules may be called for the following reasons:
- DBSI-call backup occurring to: ⟷ ARIYD11
  a. Unfix DBSI-call held buffers and release acquired locks,
  b. Initiate LUW ROLLBACK if required, or
  c. Determine if the DBSI call should be retried.
- Disconnect row T from its twin and parent rows; update SCBs referring to row T and the affected twin and, possibly, ⟷ ARIYD18



FETCH ROW from a Table

* Called from ARIYM00
ARIYD04 (FETCH ROW)

The row to be fetched can be identified by providing:
1. An ID of an active and positioned 'on' scan in SCANID, or
2. SCANID=0, DBSPACE, RID, and legal TID, or
3. SCANID=0, DBSPACE, RID, TID=0, IID, ICOMP, and a key-value in KDOMAINS (if ICOMP=FIRST, no key value is needed).

Search arguments can be checked by setting NSARGS > 0 and providing a search argument list (SARGS). In that case, DOMAINS are returned only if the search arguments are satisfied. Otherwise, the BADSARG warning code is raised, and only DBSPACE, RID, and TID are returned.

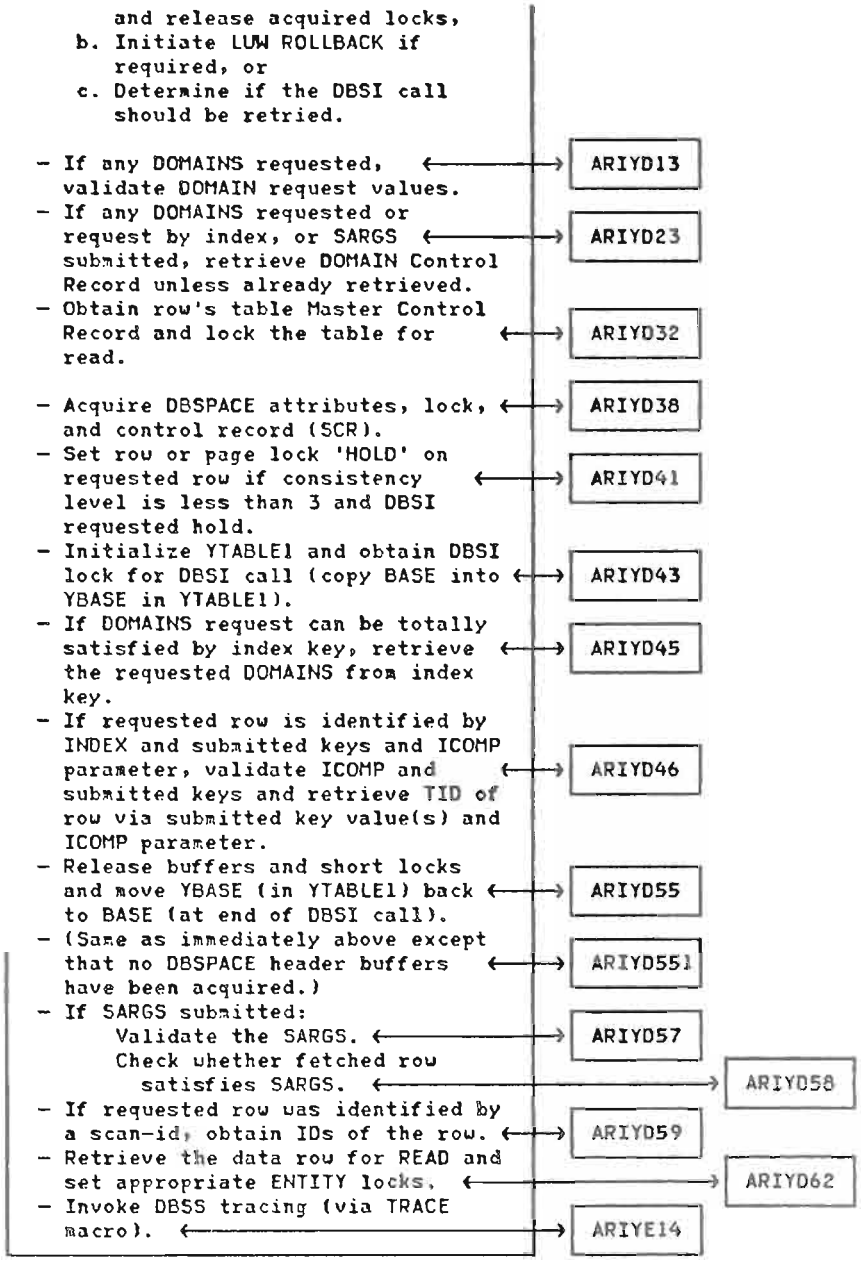
A returned row is described by DBSPACE, RID, TID, and DOMAINS. A user-controlled hold on a row can be acquired by setting HOLDIND='H'.

The number of DOMAINS to be returned is specified in NREQDOM. The DOMAINS values are pointed to by FLDPTR and are truncated to the lengths specified by FREQLTH. The actual lengths are returned in FACLTH.

For associative operation, ICOMP can take these values:
- 'M' - Match
- 'A' - First After
- 'MA' - Match or First After
- 'FI' - First

If the index has more DOMAINS than NKEYDOM, then 'DON'T CARE' values are assumed for all the non-submitted KDOMAINS.

* The following modules may be called for the following reasons:
- DBSI-call backup occuring to: ←→ ARIYD11
  a. Unfix DBSI-call held buffers



INSERT a ROW in a Table

* Called from ARIYM00
  ARIYD05 (INSERT ROW)

ARIYD05 performs the DBSI call to:
- Insert a row in a table and, optionally, connect it in a link.
- Insert a row at the end of a list.

For a table:
- The row to be inserted is described to DBSS by providing:
  1. DBSPACE, RID, DOMAINS, and TID ¬= 0, or
  2. DBSPACE, RID, DOMAINS, and TID=0.

  If TID ¬= 0, DBSS will attempt to assign to the row a TID 'close' to the submitted one. Otherwise, it will attempt to use the first index (if any) for that purpose. The point of insertion in an index is determined first by key-value (by the first-domain-value) and then by the assigned TID (by the second-domain-value if applicable). The TID actually allocated to the row is returned in TID.

- The number of submitted DOMAINS is specified in NREQDOM. If NREQDOM is smaller than the table degree, DBSS will virtually store strings of null values for the non-submitted DOMAINS. The correct lengths must be entered for fixed-length DOMAINS that are being submitted.

- If a non-zero LID is submitted, the inserted row is also connected in the link identified by SEGMENT, LID. The point of insertion is specified via PSEGMENT, PRID, PTID, and QUALF or by SCANID and QUALF as in CONNECT. The protocols for link scan update are as in CONNECT.

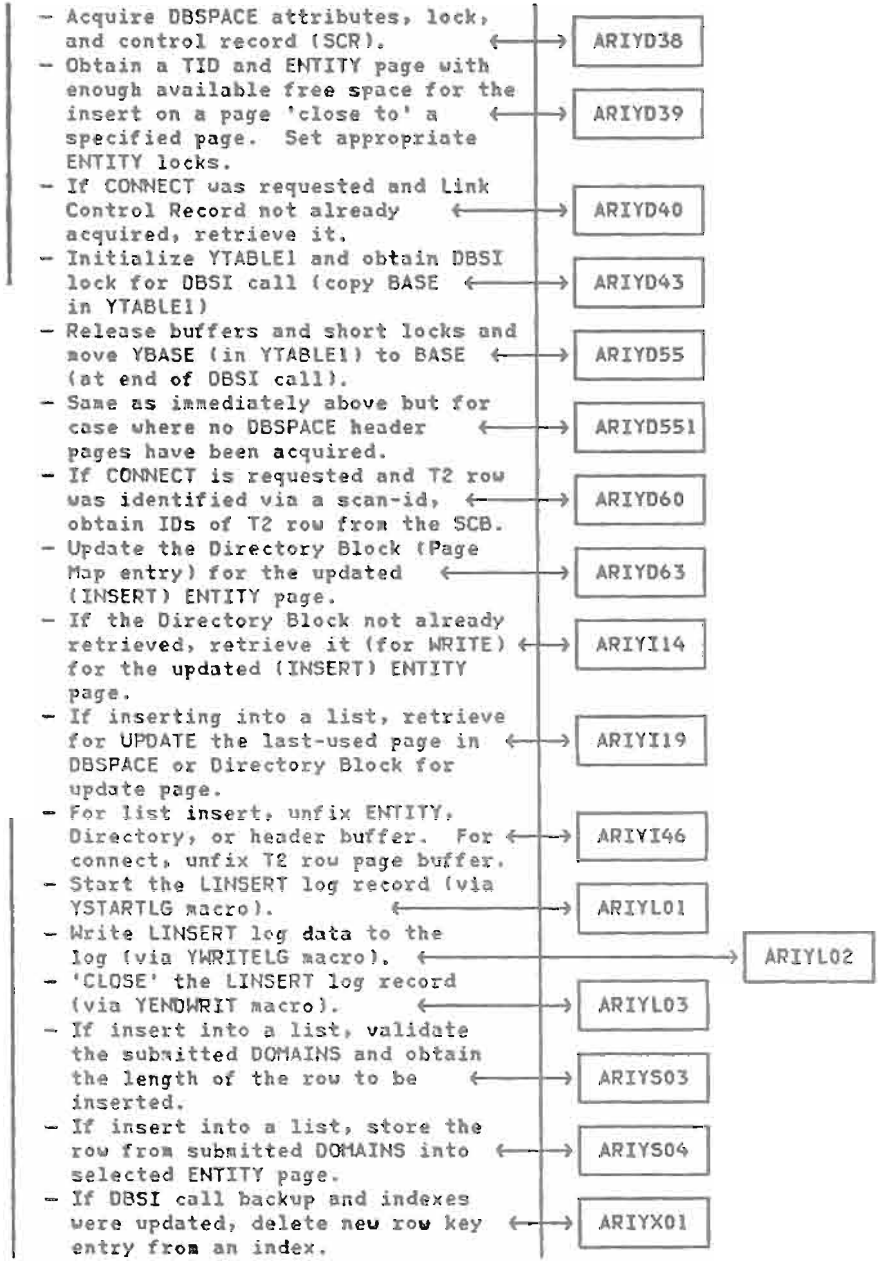
- INSERT will change to ON(T) the state of an index scan that was opened by the LUW issuing INSERT T and which is positioned in the scan order in a position adjacent to T.

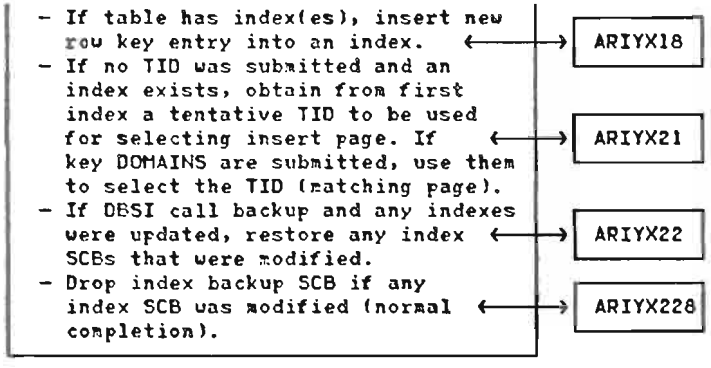
For a sequential list:
- The row to be inserted is described to the DBSI by providing DBSPACE, RID = 0, and DOMAINS. It is always inserted at the end of the sequential list.

- The number of submitted DOMAINS is specified in NREQDOM. If NREQDOM is smaller than the table degree, DBSS will actually store strings of null values for the non-submitted DOMAINS. The correct lengths must be entered for fixed-length DOMAINS that are being submitted.

* The following modules may be called for the following reasons:
  - If DBSI call backup is occurring:
    a. Unfix DBSI call held buffers release acquired DBSI locks, ←→ ARIYD11
    b. Initiate LUW ROLLBACK if required,
    c. Determine if the DBSI call should be retried.
  - If link connect requested, make the link updates in the row's right twin, and left twin or parent rows (if they exist), and update LINSERT log record to reflect these updates. Update affected SCBs. ←→ ARIYD15
  - If the table has indexes, retrieve Domain Control Record and Index Control Record(s) (separate call for each one). ←→ ARIYD23
  - If indexes to be updated, assemble index entry from submitted DOMAINS. ←→ ARIYD29
  - Obtain row's table Master Control Record and lock the table for update. ←→ ARIYD32

DBSS Data Manipulation (continued)





GET NEXT ROW During Scan

* Called from ARIYM00
ARIYD06 (GET NEXT ROW)

ARIYD06 performs the DBSI call to do the GET NEXT ROW processing. Fast next processing is done if these conditions are true:
- Previous call was for NEXT
- Same SCANID was used
- Scan was left on and clean
- Return code was 0
- Page locking was used.

ARIYD06 sets the FASTFLAG in YTABLE1 to '1'B if all the conditions are met. It also tries to fit copies of the domain control row and index control row into YTABLE1 and sets the appropriate flags YDCRFLAG and YICRFLAG.

GET NEXT is the DBSI call to search and retrieve a row from a table or sequential list using an opened scan. For a scan opened on an index of a table, the row that is returned is located by providing:
- The ID of a scan on an index in SCANID.
- A key-value restriction in ICOMP and KDOMAINS (optional). A value of NKEYDOM=0 indicates that a key restriction is not being submitted. ICOMP can take the values 'M ', 'B ', and 'MB'.
- A DOMAIN search clause in SARGS (optional). A value of NSARGS=0 indicates that a search clause is not being submitted.

The row description is returned in SEGMENT, RID, TID, and DOMAINS. The index ID on which the scan is opened is returned in IID, and LID is set to 0. The returned row is the first in the index, starting from the position presently identified in the scan and satisfying the restrictions (if any) on key-value and search clause.

If the scan is 'ON' a row, the row itself is included in the search if QUALF='N', but it is not included if QUALF='A' (AFTER).

If the scan is 'BETWEEN' two rows, the second row is always the starting point of the search. If the scan is 'EOF', no search is performed. The search is halted with a key-violation or an EOF warning in the return code when a key-value restriction (if any) cannot be satisfied or when EOF on the scan is reached.

An automatic hold on the current row is acquired by DBSS, and the hold previously acquired in the scan is released. In addition, a user-controlled hold on the current row can be acquired by setting HOLDIND='H'.

If the key restriction is violated, the scan is positioned 'ON' the first offending row that is returned in SEGMENT, RID, TID, and DOMAINS. An automatic scan hold (and, if requested, a DBSI caller hold) is acquired on this row. A warning is raised in the return code.

If EOF is reached, the scan is positioned 'EOF' after the last scanned row. SEGMENT, RID, and TID identify the row pointed to by the scan, but nothing is returned in DOMAINS. Also, an automatic hold is acquired on the last row, and the previous hold is released. No user-controlled hold is accepted in this condition. A warning is raised in the return code.

For a scan opened on a binary link, returned row is located by DBSS using SCANID and, optionally, SARGS. NSARGS must be 0 if SARGS is not to be used. The returned row (if any) is always a child because QUALF='H' is not accepted if the present row is a parent. The DBSPACE, Table ID, and TID fields of the child are returned in SEGMENT, RID,

TID, and DOMAINS. The DBSPACE, Table ID, and TID of the parent are returned in PSEGMENT, PRID, and PTID. The link ID on which the scan is opened is returned in LID, and TID is set to 0.

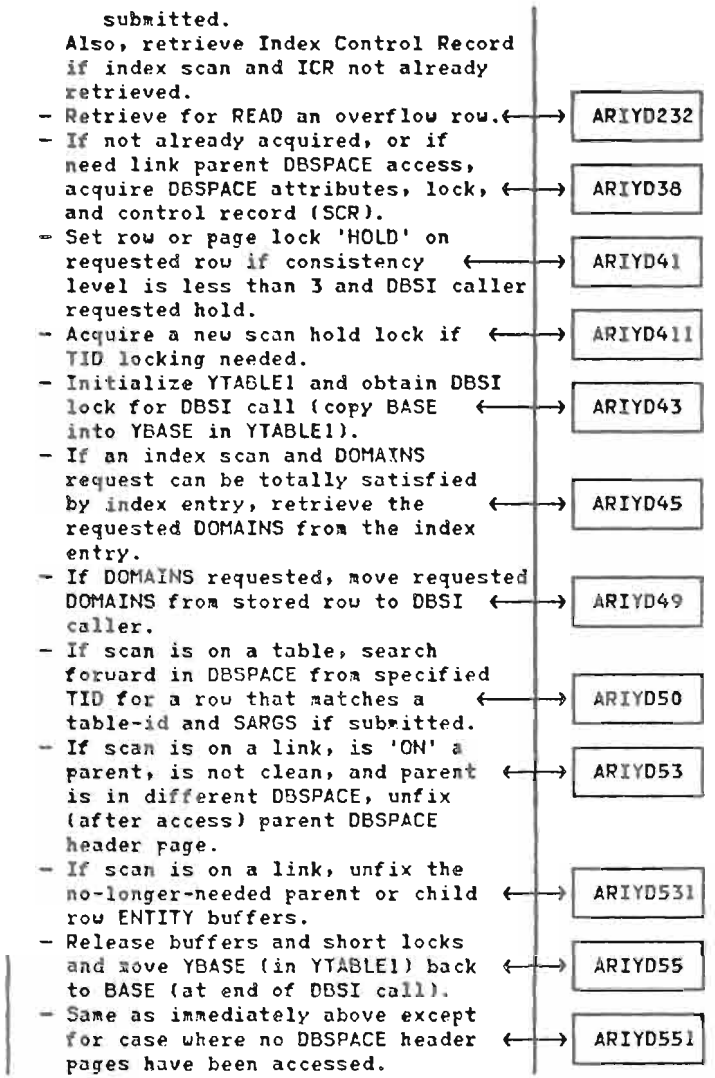
The definition of returned row, the rules for hold and for end of scanned set are similar to those given for the case of an index scan that does not use a key qualification. If the last row is a parent, then TID=0 is returned.

For a scan opened on a unary link, the definition of NEXT differs from the binary link case only in that 0's are always returned in PSEGMENT, PRID, and PTID.

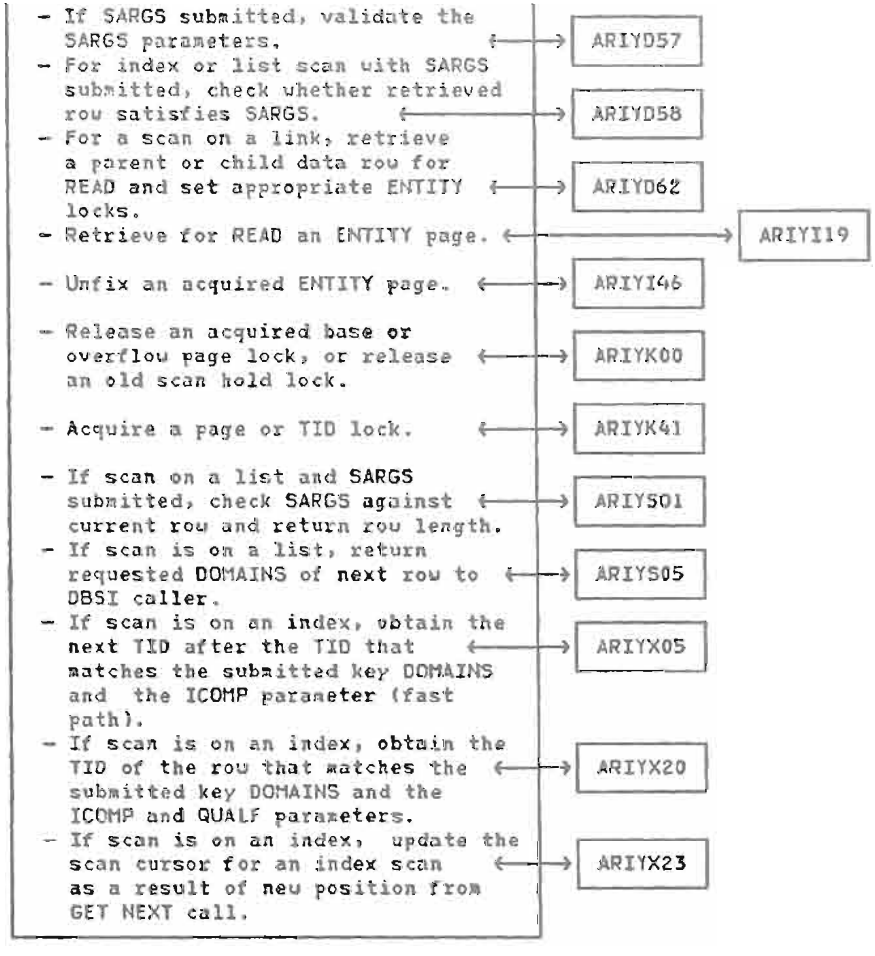
For a scan on a table, the definition of NEXT differs from the unary link case only in that 0's are returned in LID and IID.

For a scan on a sequential list, the definition of NEXT differs from the unary link case only in that 0's are returned in RID, IID, and LID.

- The following modules may be called for the following reasons:
  - If the DBSI call BACKUP occurs: ←→ ARIYD11
    a. Unfix DBSI call held buffers and release acquired DBSI locks.
    b. Initiate LUW rollback if needed.
    c. Determine if the DBSI call should be retried.
  - If DOMAINS requested, validate DOMAINS request values. ←→ ARIYD13
  - If scan is on a table, search for 'NEXT ROW' in current DBSPACE ENTITY page. ←→ ARIYD20
  - Retrieve Domain Control Record if not already retrieved and: ←→ ARIYD23
    a. this is a link scan and requested DOMAINS or SARGS submitted, or
    b. this is an index scan and SARGS or key DOMAINS submitted or requested DOMAINS, or
    c. this is a list scan, or
    d. this is a table scan and requested DOMAINS or SARGS



DBSS Data Manipulation (continued)



OPEN SCAN on a Table

* Called from ARIYM00
  ARIYD08 (OPEN SCAN)

```
ARIYD08 performs the DBSI call to
open a scan on an index or link of a
table, or a scan on a table, or a scan
on a sequential list.  OPEN SCAN also
returns the DOMAINS of the row on
which the opened scan is positioned as
specified in the DBSI caller DOMAINS
structure (unless search arguments
result in a BADSARG warning return
code).

For open of an index scan:

  A scan is opened, pointing 'TO' a
  row in the index ordering, and a
  scan-id is returned in SCANID. The
  index on which the scan is to be
  opened is identified by providing
  DBSPACE, IID, and by setting
  QUALF='I'.  The row can be identified
  by providing:
  a. The ID of an active and positioned
     'ON' scan in SCANID, or
  b. SCANID=0, RID, and a legal TID, or
  c. SCANID=0, TID=0, RID, ICOMP, and
     a key-value for IID in KDOMAINS
     (key-value not required if
     ICOMP=FIRST).
  RID and TID are always returned.

For open of a binary link scan:

  A scan is created that points 'TO' a
  child in a set of linked rows, and a
  scan-id is returned in SCANID.  The
  link is identified by providing
  SEGMENT and LID, and by setting
  QUALF='P', 'F', or 'C', depending on
  whether the link is opened on a
  parent, first child, or arbitrary
  child.

  A scan can be opened on a parent by
  providing:
   a. The ID of an active and
      positioned 'ON' scan in SCANID,
```

pointing to the desired parent row, or

b. SCANID=0, PSEGMENT, PRID, and a legal PTID.

The parent description is returned in PSEGMENT, PRID, and PTID.

A scan be be opened on a first child by providing:

a. The ID of an active and positioned 'ON' scan in SCANID, pointing to the parent row, or

b. SCANID=0, PSEGMENT, PRID, and a legal PTID.

The parent description is returned in PSEGMENT, PRID, and PTID. The table ID and row ID of the child are returned in RID and TID.

A scan can be opened on an arbitrary child by providing:

a. The ID of an active and positioned 'ON' scan in SCANID, pointing to the desired child, or

b. SCANID=0, RID, and a legal TID.

The child description is returned in RID and TID. The DBSPACE, table ID, and TID of the parent are returned in PSEGMENT, PRID, and PTID.

For open of a unary link scan:

The rules are the same as in the case of opening a binary link on a child, with the only difference being that 0's are returned in PSEGMENT, PRID, and PRID.

For open of a scan on a table:

A scan is created that points 'TO' a row in the table, and a scan ID is returned in SCANID. One must specify DBSPACE and set QUALF='R'. The row can be identified by providing:

a. The ID of an active and positioned 'ON' scan in SCANID, or

b. SCANID=0, RID, and a legal TID, or

c. SCANID=0, TID=0, and RID. In this case, the scan is opened on

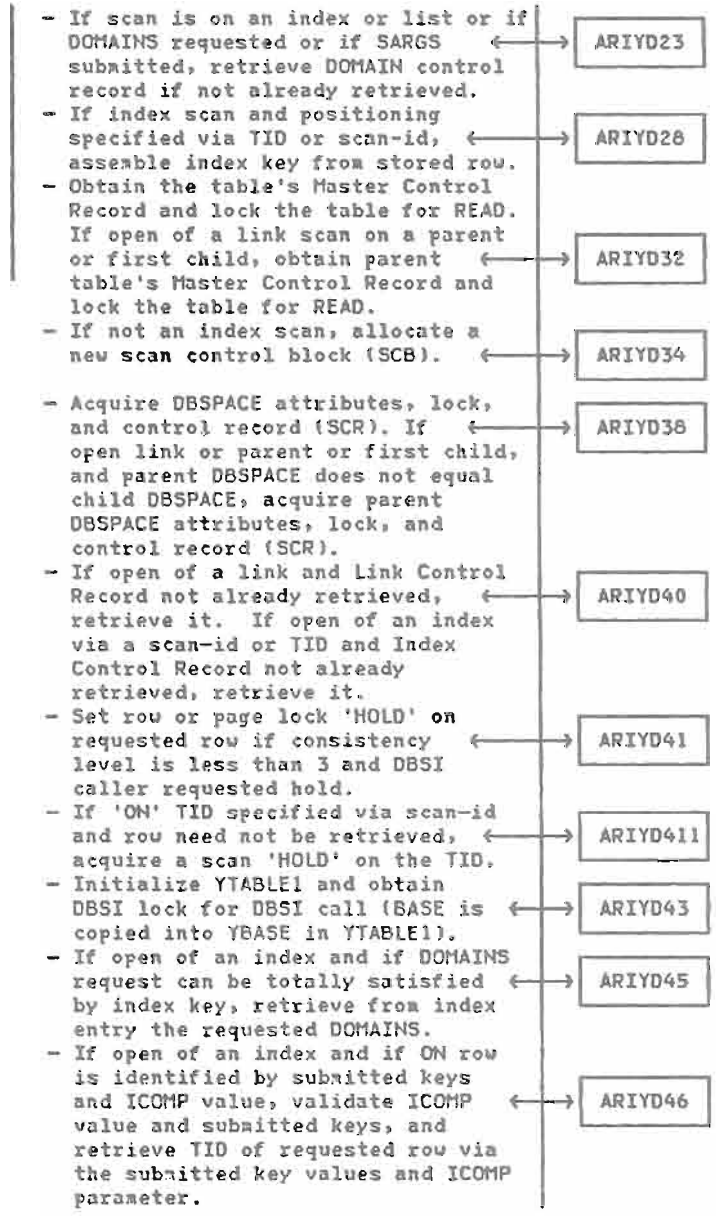
the first row of the table, in TID order.
The row description is returned in RID and TID.

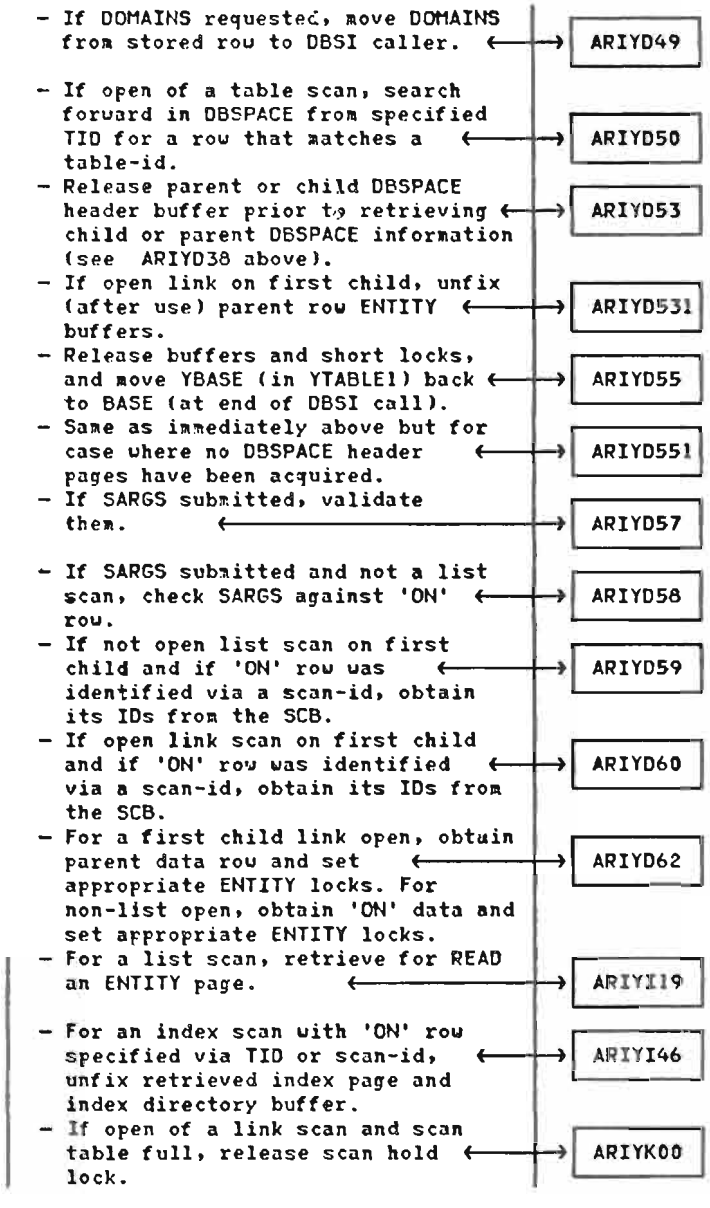
For open of a scan on a sequential list:

A scan is created that points 'TO' a row in the list, and a scan ID is returned in SCANID (in BASE). One must specify DBSPACE and set QUALF='L'. The row can be identified by providing:
a. The ID of an active and positioned 'ON' list scan in SCANID, or
b. SCANID=0. In this case, the scan is opened on the first row of the list, in physical order.
RID = 0 is returned.

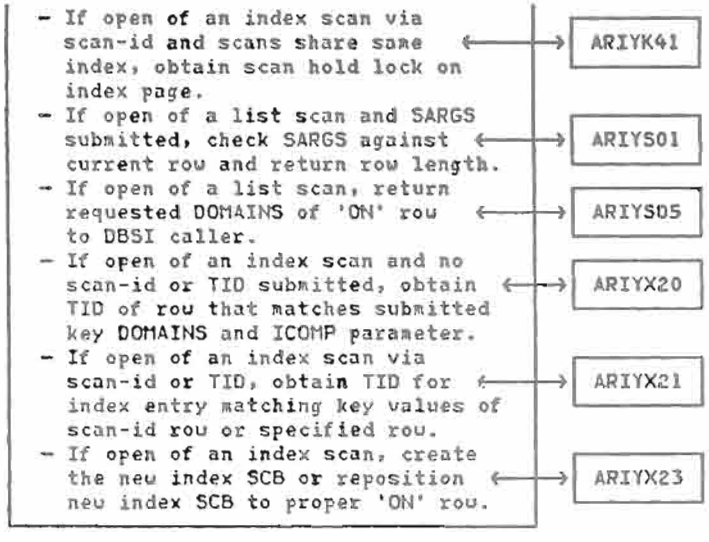
For all types of open, NSARGS > 0 causes the DBSS to check the search arguments against the resulting row. The BADSARG warning code is raised if the row does not qualify, and no DOMAINS are returned. However, the scan is still opened. An automatic hold on the row is acquired. That hold is automatically released when the scan moves to another row. In addition, a user-controlled hold can be acquired on a link, index, or table scan by setting HOLDIND='H'.

* The following modules may be called for the following reasons:
- If the DBSI call BACKUP is occurring:
  a. Unfix DBSI call held buffers and release acquired DBSI locks.
  b. Initiate LUW ROLLBACK if required.
  c. Determine if the DBSI call should be retried.
  ARIYD11
- If any DOMAINS requested, validate DOMAIN request values.
  ARIYD13


- If scan is on an index or list or if DOMAINS requested or if SARGS submitted, retrieve DOMAIN control record if not already retrieved.
ARIYD23
- If index scan and positioning specified via TID or scan-id, assemble index key from stored row.
ARIYD28
- Obtain the table's Master Control Record and lock the table for READ. If open of a link scan on a parent or first child, obtain parent table's Master Control Record and lock the table for READ.
ARIYD32
- If not an index scan, allocate a new scan control block (SCB).
ARIYD34
- Acquire DBSPACE attributes, lock, and control record (SCR). If open link or parent or first child, and parent DBSPACE does not equal child DBSPACE, acquire parent DBSPACE attributes, lock, and control record (SCR).
ARIYD38
- If open of a link and Link Control Record not already retrieved, retrieve it. If open of an index via a scan-id or TID and Index Control Record not already retrieved, retrieve it.
ARIYD40
- Set row or page lock 'HOLD' on requested row if consistency level is less than 3 and DBSI caller requested hold.
ARIYD41
- If 'ON' TID specified via scan-id and row need not be retrieved, acquire a scan 'HOLD' on the TID.
ARIYD411
- Initialize YTABLE1 and obtain DBSI lock for DBSI call (BASE is copied into YBASE in YTABLE1).
ARIYD43
- If open of an index and if DOMAINS request can be totally satisfied by index key, retrieve from index entry the requested DOMAINS.
ARIYD45
- If open of an index and if ON row is identified by submitted keys and ICOMP value, validate ICOMP value and submitted keys, and retrieve TID of requested row via the submitted key values and ICOMP parameter.
ARIYD46



DBSS Data Manipulation (continued)



RETRIEVE PARENT ROW in a Binary Link

* Called from ARIYM00
ARIYD09 (GET PARENT ROW)

ARIYD09 performs the DBSI call to retrieve parent row of a submitted child row in a specified binary link.

The link is identifed by providing DBSPACE and LID. The child row is identified by providing:
- An ID of an active and positioned 'ON' scan in SCANID, or
- SCANID=0, DBSPACE, RID and a legal TID.

The child row DBSPACE, table-id, and TID are returned in SEGMENT, RID, and TID. The parent row description is returned in PSEGMENT, PRID, PTID and DOMAINS.

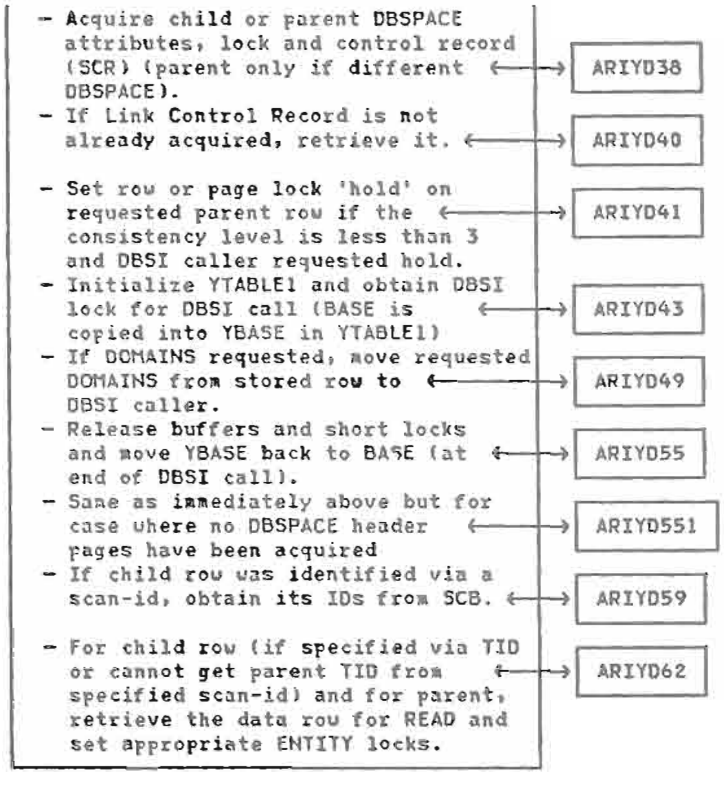
Note that search arguments cannot be used (will be ignored) on a GET PARENT call.

A user-controlled hold on the parent row can be acquired by setting HOLDIND = 'H'.

The following modules may be called (not necessarily in the order shown):

- If DBSI-call backup is occurring: ⟷ ARIYD11
  a. Unfix DBSI call held buffers and release acquired DBSI locks.
  b. Initiate LUW rollback if required.
  c. Determine if the DBSI call should be retried.
- If any DOMAINS requested, validate DOMAIN request values. ⟷ ARIYD13
- If any DOMAINS requested or SARGS submitted, retrieve Domain Control Record unless already retrieved. ⟷ ARIYD23
- Obtain the child row or parent row's table Master Control Record and lock the table for read. ⟷ ARIYD32

DBSS Data Manipulation (continued)



UPDATE ROW DOMAINS in a Table

* Called from ARIYM00
ARIYD10 (UPDATE ROW)

ARIYD10 performs the DBSI call to update some of the DOMAINS of a row of a table.

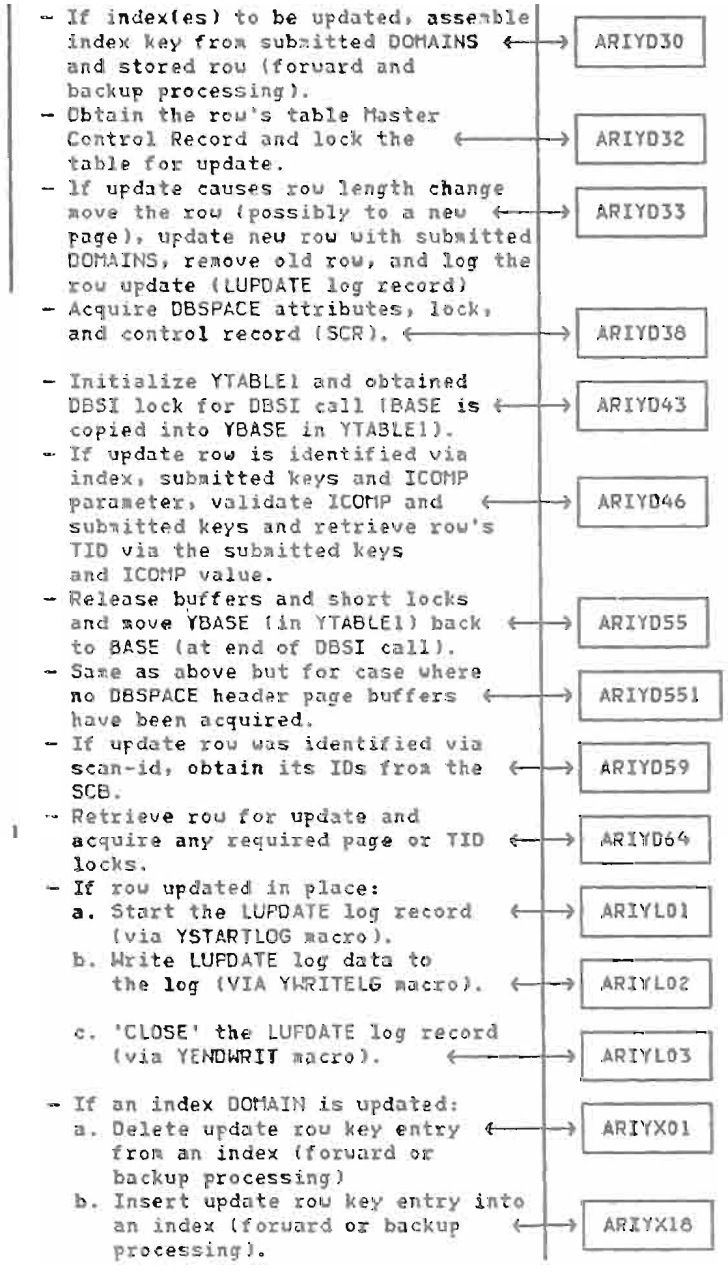
The row to be updated can be identified by providing:
- The ID of an active and positioned 'ON' scan in SCANID, or
- SCANID=0, DBSPACE, RID and a legal TID, or
- SCANID=0, DBSPACE, RID, TID=0, IID, ICOMP and a key value in KDOMAINS (key value not required if ICOMP = FIRST).

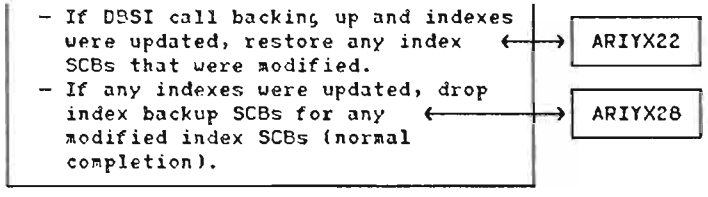
Non-negative numbers in FREQLTH entries are used to identify the DOMAINS that are being updated. The lengths for these DOMAINS are submitted in FACLTH. The correct lengths must be specified for the submitted fixed-length DOMAINS.

DBSPACE, table-id, and TID of the updated row are returned in SEGMENT, RID and TID.

The following modules may be called (not necessarily in the order shown):
- If DBSI-call backup is occurring:
  a. Unfix DBSI call held buffers and release acquired DBSI locks. ⟷ ARIYD11
  b. Initiate LUW rollback if required.
  c. Determine if the DBSI call should be retried.
- If DOMAINS submitted, retrieve Domain Control Record if not already retrieved. If index(es) are to be updated, retrieve each Index Control Record (forward or backup processing). ⟷ ARIYD23
- If index(es) to be updated, assemble index key from stored row (forward or backup processing). ⟷ ARIYD28

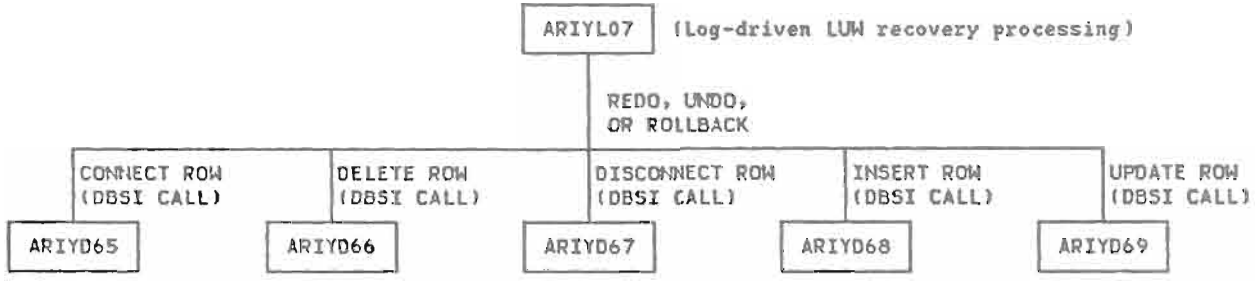
DBSS Data Manipulation (continued)





DBSS/DM RECOVERY PROCESSING - GENERAL FLOW

In this diagram, only the DBSS/DM recovery processing top modules are shown. See the following detail diagrams for other modules being called.



For:

- CONNECT ROW (ARIYD65), see page 338.
- DELETE ROW (ARIYD66), see page 339.
- DISCONNECT ROW (ARIYD67), see page 340.
- INSERT ROW (ARIYD68), see page 341.
- UPDATE ROW (ARIYD69), see page 342.

DBSS/DM RECOVERY PROCESSING - DETAIL FLOW

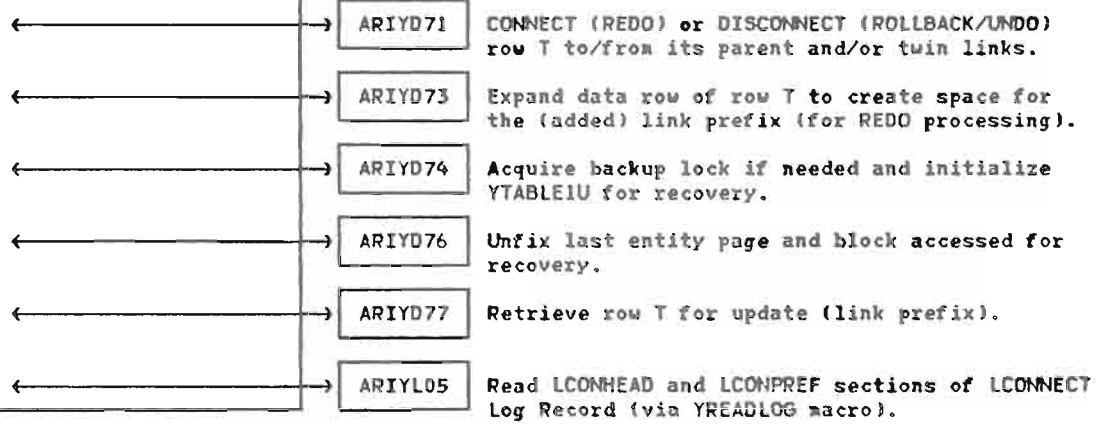
REDO/UNDO/ROLLBACK CONNECT ROW Operation (Recovery Processing)

* Called from ARIYL07
ARIYD65 (CONNECT ROW)

ARIYD65 connects row T in a REDO, or disconnects row T in UNDO or ROLLBACK using logged information.

At entry to ARIYD65, LOGL is the length of the log record, and TRANSMODE is 'R', 'U', or 'B' (REDO, UNDO, or ROLLBACK). Also at entry, log cursor points to the beginning of the logged data for the CONNECT of row T.

Modules that may be called from ARIYD65 are:



REDO/UNDO/ROLLBACK Data Row DELETE Operation (Recovery Processing)

* Called from ARIYL07
ARIYD66 (DELETE ROW)

ARIYD66 re-deletes row T for REDO (recovers DELETE operation), and, if it was DELETE with DISCONNECT, it re-disconnects any parent and twin links.

It inserts row T for UNDO or ROLLBACK (recovers DELETE operation) and, if it was DELETE with DISCONNECT, connects row T to any parent and twin links.

This is done using information logged with the DELETE operation (LDELETE data). Note that row T may be an overflow row and that in this case, both the base pointer row and the overflow data row must be deleted or inserted.

At entry, LOGL has the length of the log record and TRANMODE is 'R', 'U', or 'B' (REDO, UNDO, or ROLLBACK). Also at entry, log cursor points to the beginning of the logged data for the delete of row T.

Modules that may be called from ARIYD66 are:

ARIYD52	Unfix DBSPACE header buffers fixed because of index updating.
ARIYD72	Disconnect or connect row T from/to any parent or twin links.
ARIYD74	Initialize YTABLEIU for recovery and, if needed, acquire: backup lock and SCR and MCR and DCR if indexes are to be updated.
ARIYD75	Delete or insert all index key entries associated with row T.
ARIYD76	Unfix last entity page and block accessed for recovery.
ARIYD77	Delete row T base row, delete row T overflow row, retrieve row T base page and prepare it for pointer row insert, and retrieve row T base or overflow page and prepare it for data row insert.
ARIYL05	Read parts of the LDELETE log record (via the YREADLOG macro).

REDO/UNDO/ROLLBACK DISCONNECT Row Operation (Recovery Processing)

* Called from ARIYL07
ARIYD67 (DISCONNECT ROW)

ARIYD67 disconnects row T in REDO or reconnects row T in UNDO and ROLLBACK using logged information.

At entry, LOGL has the length of the log record and TRANMODE is 'B', 'U', or 'R' (ROLLBACK, UNDO, or REDO). Also at entry, log cursor points to the beginning of the logged data for the disconnect of row T.

Modules that may be called from ARIYD67 are:

ARIYD72	Update link prefixes of row T's right twin, left twin, and parent rows (CONNECT or DISCONNECT row T).
ARIYD74	Acquire backup lock if needed and initialize YTABLE1U for recovery.
ARIYD76	Unfix last entity page and block accessed for recovery.
ARIYD77	Retrieve row T for update (link prefix).
ARIYL05	Read LDISCONN log record (via YREADLOG macro)

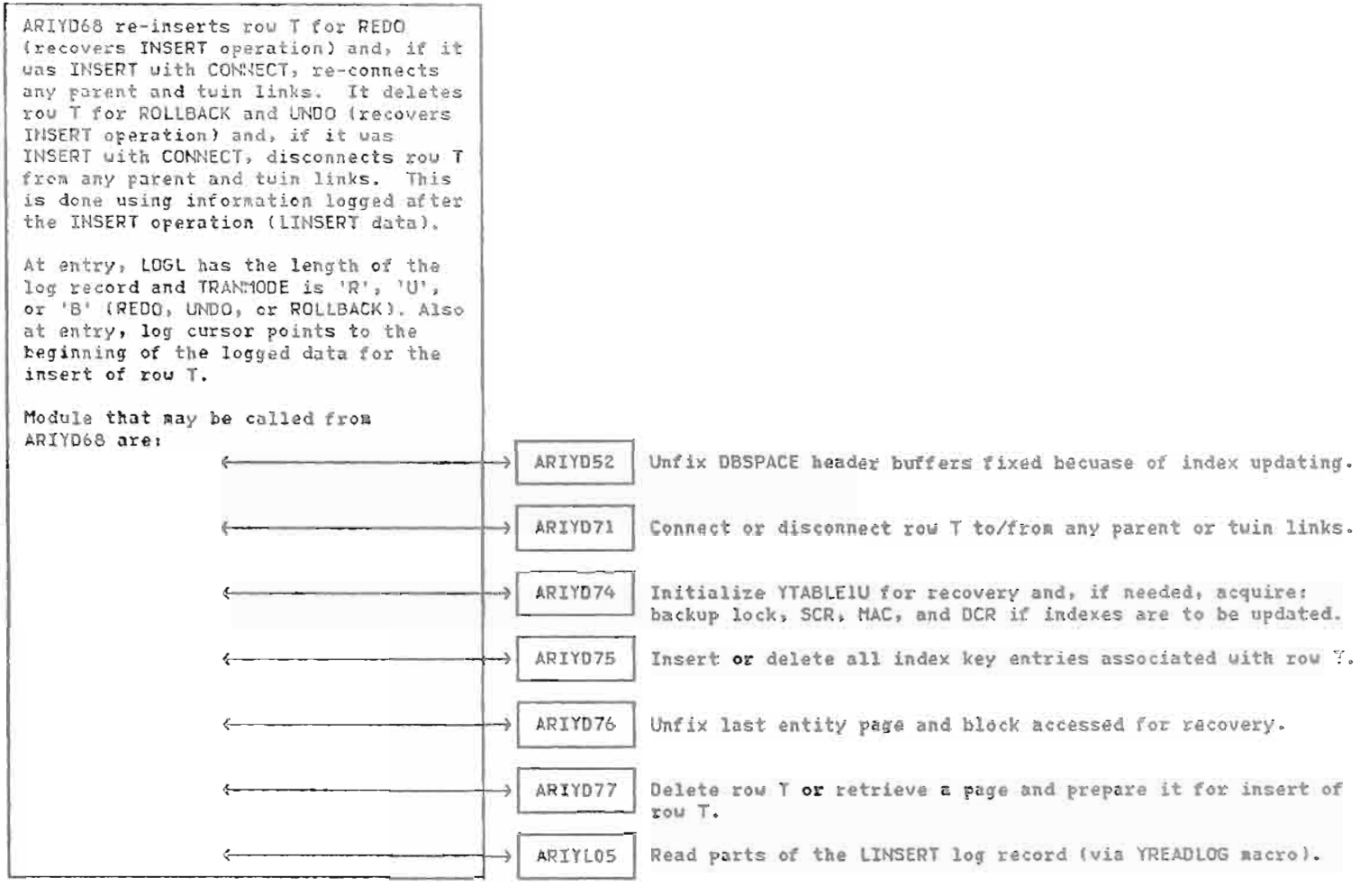
REDO/UNDO/ROLLBACK of INSERT Data Row (Recovery Processing)

* Called from ARIYL07
ARIYD68 (INSERT ROW)

ARIYD68 re-inserts row T for REDO (recovers INSERT operation) and, if it was INSERT with CONNECT, re-connects any parent and twin links. It deletes row T for ROLLBACK and UNDO (recovers INSERT operation) and, if it was INSERT with CONNECT, disconnects row T from any parent and twin links. This is done using information logged after the INSERT operation (LINSERT data).

At entry, LOGL has the length of the log record and TRANMODE is 'R', 'U', or 'B' (REDO, UNDO, or ROLLBACK). Also at entry, log cursor points to the beginning of the logged data for the insert of row T.

Module that may be called from ARIYD68 are:



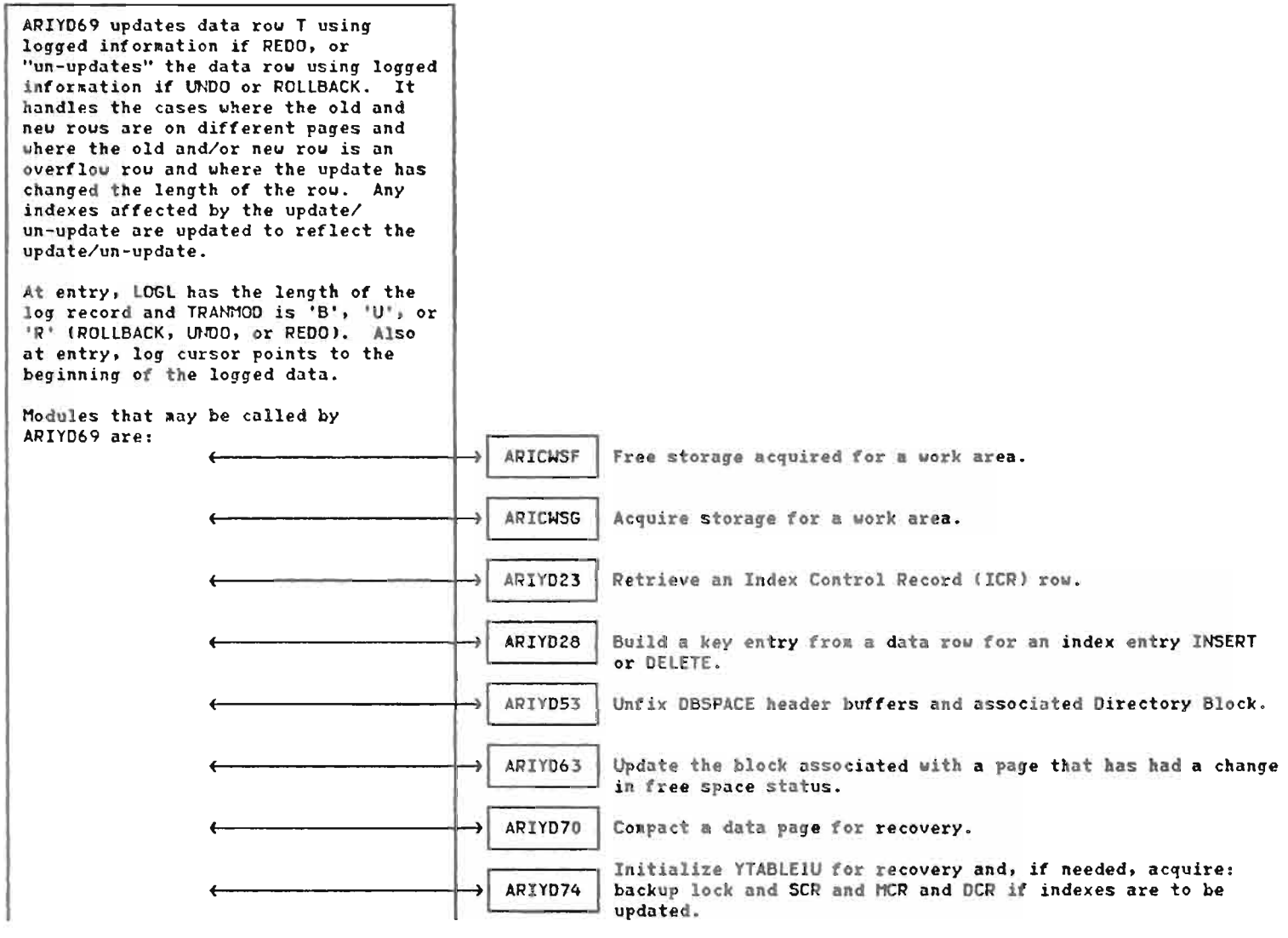
REDO/UNDO/ROLLBACK Data Row UPDATE Operation (Recovery Processing)

* Called from ARIYL07
ARIYD69 (UPDATE ROW)

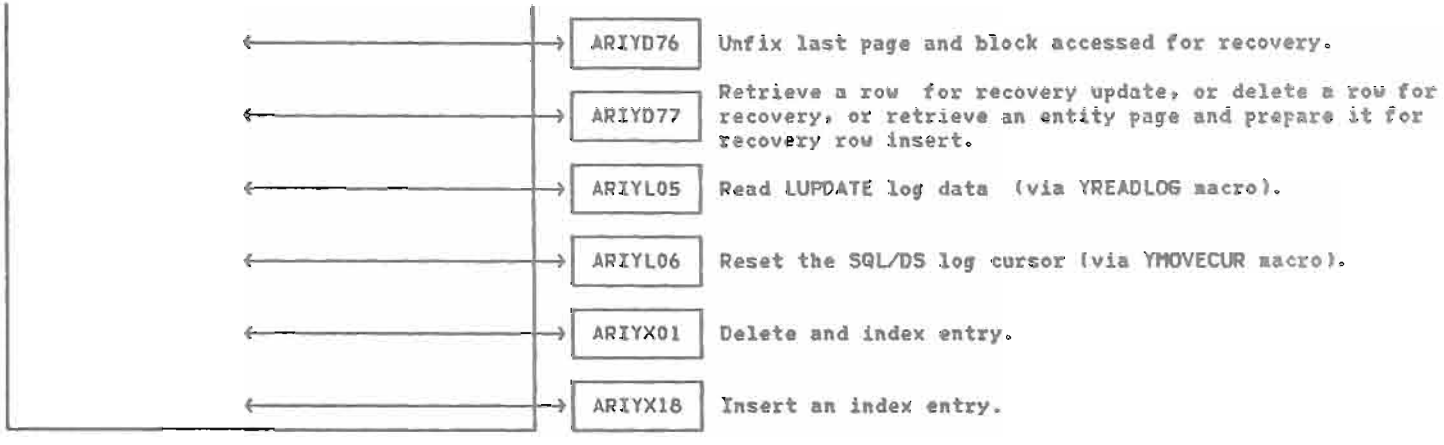
ARIYD69 updates data row T using logged information if REDO, or "un-updates" the data row using logged information if UNDO or ROLLBACK. It handles the cases where the old and new rows are on different pages and where the old and/or new row is an overflow row and where the update has changed the length of the row. Any indexes affected by the update/un-update are updated to reflect the update/un-update.

At entry, LOGL has the length of the log record and TRANMOD is 'B', 'U', or 'R' (ROLLBACK, UNDO, or REDO). Also at entry, log cursor points to the beginning of the logged data.

Modules that may be called by ARIYD69 are:

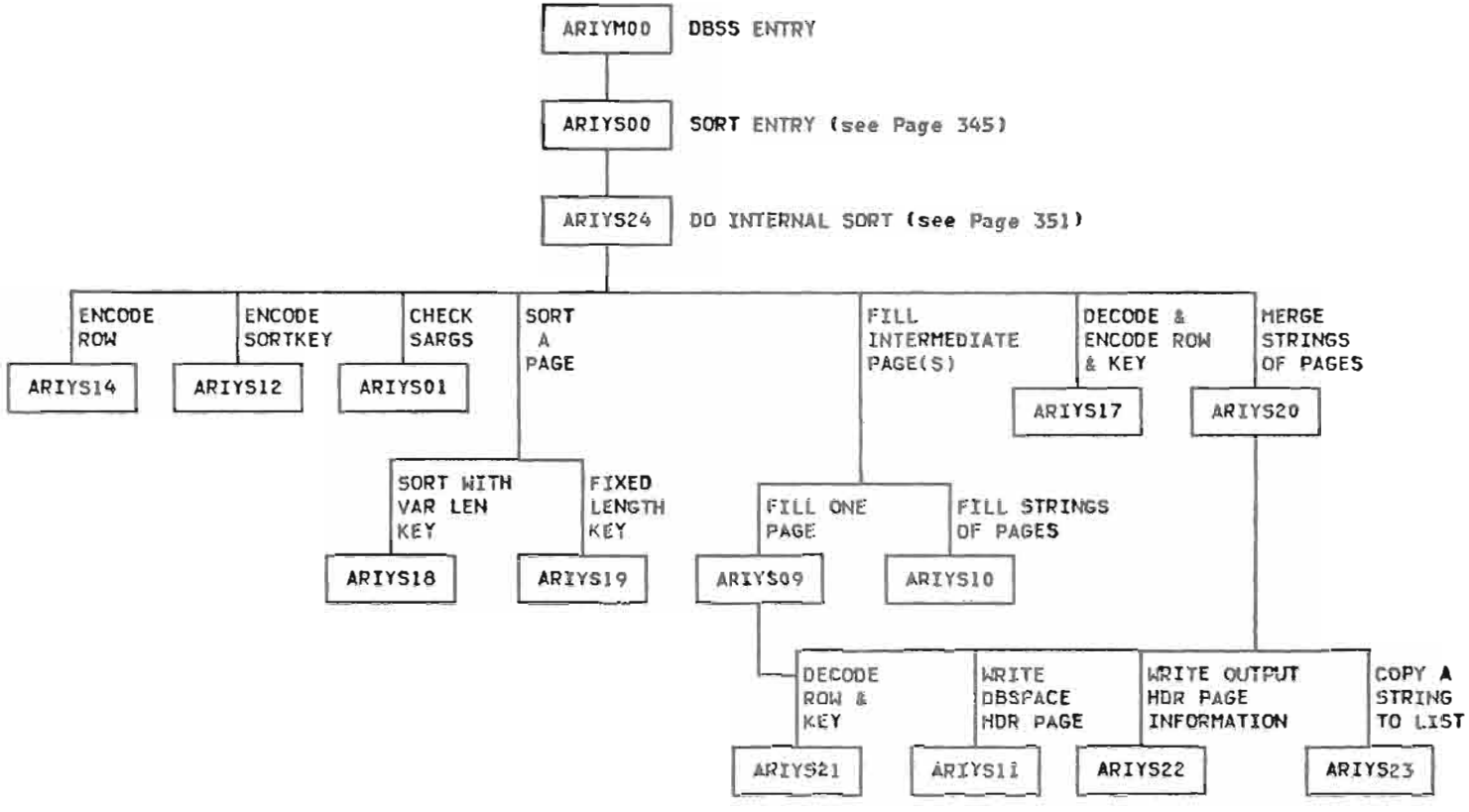


DBSS Data Manipulation (continued)

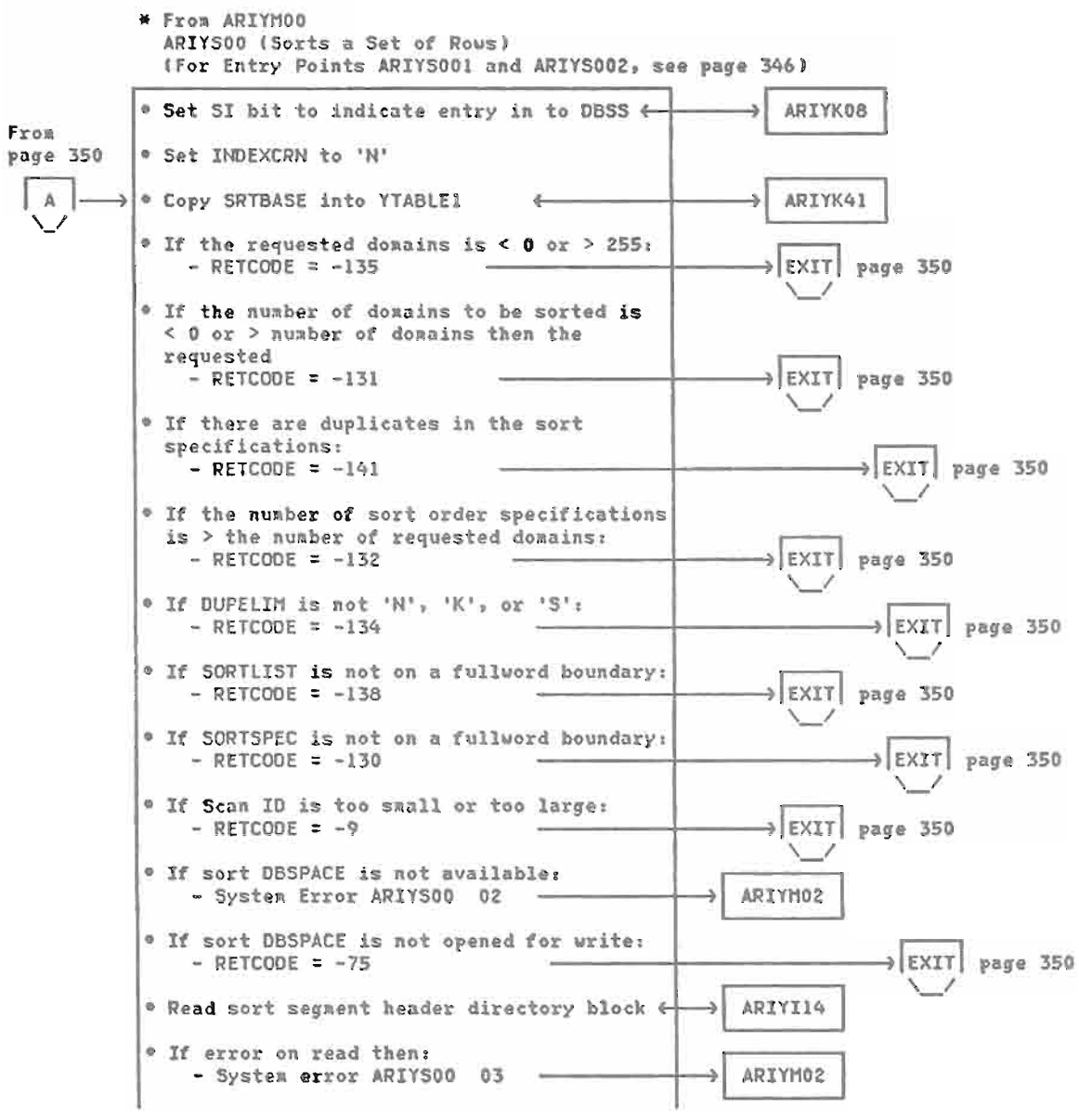


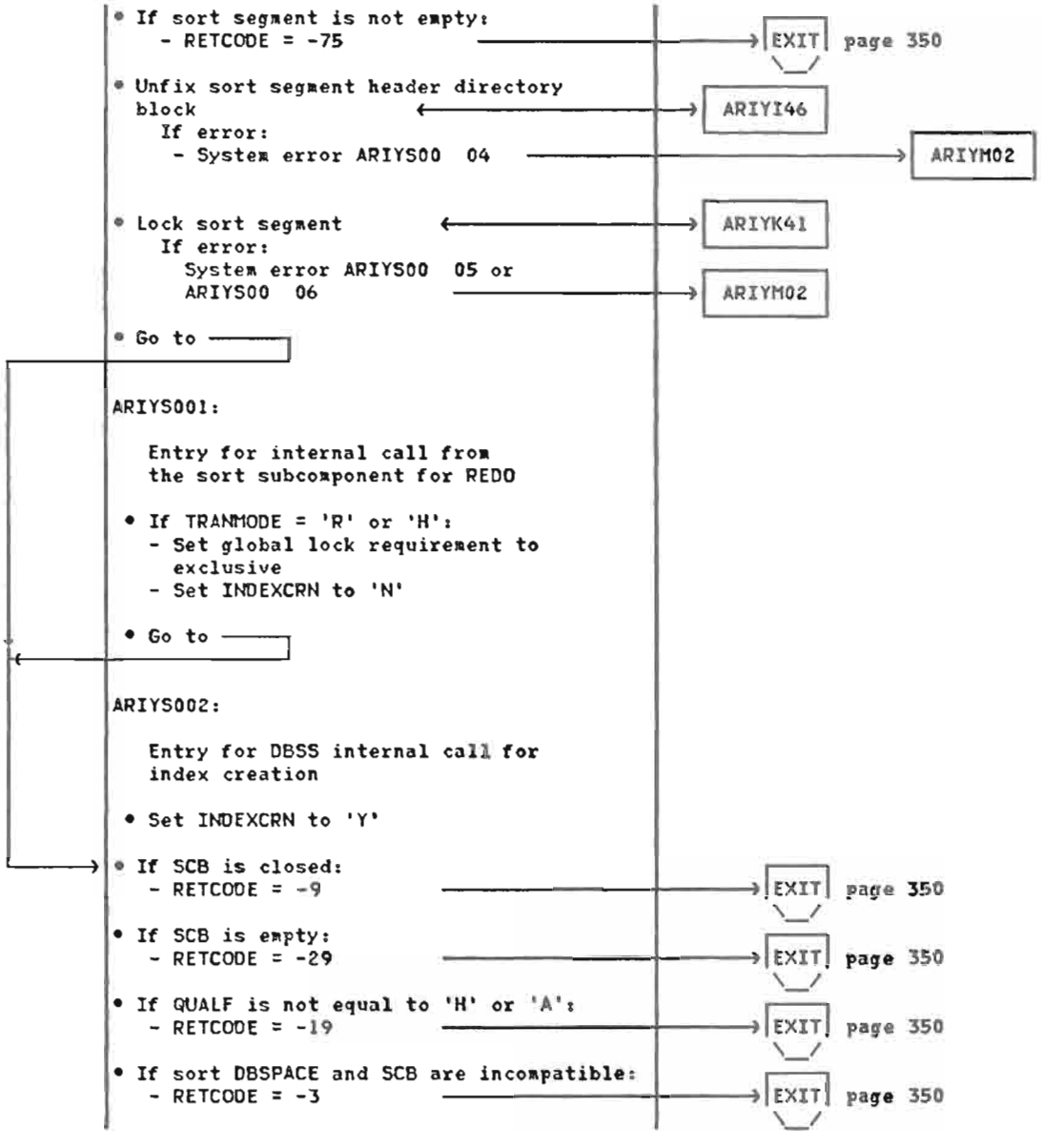
DBSS SORT - GENERAL FLOW

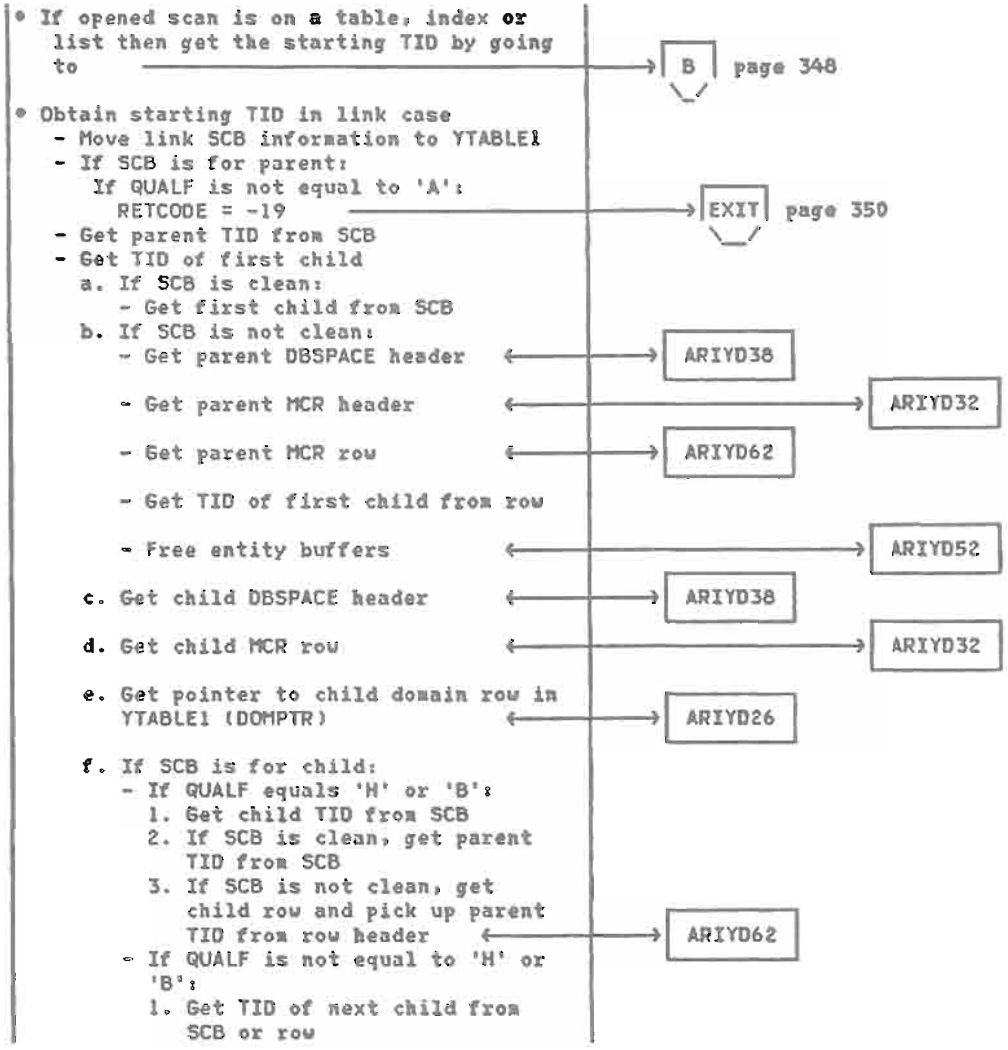
In this diagram, only the major modules of DBSS Sort are shown. For the details, see page 345.

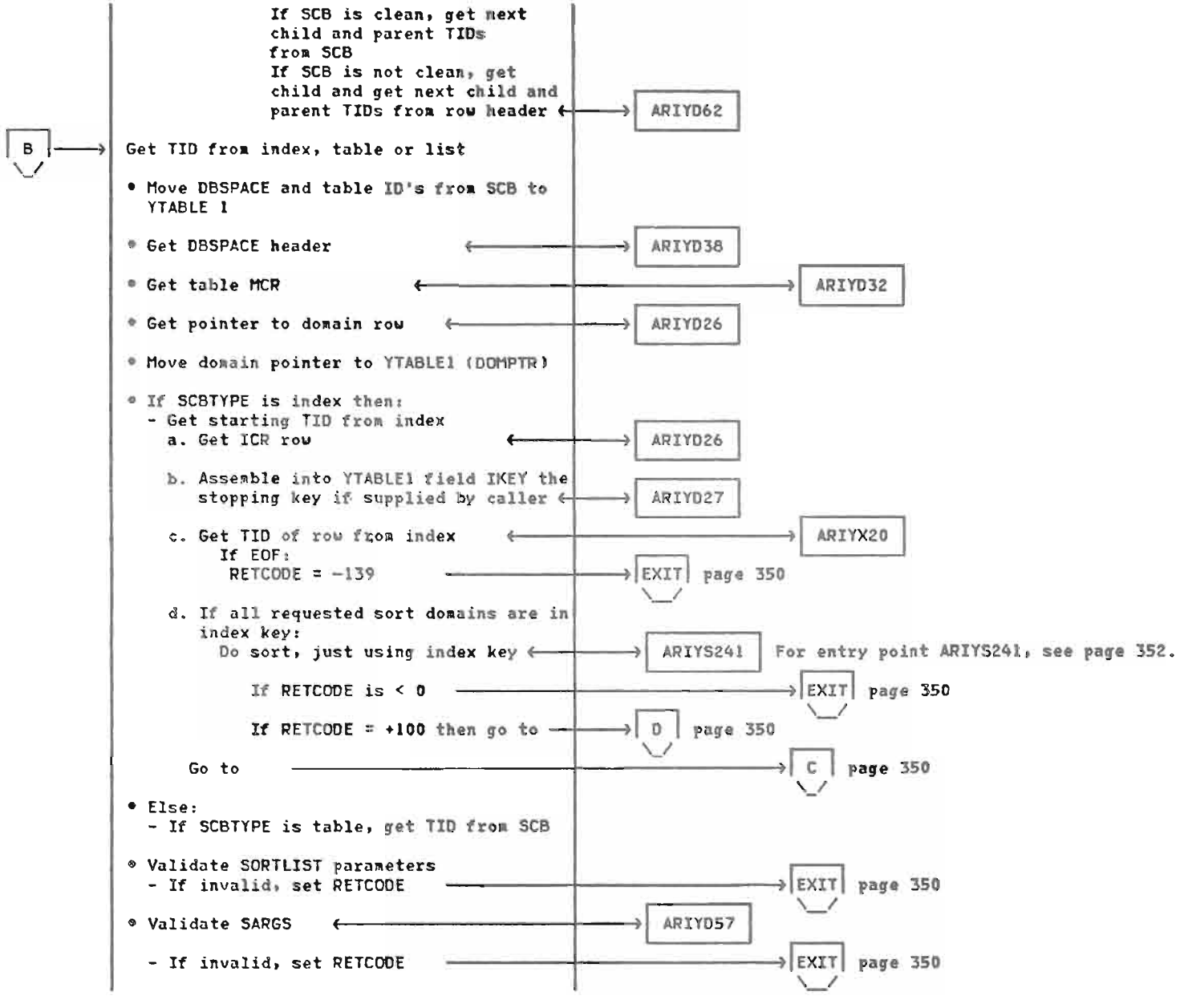


DBSS SORT - DETAIL FLOW

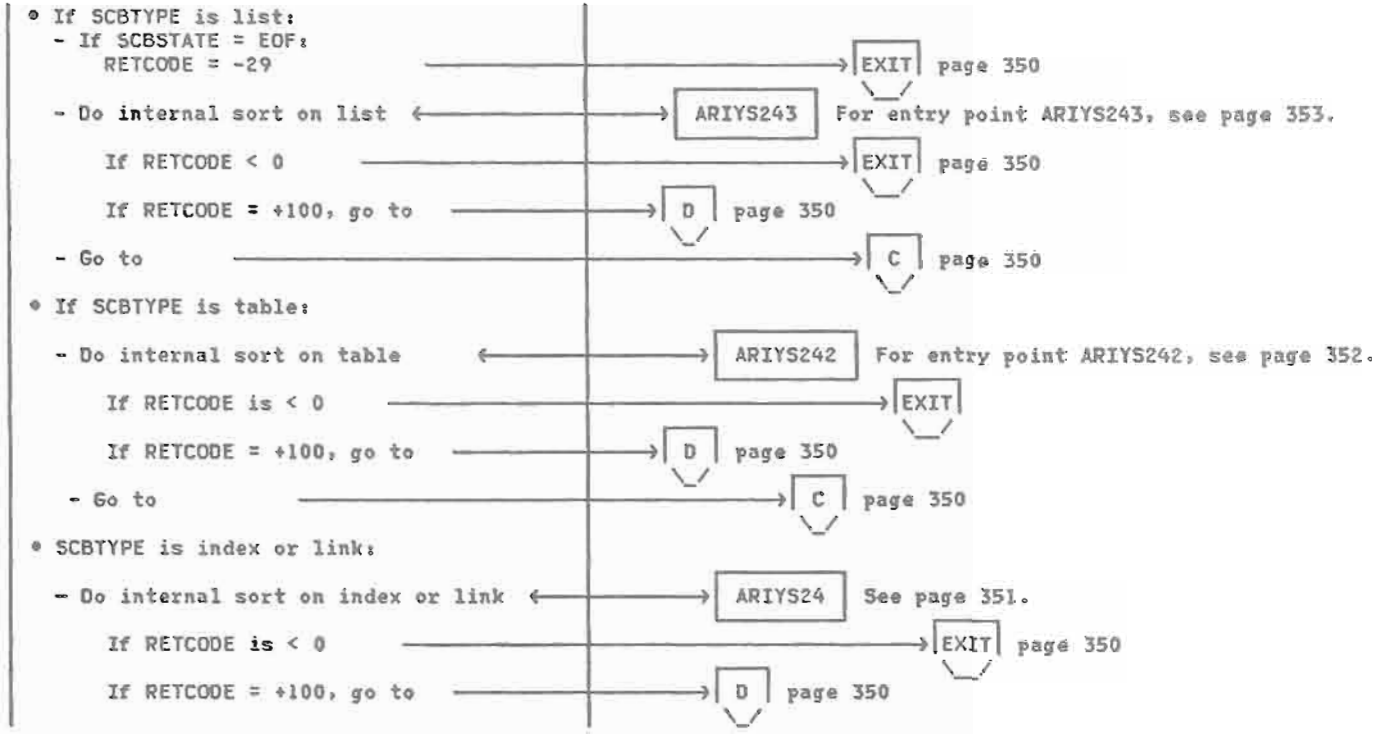


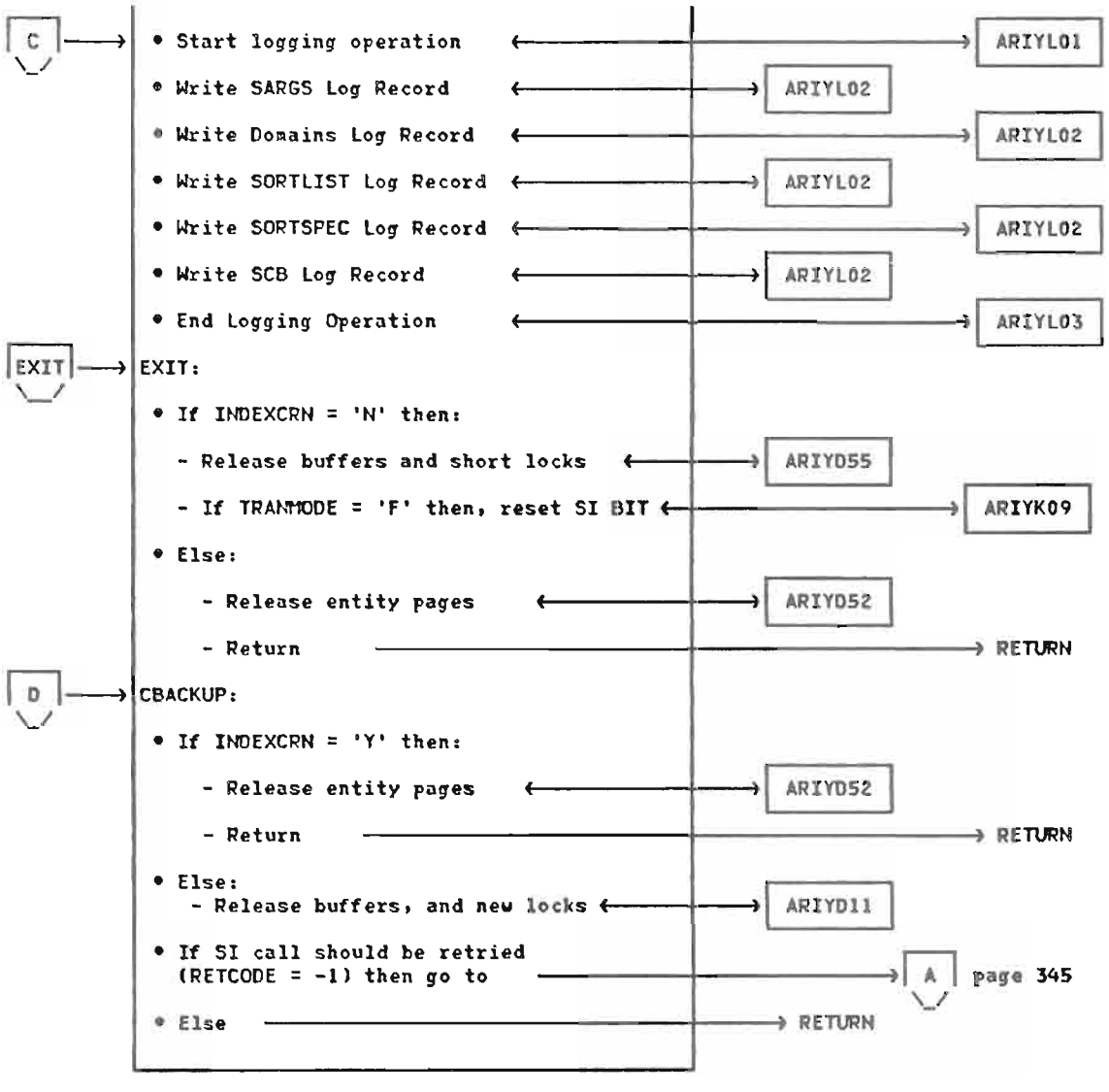



• If opened scan is on a table, index or list then get the starting TID by going to
B
page 348
• Obtain starting TID in link case
- Move link SCB information to YTABLE1
- If SCB is for parent:
If QUALF is not equal to 'A':
RETCODE = -19
EXIT
page 350
- Get parent TID from SCB
- Get TID of first child
a. If SCB is clean:
- Get first child from SCB
b. If SCB is not clean:
- Get parent DBSPACE header
ARIYD38
- Get parent MCR header
ARIYD32
- Get parent MCR row
ARIYD62
- Get TID of first child from row
- Free entity buffers
ARIYD52
c. Get child DBSPACE header
ARIYD38
d. Get child MCR row
ARIYD32
e. Get pointer to child domain row in YTABLE1 (DOMPTR)
ARIYD26
f. If SCB is for child:
- If QUALF equals 'H' or 'B':
1. Get child TID from SCB
2. If SCB is clean, get parent TID from SCB
3. If SCB is not clean, get child row and pick up parent TID from row header
ARIYD62
- If QUALF is not equal to 'H' or 'B':
1. Get TID of next child from SCB or row

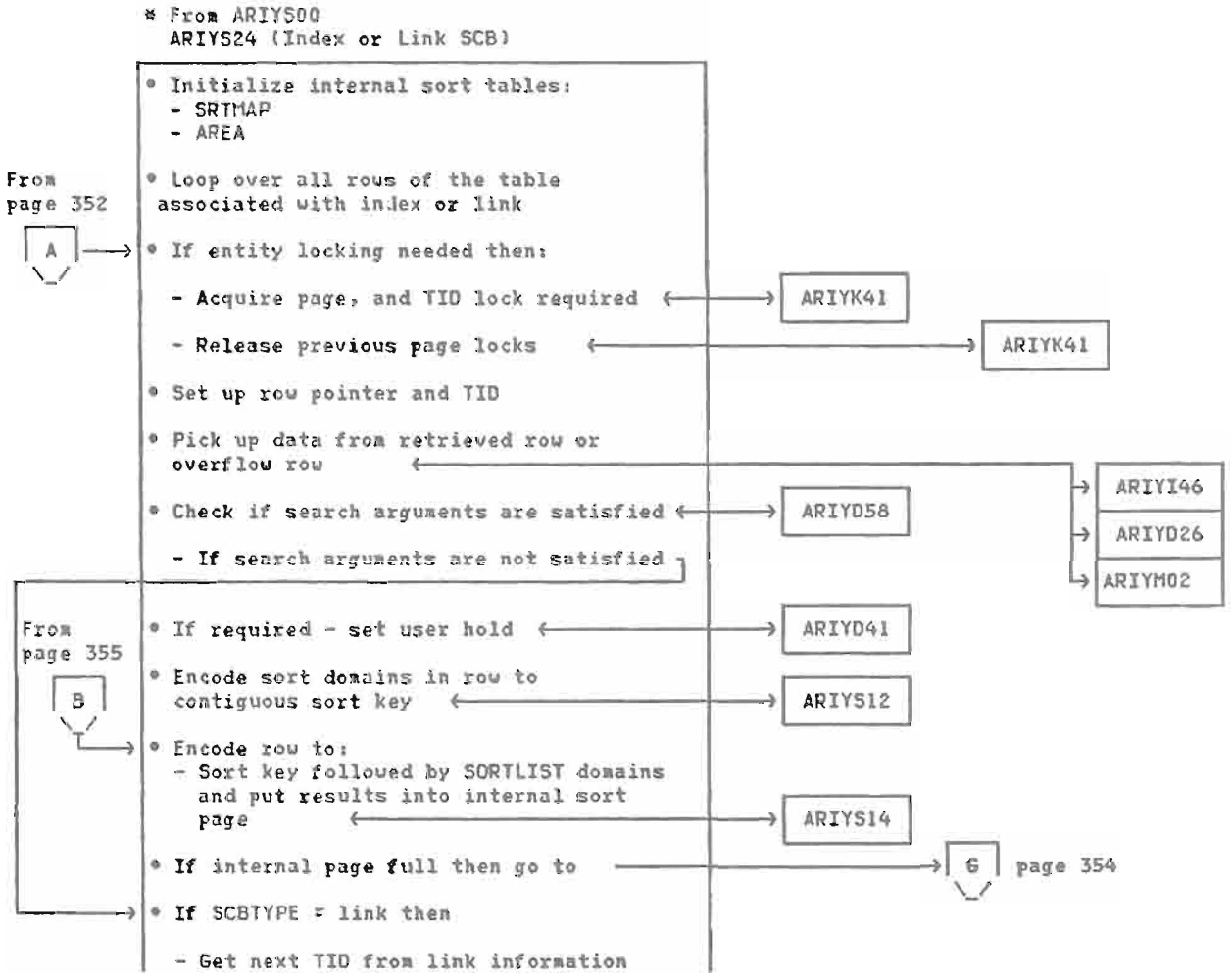


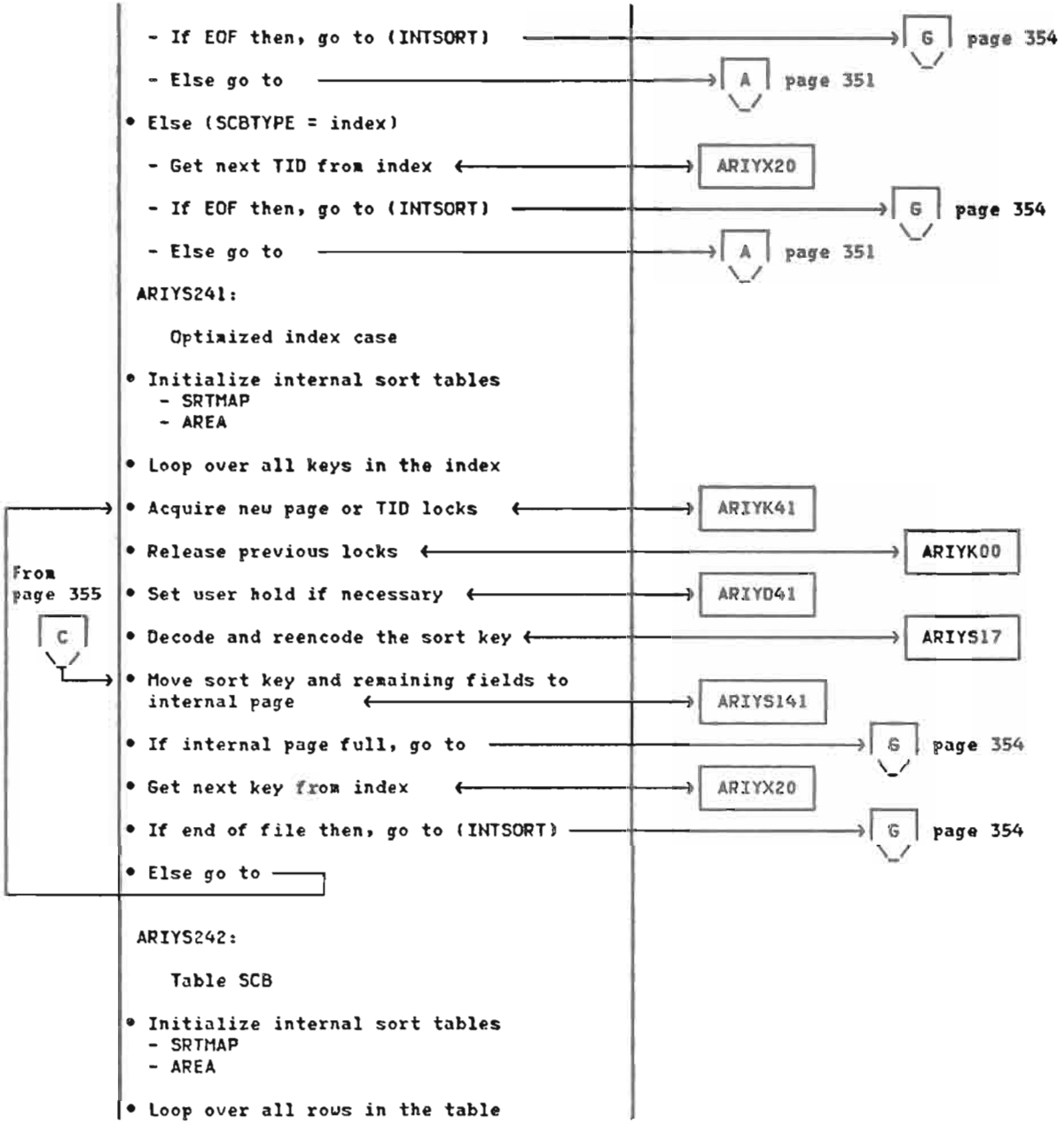
DBSS Sort (continued)

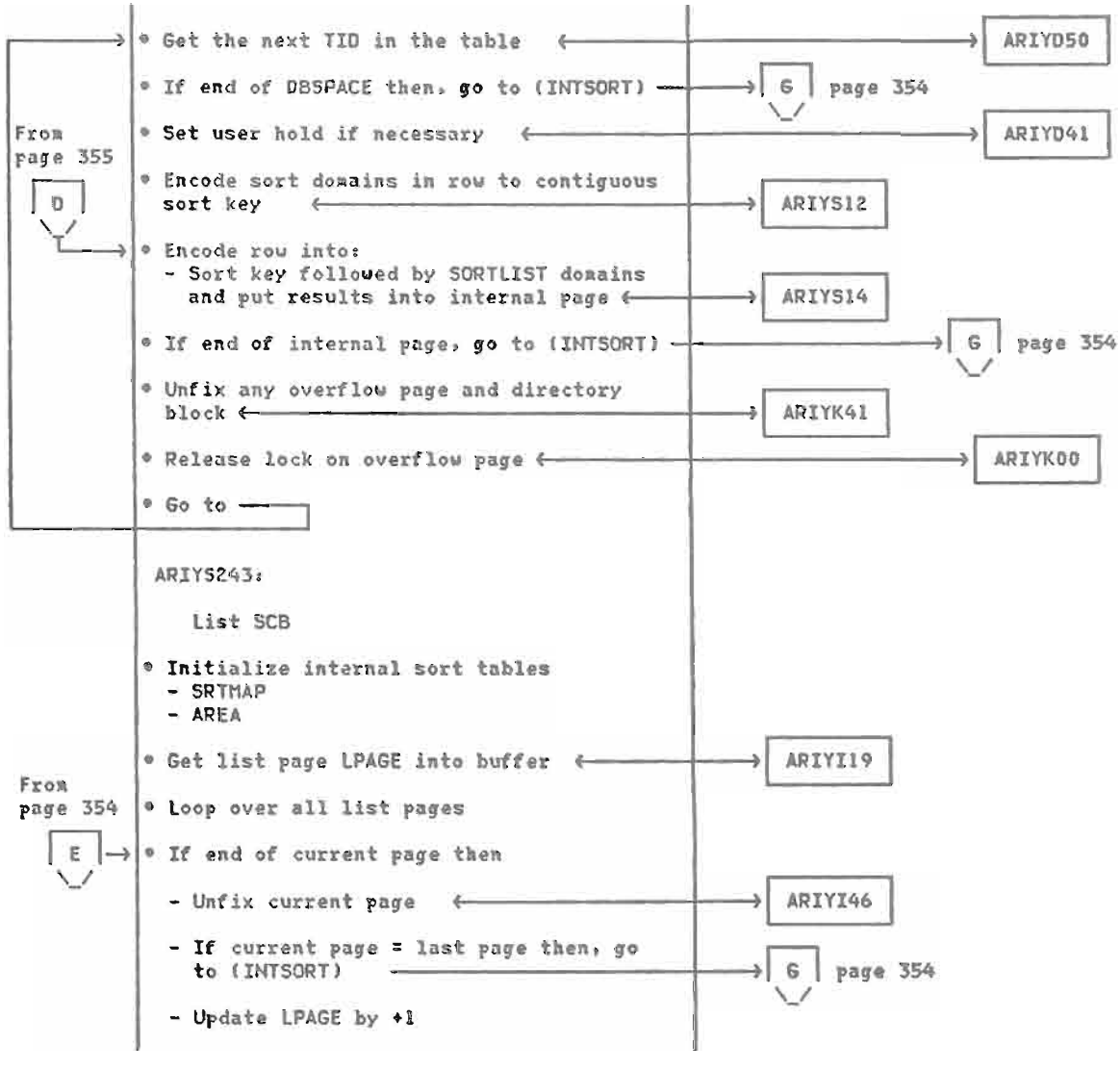


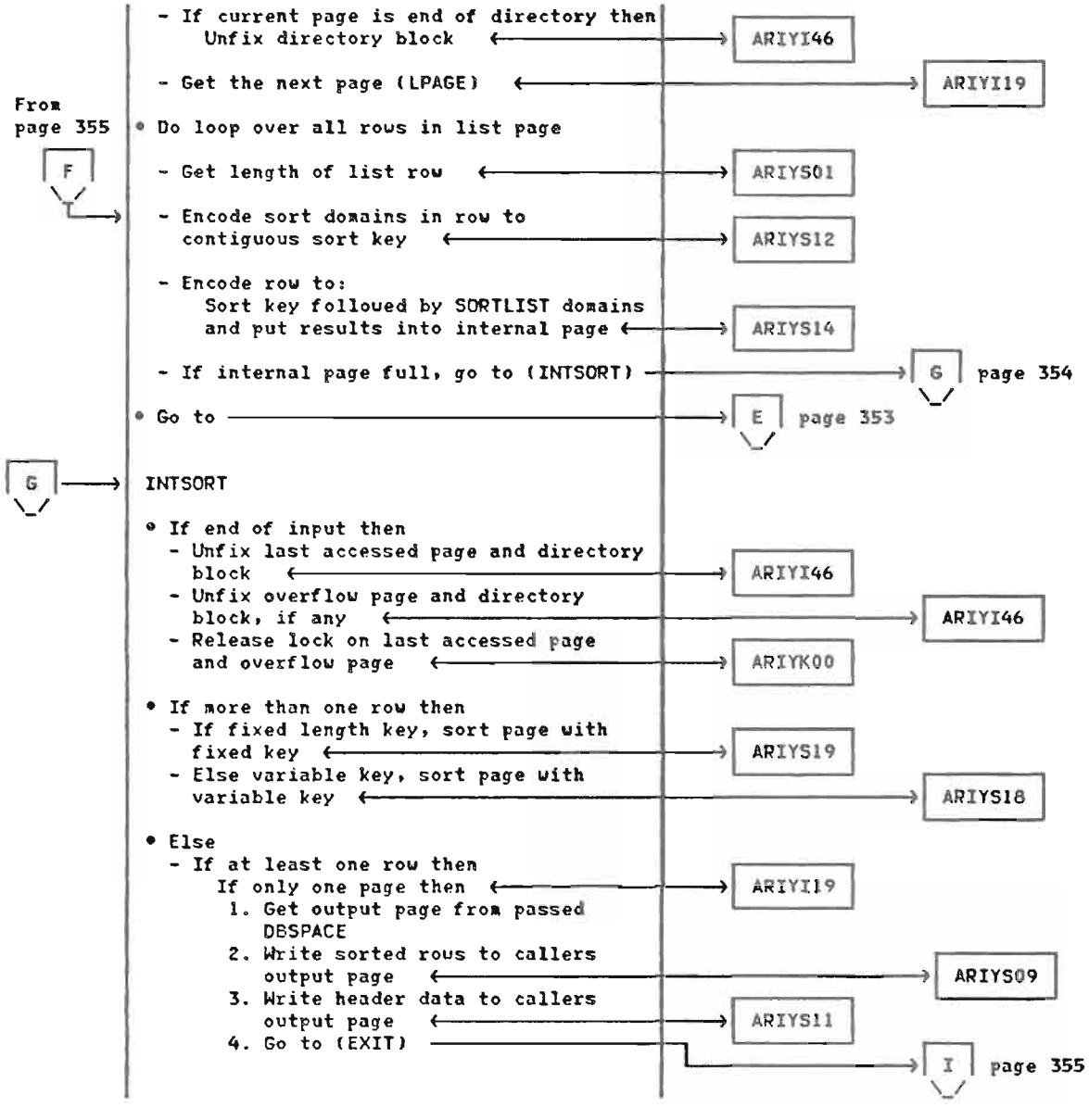


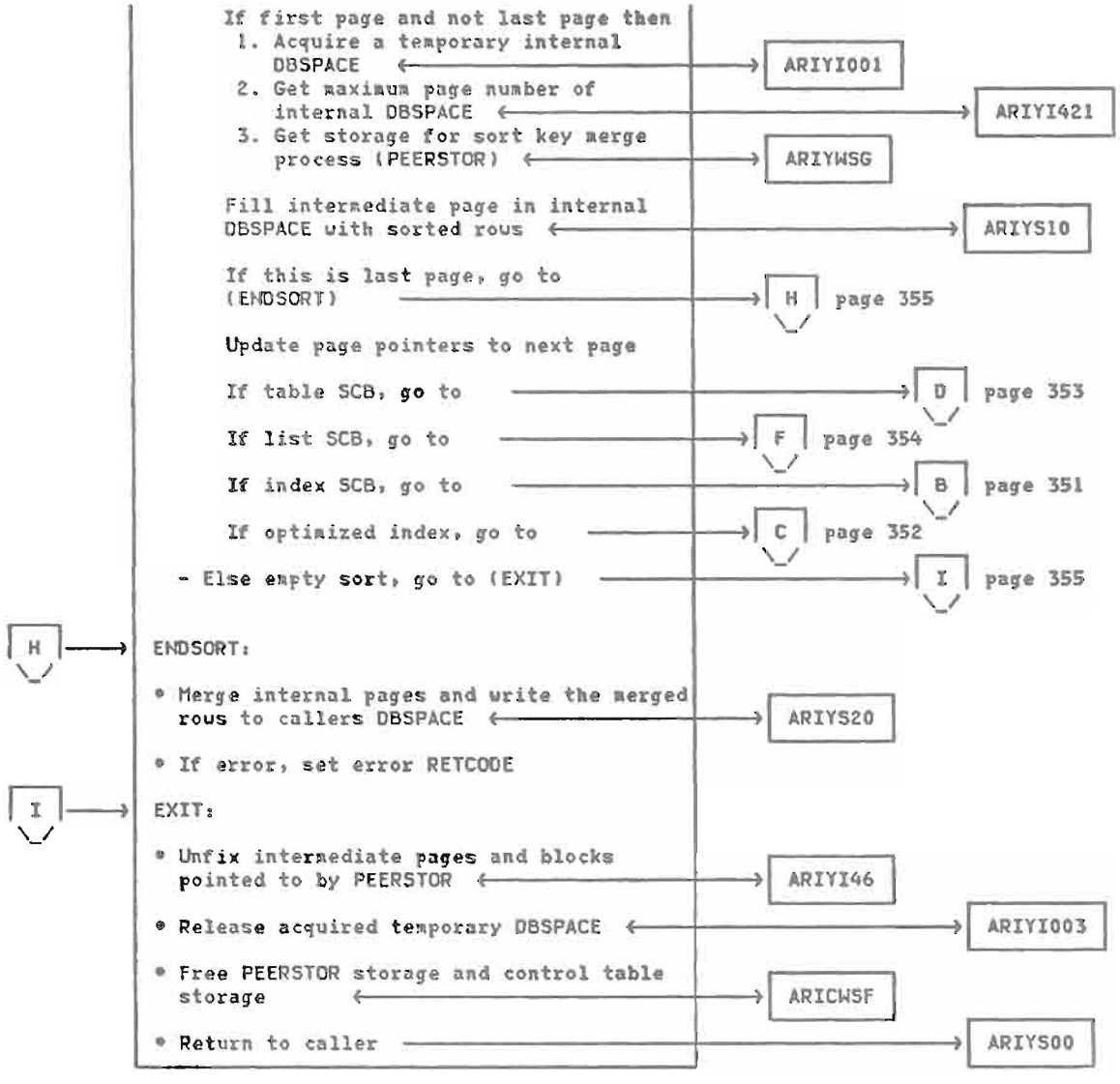
- Entry point ARIYS24 sorts rou with an index or link SCB opened.
- For entry point ARIYS241, see page 352 (ARIYS241 sorts rous with an index SCB opened and all requested domains are in index key. Optimized entry for no entity pages being touched.).
- For entry point ARIYS242, see page 352 (ARIYS242 sorts rous with a table SCB opened.).
- For entry point ARIYS243, see page 353 (ARIYS243 sorts rous with a list SCB opened.).
- Each entry point picks up rous from its respective type of object and places them in internal pages to be sorted. They then branch to a common sort process INTSORT.





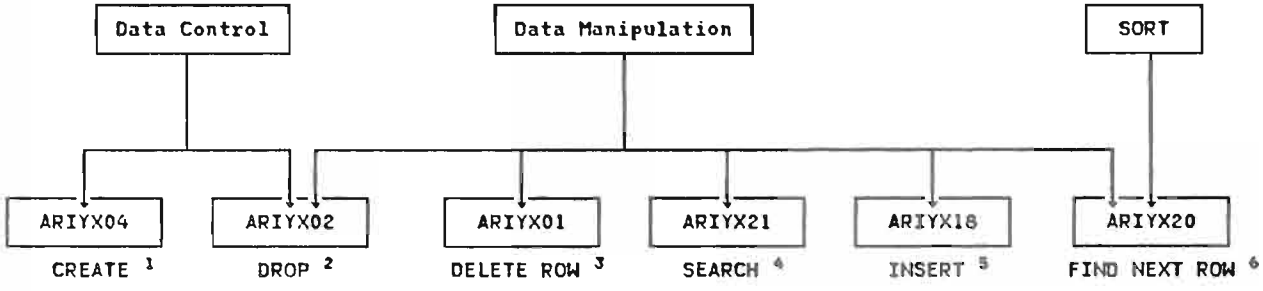

• Get the next TID in the table
ARIYD50
• If end of DBSPACE then, go to (INTSORT)
G
page 354
From page 355
D
• Set user hold if necessary
ARIYD41
• Encode sort domains in row to contiguous sort key
ARIYS12
• Encode row into:
- Sort key followed by SORTLIST domains and put results into internal page
ARIYS14
• If end of internal page, go to (INTSORT)
G
page 354
• Unfix any overflow page and directory block
ARIYK41
• Release lock on overflow page
ARIYK00
• Go to
ARIYS243:
List SCB
• Initialize internal sort tables
- SRTMAP
- AREA
• Get list page LPAGE into buffer
ARIYI19
From page 354
E
• Loop over all list pages
• If end of current page then
- Unfix current page
ARIYI46
- If current page = last page then, go to (INTSORT)
G
page 354
- Update LPAGE by +1




If first page and not last page then
1. Acquire a temporary internal DBSPACE
ARIYI001
2. Get maximum page number of internal DBSPACE
ARIYI421
3. Get storage for sort key merge process (PEERSTOR)
ARIYWSG
Fill intermediate page in internal DBSPACE with sorted rows
ARIYS10
If this is last page, go to (ENDSORT)
H page 355
Update page pointers to next page
If table SCB, go to
D page 353
If list SCB, go to
F page 354
If index SCB, go to
B page 351
If optimized index, go to
C page 352
- Else empty sort, go to (EXIT)
I page 355
H
ENDSORT:
• Merge internal pages and write the merged rows to callers DBSPACE
ARIYS20
• If error, set error RETCODE
I
EXIT:
• Unfix intermediate pages and blocks pointed to by PEERSTOR
ARIYI46
• Release acquired temporary DBSPACE
ARIYI003
• Free PEERSTOR storage and control table storage
ARICWSF
• Return to caller
ARIYS00

DBSS INDEX - GENERAL FLOW (MAJOR INDEX MODULES)

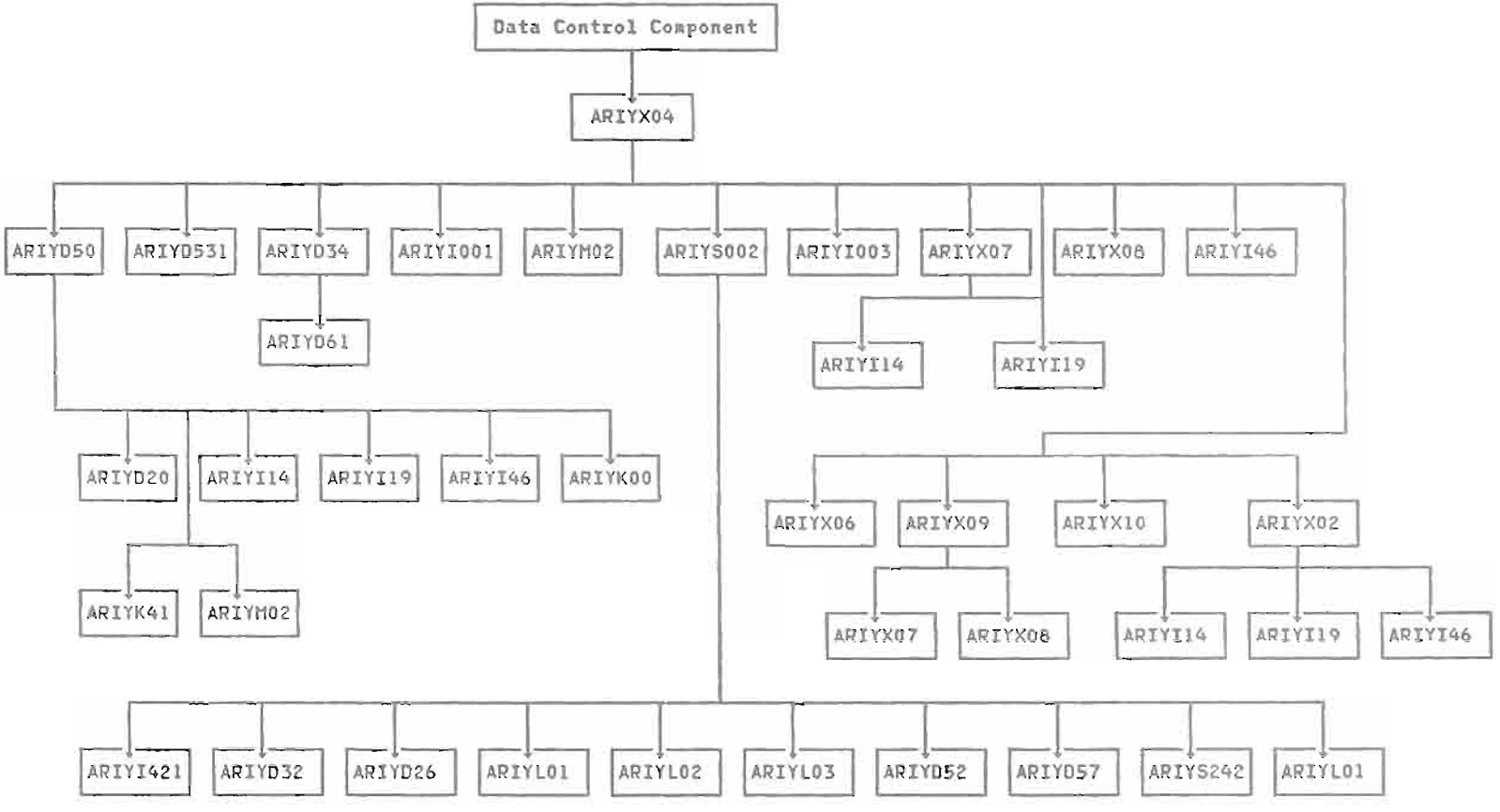
In this diagram, only the DBSS/Index major modules are shown. See the following detail diagrams for other modules being calle
Paths (and the page numbers on which they may be found) are shown at the bottom of this diagram.



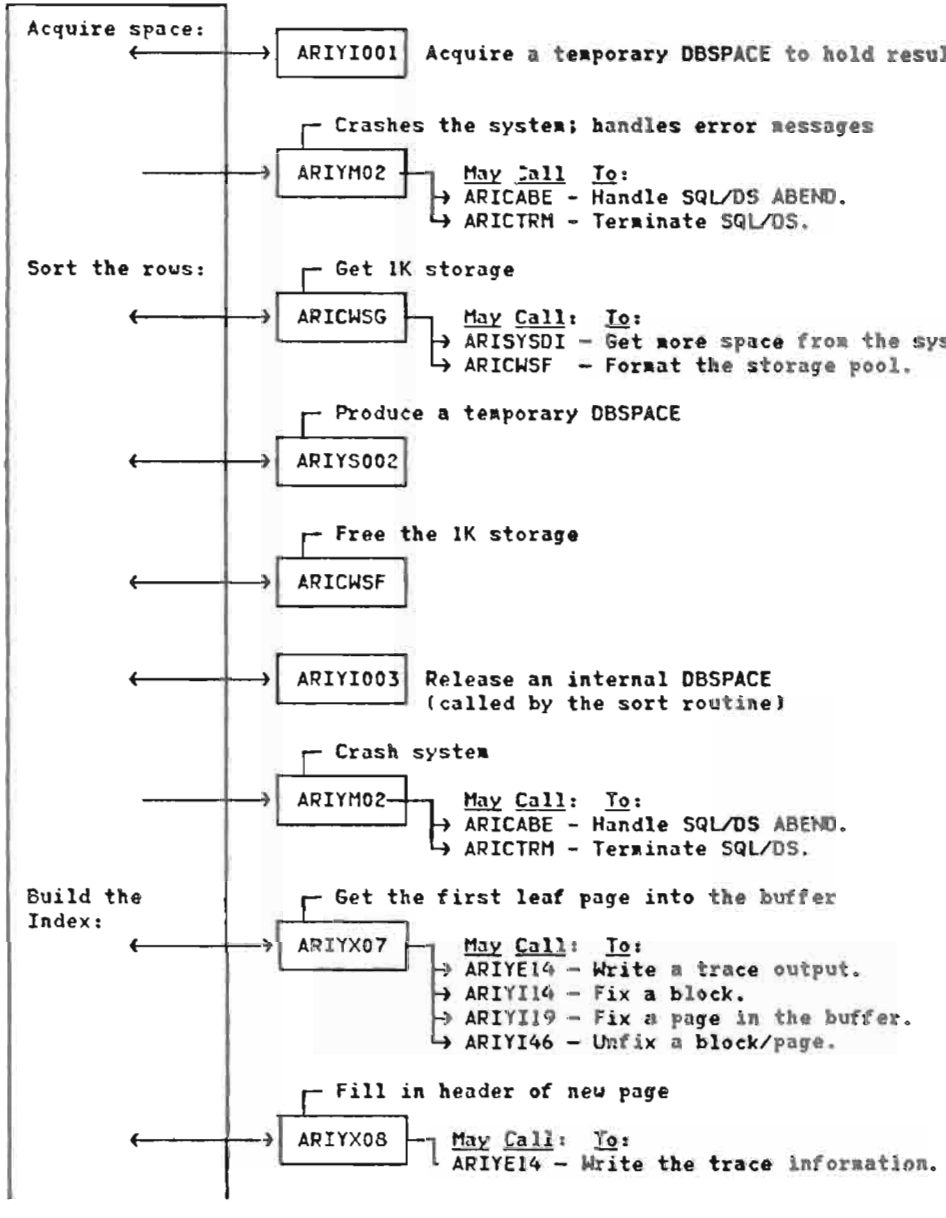
1 Fill an Index during Creation (see page 357)
2 Drop an Index (see page 362)
3 Delete a Key from an Index (see page 365)
4 Search an Index for a Key (see page 368)
5 Insert a Key into an Index (see page 373)
6 Find the Next Sequential Key in an Index (see page 379)

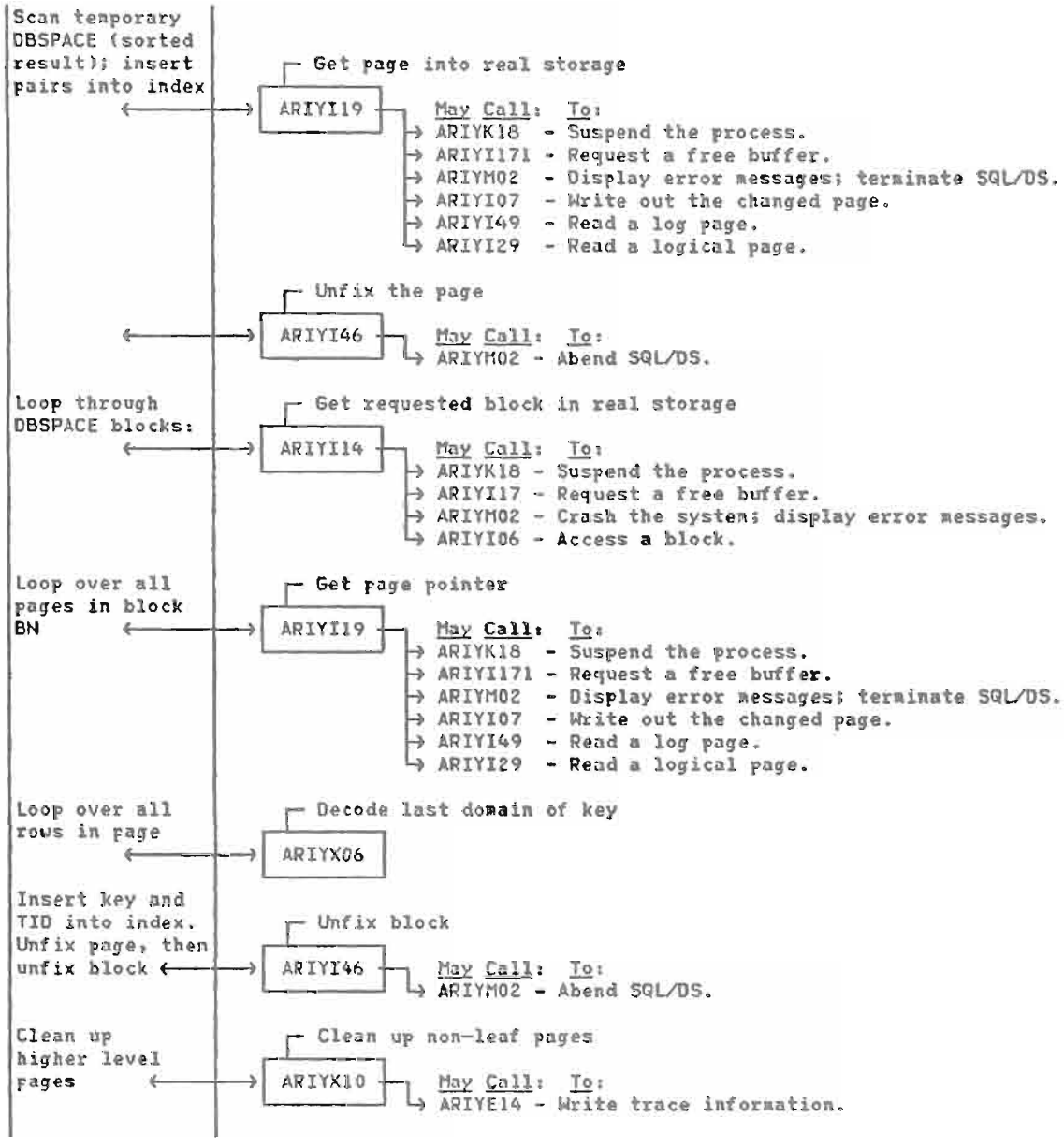
DBSS INDEX - DETAIL FLOW

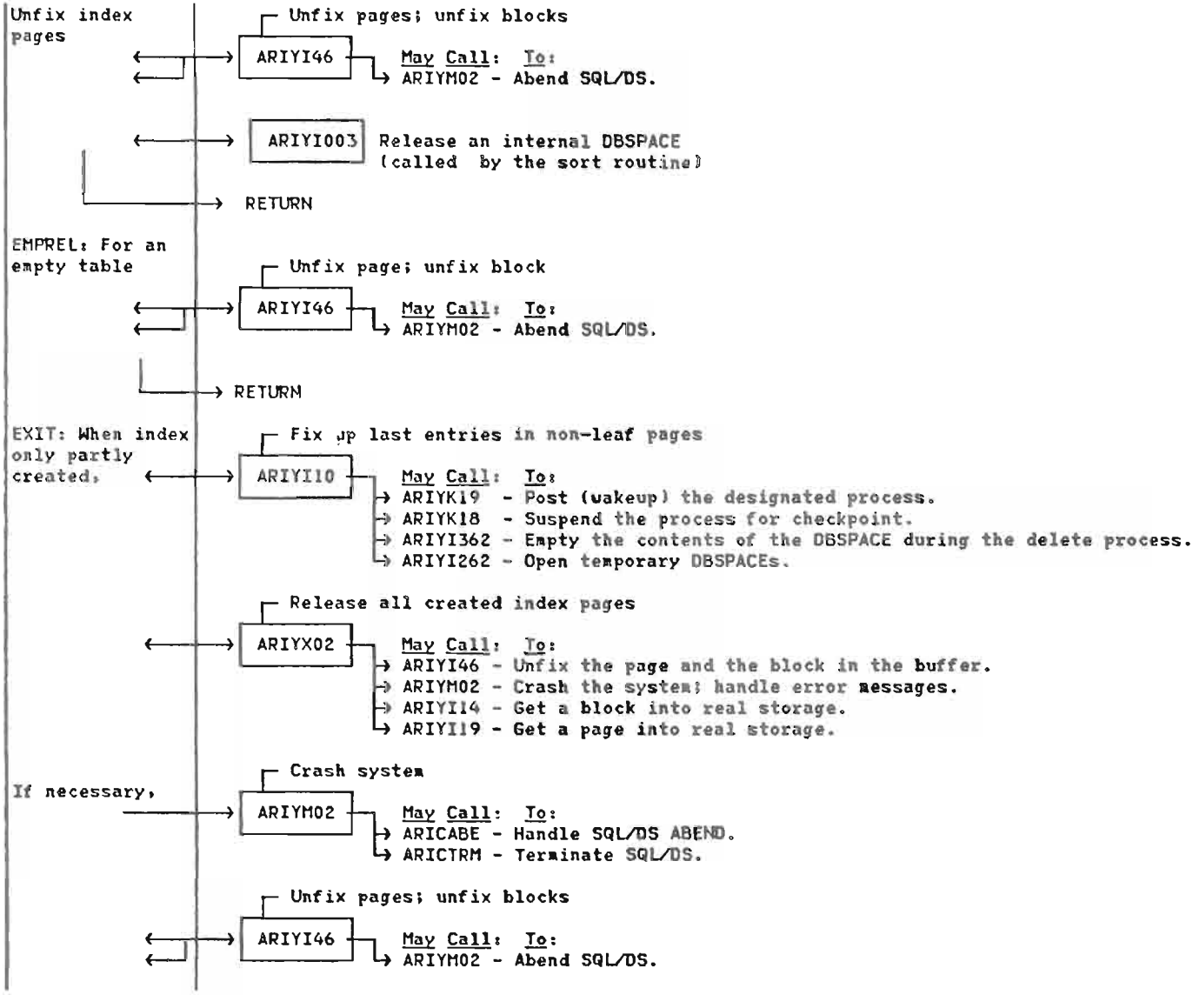
Fill an Index During Creation

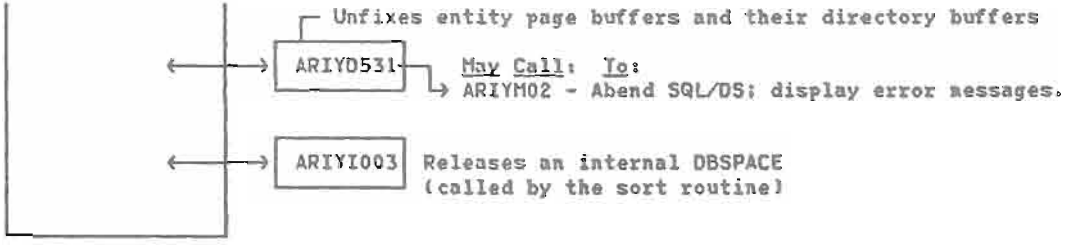


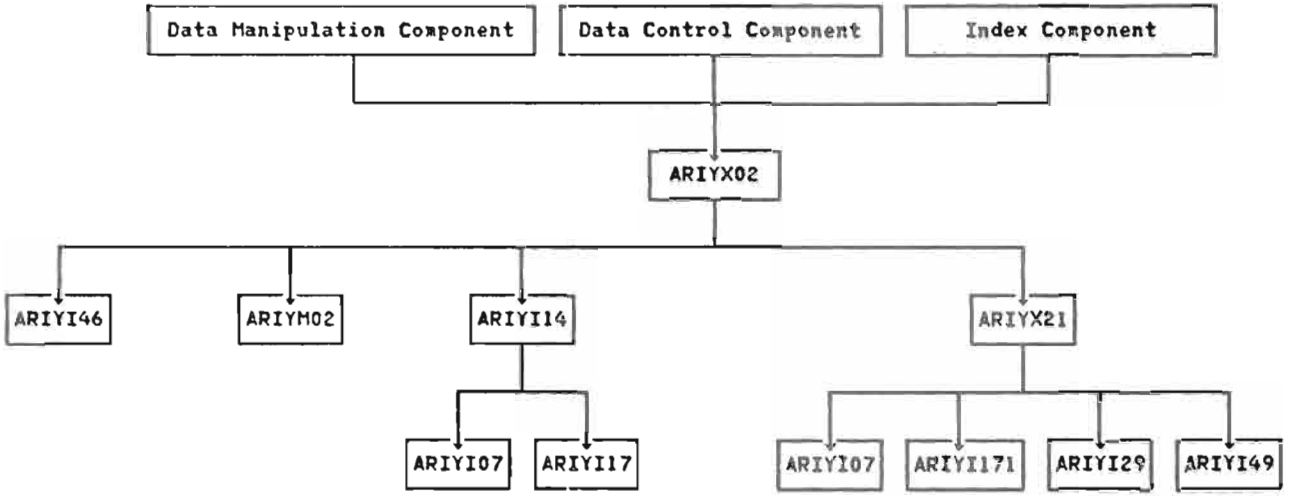
(From Data Control)
ARIYX04



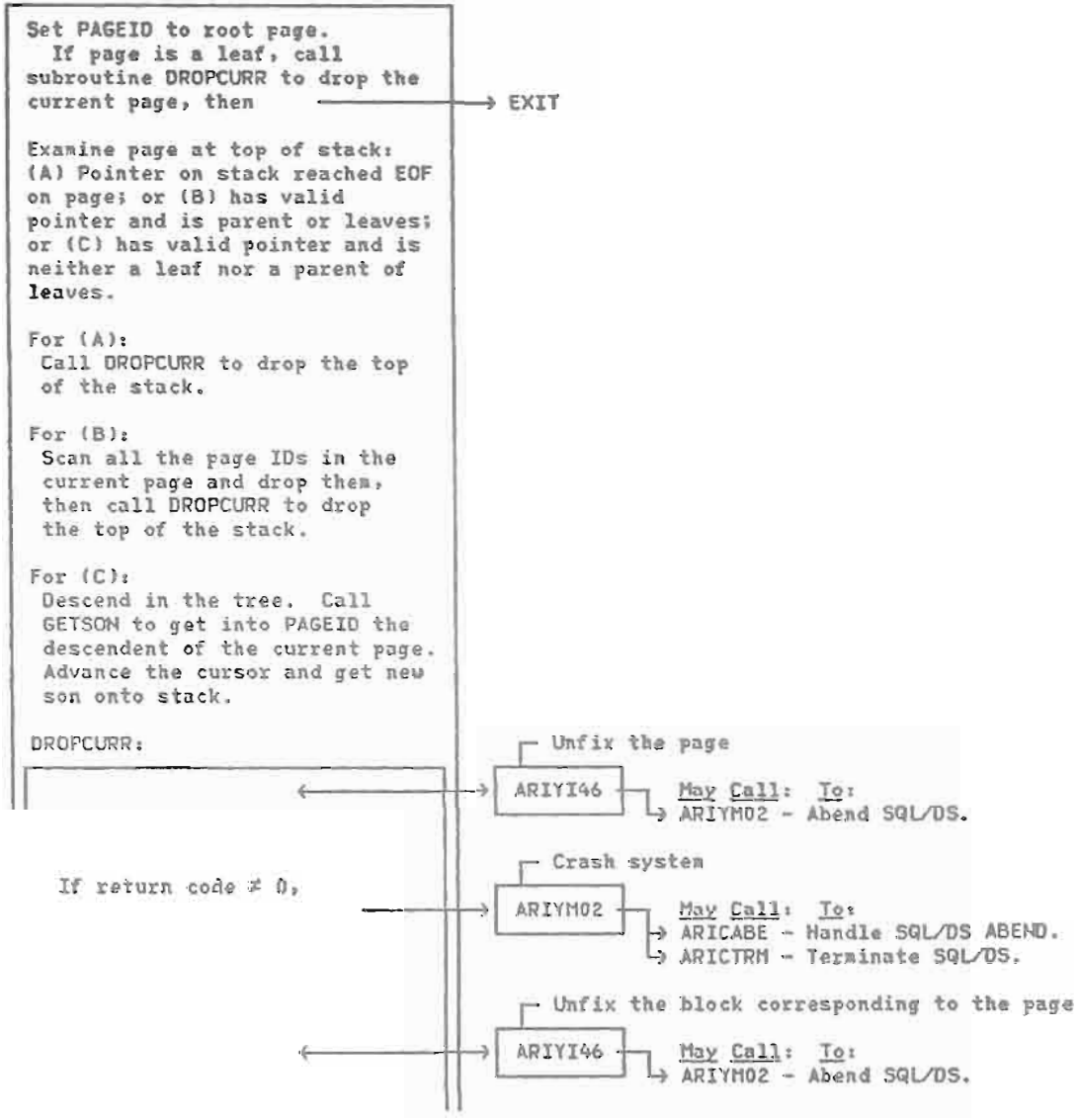

Scan temporary DBSPACE (sorted result); insert pairs into index
Get page into real storage
ARIYI19
May Call: To:
ARIYK18 - Suspend the process.
ARIYI171 - Request a free buffer.
ARIYM02 - Display error messages; terminate SQL/DS.
ARIYI07 - Write out the changed page.
ARIYI49 - Read a log page.
ARIYI29 - Read a logical page.
Unfix the page
ARIYI46
May Call: To:
ARIYM02 - Abend SQL/DS.
Loop through DBSPACE blocks:
Get requested block in real storage
ARIYI14
May Call: To:
ARIYK18 - Suspend the process.
ARIYI17 - Request a free buffer.
ARIYM02 - Crash the system; display error messages.
ARIYI06 - Access a block.
Loop over all pages in block BN
Get page pointer
ARIYI19
May Call: To:
ARIYK18 - Suspend the process.
ARIYI171 - Request a free buffer.
ARIYM02 - Display error messages; terminate SQL/DS.
ARIYI07 - Write out the changed page.
ARIYI49 - Read a log page.
ARIYI29 - Read a logical page.
Loop over all rows in page
Decode last domain of key
ARIYX06
Insert key and TID into index. Unfix page, then unfix block
Unfix block
ARIYI46
May Call: To:
ARIYM02 - Abend SQL/DS.
Clean up higher level pages
Clean up non-leaf pages
ARIYX10
May Call: To:
ARIYE14 - Write trace information.



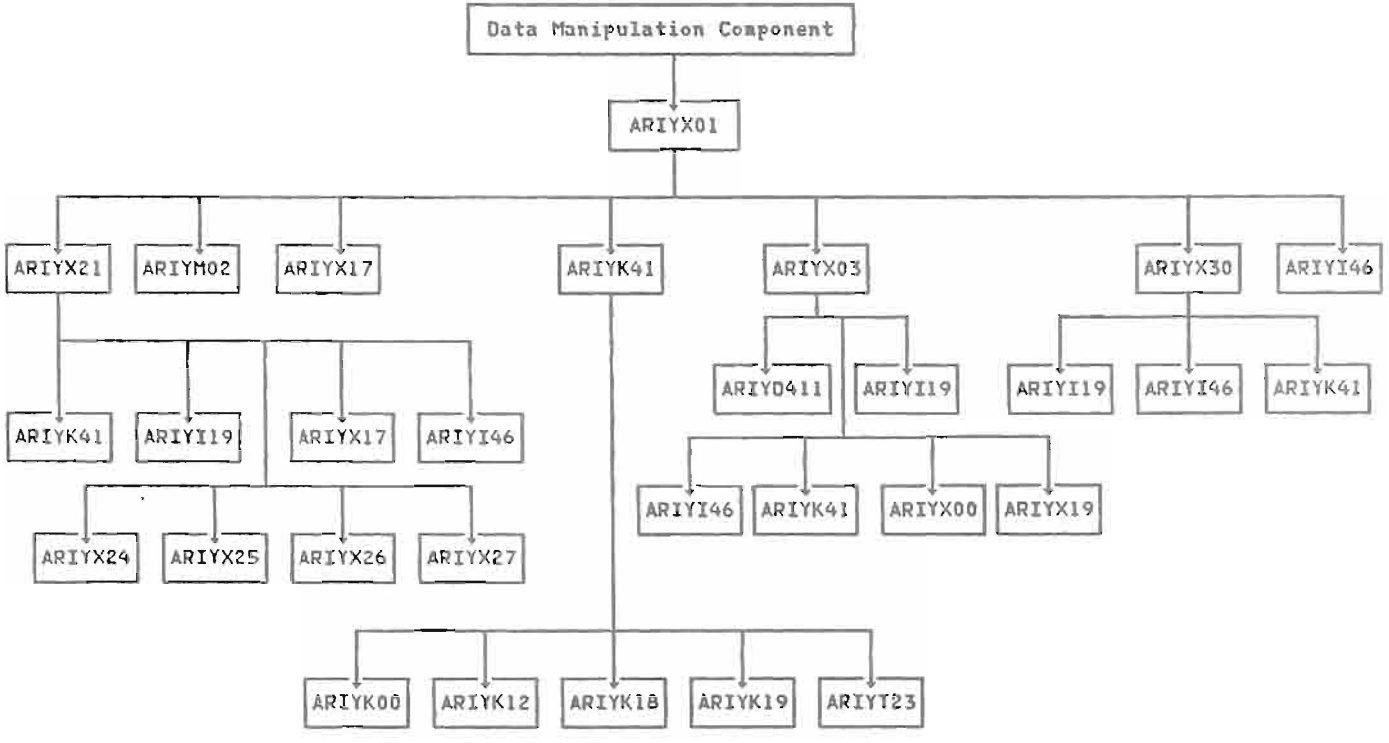

Unfixes entity page buffers and their directory buffers
ARIYD531
May Call: To:
ARIYM02 - Abend SQL/DS; display error messages.
ARIYI003
Releases an internal DBSPACE
(called by the sort routine)

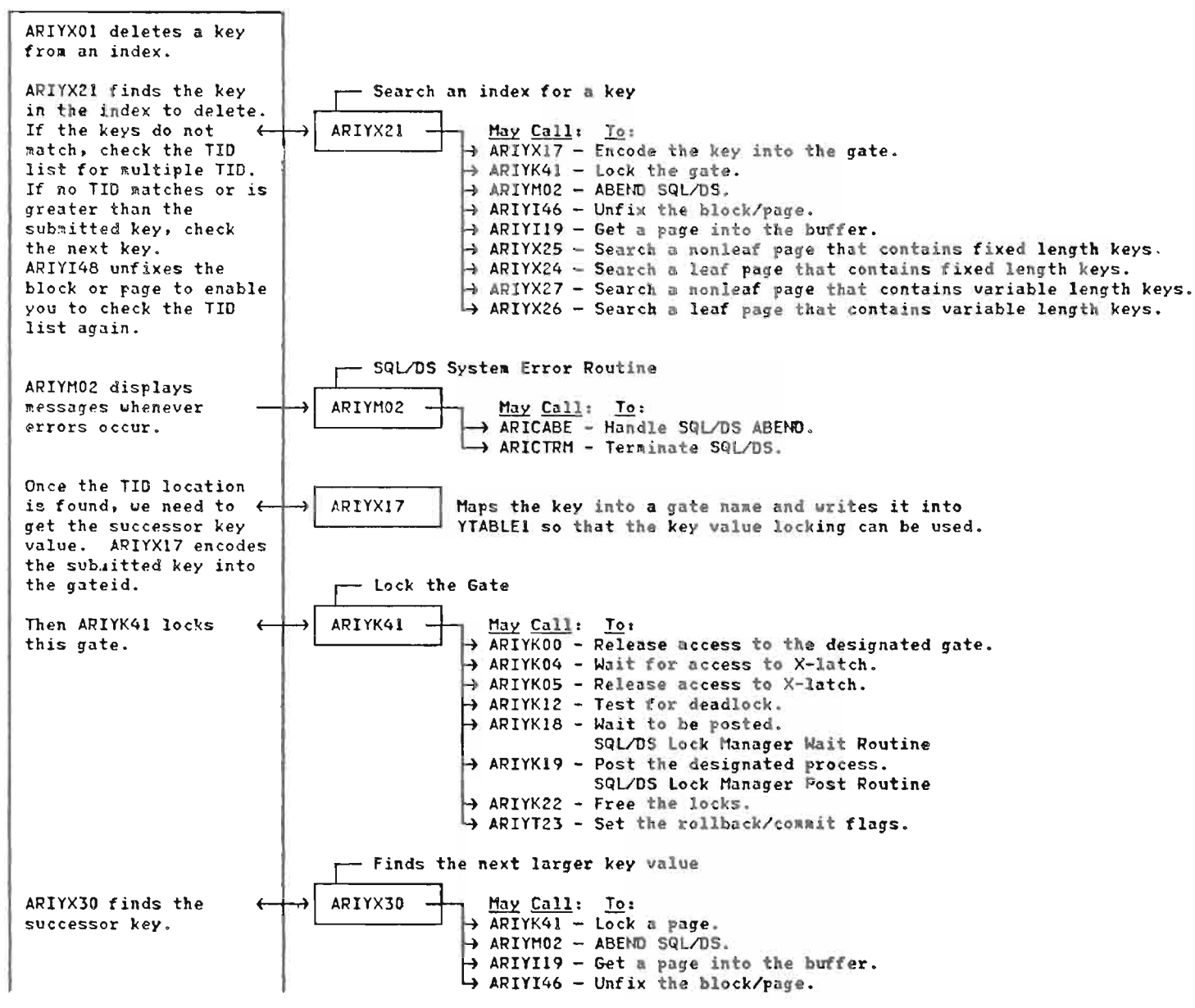


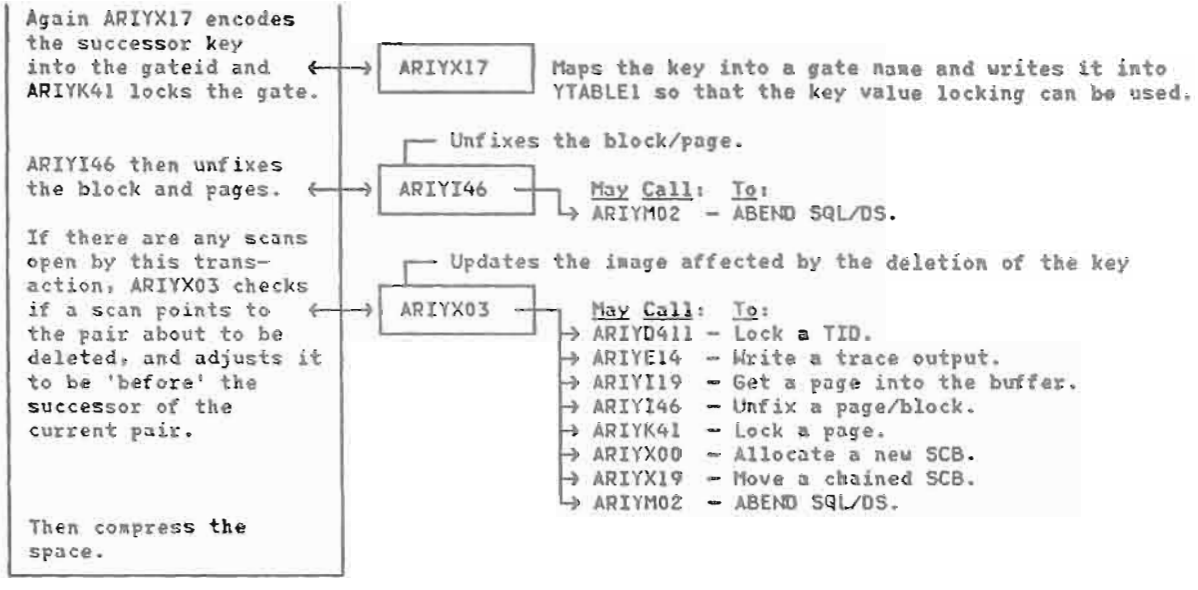
(From Data Manipulation, Data Control, or Index)
ARIYX02

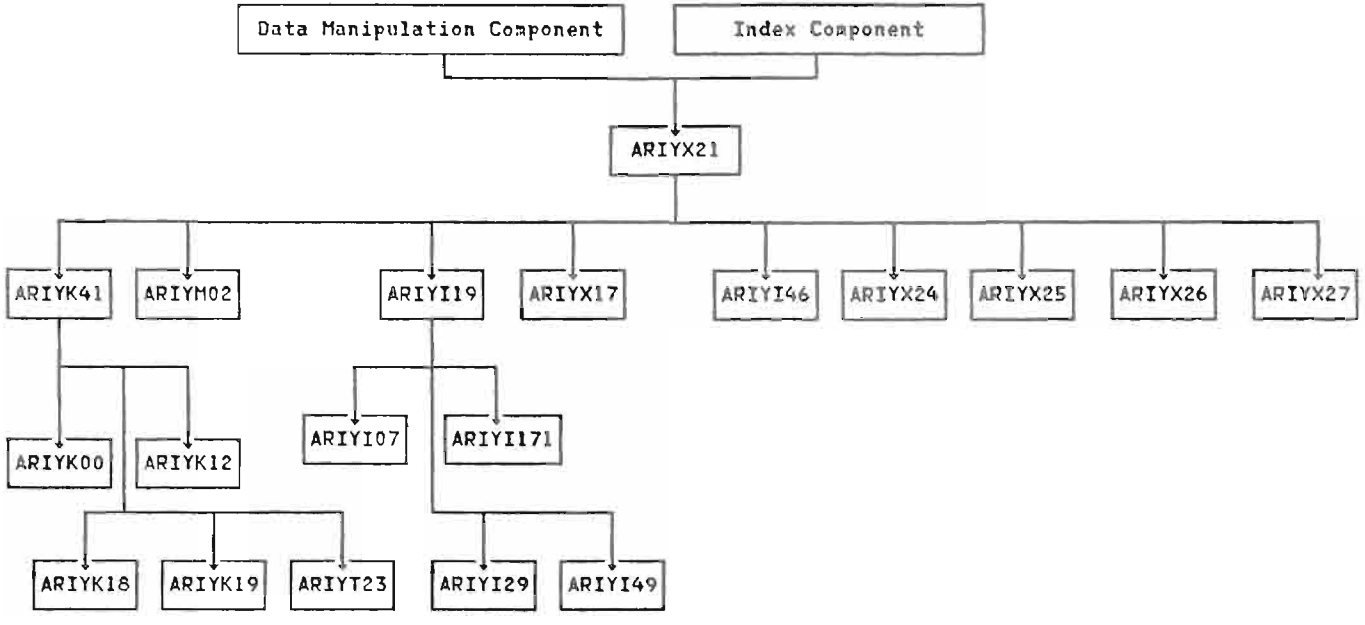


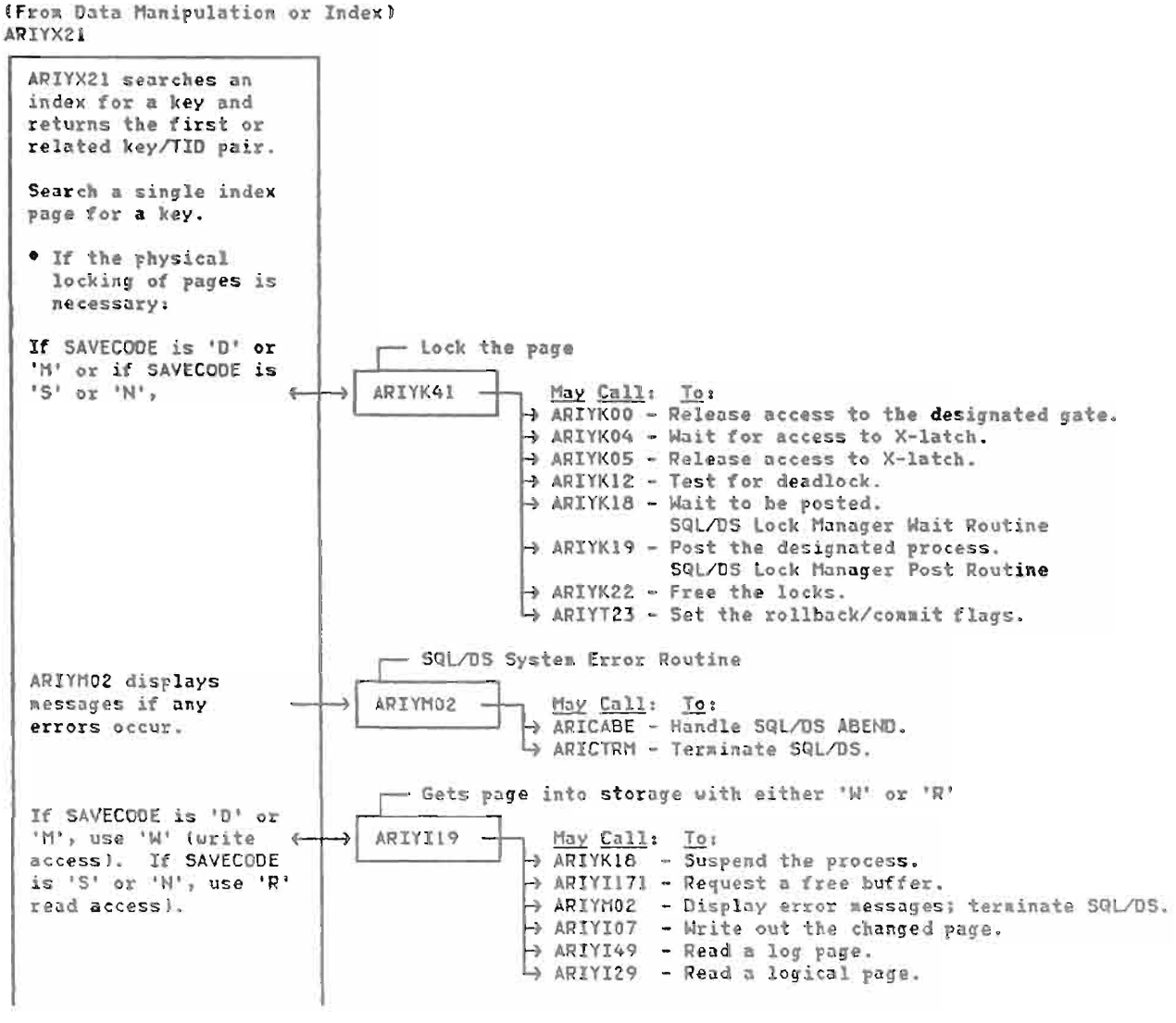
Delete a Key from an Index

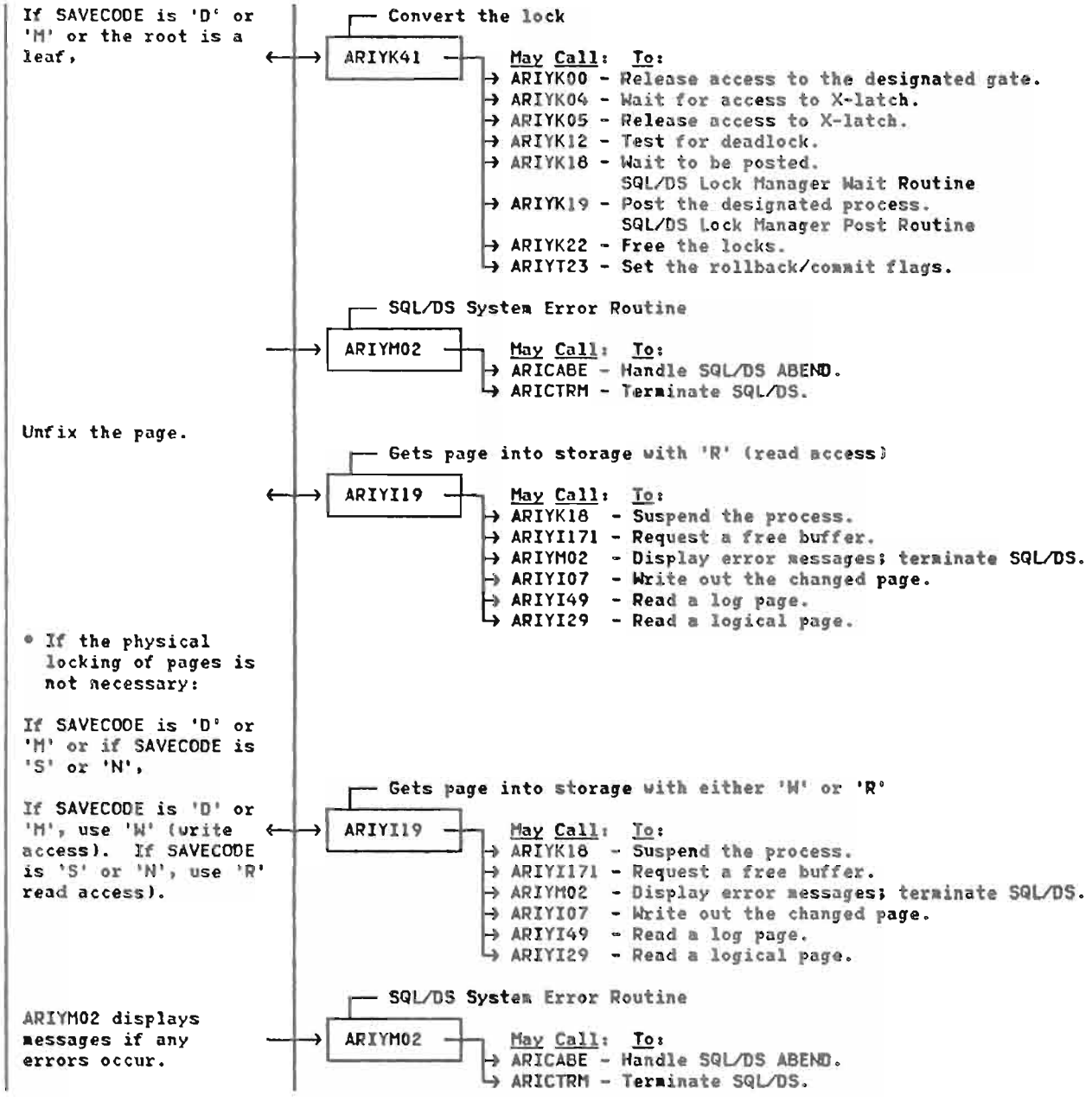


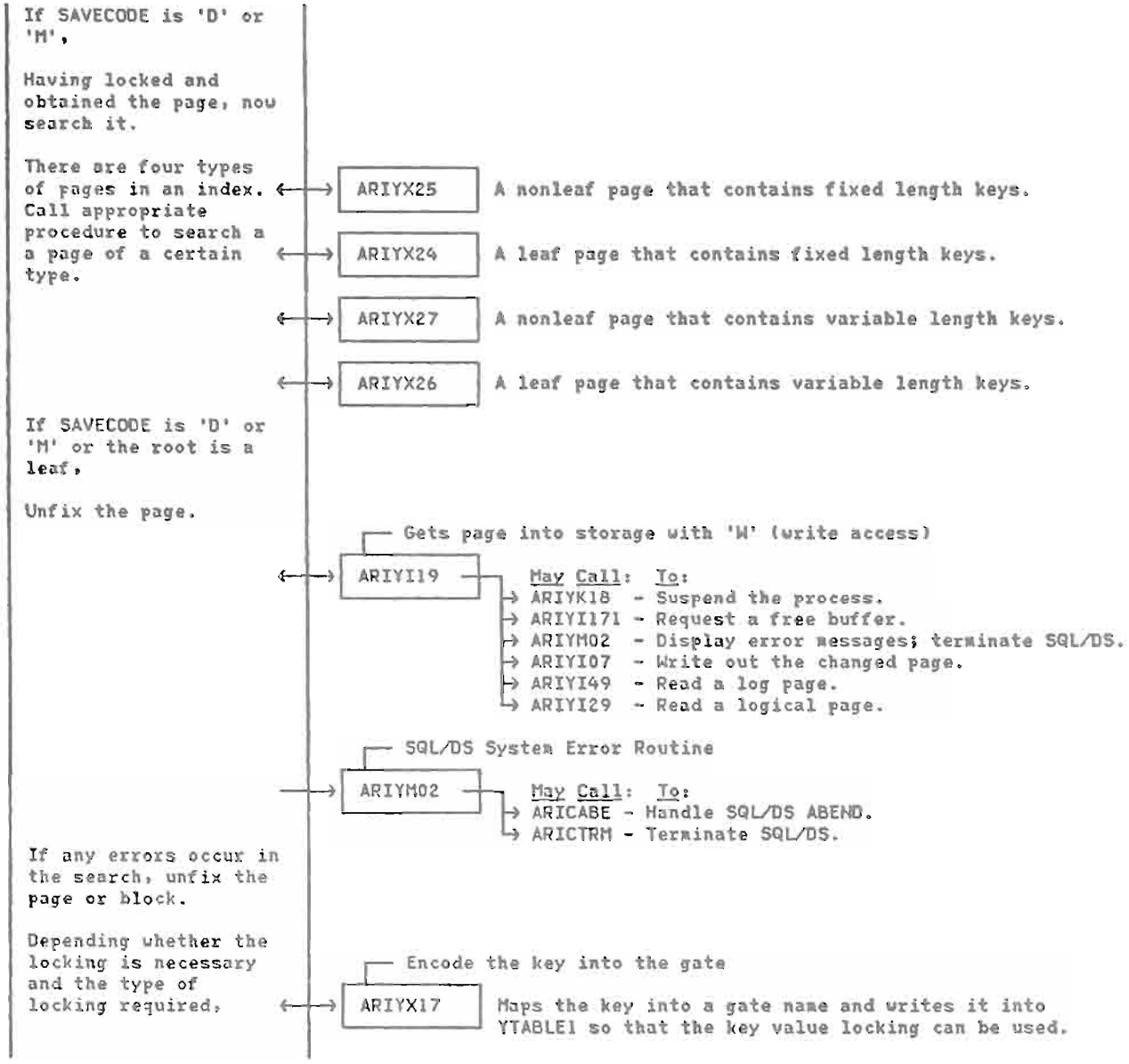


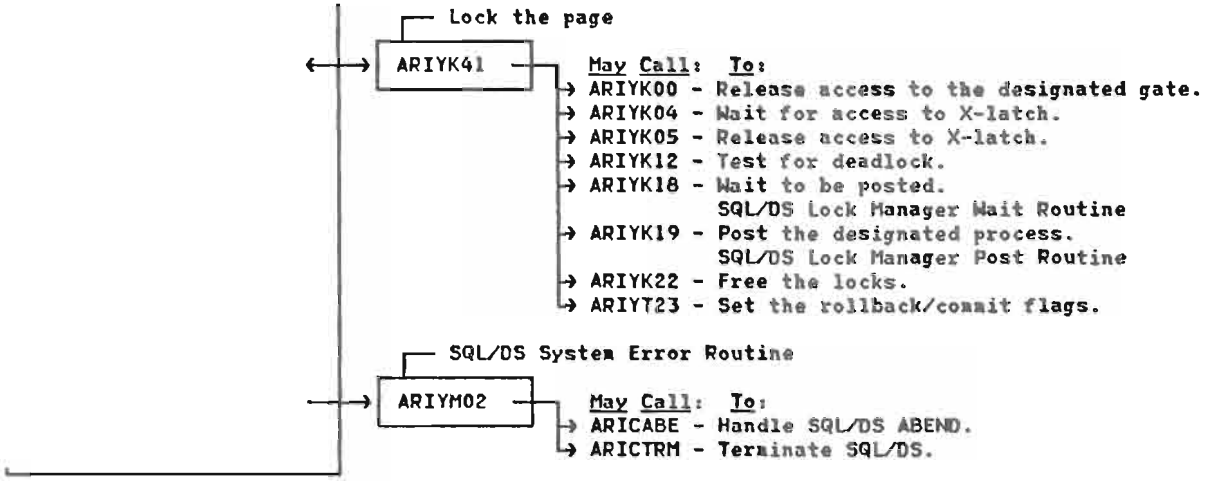




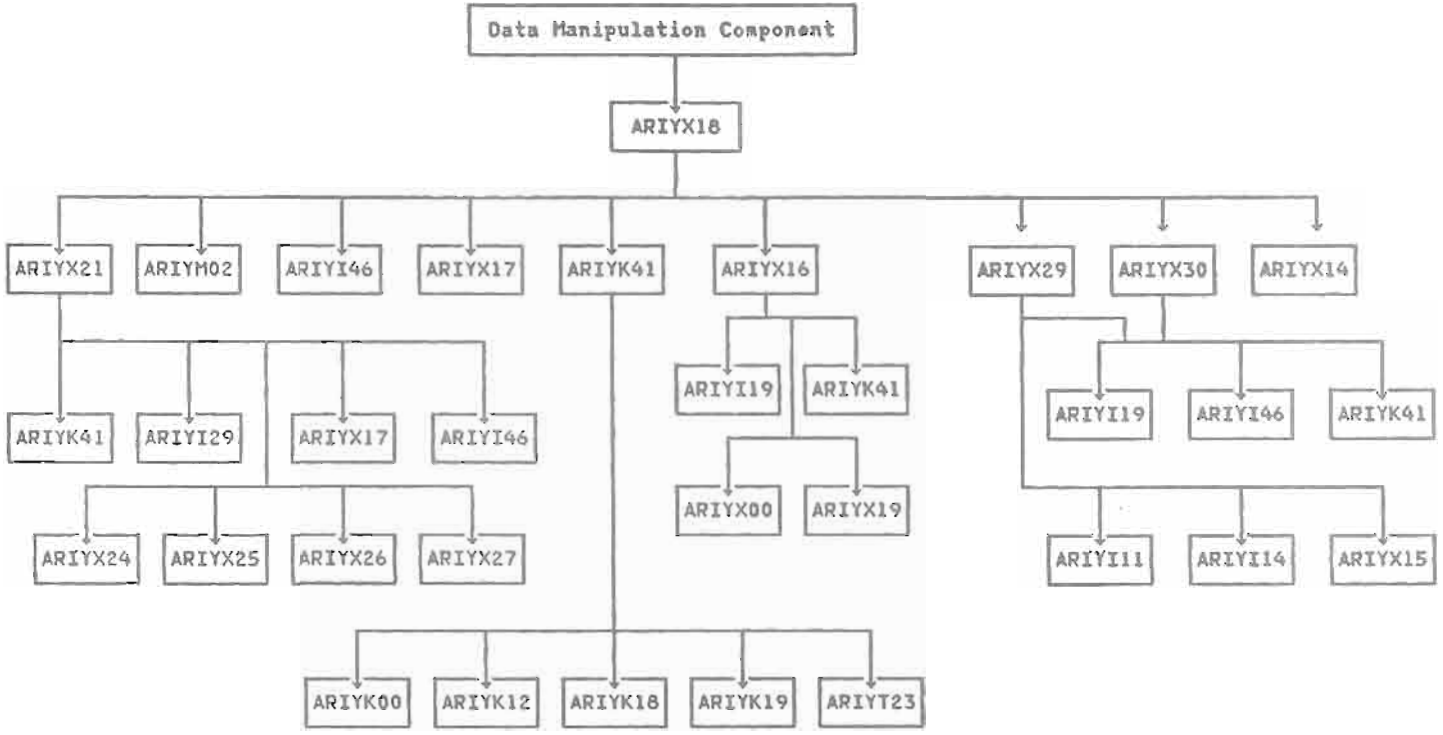

(From Data Manipulation or Index)
ARIYX21
ARIYX21 searches an index for a key and returns the first or related key/TID pair.
Search a single index page for a key.
• If the physical locking of pages is necessary:
If SAVECODE is 'D' or 'M' or if SAVECODE is 'S' or 'N',
Lock the page
ARIYK41
May Call: To:
ARIYK00 - Release access to the designated gate.
ARIYK04 - Wait for access to X-latch.
ARIYK05 - Release access to X-latch.
ARIYK12 - Test for deadlock.
ARIYK18 - Wait to be posted.
SQL/DS Lock Manager Wait Routine
ARIYK19 - Post the designated process.
SQL/DS Lock Manager Post Routine
ARIYK22 - Free the locks.
ARIYT23 - Set the rollback/commit flags.
ARIYM02 displays messages if any errors occur.
SQL/DS System Error Routine
ARIYM02
May Call: To:
ARICABE - Handle SQL/DS ABEND.
ARICTRM - Terminate SQL/DS.
If SAVECODE is 'D' or 'M', use 'W' (write access). If SAVECODE is 'S' or 'N', use 'R' read access).
Gets page into storage with either 'W' or 'R'
ARIYI19
May Call: To:
ARIYK18 - Suspend the process.
ARIYI171 - Request a free buffer.
ARIYM02 - Display error messages; terminate SQL/DS.
ARIYI07 - Write out the changed page.
ARIYI49 - Read a log page.
ARIYI29 - Read a logical page.



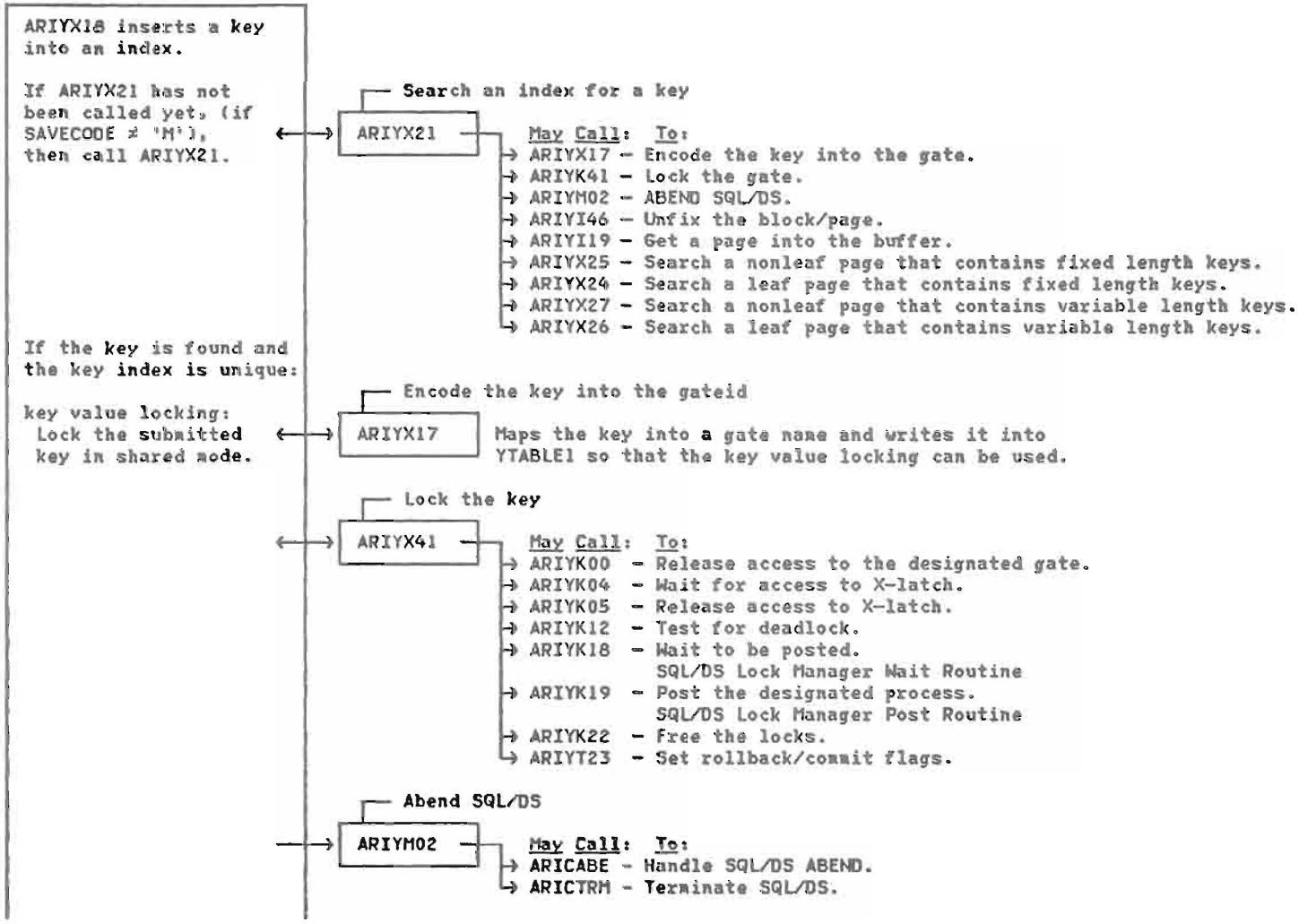

If SAVECODE is 'D' or 'M',
Having locked and obtained the page, now search it.
There are four types of pages in an index. Call appropriate procedure to search a a page of a certain type.
ARIYX25
A nonleaf page that contains fixed length keys.
ARIYX24
A leaf page that contains fixed length keys.
ARIYX27
A nonleaf page that contains variable length keys.
ARIYX26
A leaf page that contains variable length keys.
If SAVECODE is 'D' or 'M' or the root is a leaf,
Unfix the page.
Gets page into storage with 'W' (write access)
ARIYI19
May Call: To:
ARIYK18 - Suspend the process.
ARIYI171 - Request a free buffer.
ARIYM02 - Display error messages; terminate SQL/DS.
ARIYI07 - Write out the changed page.
ARIYI49 - Read a log page.
ARIYI29 - Read a logical page.
SQL/DS System Error Routine
ARIYM02
May Call: To:
ARICABE - Handle SQL/DS ABEND.
ARICTRM - Terminate SQL/DS.
If any errors occur in the search, unfix the page or block.
Depending whether the locking is necessary and the type of locking required,
Encode the key into the gate
ARIYX17
Maps the key into a gate name and writes it into YTABLE1 so that the key value locking can be used.

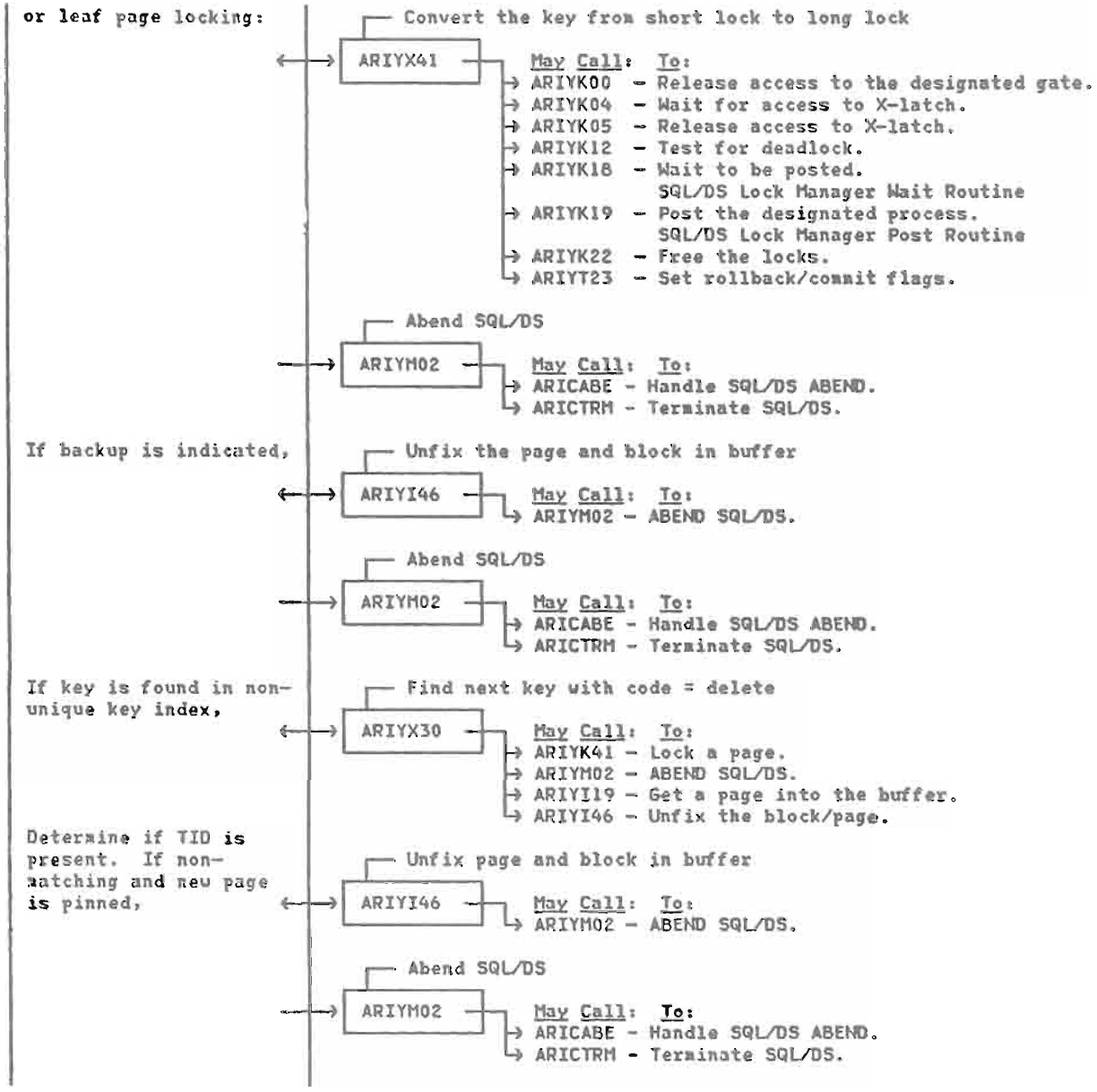


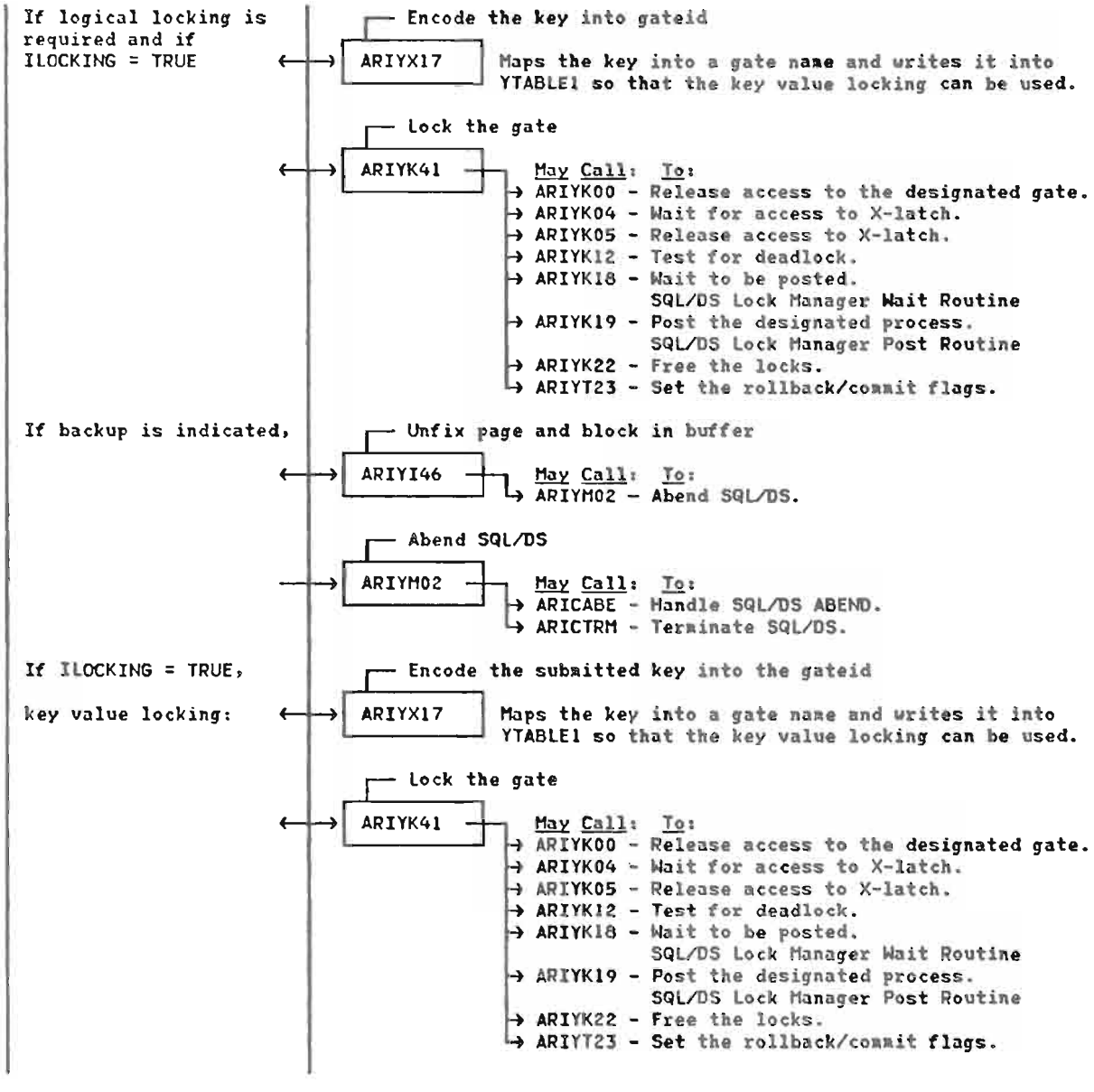
Insert a Key into an Index

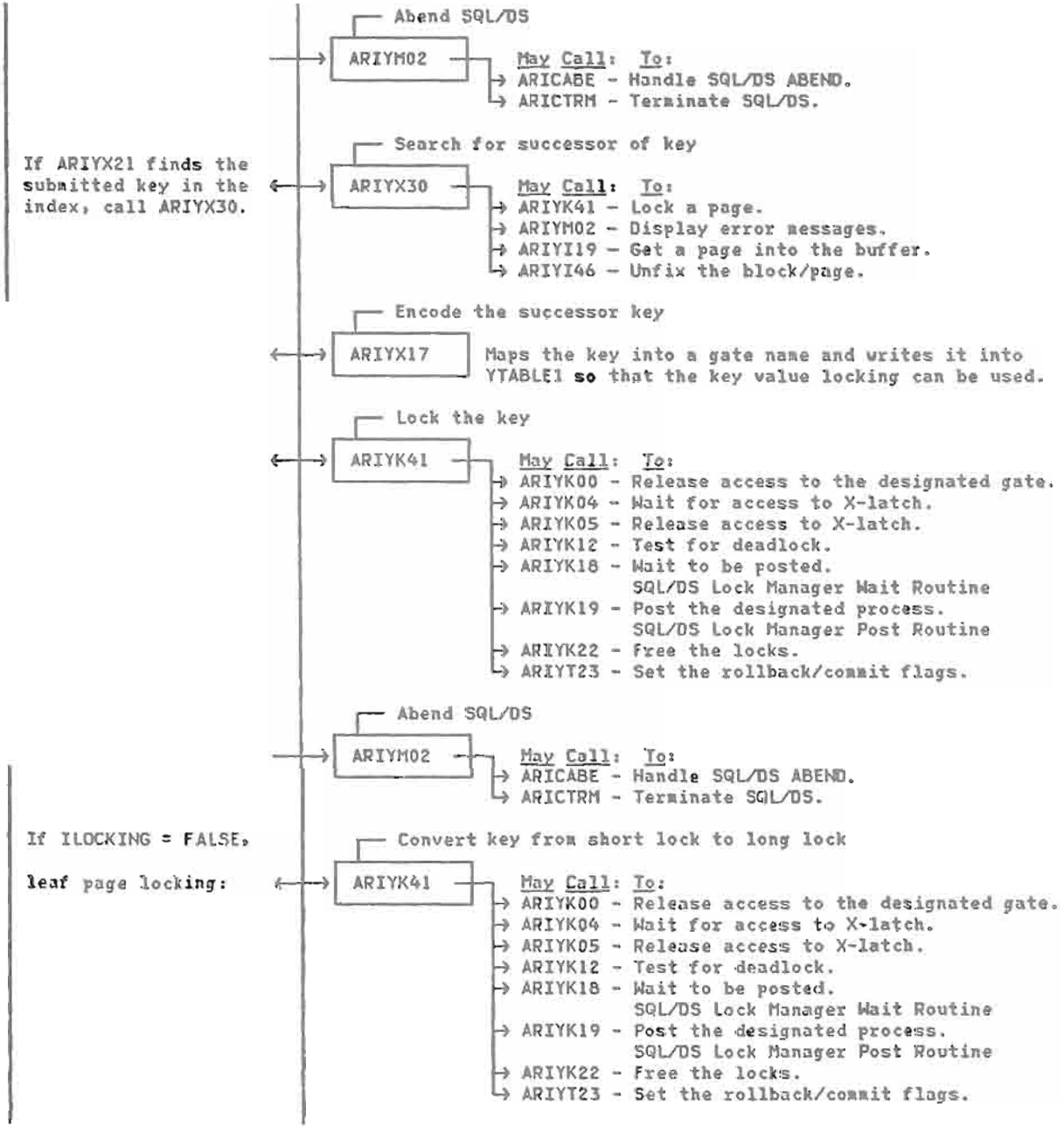


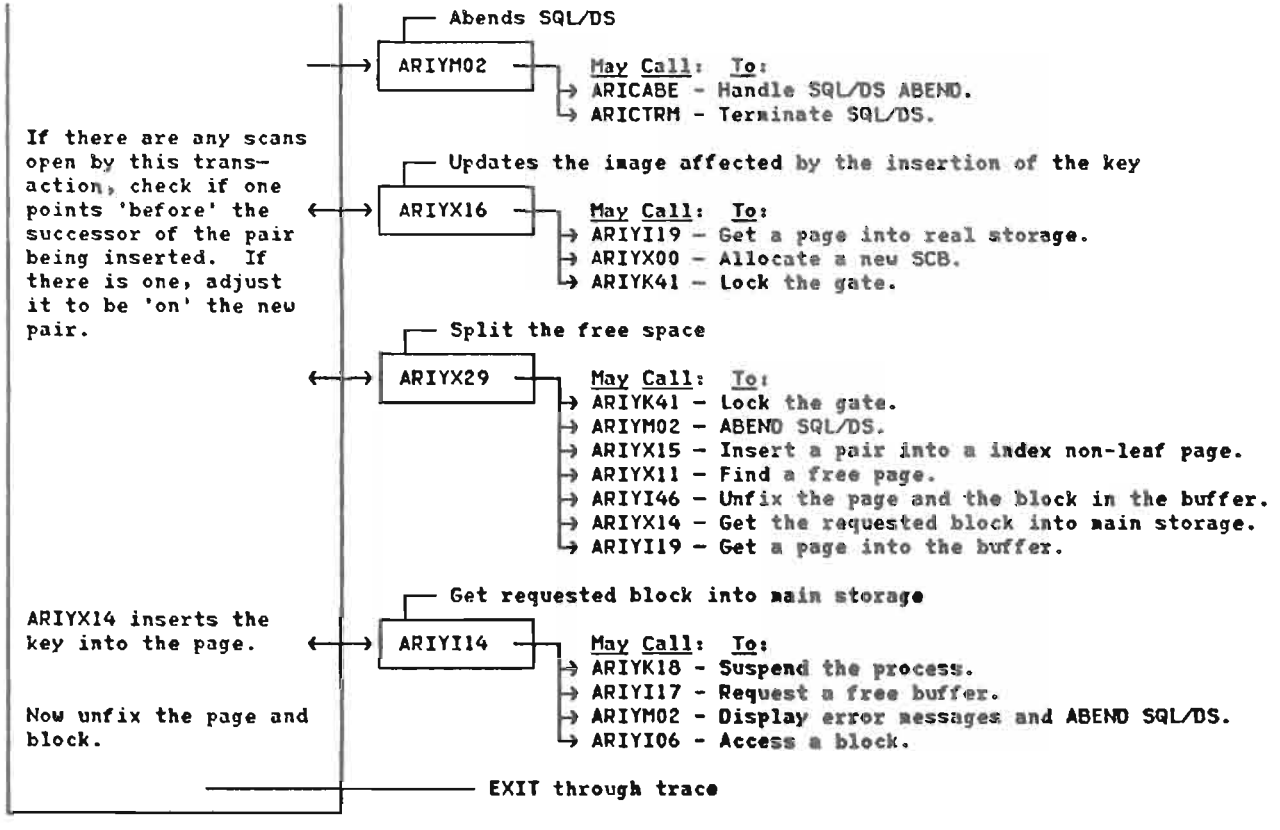
(From Data Manipulation)
ARIYX18



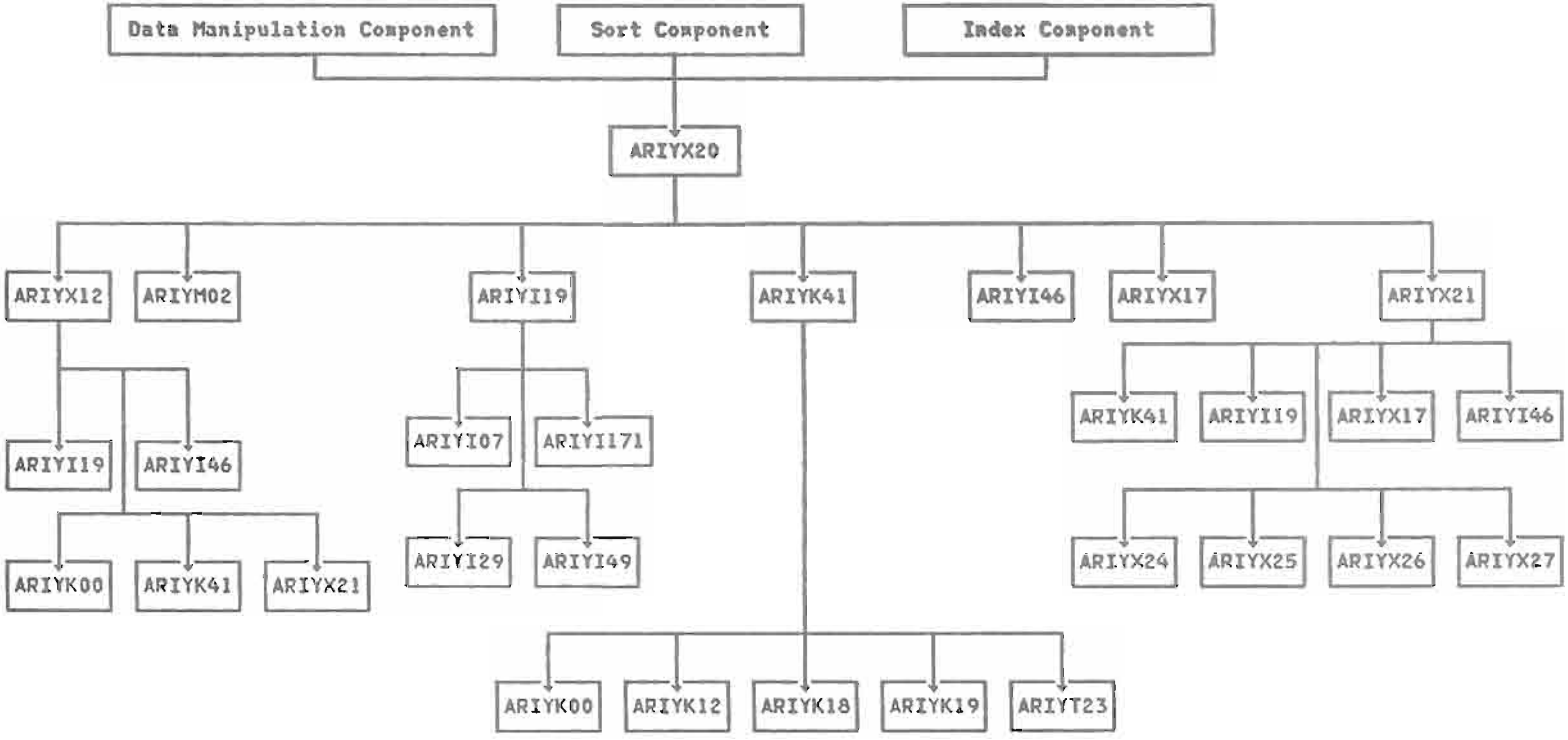

or leaf page locking:
Convert the key from short lock to long lock
ARIYX41
May Call: To:
ARIYK00 - Release access to the designated gate.
ARIYK04 - Wait for access to X-latch.
ARIYK05 - Release access to X-latch.
ARIYK12 - Test for deadlock.
ARIYK18 - Wait to be posted.
SQL/DS Lock Manager Wait Routine
ARIYK19 - Post the designated process.
SQL/DS Lock Manager Post Routine
ARIYK22 - Free the locks.
ARIYT23 - Set rollback/commit flags.
Abend SQL/DS
ARIYM02
May Call: To:
ARICABE - Handle SQL/DS ABEND.
ARICTRM - Terminate SQL/DS.
If backup is indicated,
Unfix the page and block in buffer
ARIYI46
May Call: To:
ARIYM02 - ABEND SQL/DS.
Abend SQL/DS
ARIYM02
May Call: To:
ARICABE - Handle SQL/DS ABEND.
ARICTRM - Terminate SQL/DS.
If key is found in non-unique key index,
Find next key with code = delete
ARIYX30
May Call: To:
ARIYK41 - Lock a page.
ARIYM02 - ABEND SQL/DS.
ARIYI19 - Get a page into the buffer.
ARIYI46 - Unfix the block/page.
Determine if TID is present. If non-matching and new page is pinned,
Unfix page and block in buffer
ARIYI46
May Call: To:
ARIYM02 - ABEND SQL/DS.
Abend SQL/DS
ARIYM02
May Call: To:
ARICABE - Handle SQL/DS ABEND.
ARICTRM - Terminate SQL/DS.



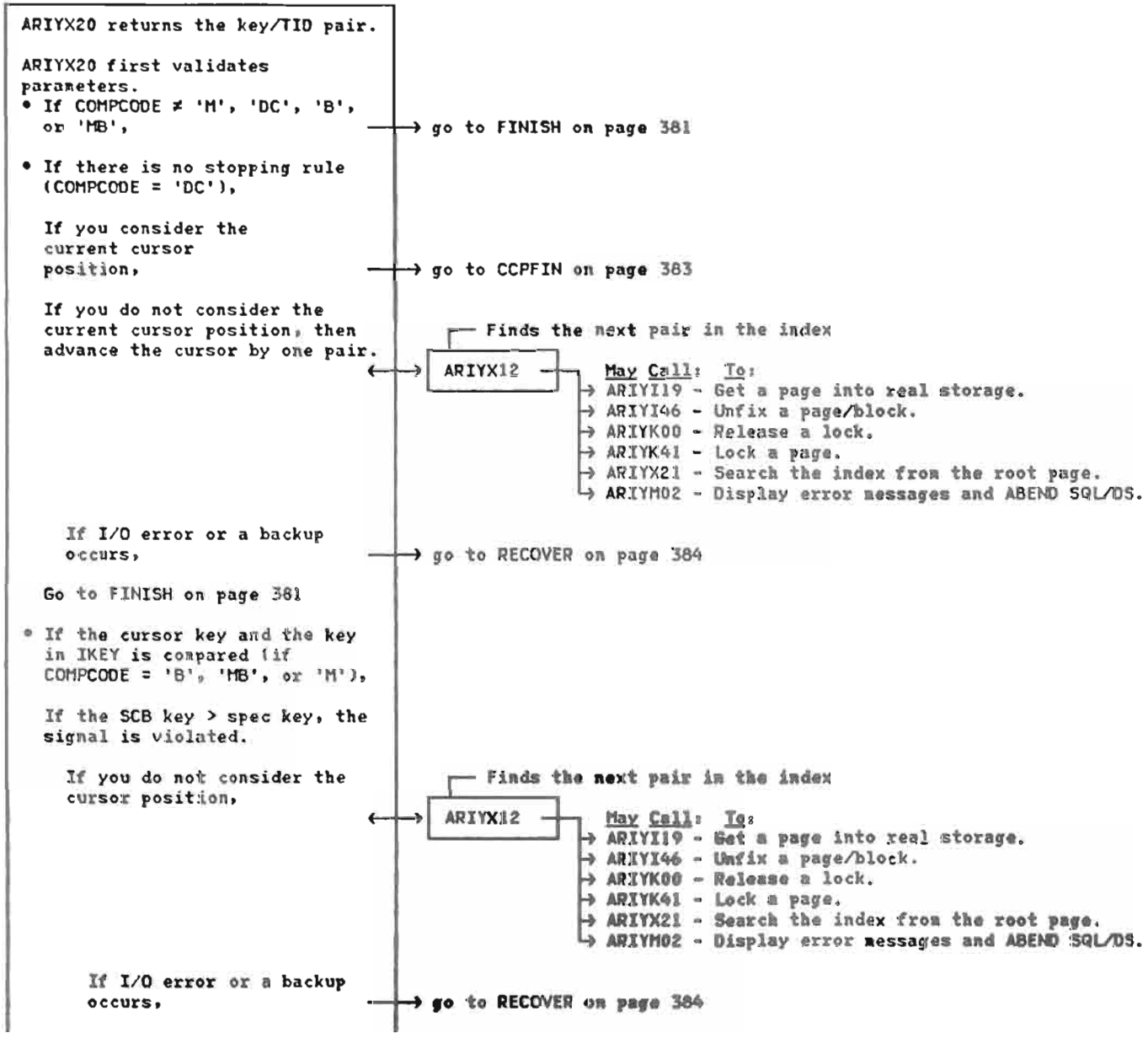

Abend SQL/DS
ARIYM02
May Call: To:
ARICABE - Handle SQL/DS ABEND.
ARICTRM - Terminate SQL/DS.
Search for successor of key
If ARIYX21 finds the submitted key in the index, call ARIYX30.
ARIYX30
May Call: To:
ARIYK41 - Lock a page.
ARIYM02 - Display error messages.
ARIYI19 - Get a page into the buffer.
ARIYI46 - Unfix the block/page.
Encode the successor key
ARIYX17
Maps the key into a gate name and writes it into YTABLE1 so that the key value locking can be used.
Lock the key
ARIYK41
May Call: To:
ARIYK00 - Release access to the designated gate.
ARIYK04 - Wait for access to X-latch.
ARIYK05 - Release access to X-latch.
ARIYK12 - Test for deadlock.
ARIYK18 - Wait to be posted.
SQL/DS Lock Manager Wait Routine
ARIYK19 - Post the designated process.
SQL/DS Lock Manager Post Routine
ARIYK22 - Free the locks.
ARIYT23 - Set the rollback/commit flags.
Abend SQL/DS
ARIYM02
May Call: To:
ARICABE - Handle SQL/DS ABEND.
ARICTRM - Terminate SQL/DS.
If ILOCKING = FALSE,
leaf page locking:
Convert key from short lock to long lock
ARIYK41
May Call: To:
ARIYK00 - Release access to the designated gate.
ARIYK04 - Wait for access to X-latch.
ARIYK05 - Release access to X-latch.
ARIYK12 - Test for deadlock.
ARIYK18 - Wait to be posted.
SQL/DS Lock Manager Wait Routine
ARIYK19 - Post the designated process.
SQL/DS Lock Manager Post Routine
ARIYK22 - Free the locks.
ARIYT23 - Set the rollback/commit flags.



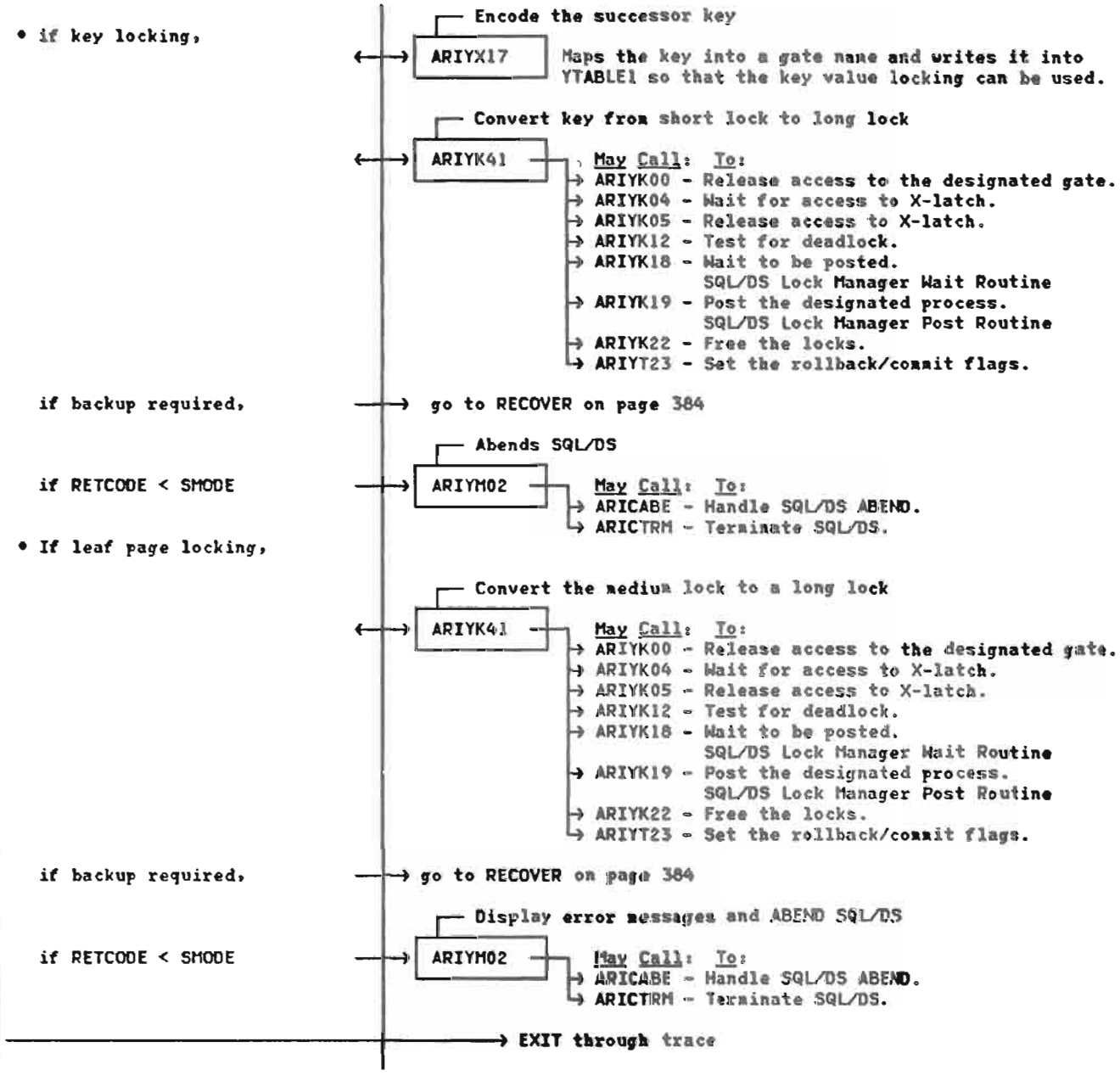
Find the Next Sequential Key in an Index



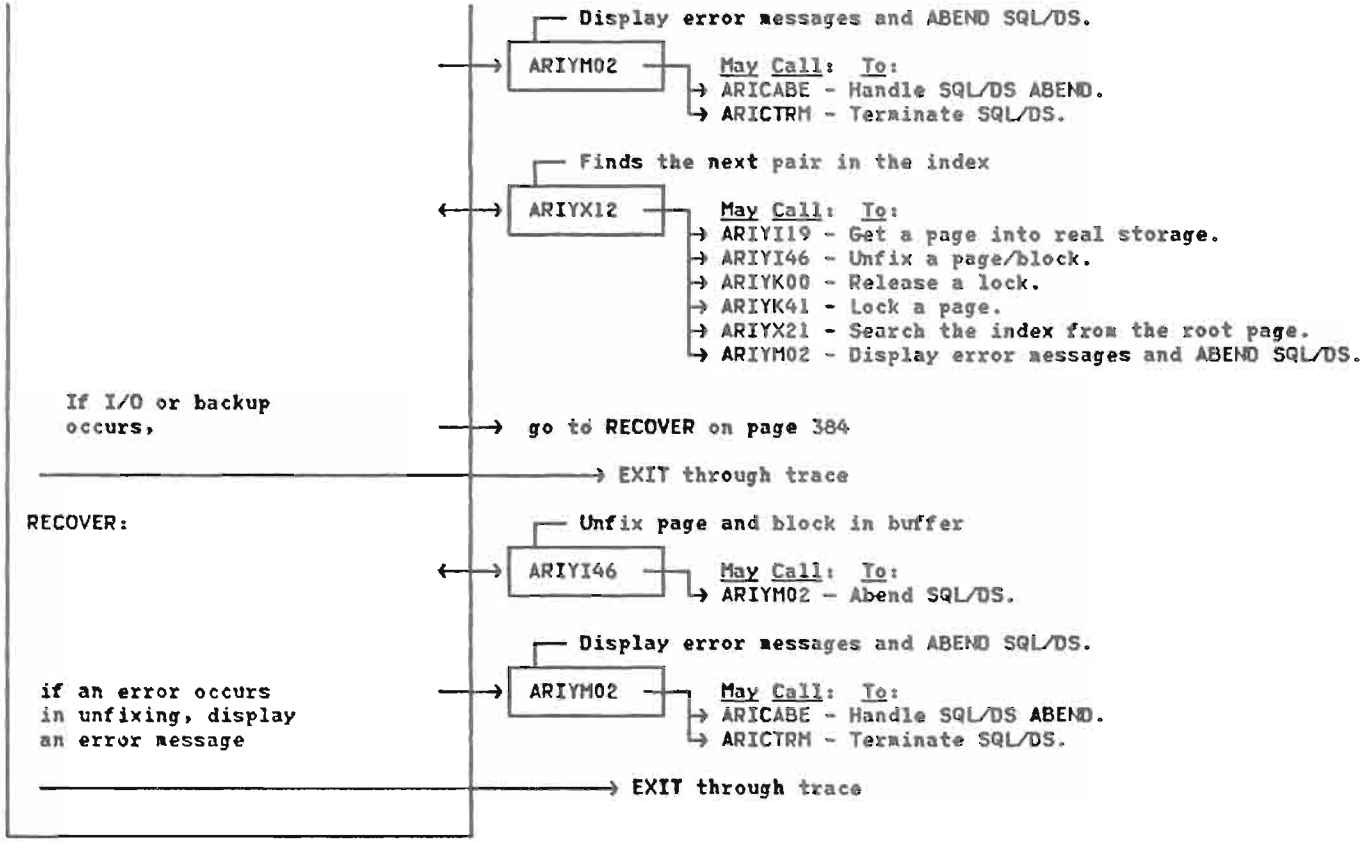
(From Data Manipulation, Sort, or ARIYX05 of Index)
ARIYX20




Go to FINISH on page 381
If the current cursor position is included (QUALCODE = 'H'),
If COMPCODE = 'MB', 'M', 'B',
go to CCFPIN on page 383
• Scan through the index for a key.
Finds the next pair in the index
ARIYX12
May Call: To:
ARIYI19 - Get a page into real storage.
ARIYI46 - Unfix a page/block.
ARIYK00 - Release a lock.
ARIYK41 - Lock a page.
ARIYX21 - Search the index from the root page.
ARIYM02 - Display error messages and ABEND SQL/DS.
If I/O error or backup occurs,
go to RECOVER on page 384
• If a new key is hit and the stopping rule is violated (SCB key > spec key),
go to FINISH on page 381
• Since the new key is not beyond the stopping key,
If COMPCODE = 'MB', 'B', 'M',
go to FINISH on page 381
• If a matched pair is not yet found, find the next pair in the index. Also put the page in the buffer and validate the information.
Finds the next pair in the index and validates the information
ARIYX121
May Call: To:
ARIYI19 - Get a page into real storage.
ARIYI46 - Unfix a page/block.
ARIYK00 - Release a lock.
ARIYK41 - Lock a page.
ARIYX21 - Search the index from the root page.
ARIYM02 - Display error messages and ABEND SQL/DS.
If I/O error or backup occurs,
go to RECOVER on page 384
Abends SQL/DS
If any errors occur,
ARIYM02
May Call: To:
ARICABE - Handle SQL/DS ABEND.
ARICTRM - Terminate SQL/DS.
FINISH:
This routine returns the pair under the cursor.
If logical locking is required:

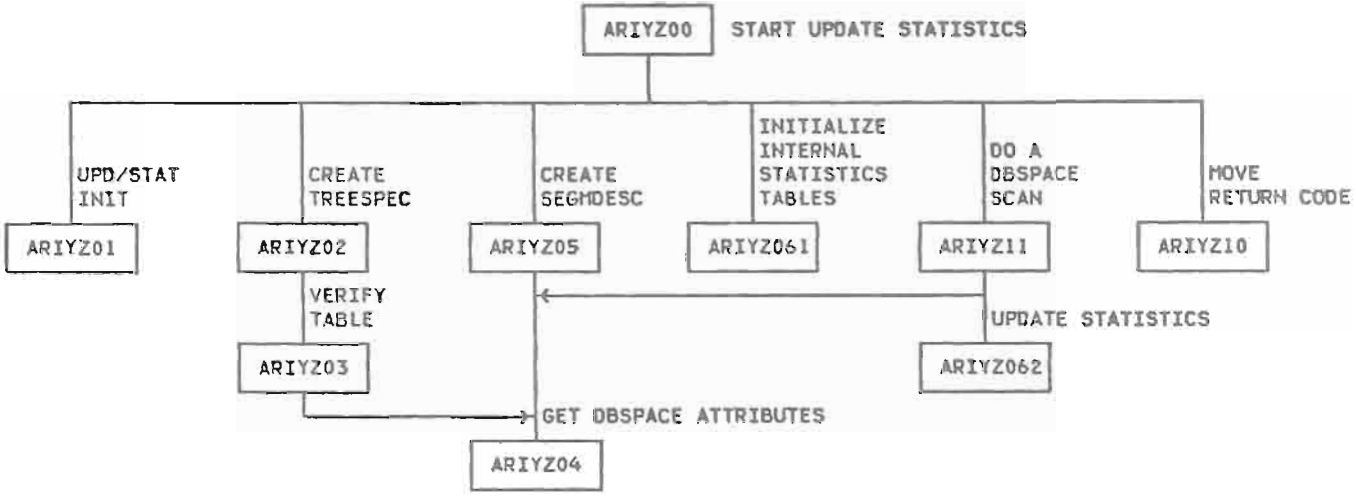



CCPFIN:
This routine returns the current cursor position.
If there is no key value locking or page locking,
Convert key from short lock to long lock
ARIYK41
May Call: To:
ARIYK00 - Release access to the designated gate.
ARIYK04 - Wait for access to X-latch.
ARIYK05 - Release access to X-latch.
ARIYK12 - Test for deadlock.
ARIYK18 - Wait to be posted.
SQL/DS Lock Manager Wait Routine
ARIYK19 - Post the designated process.
SQL/DS Lock Manager Post Routine
ARIYK22 - Free the locks.
ARIYT23 - Set the rollback/commit flags.
if backup required,
go to RECOVER on page 384
Display error messages and ABEND SQL/DS.
if RETCODE < SMODE,
ARIYM02
May Call: To:
ARICABE - Handle SQL/DS ABEND.
ARICTRM - Terminate SQL/DS.
Gets page into storage
ARIYI19
May Call: To:
ARIYK18 - Suspend the process.
ARIYI171 - Request a free buffer.
ARIYM02 - Display error messages; ABEND SQL/DS.
ARIYI07 - Write out the changed page.
ARIYI49 - Read a log page.
ARIYI29 - Read a logical page.
If I/O error occurs,
go to RECOVER on page 384
Display error messages and ABEND SQL/DS.
If unfix did not occur (RETCODE ≠ 0),
ARIYM02
May Call: To:
ARICABE - Handle SQL/DS ABEND.
ARICTRM - Terminate SQL/DS.
Unfix page and block in buffer
If SCB version is not the same as the page version,
ARIYI146
May Call: To:
ARIYM02 - ABEND SQL/DS.

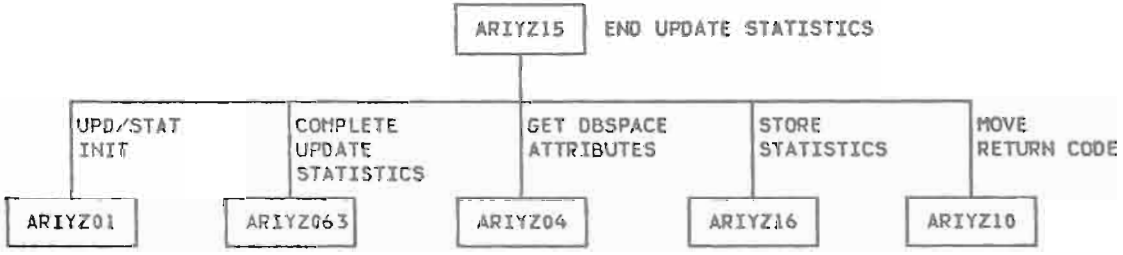


DBSS STATISTICS - GENERAL FLOW

START UPDATE STATISTICS



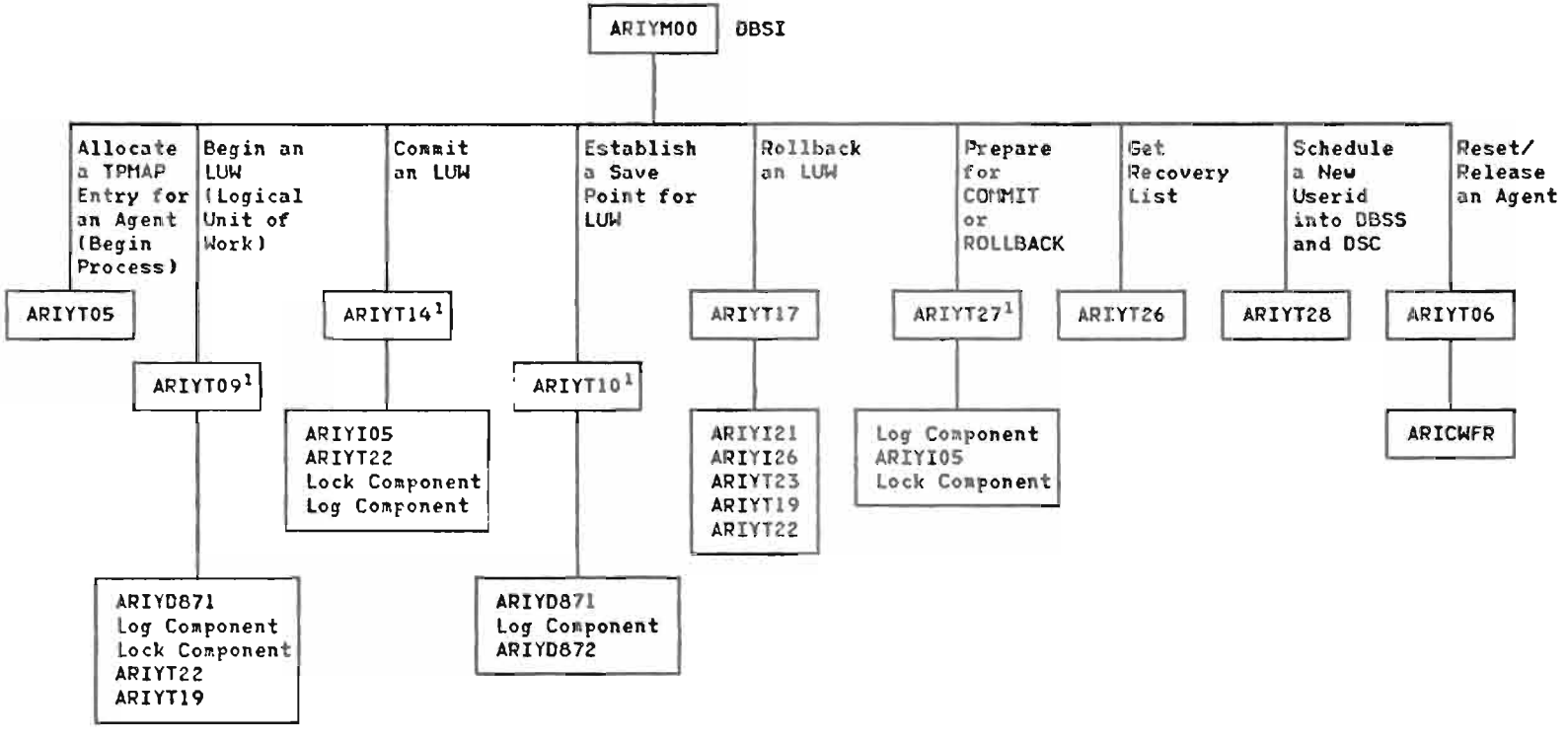
END UPDATE STATISTICS



DBSS WORK - GENERAL FLOW

In this diagram, only the major modules of DBSS Work and those called by ARIYM00 are shown. For the details, see page:
- 387 for Allocate a TPMAP Entry (ARIYT05)
- 388 for Begin an LUW (ARIYT09)
- 390 for Commit an LUW (ARIYT14)
- 392 for Establish an Save Point (ARIYT10)
- 393 for Rollback an LUW (ARIYT17)
- 394 for Prepare for COMMIT or ROLLBACK (ARIYT27)
- 397 for Reset/Release an Agent (ARIYT06)

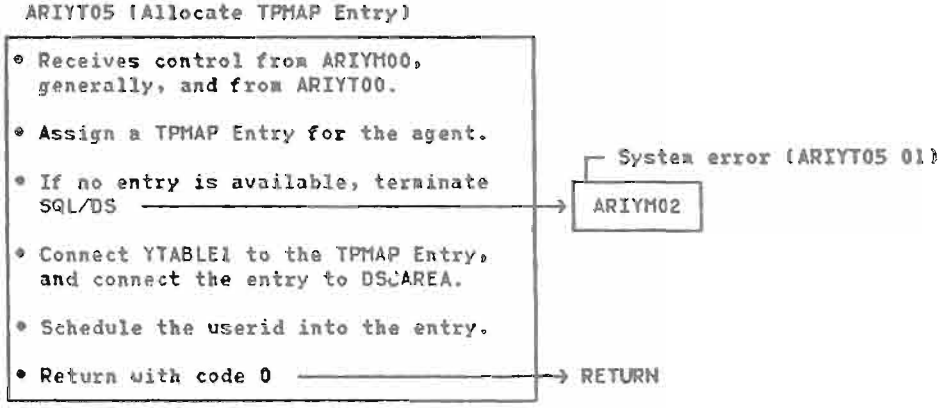
For Build In-Doubt Agents (ARIYT29 - not shown in this diagram), see page 398. ARIYT29 is called by ARIYL13 (see pages 271 and 413).



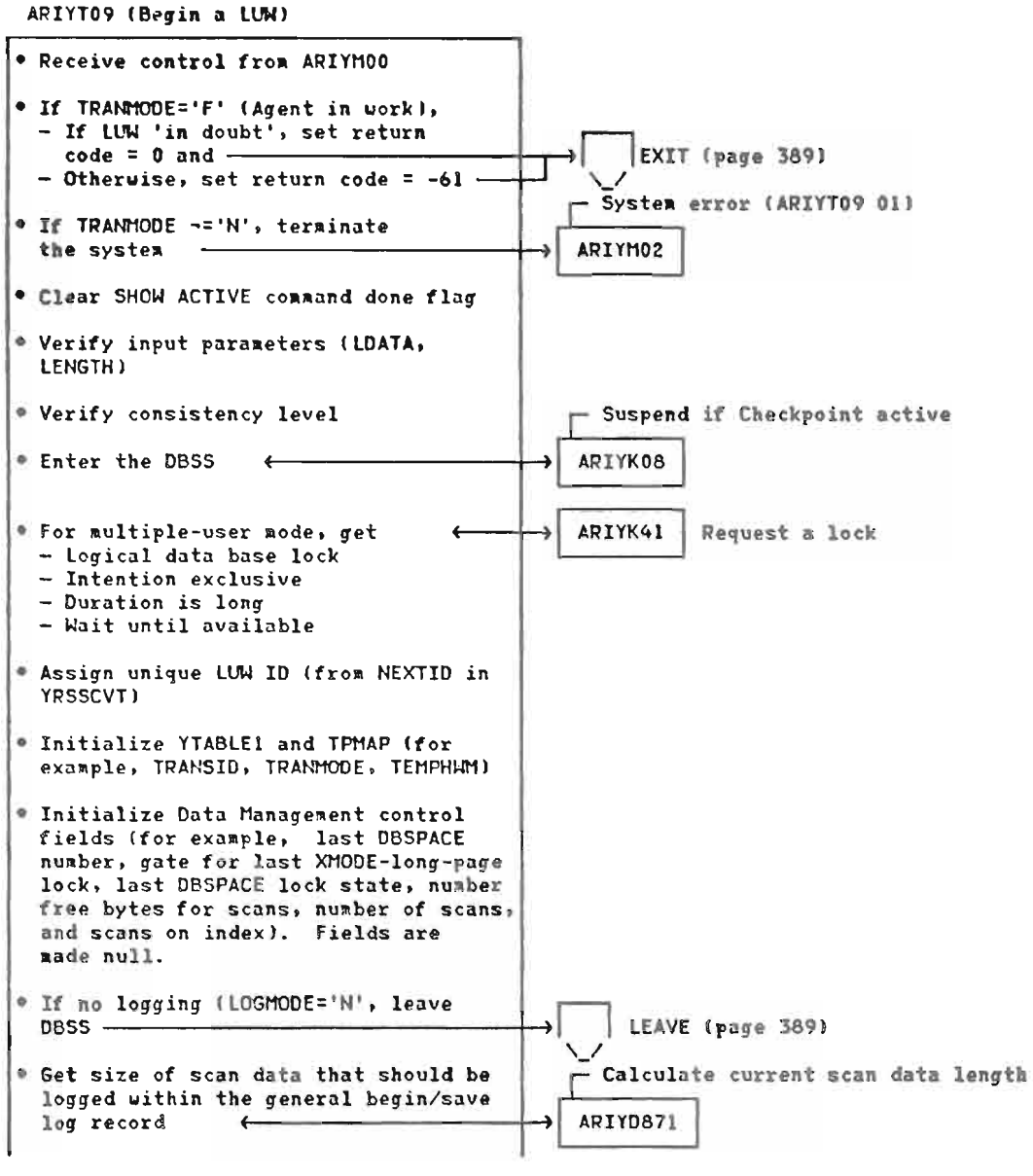
[1] Access ARIYK08, ARIYK09, and ARIYM02

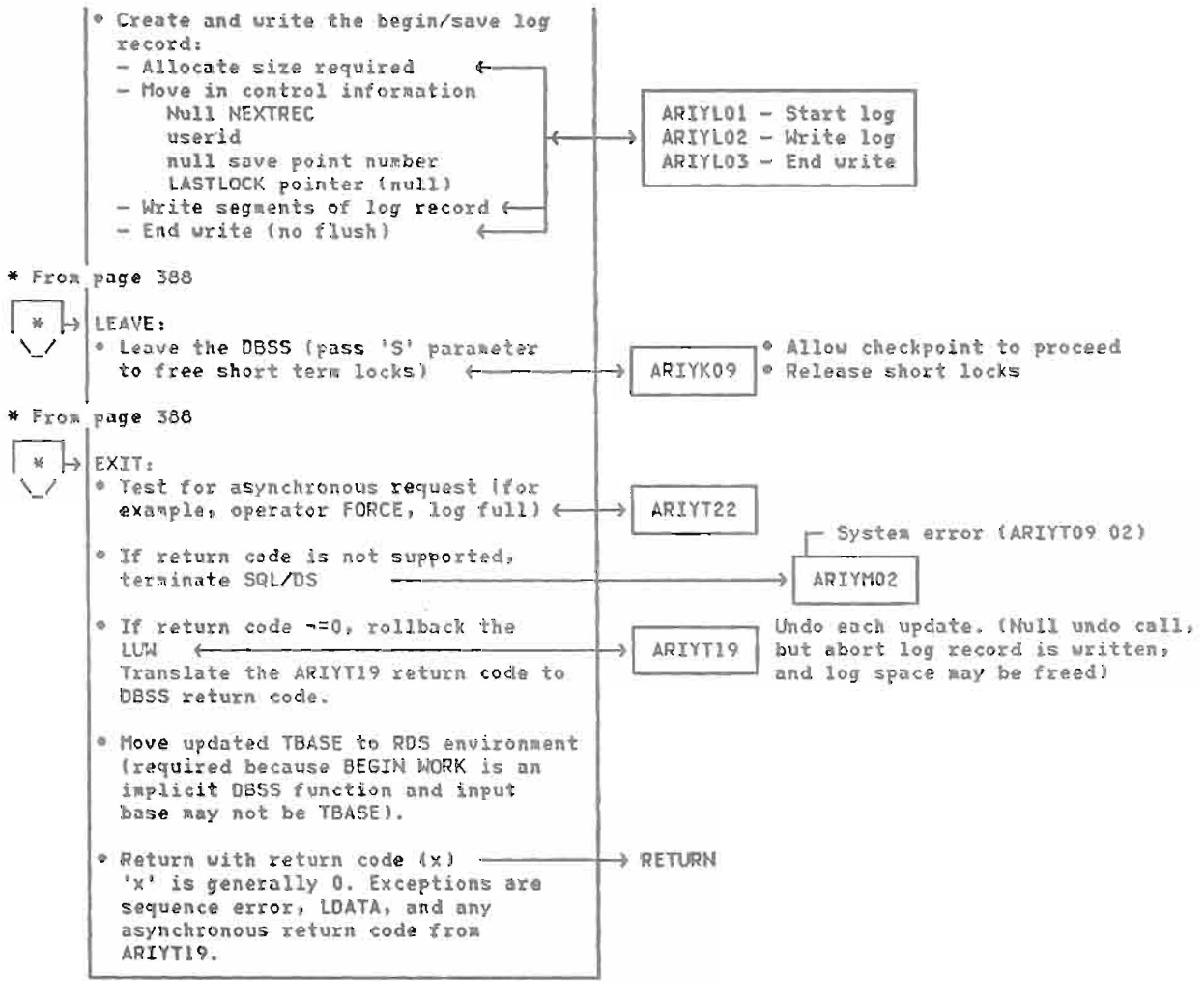
DBSS WORK - DETAILS

Allocate a TPMAP Entry (Begin Process)

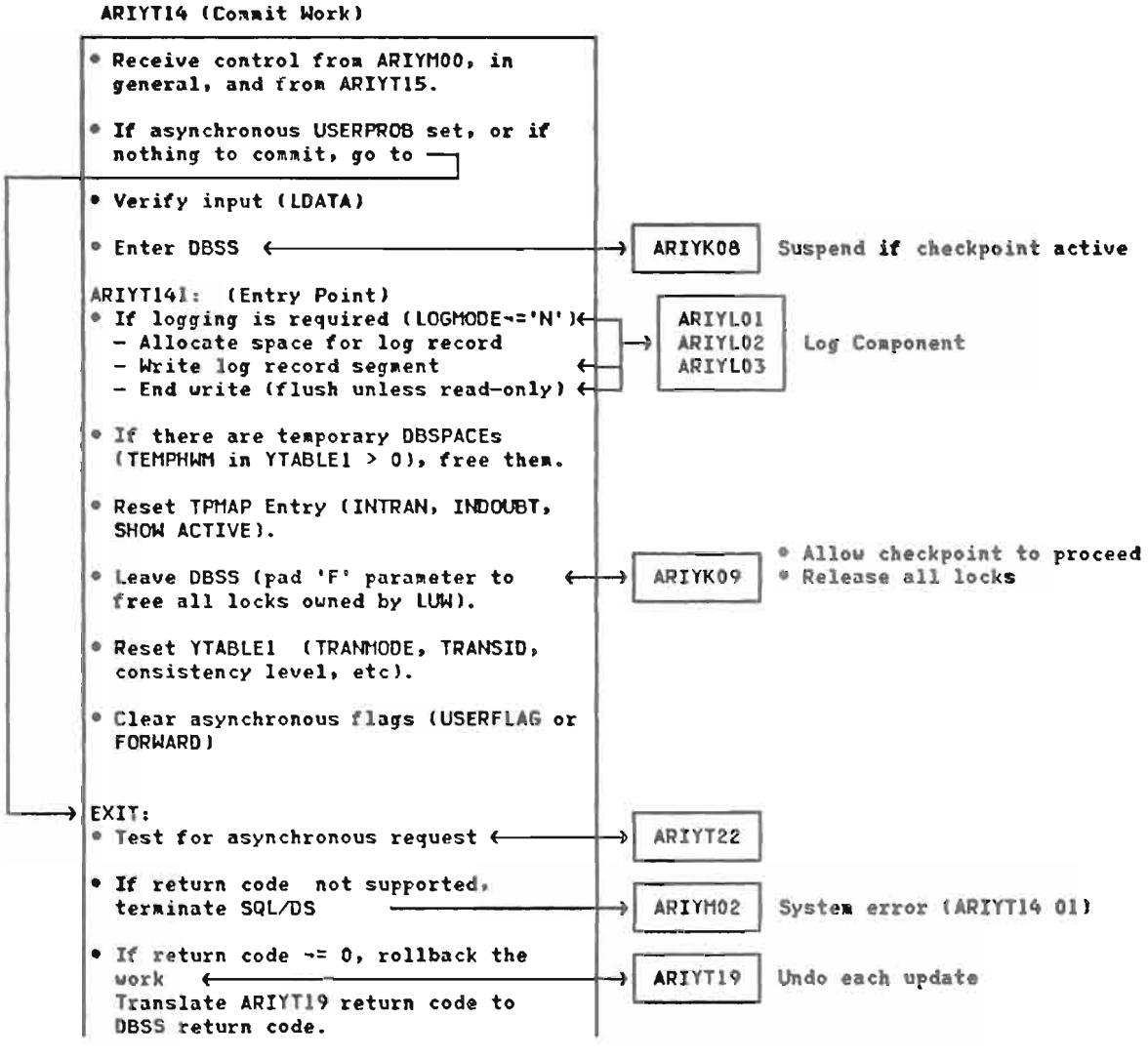


Begin a Logical Unit of Work (LUW)

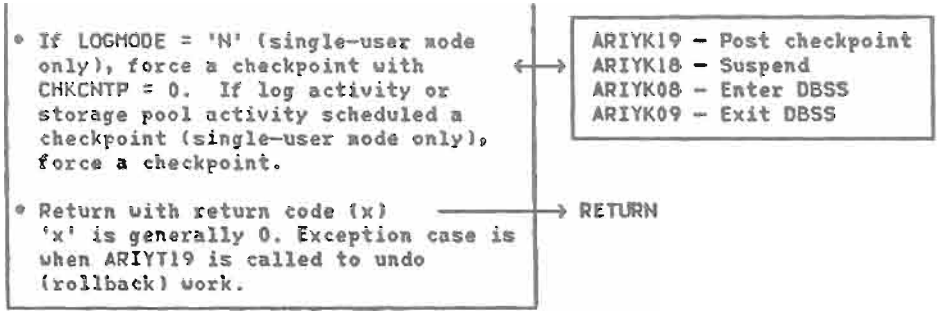



* Create and write the begin/save log record:
- Allocate size required
- Move in control information
Null NEXTREC
userid
null save point number
LASTLOCK pointer (null)
- Write segments of log record
- End write (no flush)
ARIYL01 - Start log
ARIYL02 - Write log
ARIYL03 - End write
* From page 388
*
LEAVE:
* Leave the DBSS (pass 'S' parameter to free short term locks)
ARIYK09
* Allow checkpoint to proceed
* Release short locks
* From page 386
*
EXIT:
* Test for asynchronous request (for example, operator FORCE, log full)
ARIYT22
* If return code is not supported, terminate SQL/DS
System error (ARIYT09 02)
ARIYM02
* If return code ¬=0, rollback the LUW
Translate the ARIYT19 return code to DBSS return code.
ARIYT19
Undo each update. (Null undo call, but abort log record is written, and log space may be freed)
* Move updated TBASE to RDS environment (required because BEGIN WORK is an implicit DBSS function and input base may not be TBASE).
* Return with return code (x)
'x' is generally 0. Exceptions are sequence error, LDATA, and any asynchronous return code from ARIYT19.
RETURN

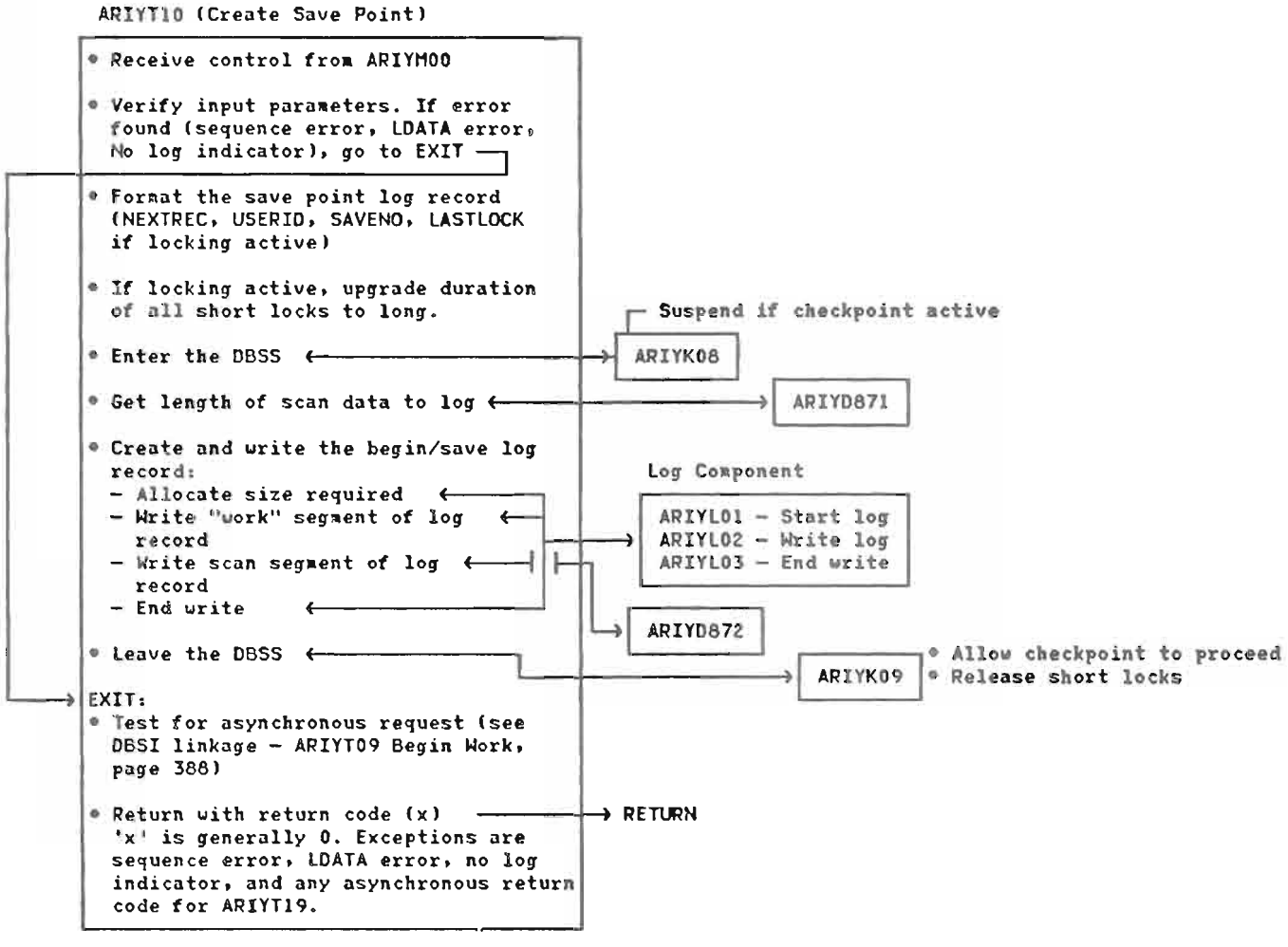
Commit an LUW (Commit Work)



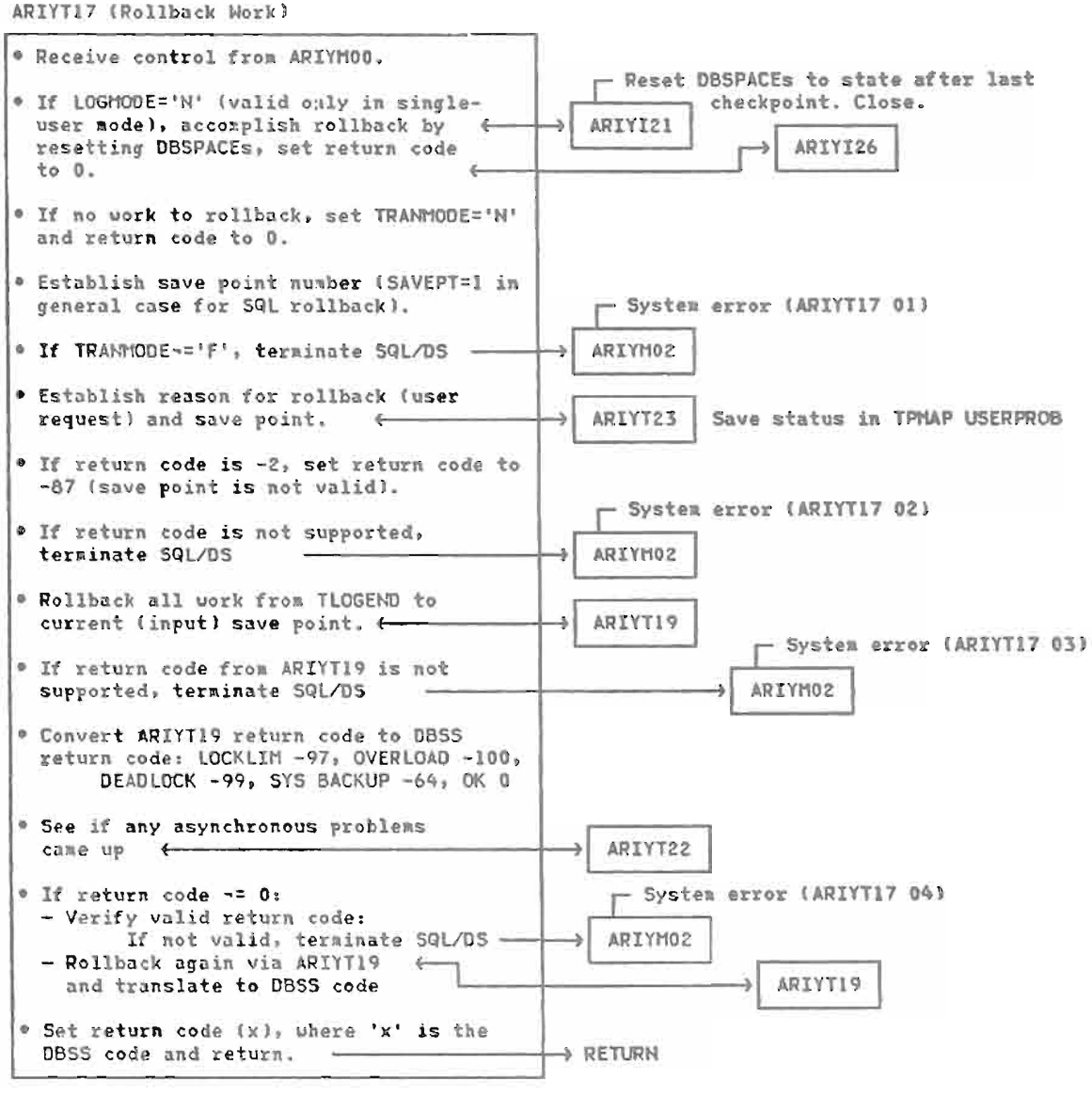
DBSS Work (continued)



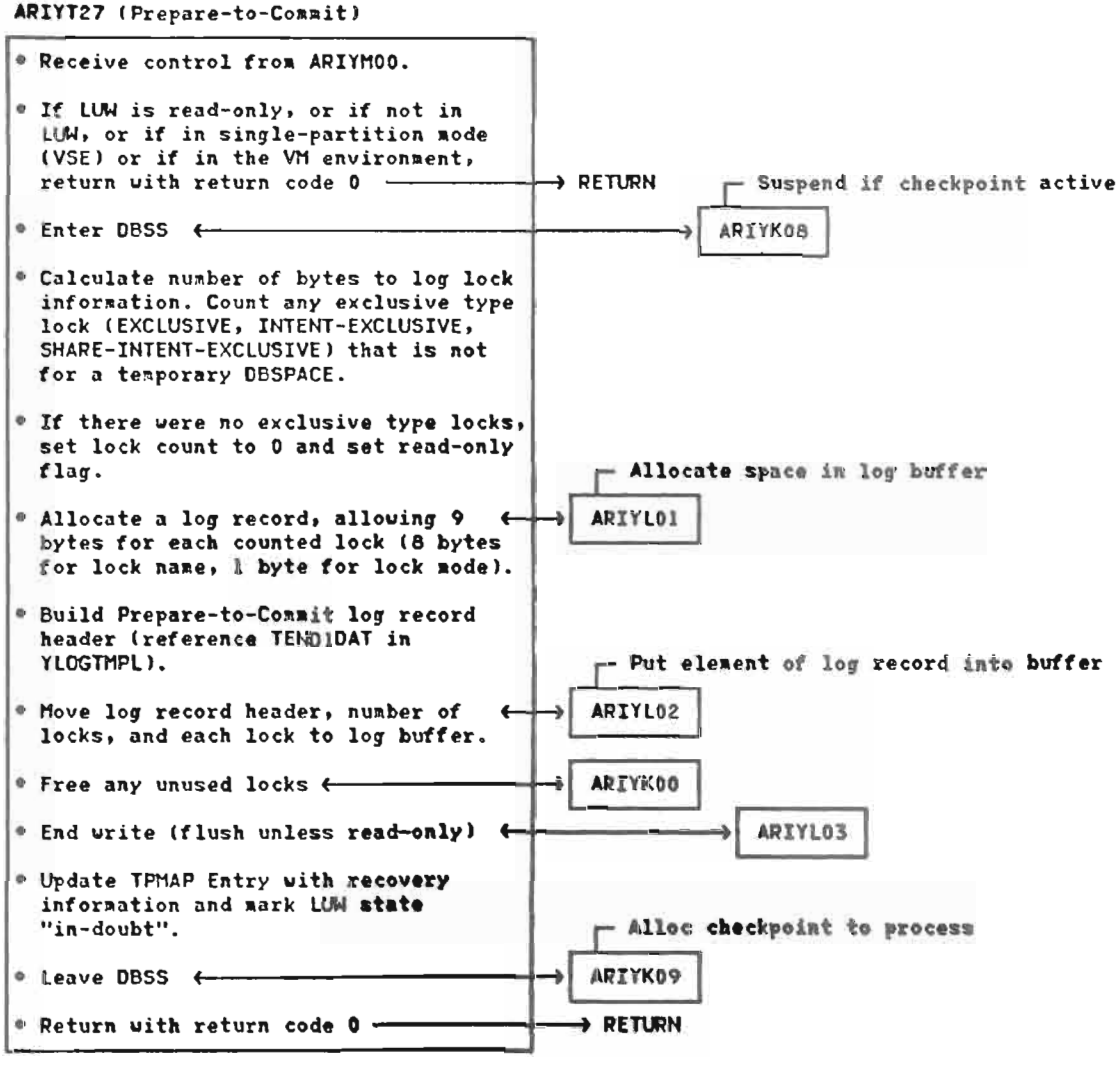
Create (Establish) a Save Point



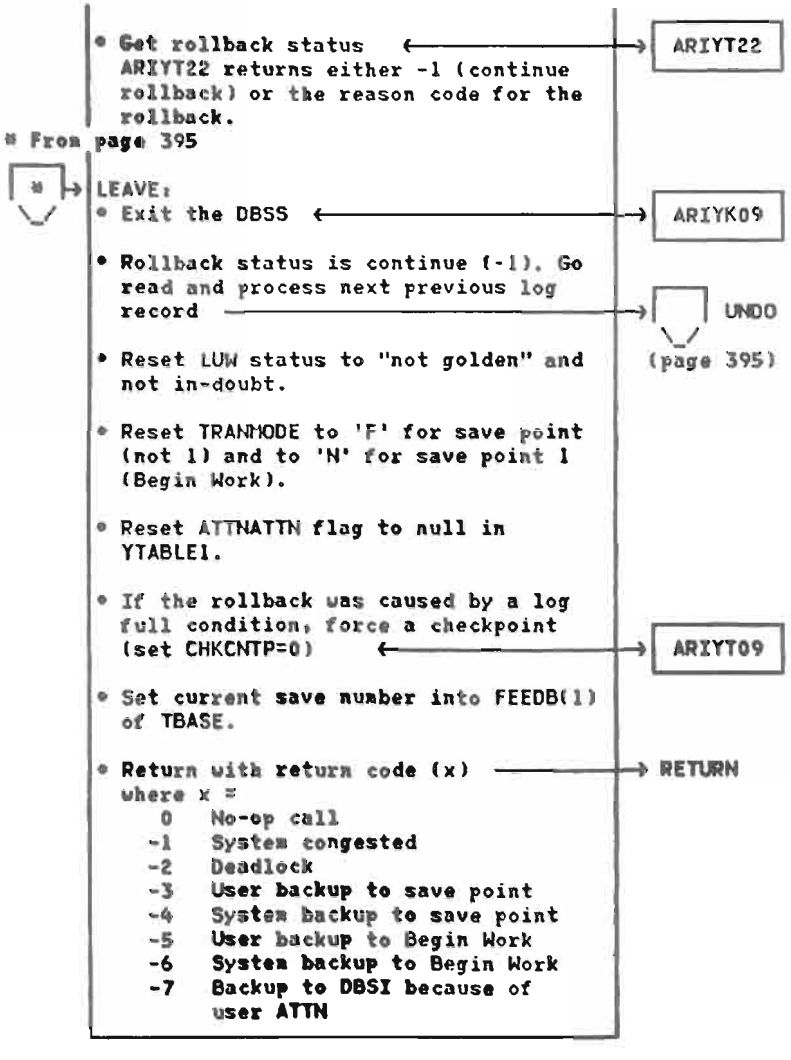
Rollback Work



Prepare to Commit

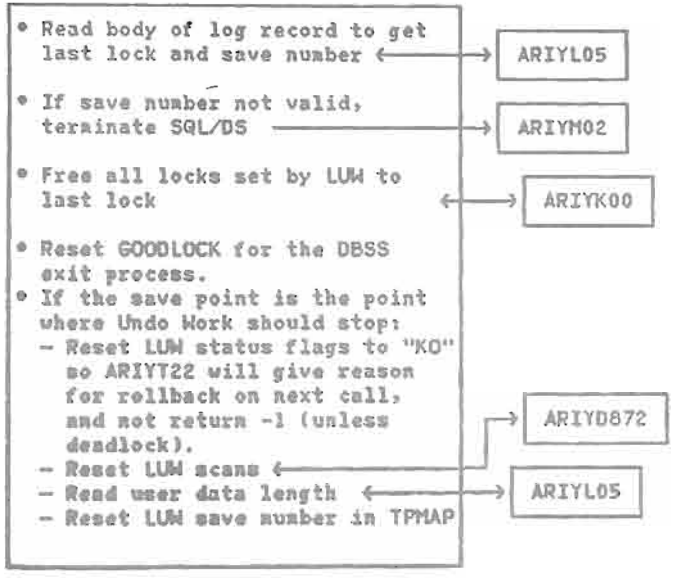


Undo Work

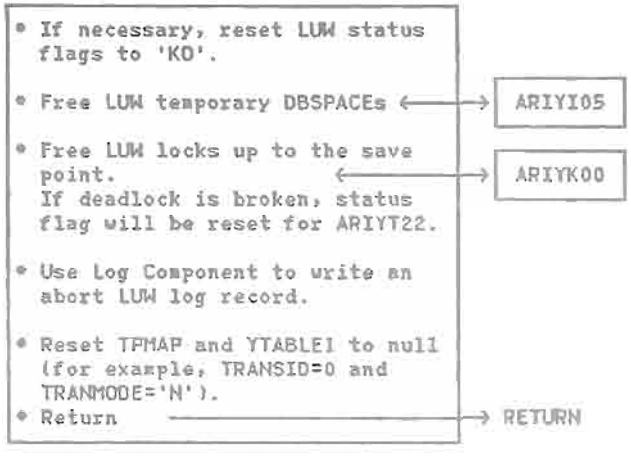


DBSS Work (continued)

ARIYT21 (Undo Begin/Save)

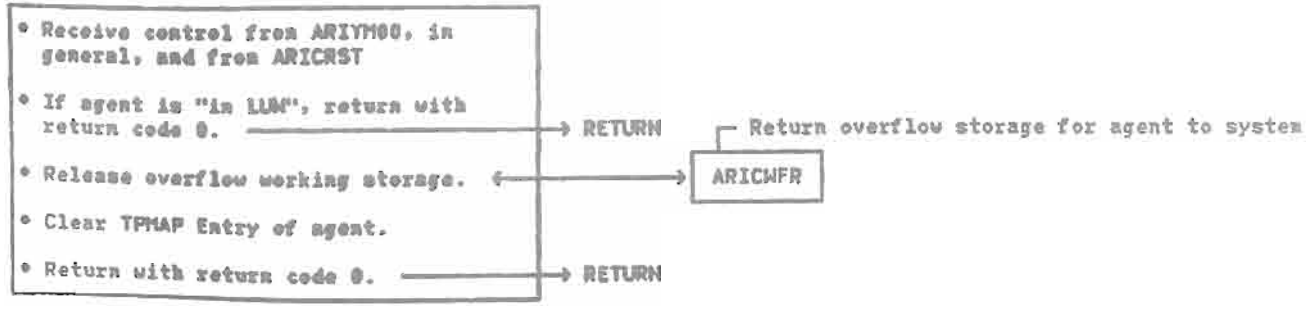


ARIYT20 (Abort LUW)

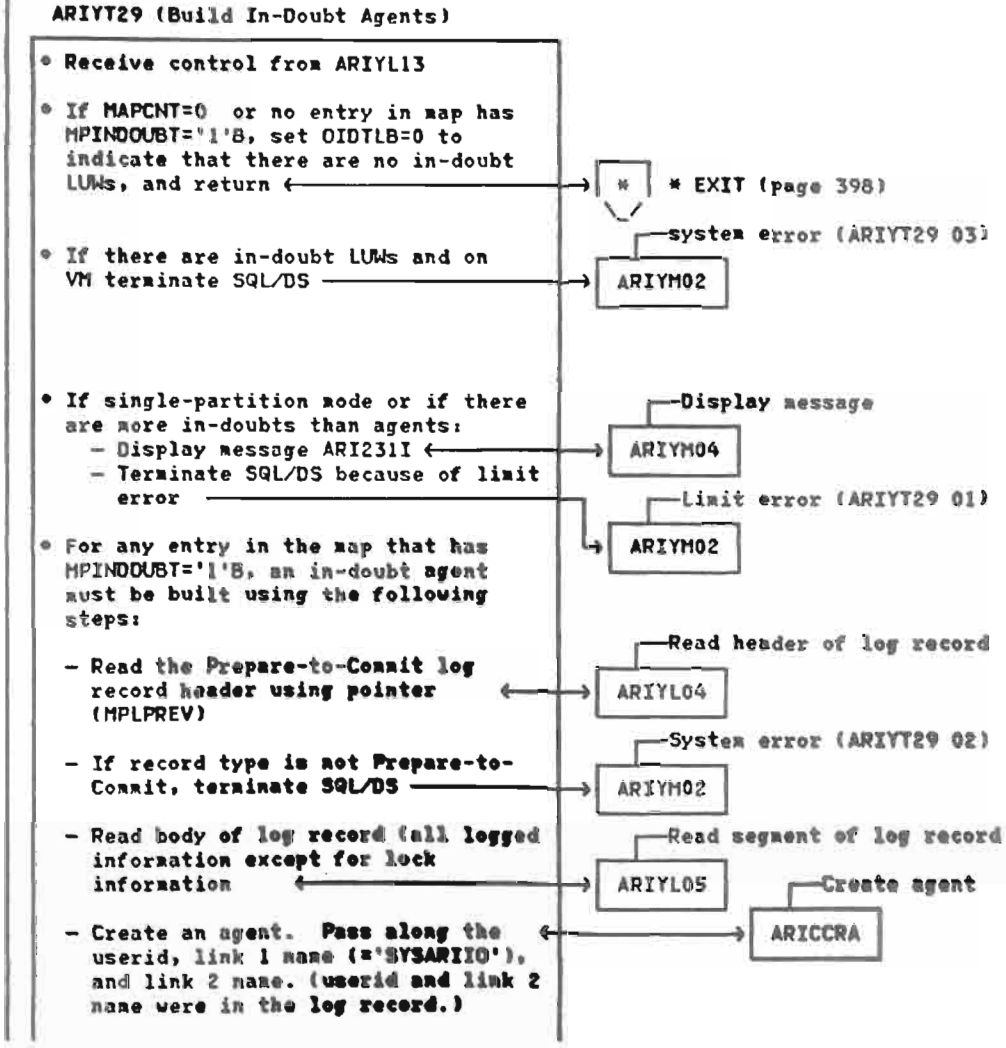


Reset/Release an Agent

ARIYT06 (Reset/Release)

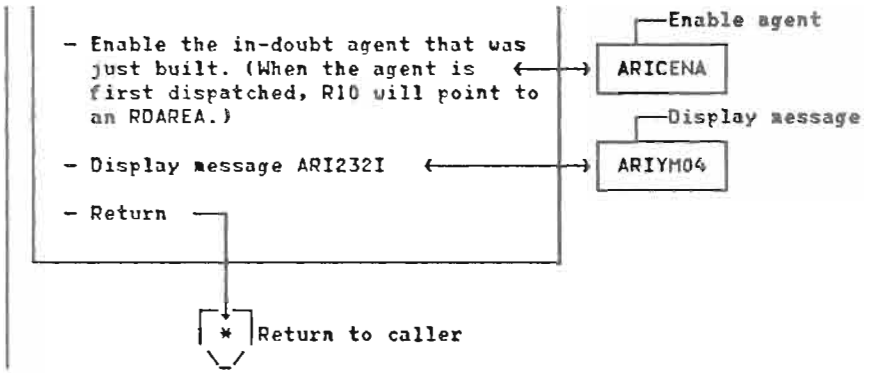


Build In-doubt Agents



DBSS Work (continued)

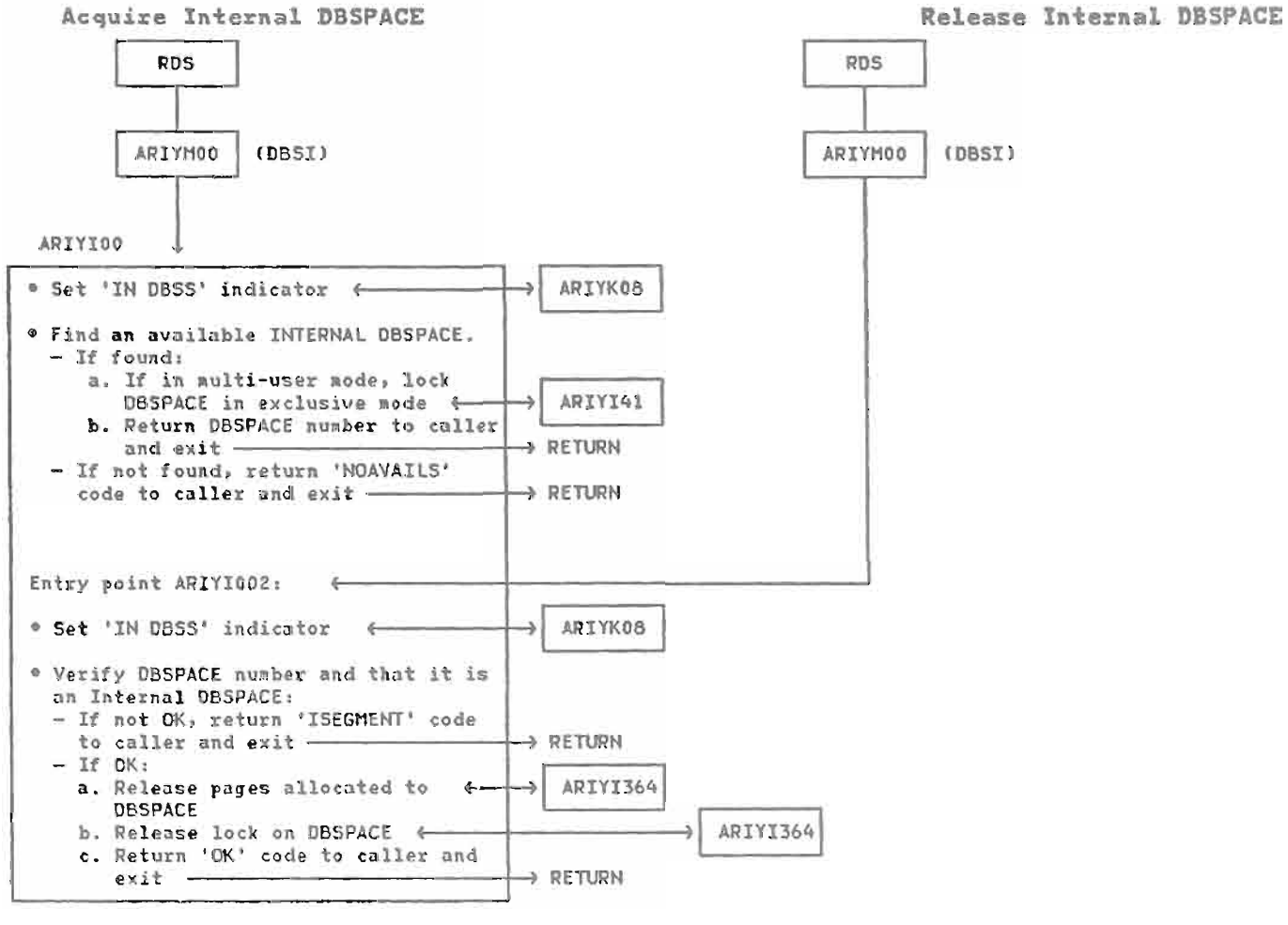
- Set pointer to DSCAREA of new agent (output of ARICCRA)
- If this is first in-doubt agent, set DS2USAG1. This sets the value in DS2CVT, which indicates WHOAMI of first non-system agent.
- Set WHOAMI and WHOAMIP for the new agent structure.
- From the log record, rebuild the YTABLE1 and TPMAP entry of the new agent. Many fields are set and enough structure is built so that SQL/DS can eventually do a COMMIT WORK or ROLLBACK WORK for the LUW.
- If this is the oldest in-doubt agent, reset IODTLB. (OIDTLB will be set to the pointer of the Begin Work log record of the oldest in-doubt LUW.)
- Read the number of logged locks ↔ ARIYL05 (Read segment of log record)
- For each logged lock, read the lock name and mode (Exclusive or Share) from the log. Then go to lock manager to obtain the lock. ↔ ARIYL41 (Get lock)
  (Note: for call to lock manager, SYSMODE must be set to 'M' or lock request call is a NOP.)
- If any request for a lock fails, terminate SQL/DS → ARIYM02 (System error (ARIYT29 04))
- Terminate reading of the log record ↔ ARIYI46 (Unfix page containing the log record)
- If unfix failed, terminate SQL/DS → ARIYM02 (System error (ARIYT29 05))
- Reset WHOAMI and WHOAMIP to the initialization (operator) agent structure



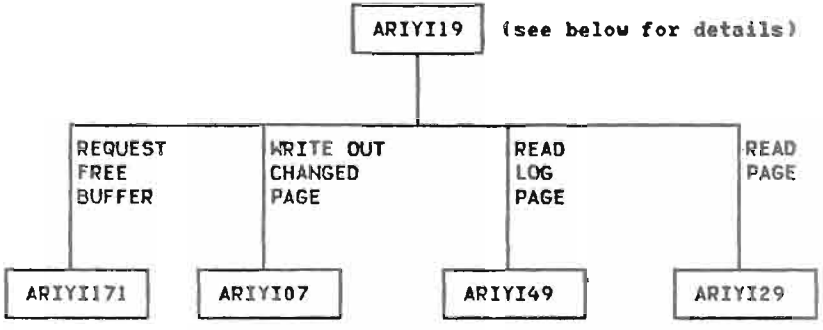
DBSS STORAGE

See page 401 for Acquire/Release Internal DBSPACE
See page 402 for Get Page
See page 404 for Get Block

ACQUIRE/RELEASE INTERNAL DBSPACE



GET PAGE



* (ARIYI19 is called from many modules. See the Module-to-Module Cross Reference in Volume 3.)

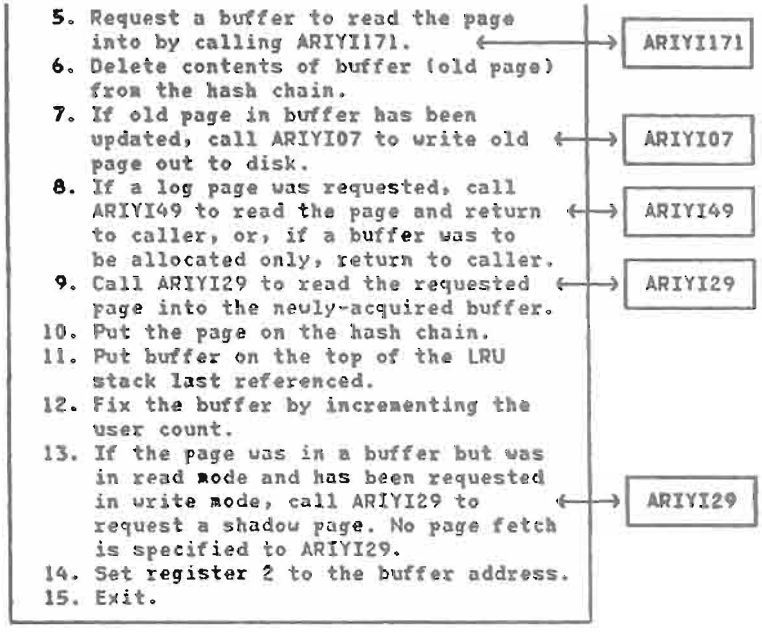
ARIYI19 (GET PAGE)

ARIYI19 gets a requested page into a page buffer for access. The page number is passed as input and is expressed as a logical page number starting with X'000080' as page zero. ARIYI19 is called by the Log component also to read a log page from either log, or from the primary log only, or from the secondary log only, or to allocate a page buffer to contain a new log page. ARIYI19 processing is as follows:

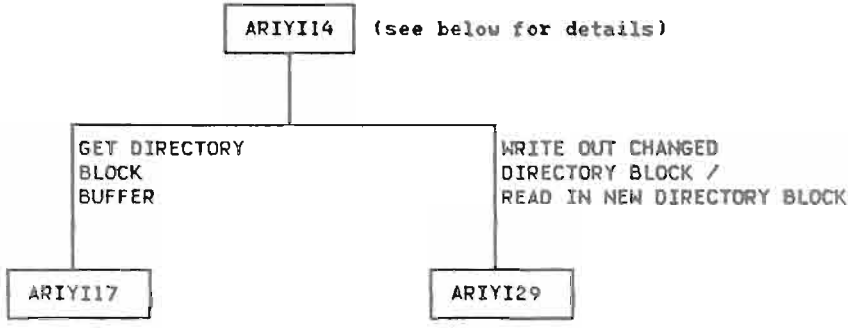
1. Check for valid page number.
   NO → RETURN (-2).
2. See if the page is the last referenced page. The pointer to the last referenced buffer is in YTABLE4.
   YES → GO TO 12.
3. Using the DBSPACE number and page number, compute hash code and search hash chain for page.
   FOUND → GO TO 12.
4. Scan the transit list to see if the page is in the process of being written out to disk.
   YES → Go back to the Dispatcher and try later.

Note: ARIYI19 will call ARIYM02 on a severe error.

DBSS Storage (continued)



GET DIRECTORY BLOCK



* (ARIYI14 is called from many modules. See the Module-to-Module Cross Reference in Volume 3.)

ARIYI14 (GET DIRECTORY BLOCK)

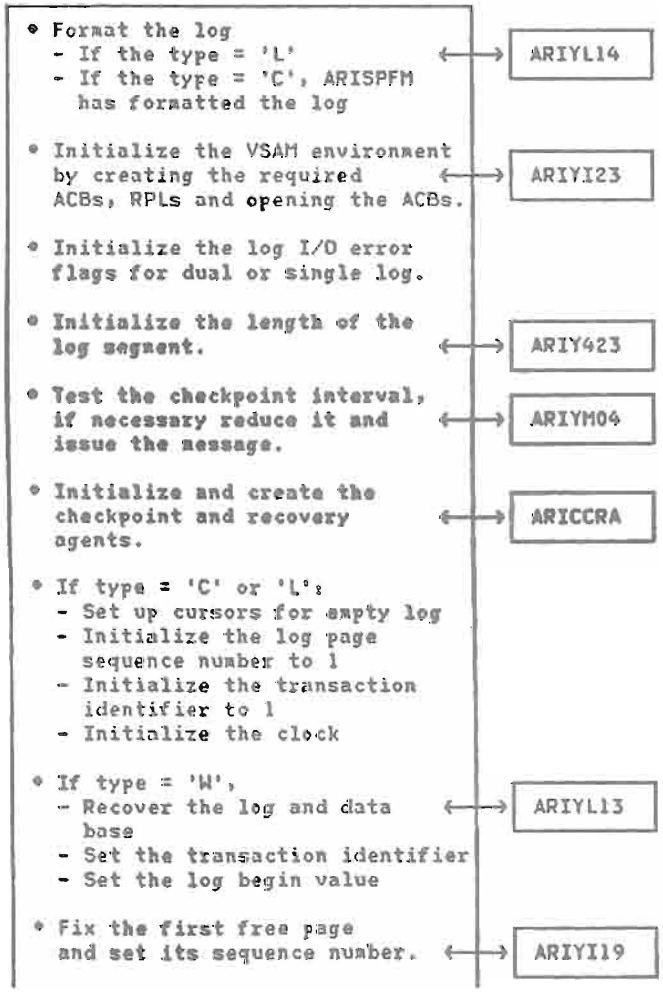
ARIYI14 gets the requested directory block into storage as follows:

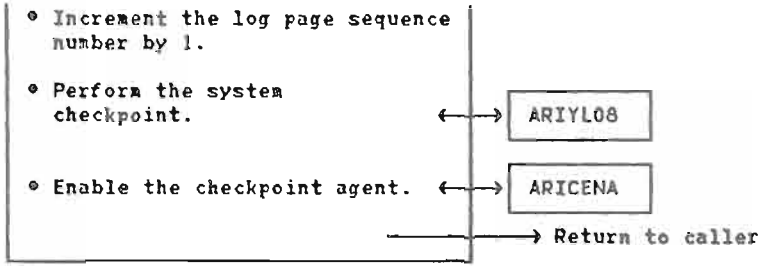
- Verifies valid directory block number.
- If multi-user mode, obtains directory block buffer contents lock.
- Sees if it is in the last-referenced buffer. If so, up fix count and exit.
- Search hash chain for buffer. Make it last-referenced buffer, up fix count and exit.
- Request a buffer to read in directory block. ⟷ ARIYI17
- Delete directory block in buffer from hash chain.
- Write out old directory block if it was changed. ⟷ ARIYI06
- Read directory block into buffer. ⟷ ARIYI06
- Add directory block to hash chain.
- Up fix count.
- If directory block is intended to be updated, indicate in MODMAP.
- Exit.

INITIALIZING THE LOG

Types of initialization:
- 'C' - cold start the data base
- 'L' - cold start the log
- 'W' - warm start the log

ARIYL00 (Called by ARIYT00)





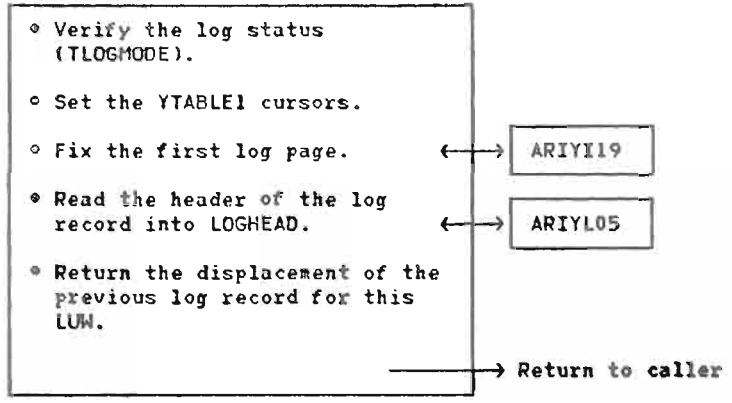
READING THE LOG

ARIYL04 and ARIYL05 are the modules needed to read the log.

Read Log Header and Status

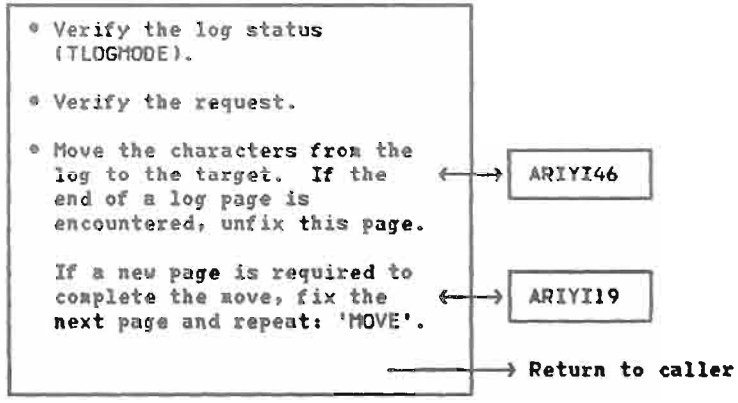
Read Log Header

ARIYL04 (Read the log header)
(Called by many DBSS modules
via YSTARTRD macro)



Read Log Status

ARIYL05 (Read the log status)
(Called by many DBSS modules
via YREADLOG macro)



WRITING THE LOG

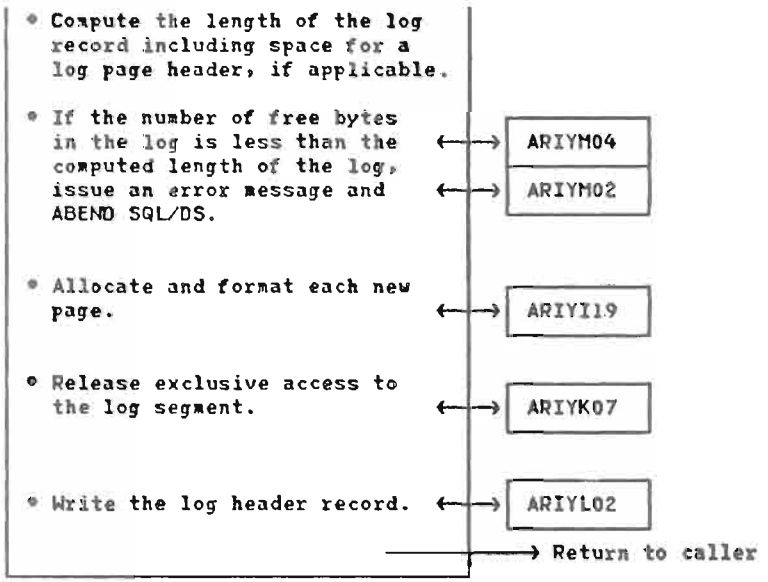
ARIYL01, ARIYL02, and ARIYL03 are the modules needed to write the log.

Write Log Header

ARIYL01 (Write the log header)
(Called by many DBSS modules
via YSTARTLG macro)

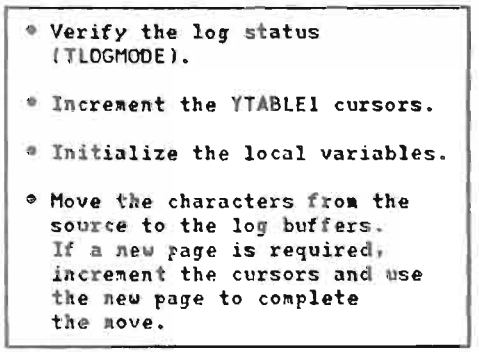
- Verify the log status (TLOGMODE).
- If the system mode is multiple users, get exclusive update access to the log segment. ⟷ ARIYK06
- Set the YTABLE1 cursors.
- Compute the number of free bytes in the log.
- If the log cushion (SLOGCUSH) is exceeded and an archive is not in progress, mark any transaction that begins in the first log page to be aborted. ⟷ ARIYT16
- If the log cushion (SLOGCUSH) is exceeded and an archive is in progress, turn on an indicator to warn the archive process (ARCHEX).
- If archiving is enabled, but an archive is not in progress or scheduled and if an implicit archive point (ARCHPCT) is exceeded, post the checkpoint agent unless a LUW began before the most recent checkpoint. ⟷ ARIYK19
- Compute the number of page boundaries crossed by writing this record.

DBSS Log/Recovery (continued)



Write Log Record

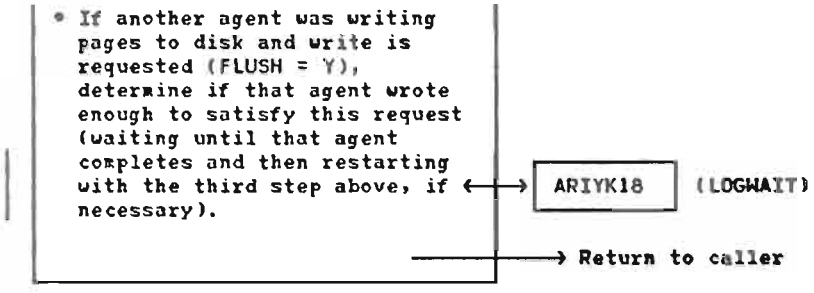
ARIYL02 (Write the log record)
(Called by many DBSS modules
via YWRITELG macro)



Terminate Writing Log Record

ARIYL03 (Terminate writing the log record)
(Called by many DBSS modules
via YENDWRIT macro)

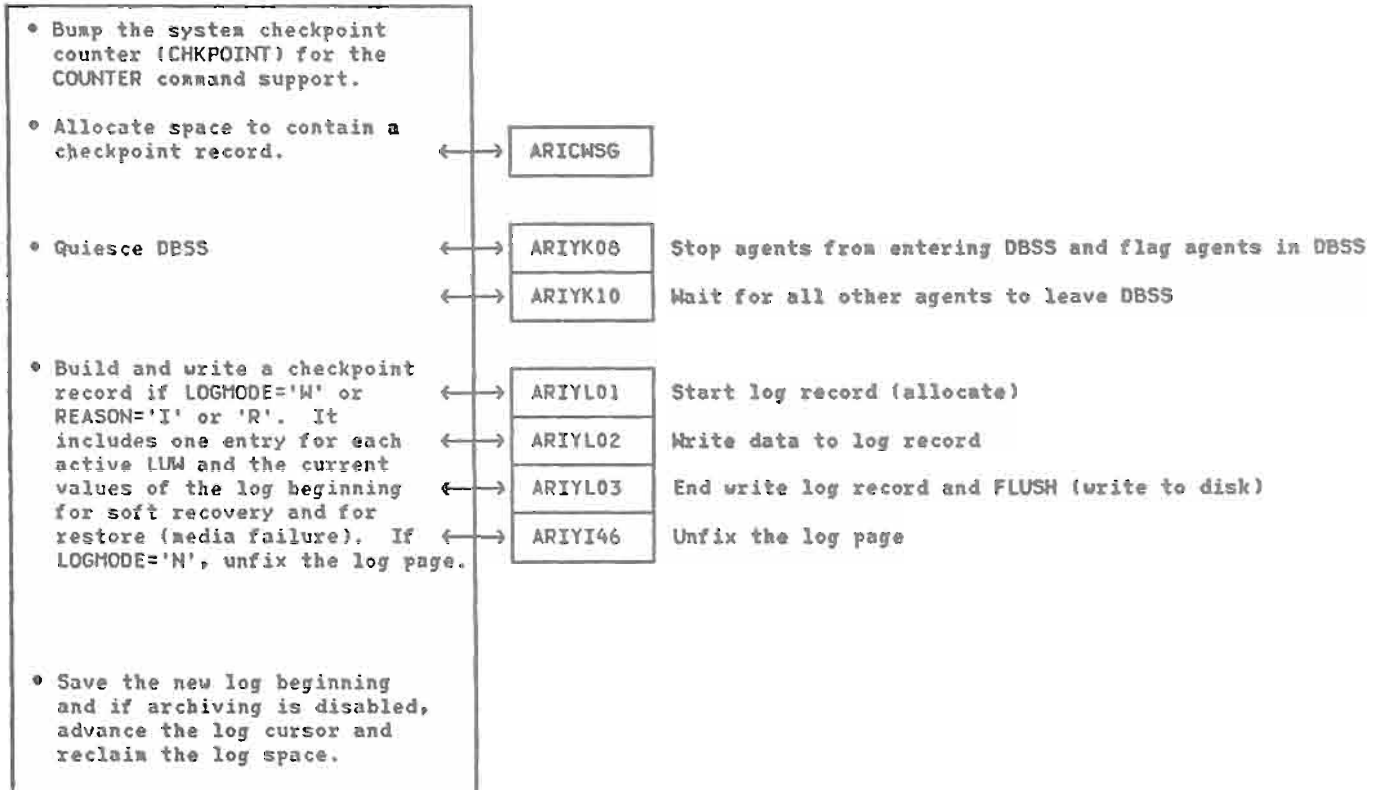
- Verify the log status (TLOGMODE, TLOGBYTE).
- If writing the log record to disk is requested (FLUSH = Y), compute the number of valid bytes in this page (TLOGWANT).
- Initialize the YTPMAP variables.
- Compute the maximum page and byte within that page that is completed by all LUWs.
- If there are no incomplete log records prior to this one, or if this record is not on the same page as the cursor pointing to the next log byte to be written (SLOGWRIT), or if writing is requested (FLUSH = Y), then the log buffer(s) should be written to disk now.
- Write all prior valid pages to disk, unless another agent is writing them. ⟷ ARIYI48
- Unfix each page as it is written. ⟷ ARIYI46
- If the checkpoint is needed, post the checkpoint agent. ⟷ ARIYK19
- Post each agent that was waiting for pages to be written. ⟷ ARIYK19
- If any agent wants the last page written, then write it. ⟷ ARIYI46

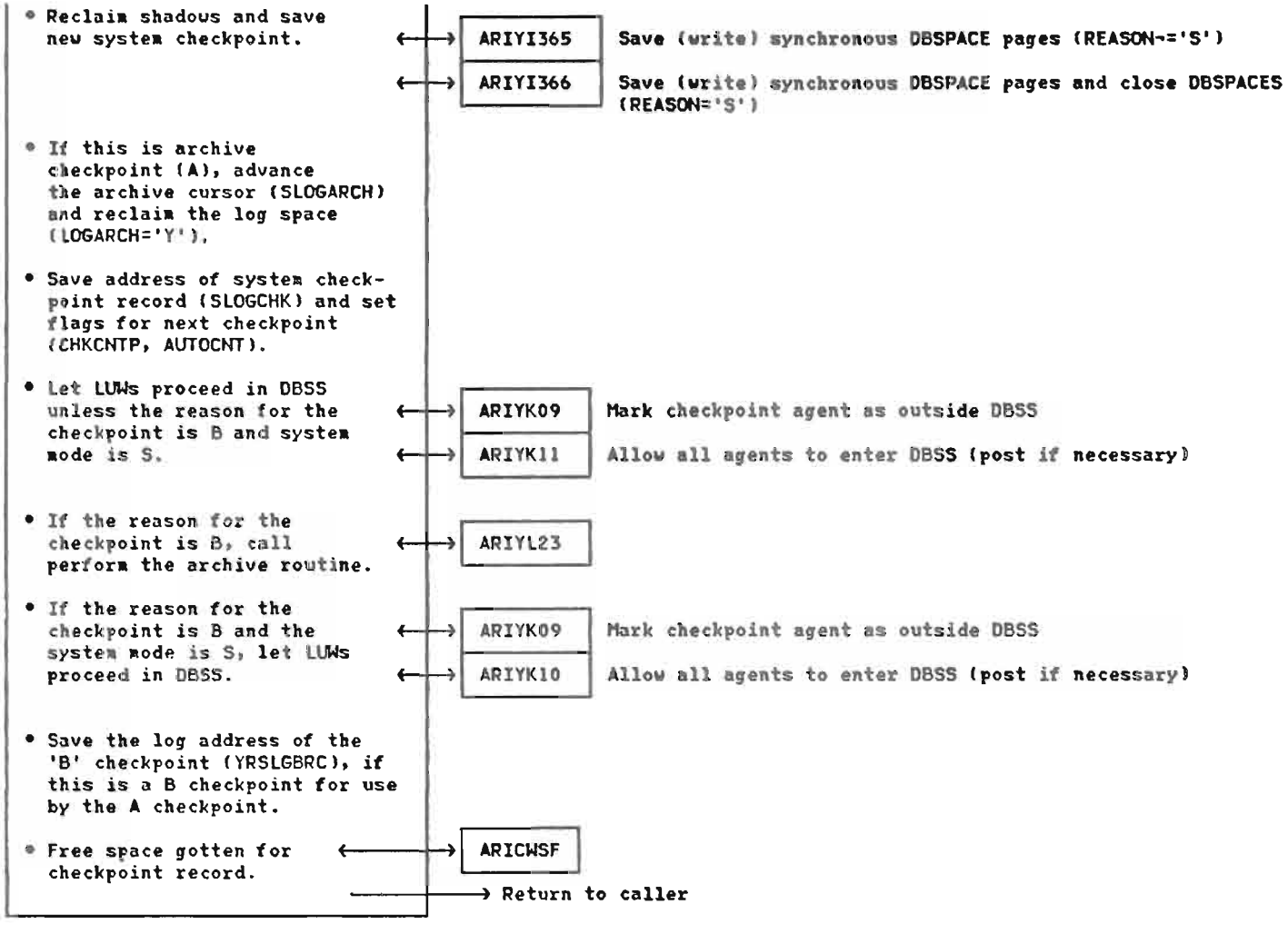


PERFORM CHECKPOINT

Reasons for checkpoint:
- I = Log just initialized
- R = System just recovered
- C = Periodic system checkpoint
- B = Archive of data base started
- A = Archive of data base completed
- S = System being shutdown

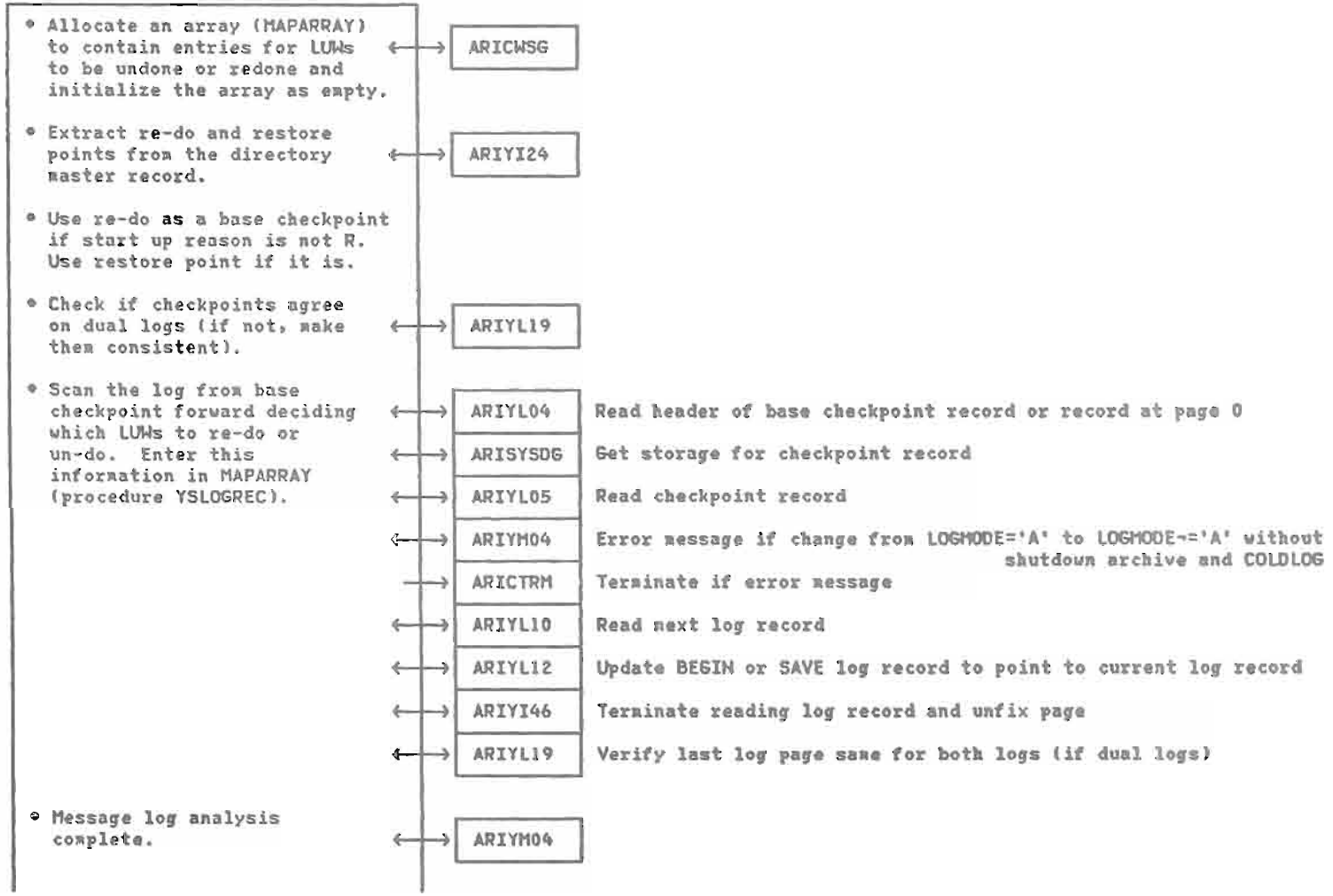
ARIYL08 (Performs the checkpoint)
(Called by ARISEGA, ARIYL00, ARIYL09)

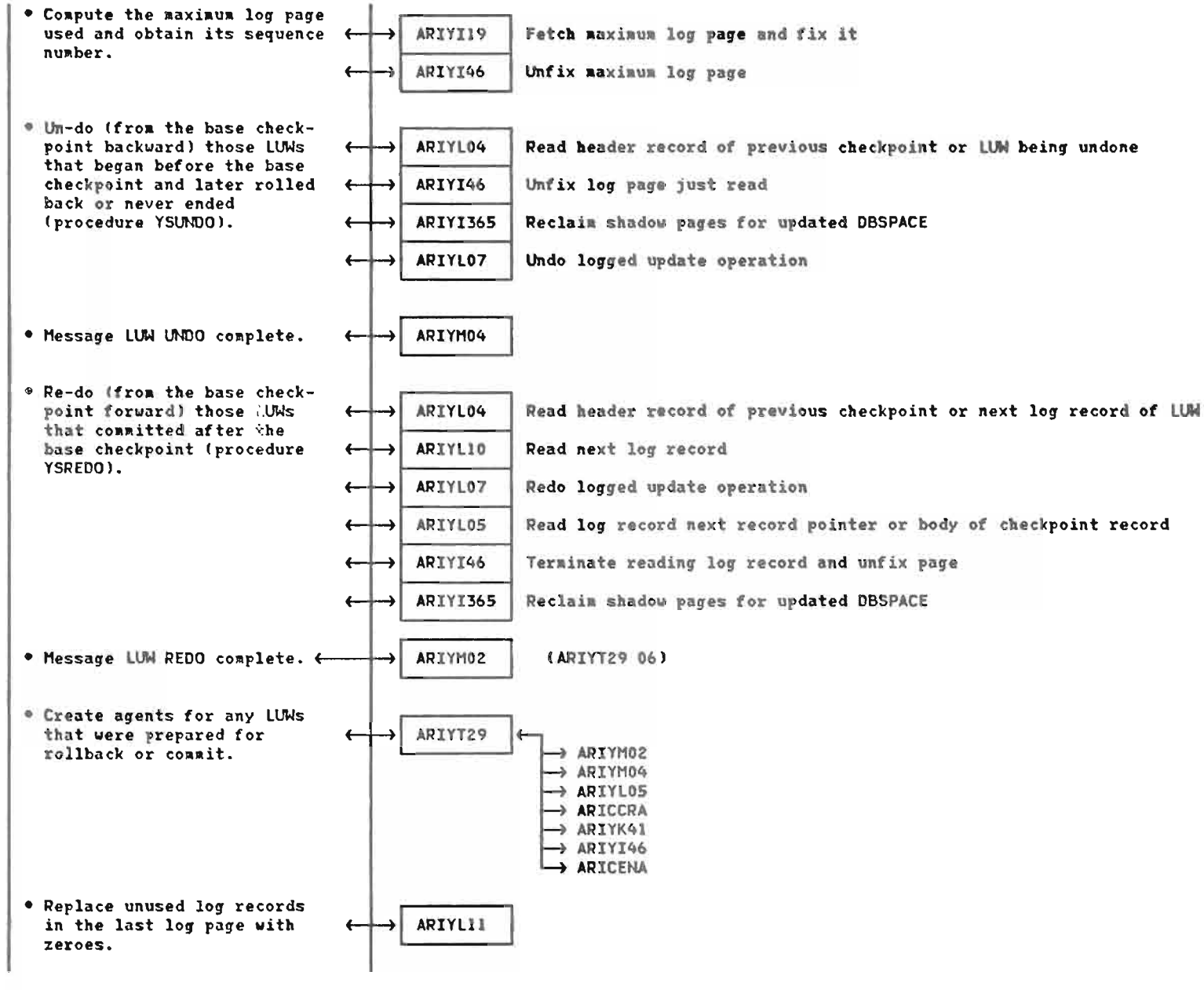




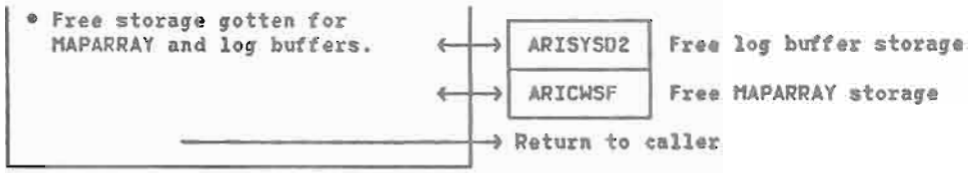
RECOVERY OF THE LOG

ARIYL13 (Recovery of the log)
(Called by ARIYL00)



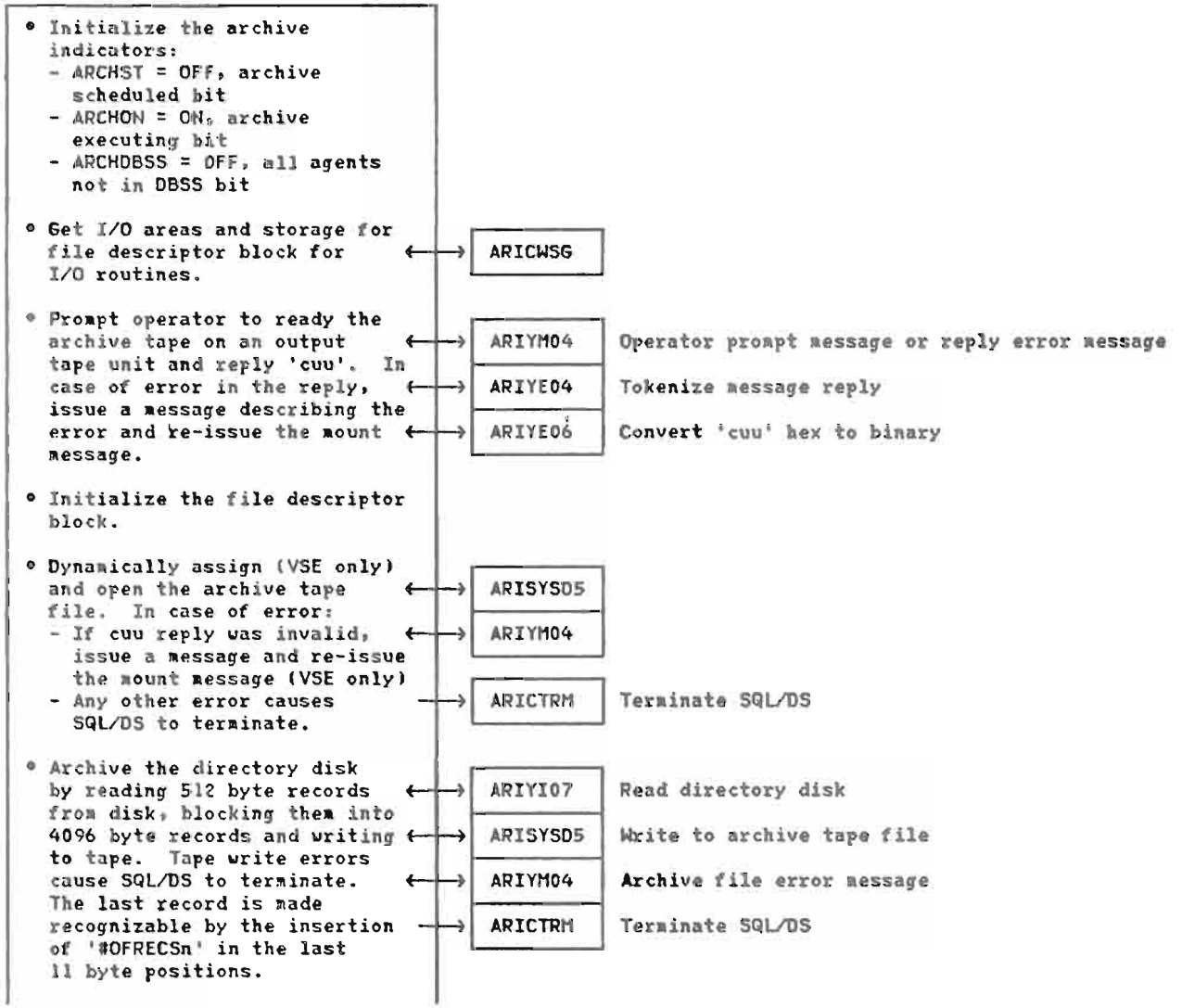


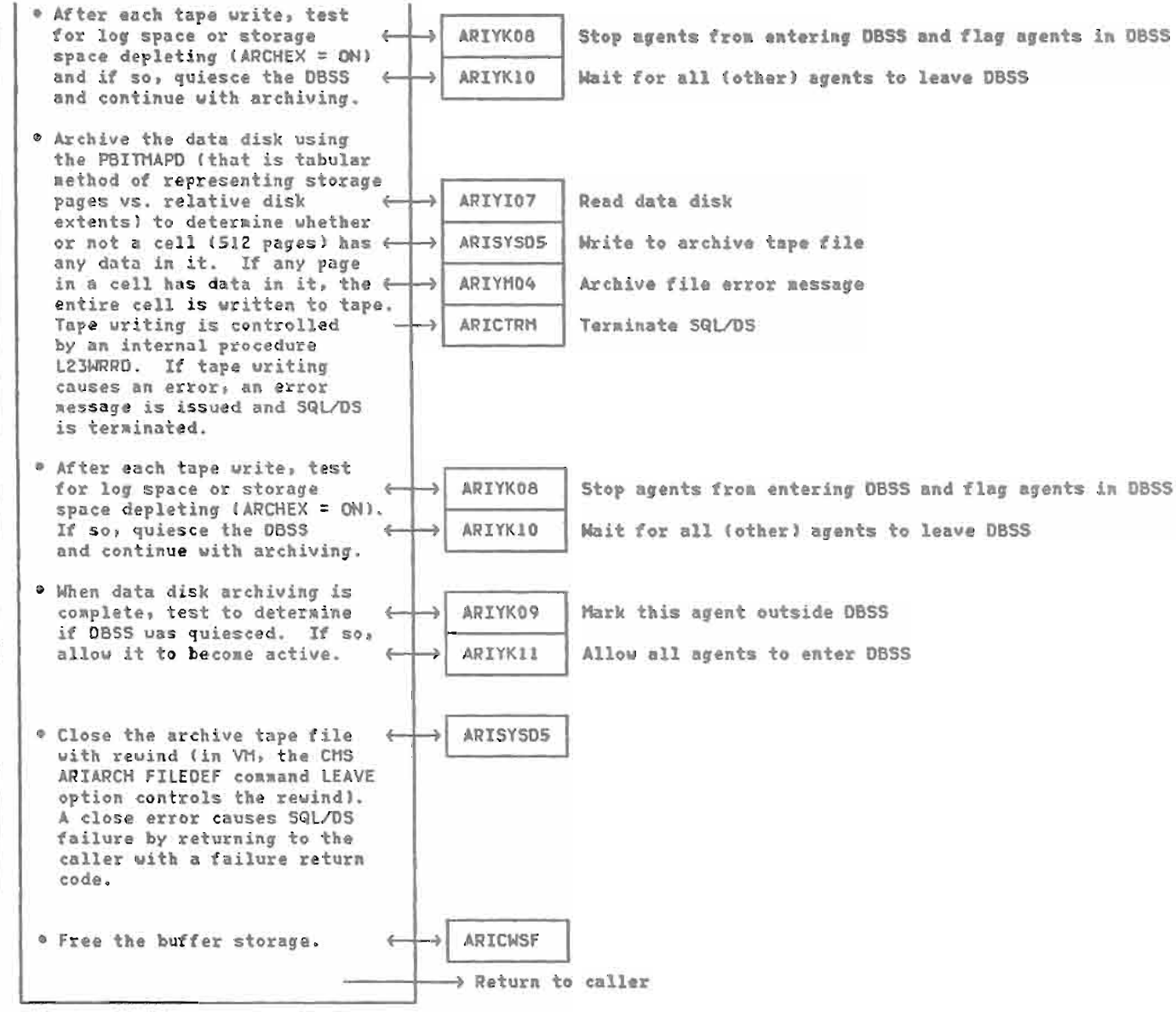
DBSS Log/Recovery (continued)



PERFORM ARCHIVE

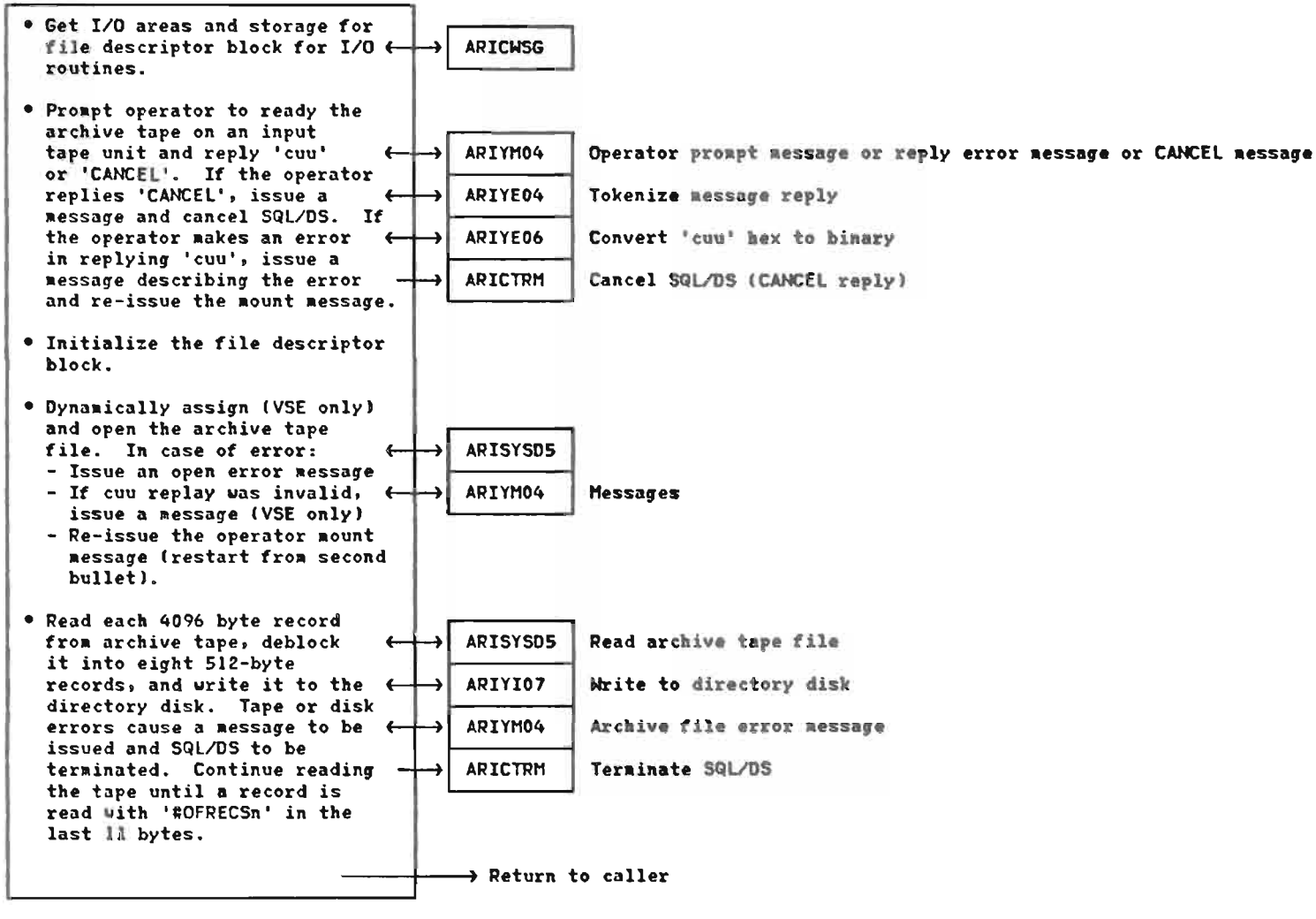
ARIYL23 (Perform archive)
(Called by ARIYL08)




• After each tape write, test for log space or storage space depleting (ARCHEX = ON) and if so, quiesce the DBSS and continue with archiving.
ARIYK08
Stop agents from entering DBSS and flag agents in DBSS
ARIYK10
Wait for all (other) agents to leave DBSS
• Archive the data disk using the PBITMAPD (that is tabular method of representing storage pages vs. relative disk extents) to determine whether or not a cell (512 pages) has any data in it. If any page in a cell has data in it, the entire cell is written to tape. Tape writing is controlled by an internal procedure L23WRRD. If tape writing causes an error, an error message is issued and SQL/DS is terminated.
ARIYI07
Read data disk
ARISYSD5
Write to archive tape file
ARIYM04
Archive file error message
ARICTRM
Terminate SQL/DS
• After each tape write, test for log space or storage space depleting (ARCHEX = ON). If so, quiesce the DBSS and continue with archiving.
ARIYK08
Stop agents from entering DBSS and flag agents in DBSS
ARIYK10
Wait for all (other) agents to leave DBSS
• When data disk archiving is complete, test to determine if DBSS was quiesced. If so, allow it to become active.
ARIYK09
Mark this agent outside DBSS
ARIYK11
Allow all agents to enter DBSS
• Close the archive tape file with rewind (in VM, the CMS ARIARCH FILEDEF command LEAVE option controls the rewind). A close error causes SQL/DS failure by returning to the caller with a failure return code.
ARISYSD5
• Free the buffer storage.
ARICWSF
Return to caller

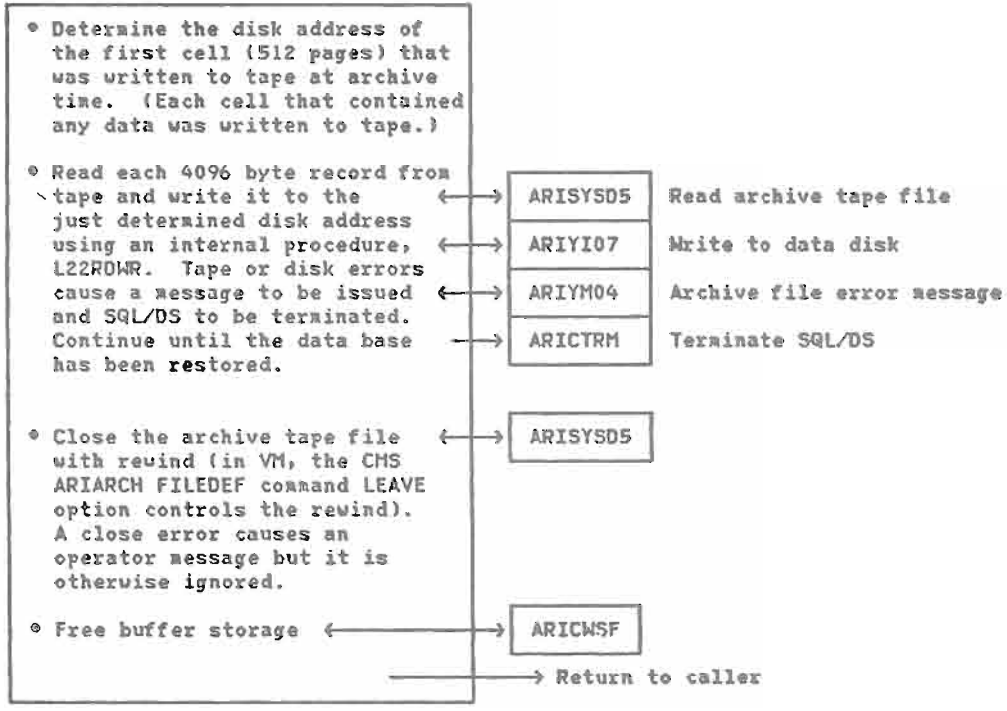
RESTORE THE DIRECTORY DISK FROM ARCHIVE

ARIYL21 (Restore the directory disk from archive)
(Called by ARIYI22D)



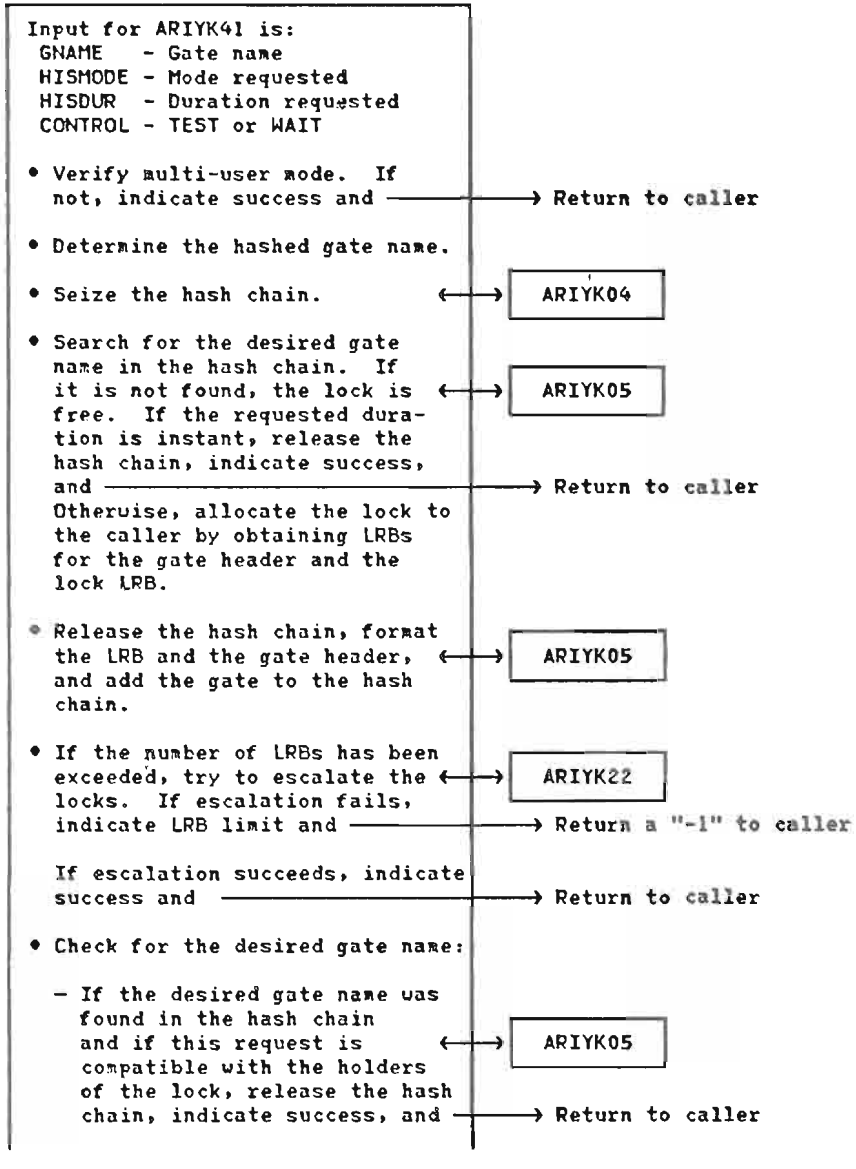
RESTORE THE DATA DISK FROM ARCHIVE

ARIYL22 (Restore the data from the archive)
(Called by ARIYT00)

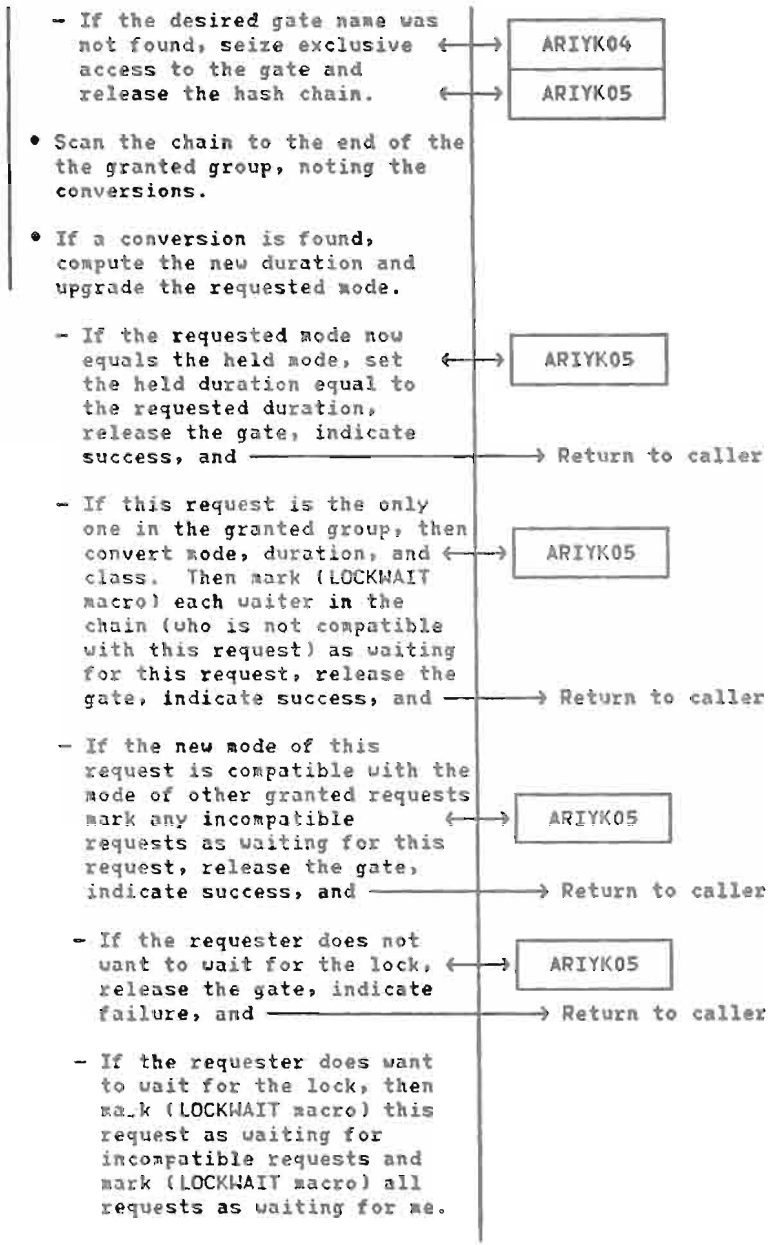


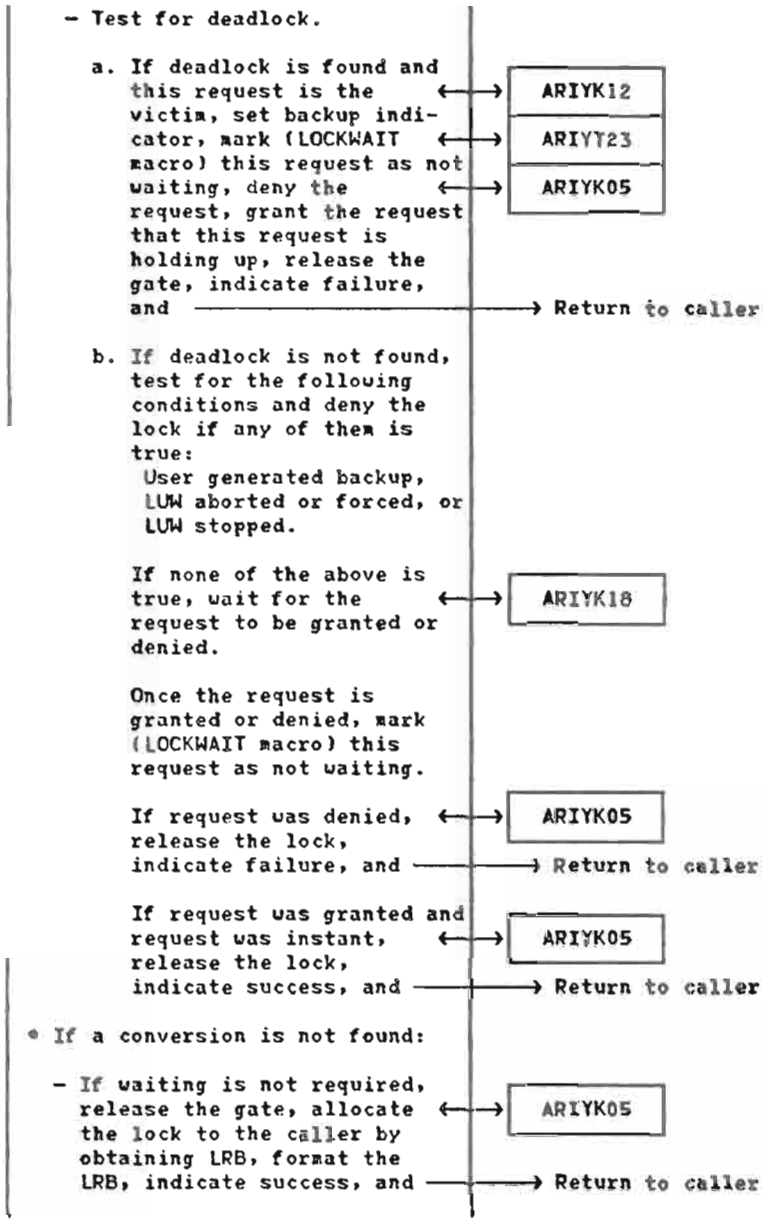
DBSS LOCK

* Called from numerous modules (see "Module-to-Module Cross Reference"
in Volume 2), including ARIYK01 and ARIYK28
ARIYK41 (Request Access to a Named Gate)

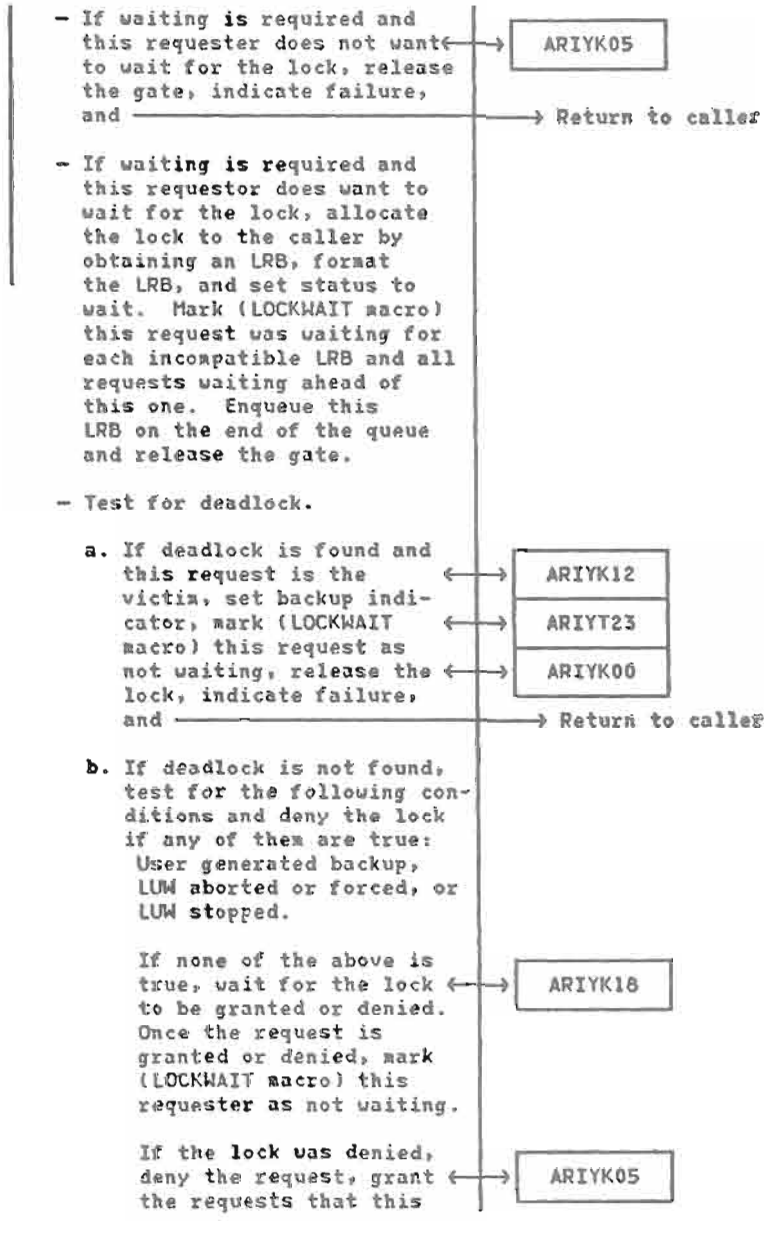


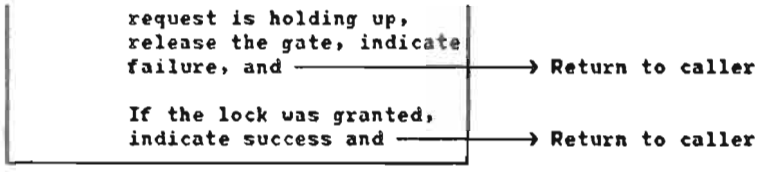
DBSS Lock (continued)





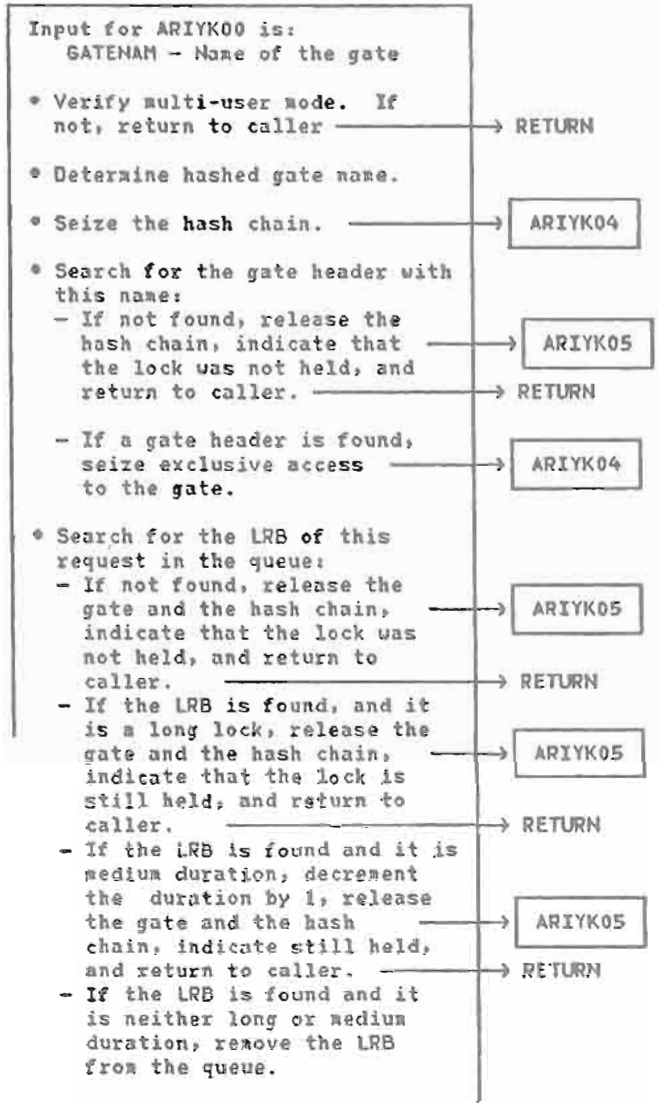
DBSS Lock (continued)

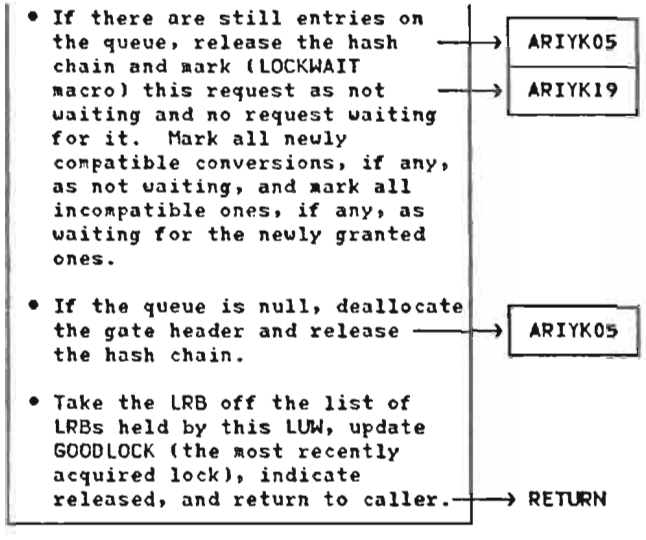




* Called from numerous modules (see "Module-to-Module Cross Reference" in Volume 2), including ARIYK01 and ARIYK28

ARIYK00 (Release Access to a Named Gate)





ISQL (INTERACTIVE STRUCTURED QUERY LANGUAGE)

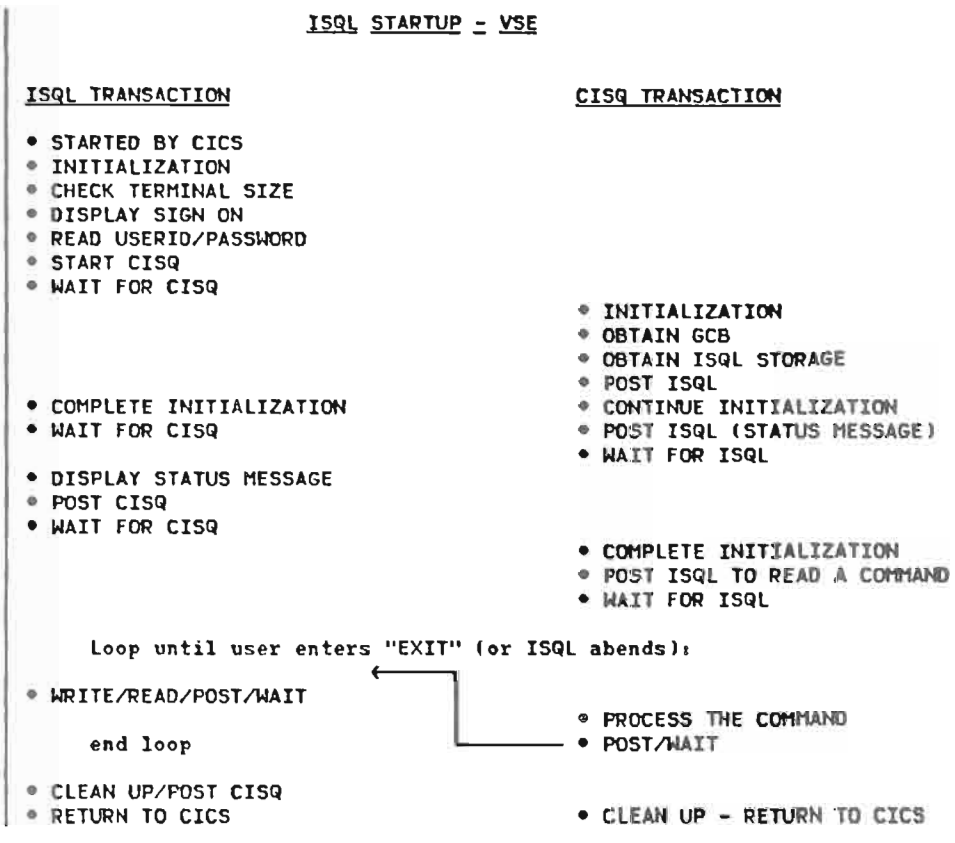
ISQL OVERVIEW

Running ISQL on VSE or VM differs with respect to ISQL startup. On VSE, two CICS transactions are required (see page 428 for an overview). On VM, only one load module is required. Once the Mainline Loop Controller (ARIIDBS) is called, processing is the same for both VM and VSE (page 432).

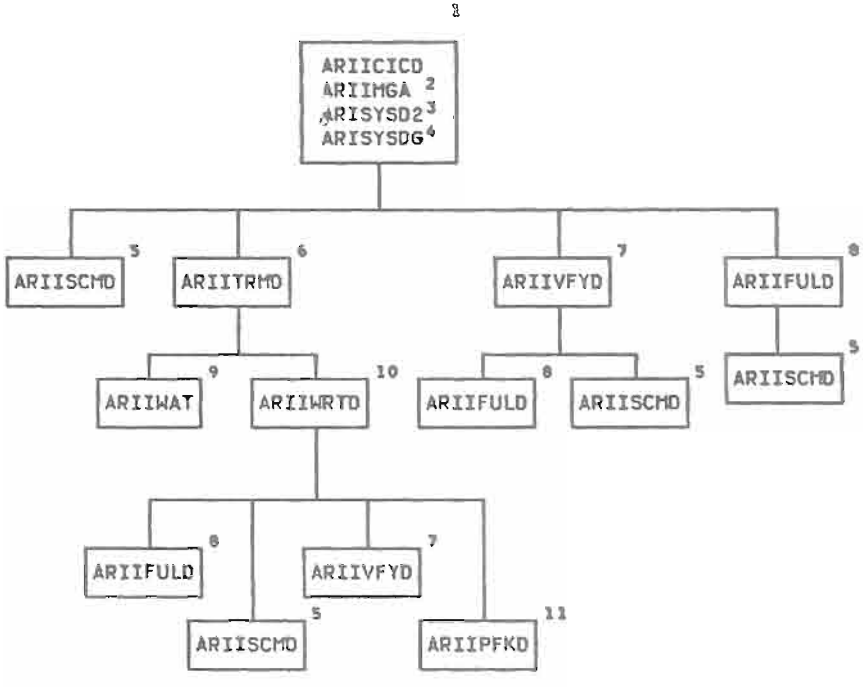
On VSE, the first of the two transactions, ISQL, is the terminal control transaction. The ISQL transaction is started by CICS when the user enters "ISQL" on a terminal. (See page 429 for an overview of the module flow for the ISQL transaction.) The second transaction, CISQ, is the command processor transaction. CISQ is started by the ISQL transaction and interacts with the Online Resource Manager to access the data base. (See page 430 for an overview of the module flow for the CISQ transaction.) A block of shared storage (ARIIGCB) is used for communication between the two tasks (see Volume 2).

On VM, the ISQL load module (ARIISQL) is loaded by a bootstrap (ARISISBT) when the user enters "ISQL" on a terminal. (See page 431 for an overview of the module flow for the ISQL load module.)

Upon completion of initialization, whether VSE or VM, control is passed to the Mainline Loop Controller (ARIIDBS), and ISQL processing is the same for both VSE and VM.

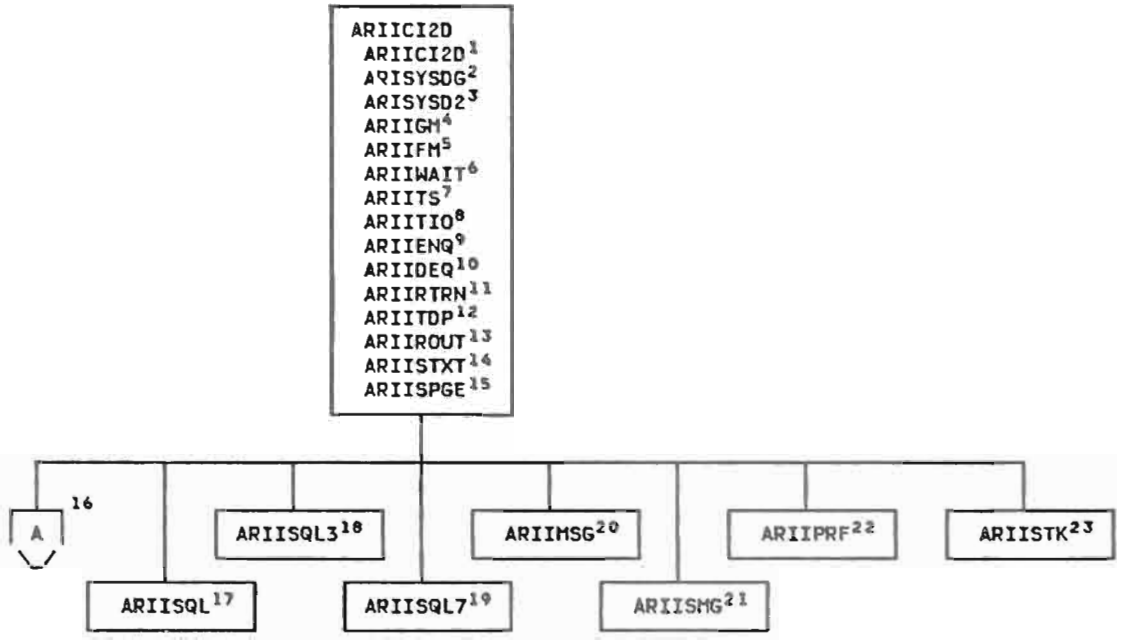


ISQL Transaction



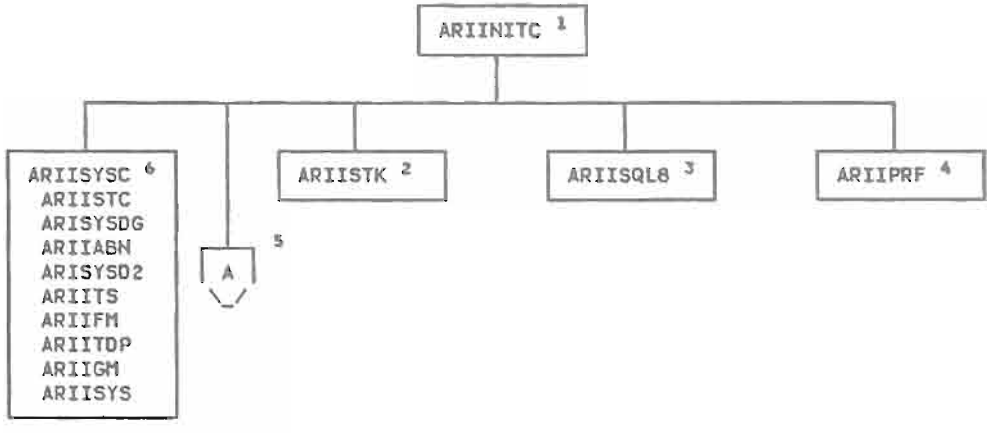
1 See page 434 for details
2 PF-key error messages
3 Dynamic storage FREEMAIN
4 dynamic storage GETMAIN
5 CICS SEND/RECEIVE/WAIT
6 WAIT/POST loop
7 Verify Cancel
8 Full-screen processing
9 15-second wait
10 Display Formatter
11 PF-key processing

CISQ Transaction



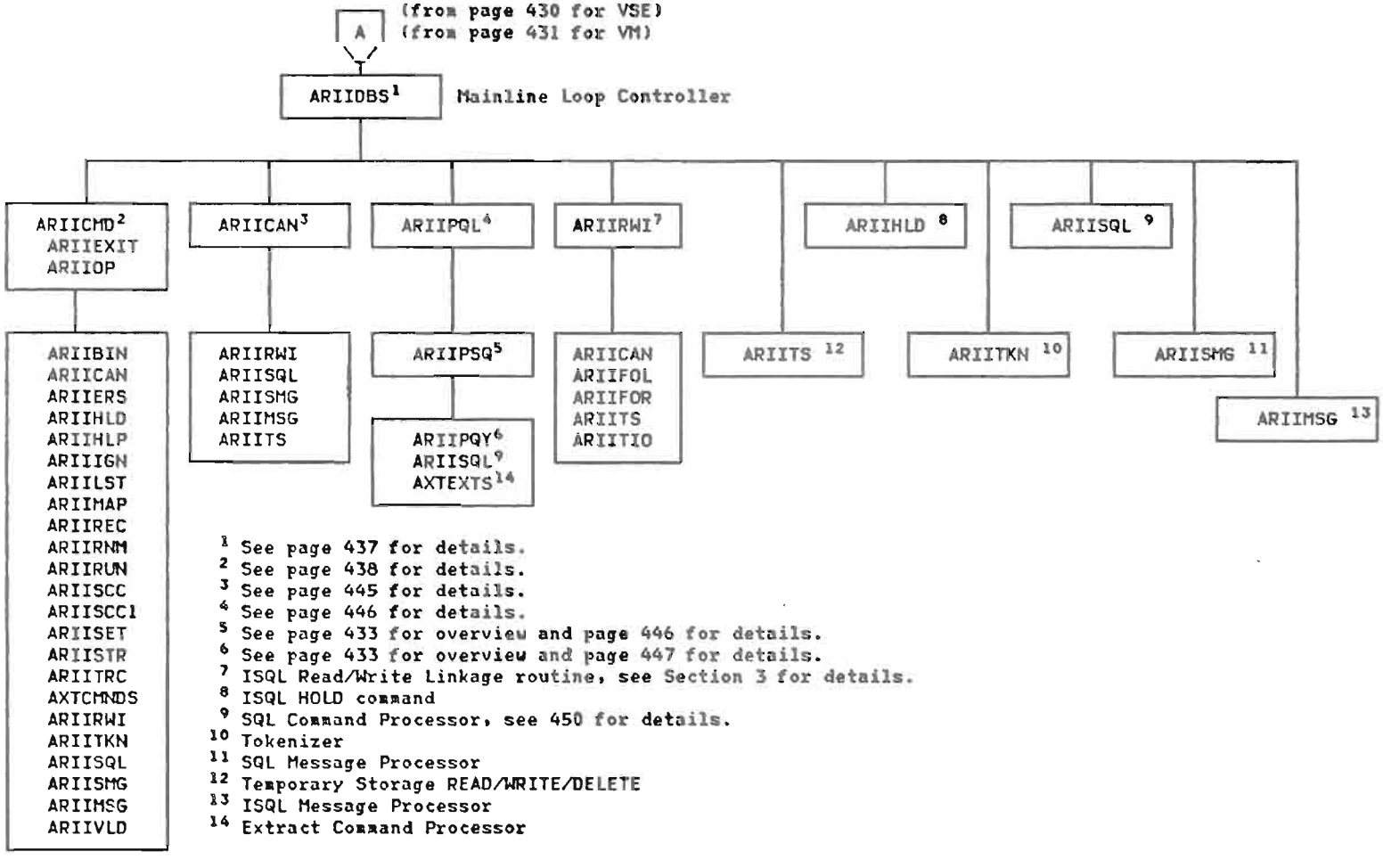
1 Mainline initialization
2 Automatic storage GETMAIN
3 Automatic storage FREEMAIN
4 Buffer storage GETMAIN
5 Buffer storage FREEMAIN
6 Wait for terminal I/O
7 Temporary storage control
8 Restart ISQL transaction
9 Resource ENQ routine
10 Resource DEQ routine
11 Return to CICS routine
12 Trace Dump Routine
13 CICS ROUTE Routing
14 CICS Send Text Routine
15 CICS Send Page Routine
16 Mainline loop, see page 432 for details
17 Setup online resource manager exit routine
18 Connect user ID/password
19 Obtain SQL/DS userid
20 ISQL message module
21 SQL message module
22 Profile routine
23 Obtain Stack Storage, see page 451

ISQL on VM



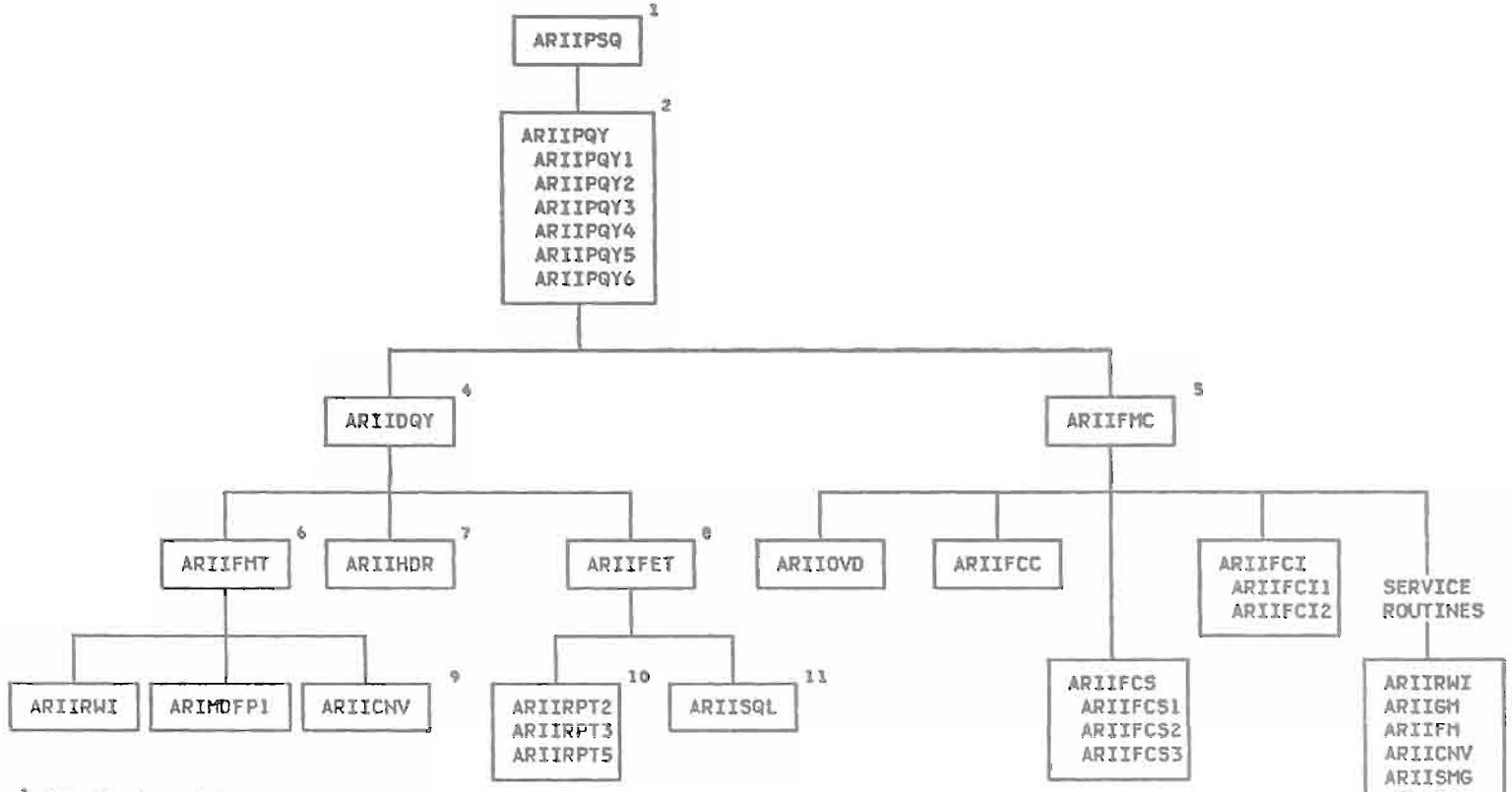
1 Mainline initialization (ARIINITC called by bootstrap module ARISISBT)
2 Obtain storage for ISQL dynamic storage stack (see page 451)
3 Establish CANCEL as an immediate command
4 Profile routine processing
5 Mainline loop (see page 432 for details)
6 System dependent subroutines for CMS (CMS version of ARIINITC subroutines:

- ARISYSDG - Get automatic storage
- ARISYSD2 - Free automatic storage
- ARIIGM - Get buffer storage
- ARIIFM - Free buffer storage
- ARIIABN - Abend exit/cleanup
- ARIITS - Temporary storage processing for routines
- ARIITDP - ISQLTRACE DUMP processing
- ARIISYS - Get storage for ARIISYSC automatic storage area
- ARIISTC - Check program stack



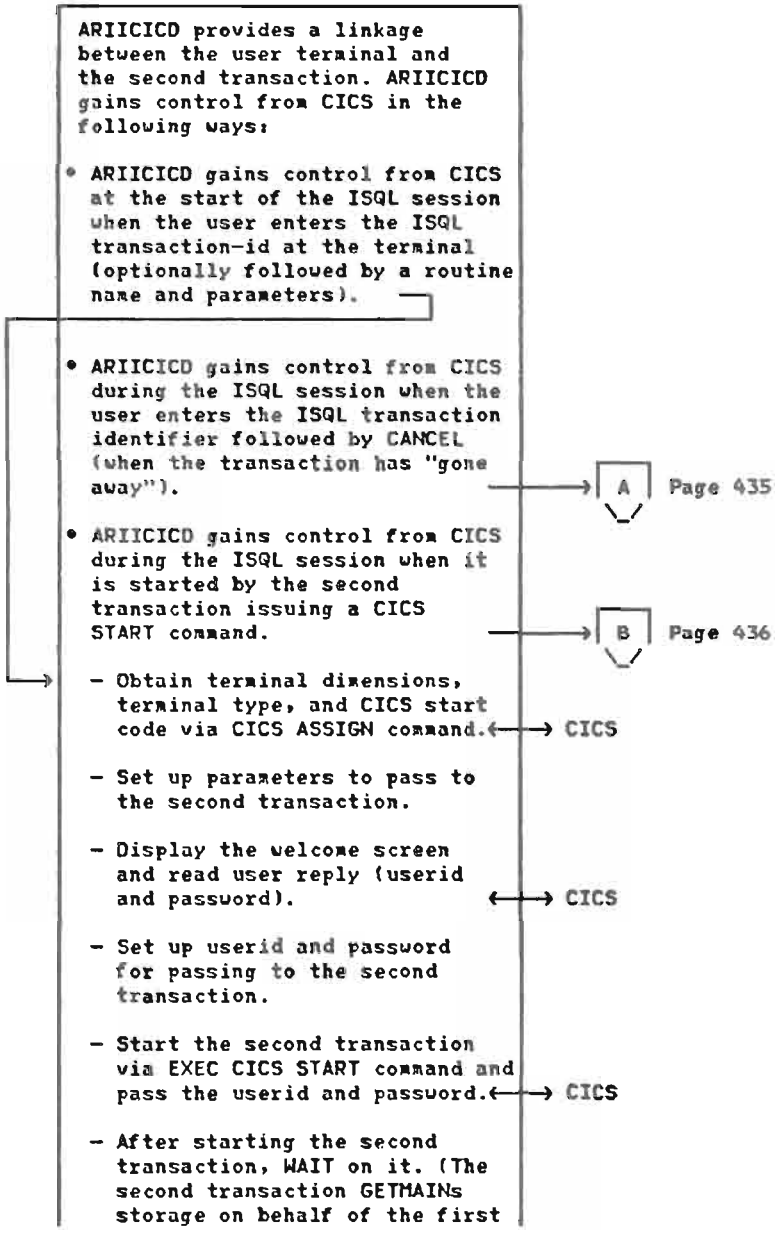
[1] See page 437 for details.
[2] See page 438 for details.
[3] See page 445 for details.
[4] See page 446 for details.
[5] See page 433 for overview and page 446 for details.
[6] See page 433 for overview and page 447 for details.
[7] ISQL Read/Write Linkage routine, see Section 3 for details.
[8] ISQL HOLD command
[9] SQL Command Processor, see 450 for details.
[10] Tokenizer
[11] SQL Message Processor
[12] Temporary Storage READ/WRITE/DELETE
[13] ISQL Message Processor
[14] Extract Command Processor

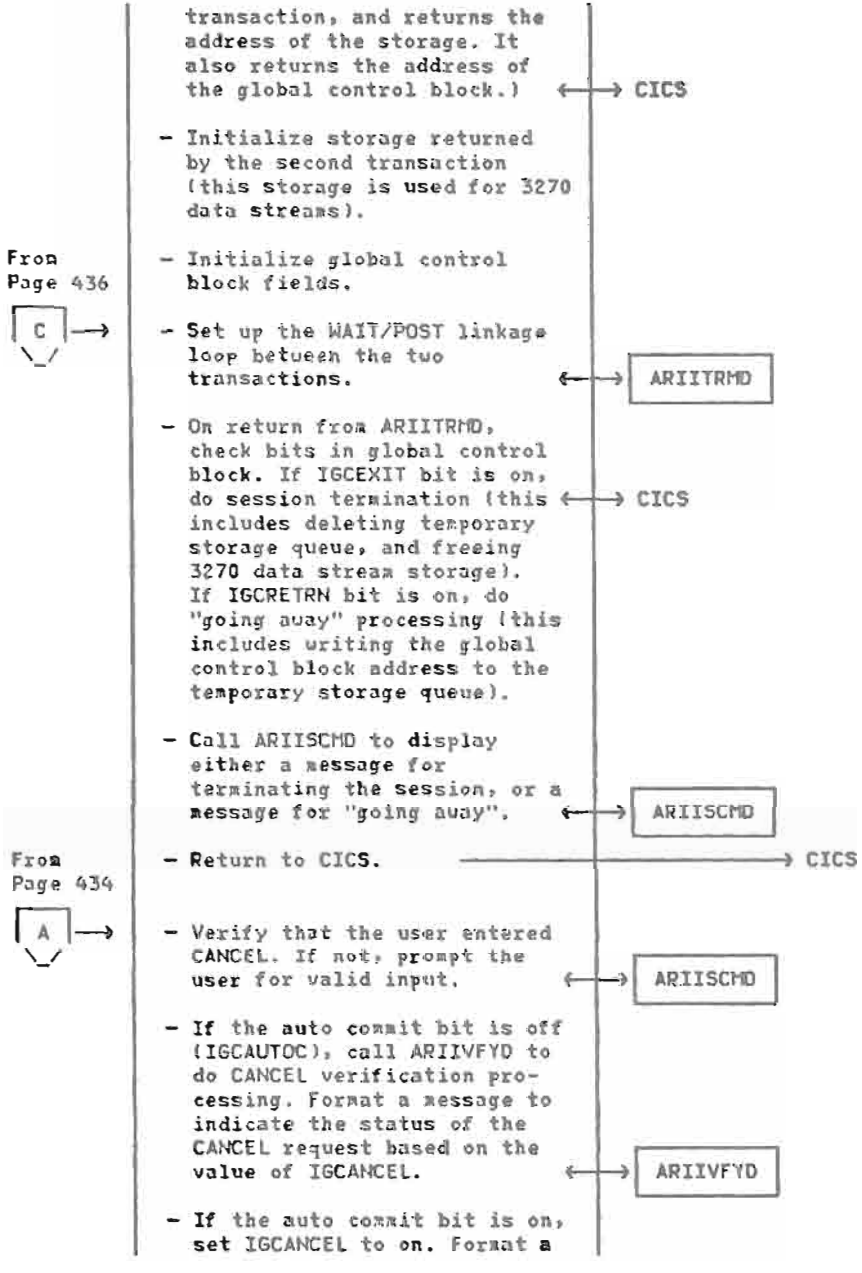
DISPLAY PROCESSING

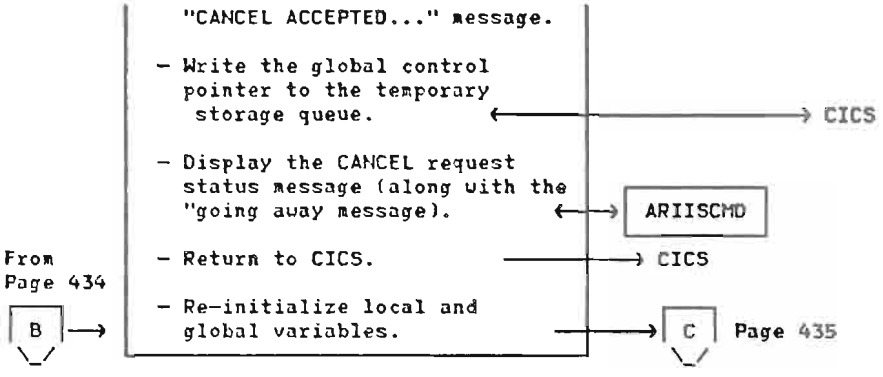


1 SQL Statement Processor
2 SQL Query Processor
3 ISQL Print Routine: ARIIPRTD (VSE), ARIIPRTC (VM/CMS)
4 SQL Query Display Manager
5 FORMAT Command Processor
6 Display Manager - Row Formatter
7 Displays Column Headers
8 Display Manager - Result Fetch
9 Conversion Routine
10 Report Formatter
11 SQL Executor

ARIICICD

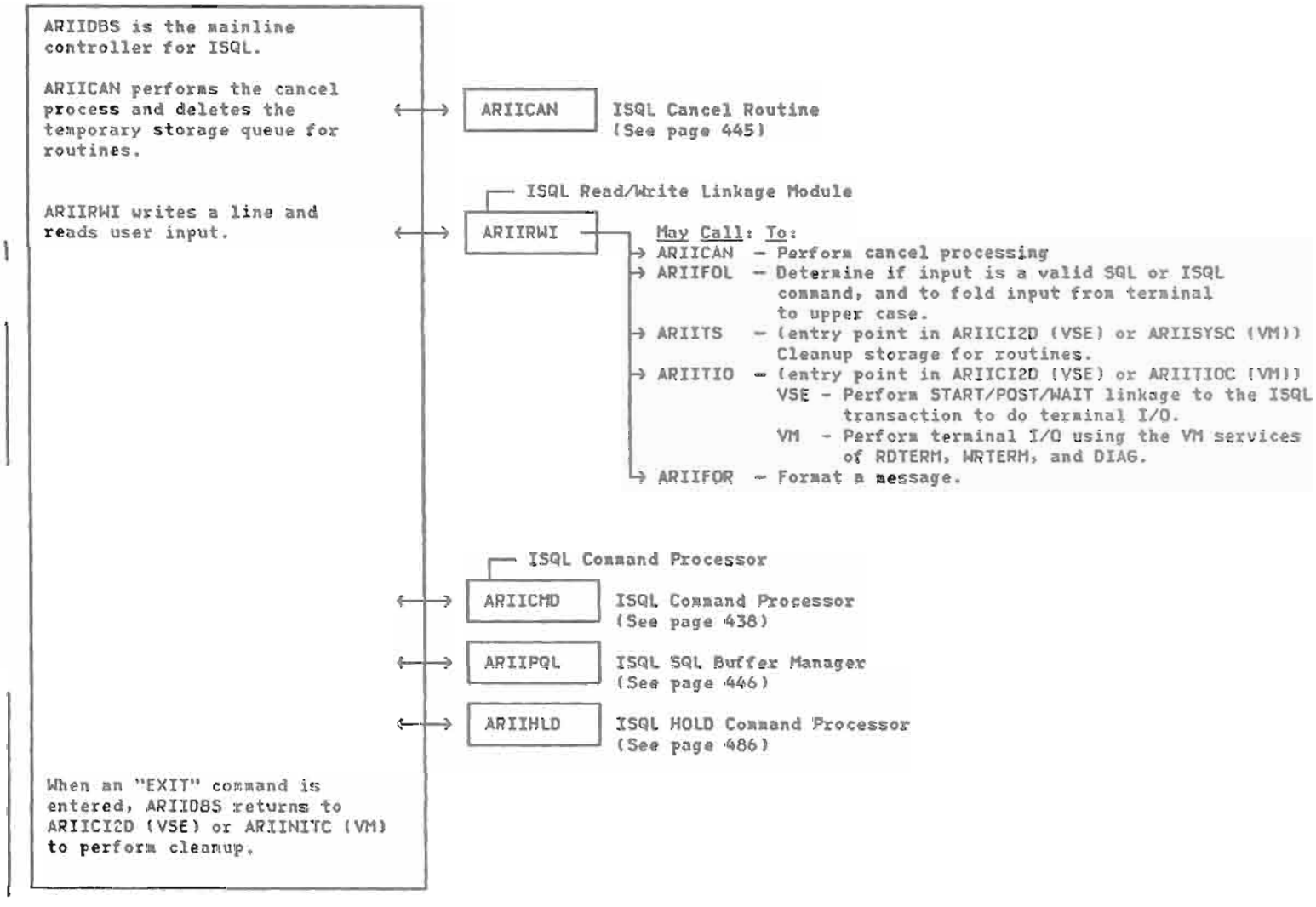



transaction, and returns the address of the storage. It also returns the address of the global control block.)
CICS
- Initialize storage returned by the second transaction (this storage is used for 3270 data streams).
From Page 436
- Initialize global control block fields.
C
- Set up the WAIT/POST linkage loop between the two transactions.
ARIITRMD
- On return from ARIITRMD, check bits in global control block. If IGCEXIT bit is on, do session termination (this includes deleting temporary storage queue, and freeing 3270 data stream storage). If IGCRETRN bit is on, do "going away" processing (this includes writing the global control block address to the temporary storage queue).
CICS
- Call ARIISCMD to display either a message for terminating the session, or a message for "going away".
ARIISCMD
From Page 434
- Return to CICS.
CICS
A
- Verify that the user entered CANCEL. If not, prompt the user for valid input.
ARIISCMD
- If the auto commit bit is off (IGCAUTOC), call ARIIVFYD to do CANCEL verification processing. Format a message to indicate the status of the CANCEL request based on the value of IGCANCEL.
ARIIVFYD
- If the auto commit bit is on, set IGCANCEL to on. Format a

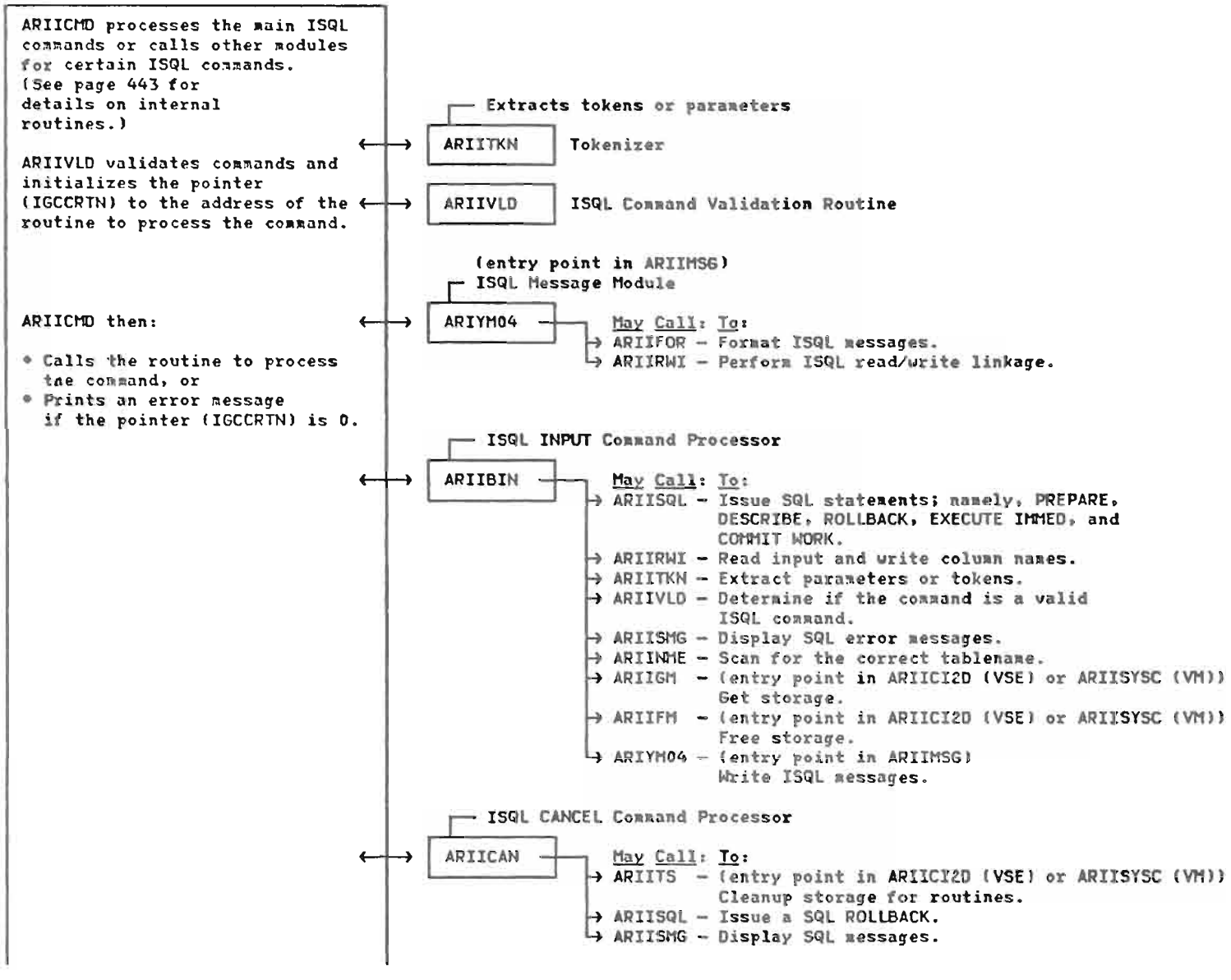


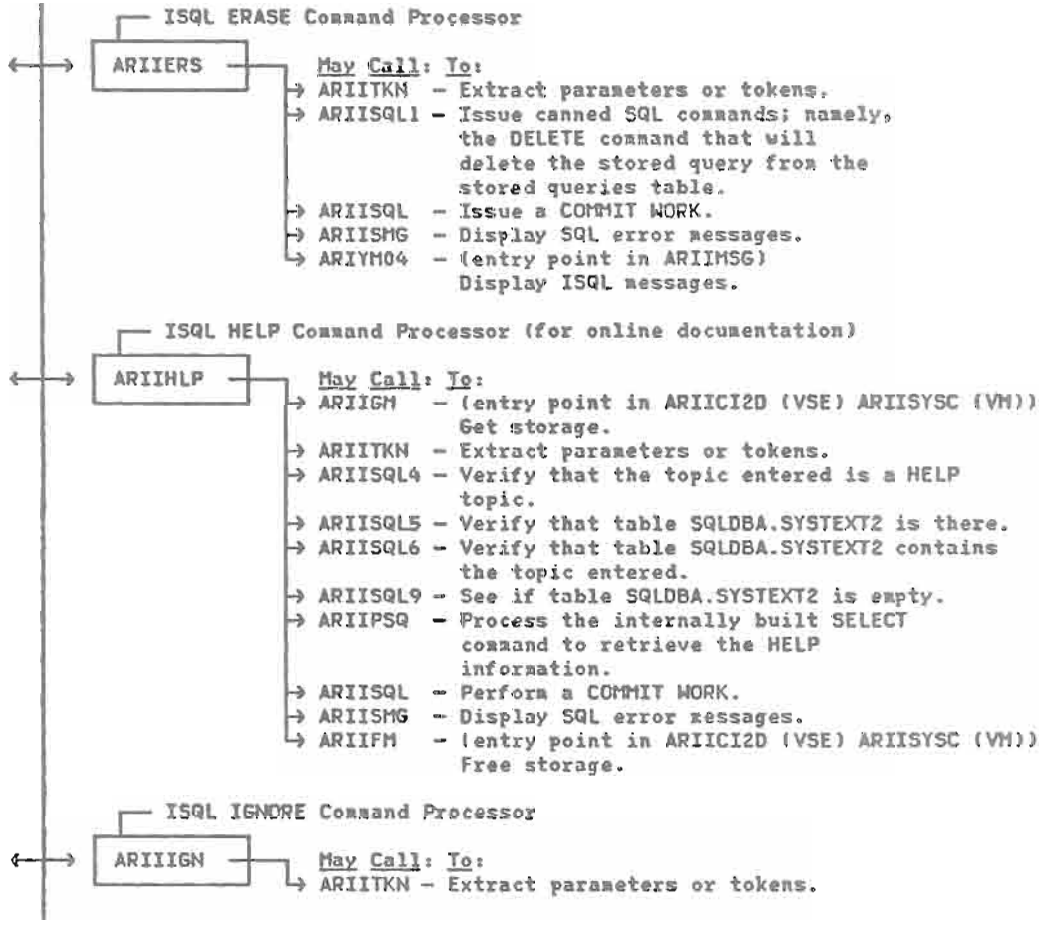
ISQL (continued)

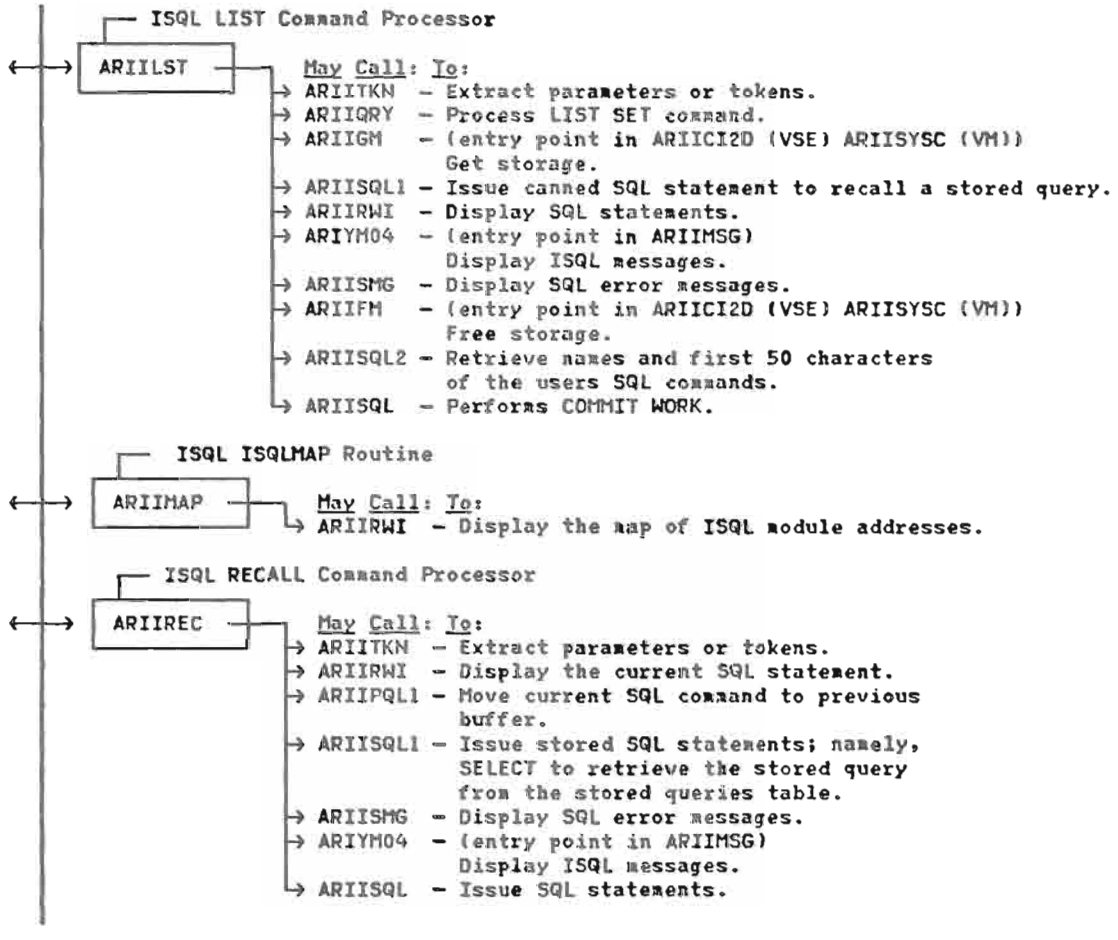
ARIIDBS

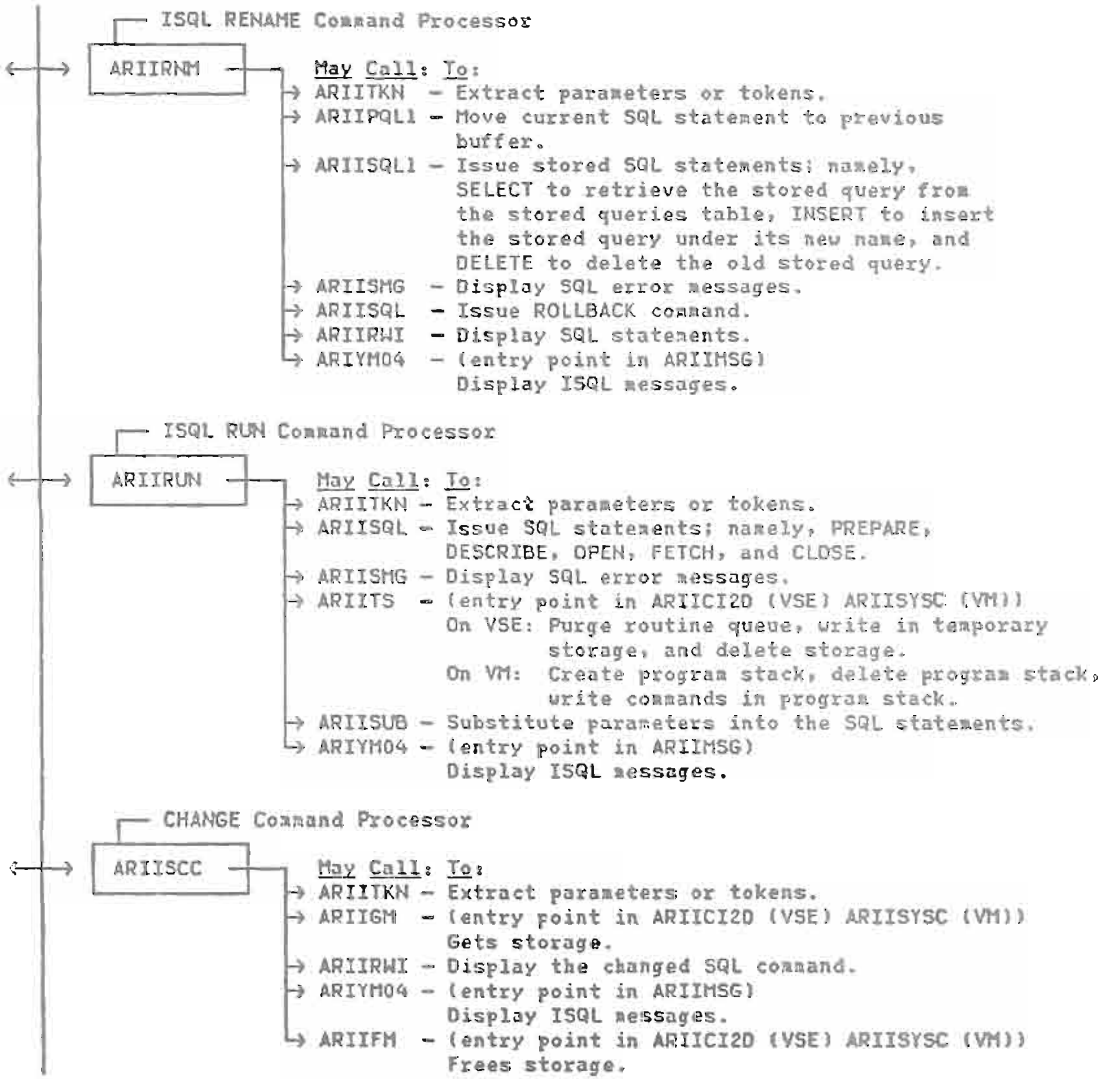


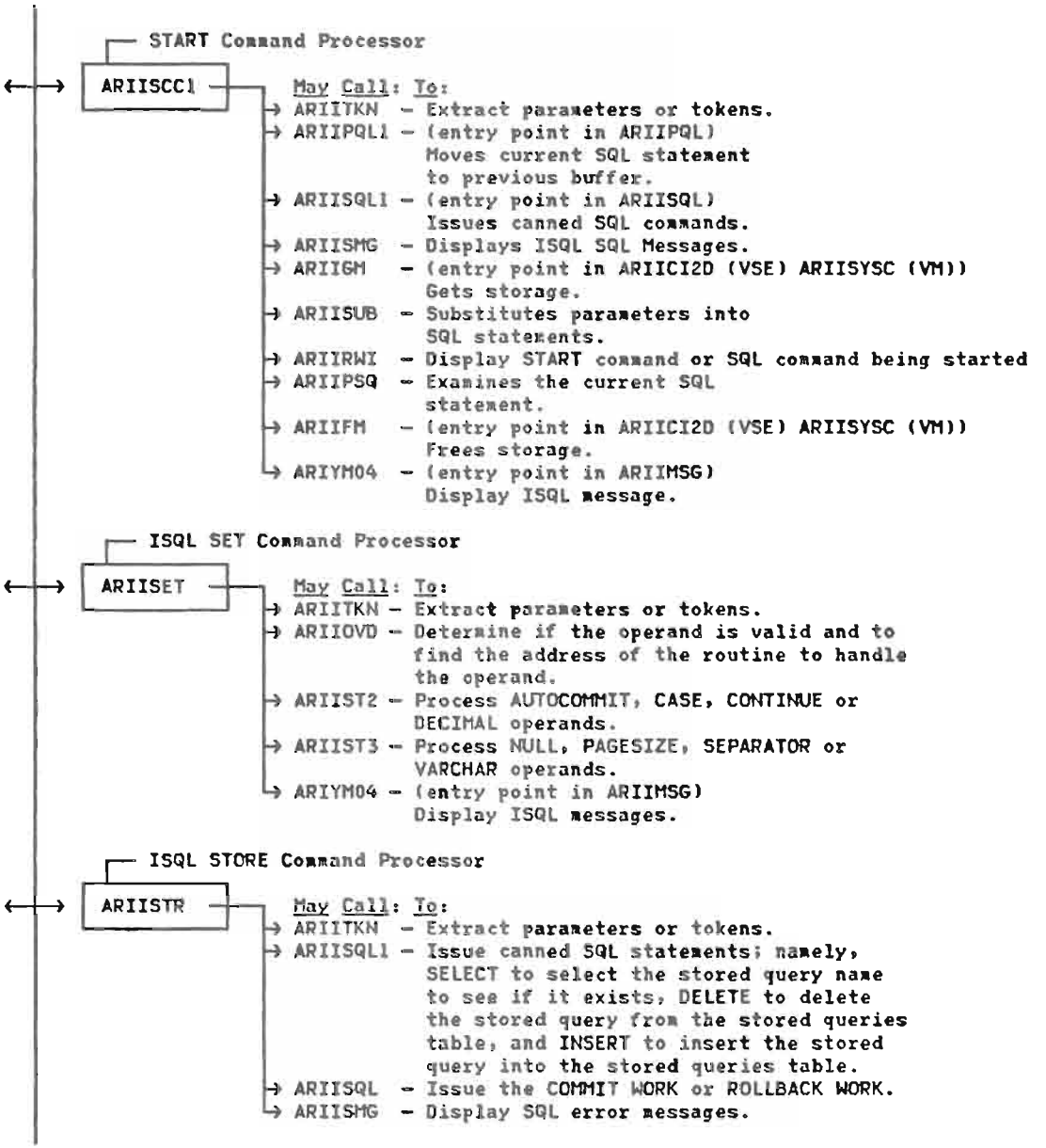
ARIICMD (Called by ARIIDBS)



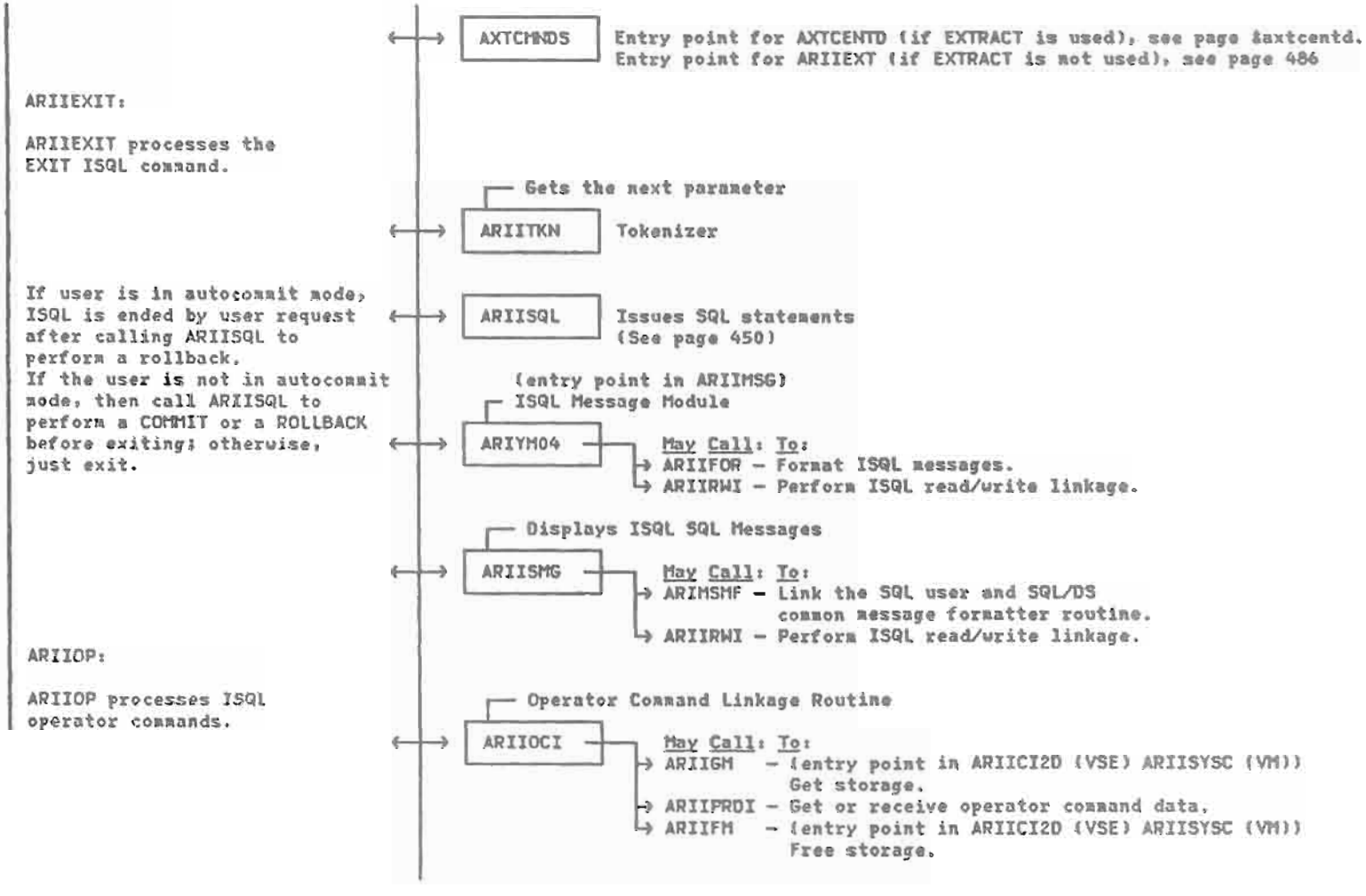

ISQL ERASE Command Processor
ARIIERS
May Call: To:
ARIITKN - Extract parameters or tokens.
ARIISQL1 - Issue canned SQL commands; namely, the DELETE command that will delete the stored query from the stored queries table.
ARIISQL - Issue a COMMIT WORK.
ARIISMG - Display SQL error messages.
ARIYM04 - (entry point in ARIIMSG) Display ISQL messages.
ISQL HELP Command Processor (for online documentation)
ARIIHLP
May Call: To:
ARIIGM - (entry point in ARIICI2D (VSE) ARIISYSC (VM)) Get storage.
ARIITKN - Extract parameters or tokens.
ARIISQL4 - Verify that the topic entered is a HELP topic.
ARIISQL5 - Verify that table SQLDBA.SYSTEXT2 is there.
ARIISQL6 - Verify that table SQLDBA.SYSTEXT2 contains the topic entered.
ARIISQL9 - See if table SQLDBA.SYSTEXT2 is empty.
ARIIPSQ - Process the internally built SELECT command to retrieve the HELP information.
ARIISQL - Perform a COMMIT WORK.
ARIISMG - Display SQL error messages.
ARIIFM - (entry point in ARIICI2D (VSE) ARIISYSC (VM)) Free storage.
ISQL IGNORE Command Processor
ARIIIGN
May Call: To:
ARIITKN - Extract parameters or tokens.

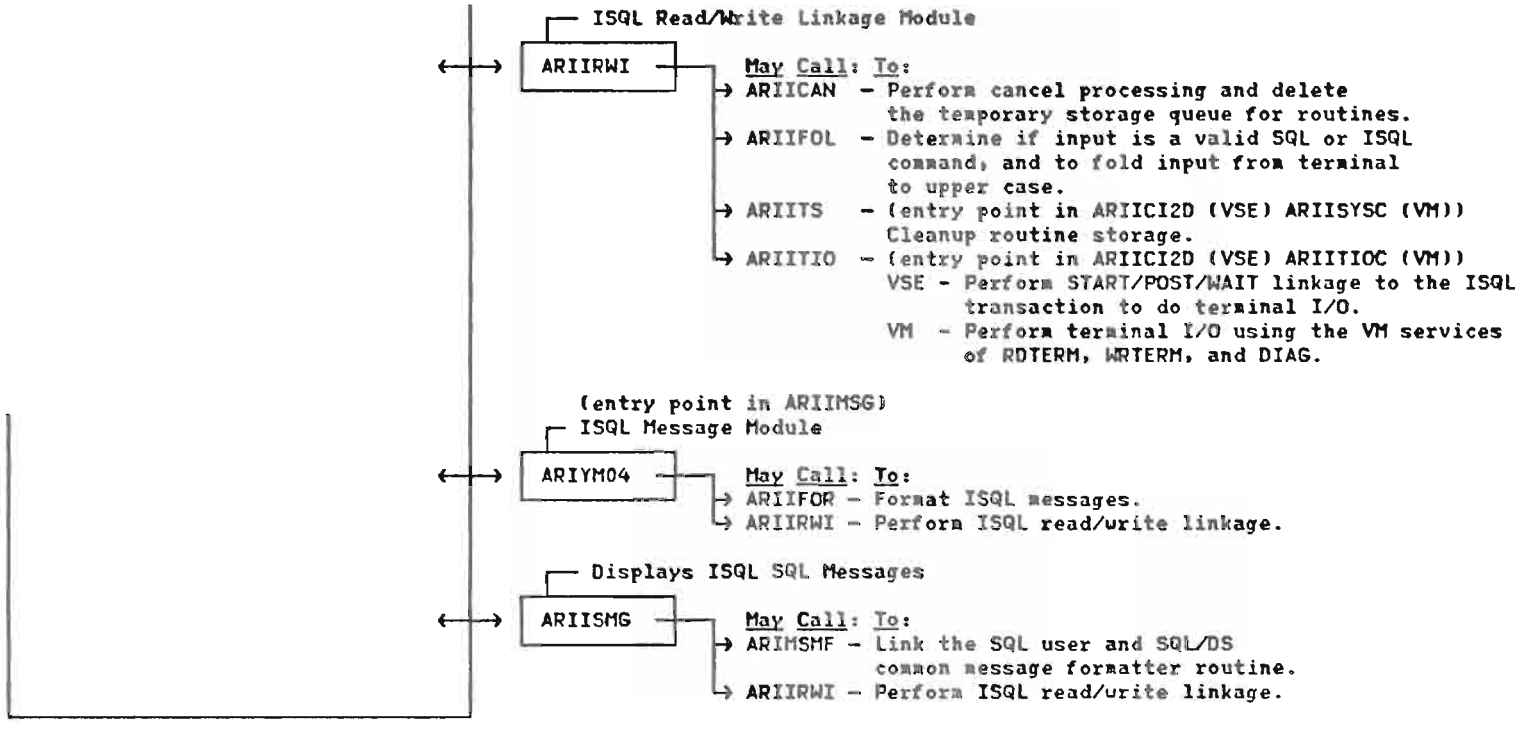



ISQL RENAME Command Processor
ARIIRNM
May Call: To:
ARIITKN - Extract parameters or tokens.
ARIIPQL1 - Move current SQL statement to previous buffer.
ARIISQL1 - Issue stored SQL statements: namely, SELECT to retrieve the stored query from the stored queries table, INSERT to insert the stored query under its new name, and DELETE to delete the old stored query.
ARIISMG - Display SQL error messages.
ARIISQL - Issue ROLLBACK command.
ARIIRWI - Display SQL statements.
ARIYM04 - (entry point in ARIIMSG) Display ISQL messages.
ISQL RUN Command Processor
ARIIRUN
May Call: To:
ARIITKN - Extract parameters or tokens.
ARIISQL - Issue SQL statements; namely, PREPARE, DESCRIBE, OPEN, FETCH, and CLOSE.
ARIISMG - Display SQL error messages.
ARIITS - (entry point in ARIICI2D (VSE) ARIISYSC (VM)) On VSE: Purge routine queue, write in temporary storage, and delete storage. On VM: Create program stack, delete program stack, write commands in program stack.
ARIISUB - Substitute parameters into the SQL statements.
ARIYM04 - (entry point in ARIIMSG) Display ISQL messages.
CHANGE Command Processor
ARIISCC
May Call: To:
ARIITKN - Extract parameters or tokens.
ARIIGM - (entry point in ARIICI2D (VSE) ARIISYSC (VM)) Gets storage.
ARIIRWI - Display the changed SQL command.
ARIYM04 - (entry point in ARIIMSG) Display ISQL messages.
ARIIFM - (entry point in ARIICI2D (VSE) ARIISYSC (VM)) Frees storage.



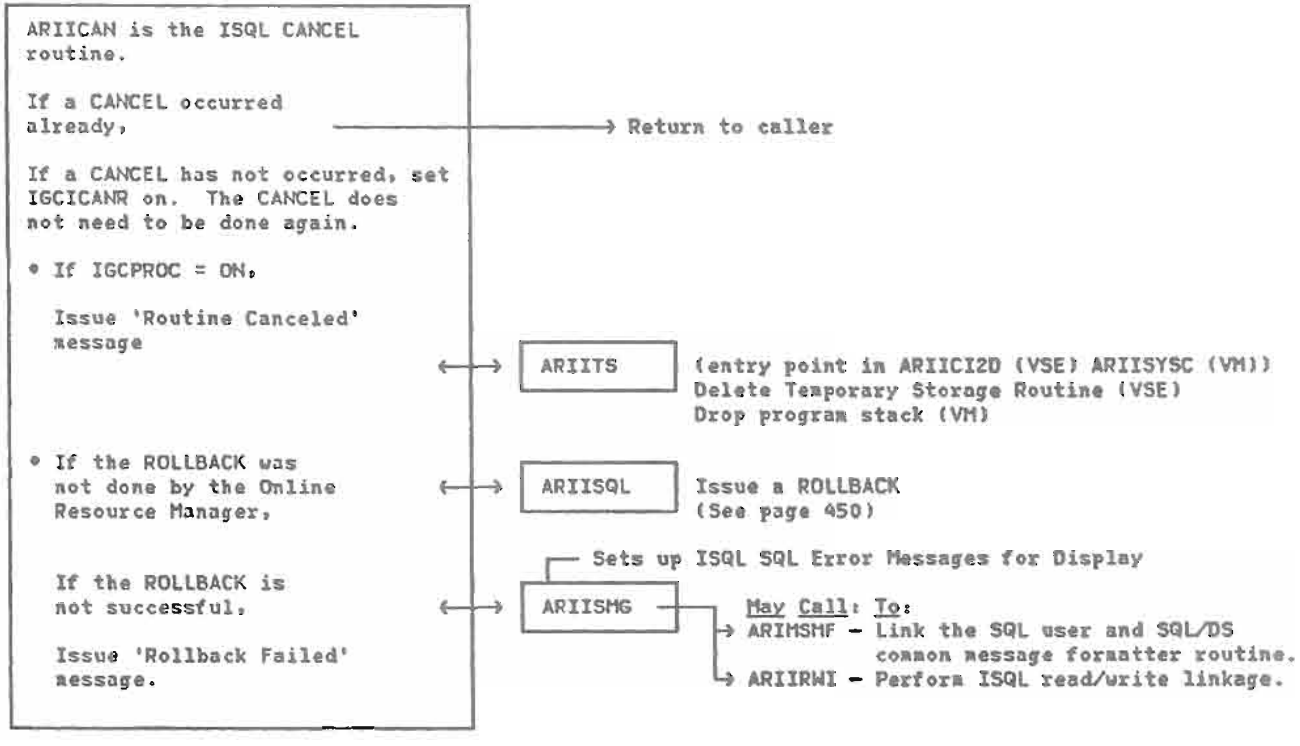
ISQL (continued)





ISQL (continued)

ARIICAN



ARIIPQL (Called by ARIIDBS)

ARIIPQL is the SQL buffer manager. It copies SQL statements from the current buffer to the previous buffer and from the input buffer to the current buffer.

ARIIPSQ examines the current SQL statement and process it accordingly.

ARIIPSQ (Called by ARIIPQL)

ARIIPSQ checks for valid types of SQL statements and processes them accordingly.

Extract parameters or tokens.

ARIITKN — Tokenizer

ARIYM04 (entry in ARIIMSG) builds and displays ISQL messages. The messages inform you what has just been executed or prompt you for a response.

(entry point in ARIIMSG)
ISQL Message Module

ARIYM04

May Call: To:
- ARIIFOR – Format ISQL messages.
- ARIIRWI – Perform ISQL read/write linkage.

ARIISMG builds and issues formatted SQL messages.

Displays ISQL SQL Messages

ARIISMG

May Call: To:
- ARIMSMF – Link the SQL user and SQL/DS common message formatter routine.
- ARIIRWI – Perform ISQL read/write linkage.

ARIISQL issues SQL statements.

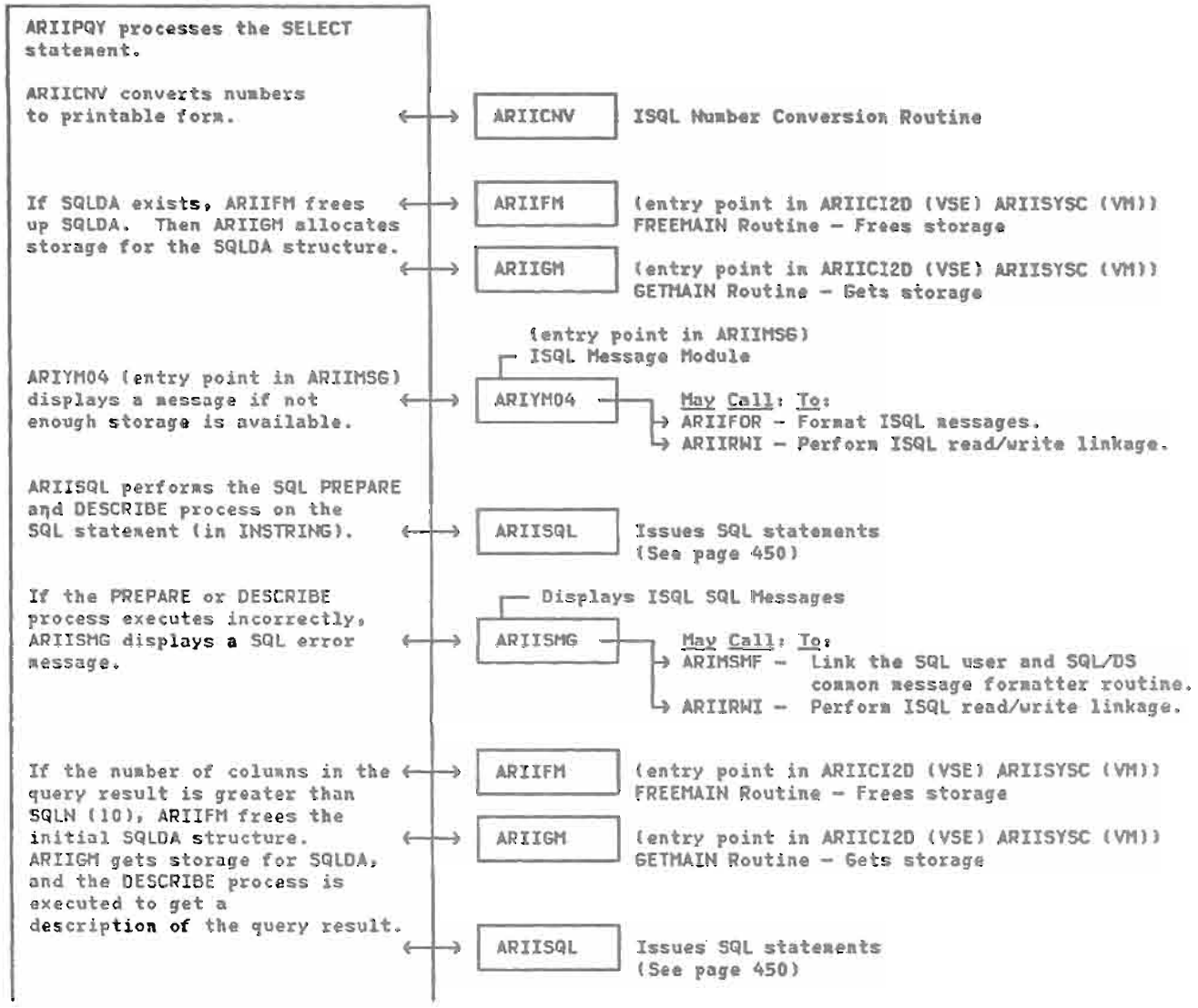
ARIISQL — Issues SQL statements
(See page 450)

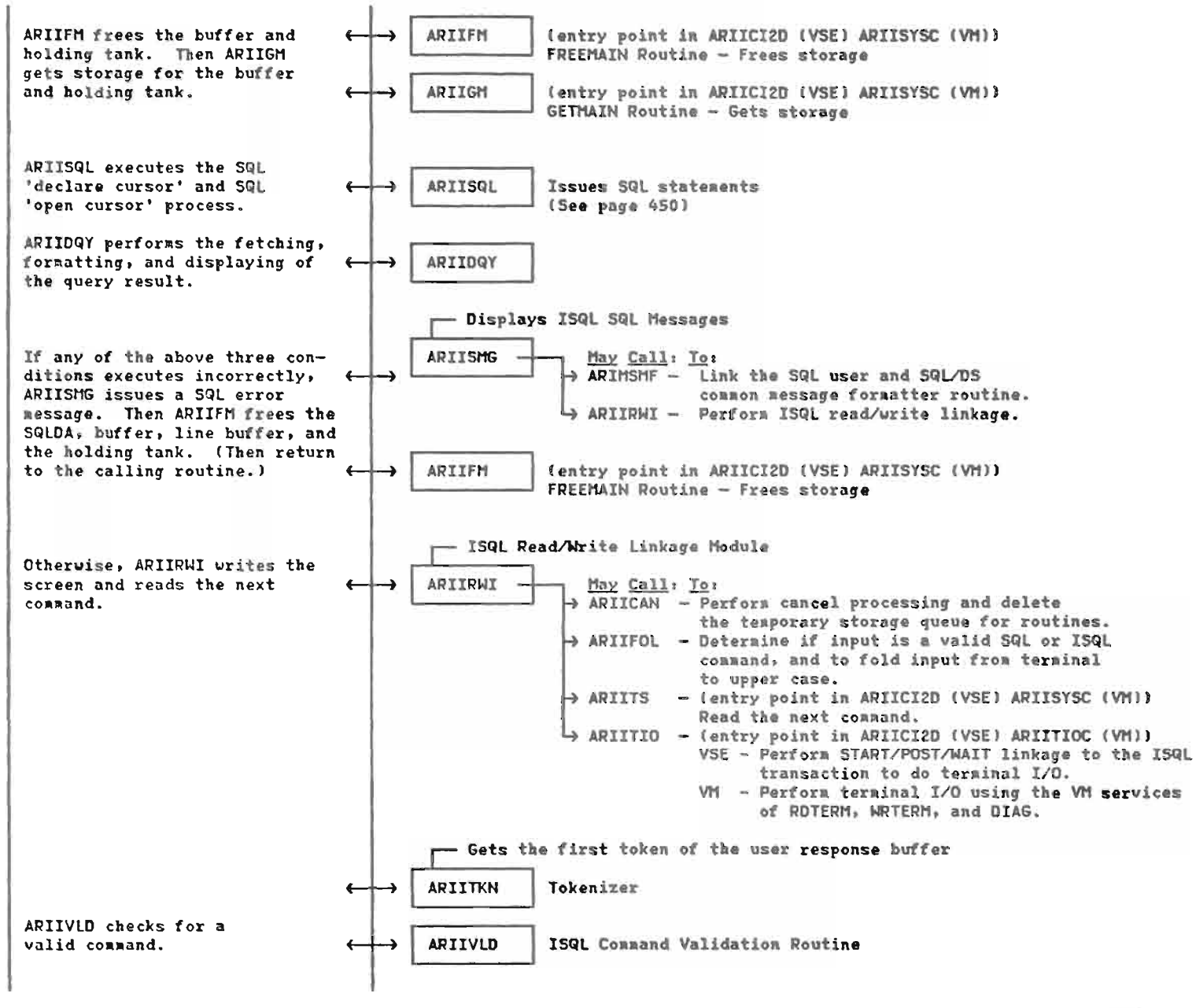
ARIIPQY processes the query statements.

ARIIPQY — Processes the SELECT statement
(See page 447)

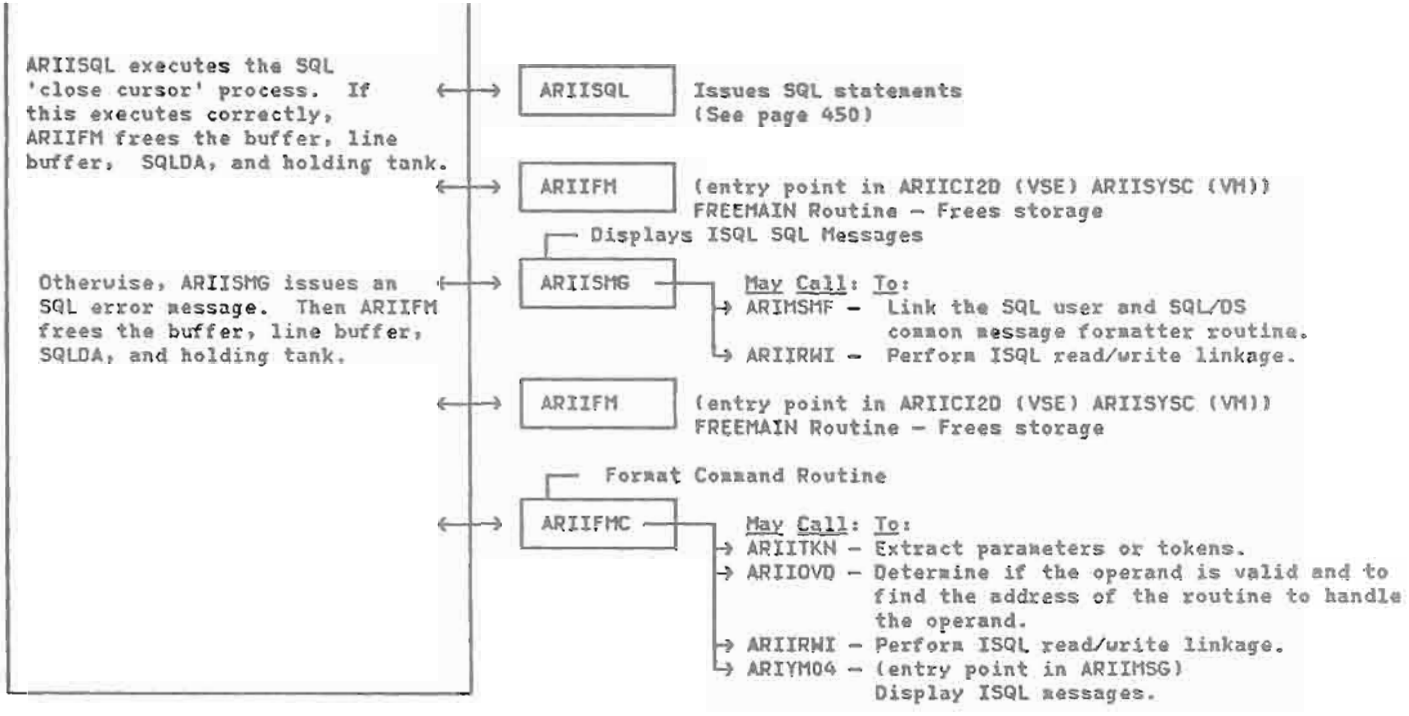
ISQL (continued)

ARIIPQY (Called by ARIIPSQ)

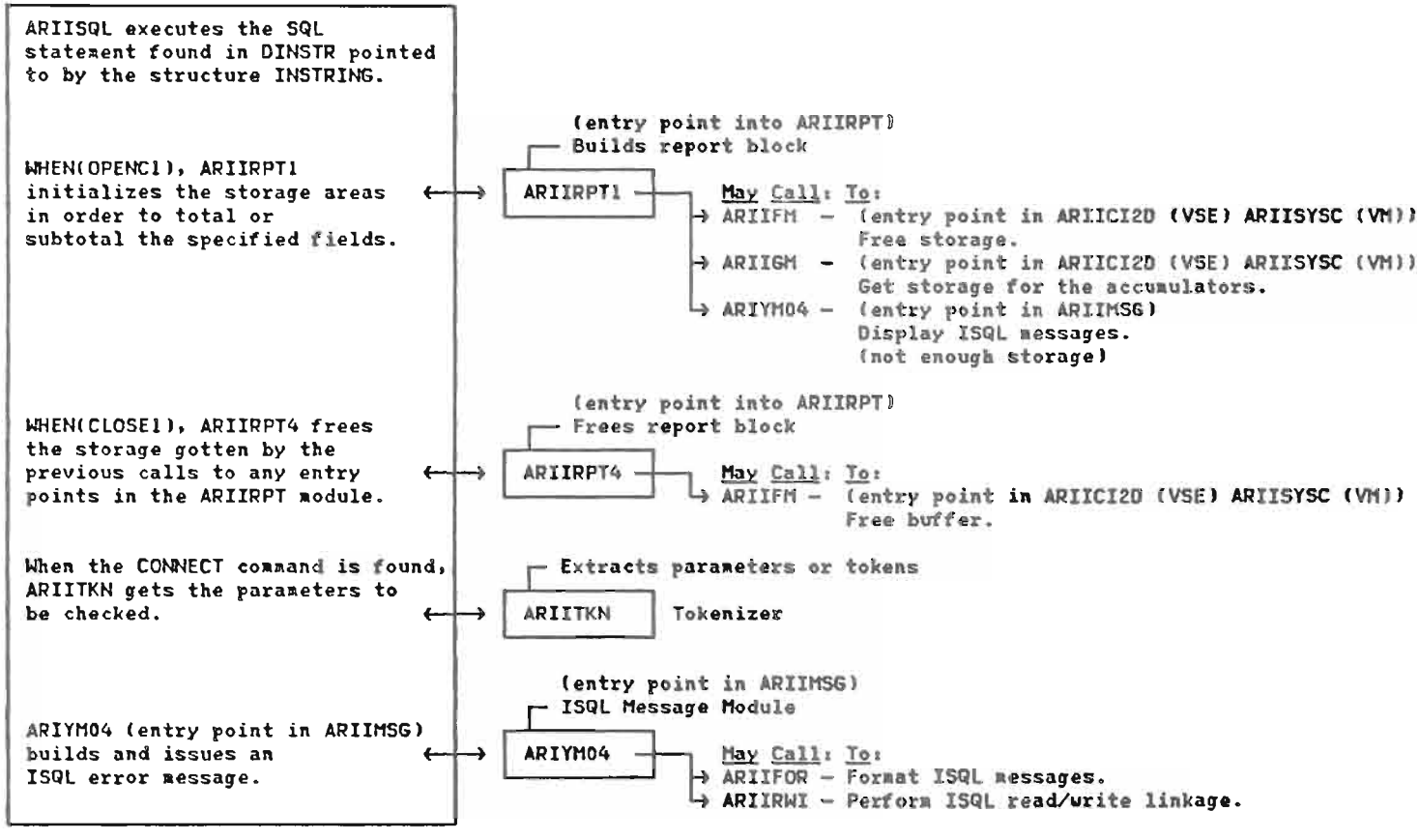




ISQL (continued)

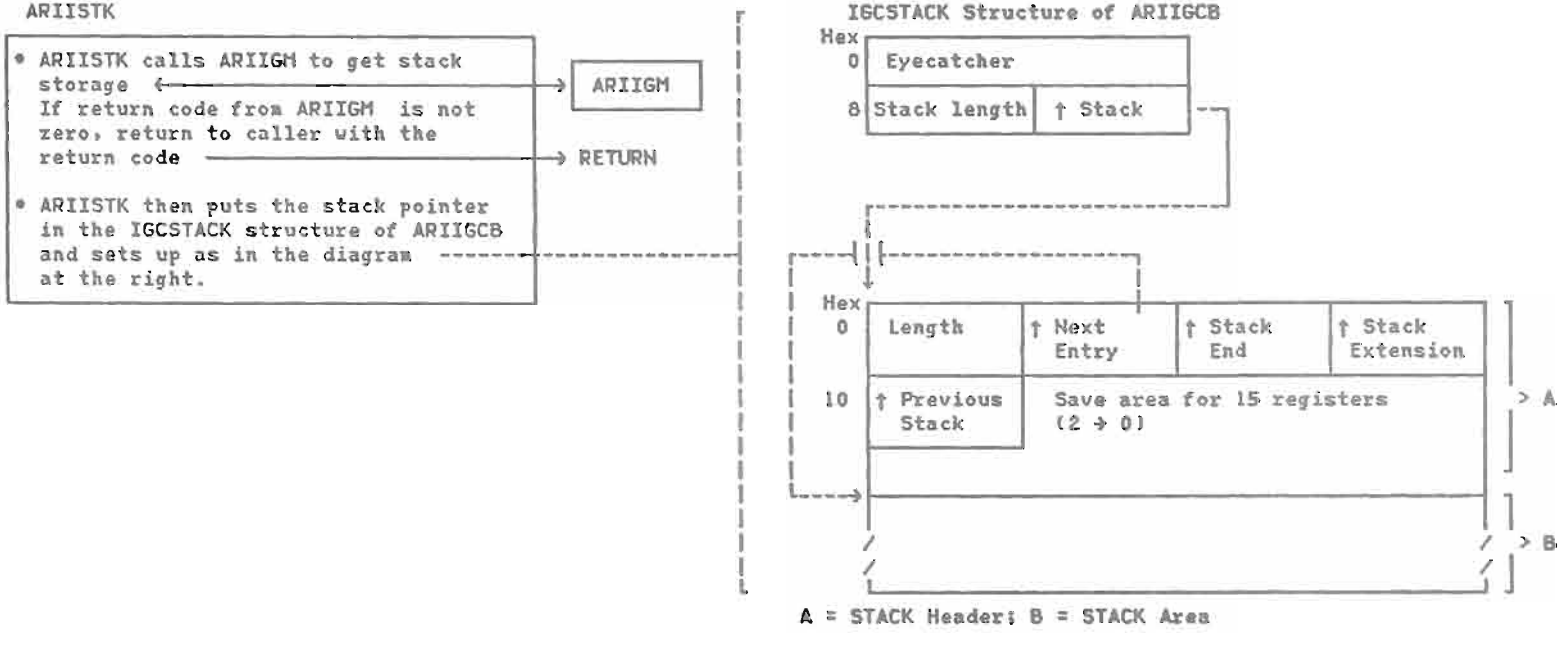


ARIISQL (Called by ARIIPSQ)



ISQL Storage Services

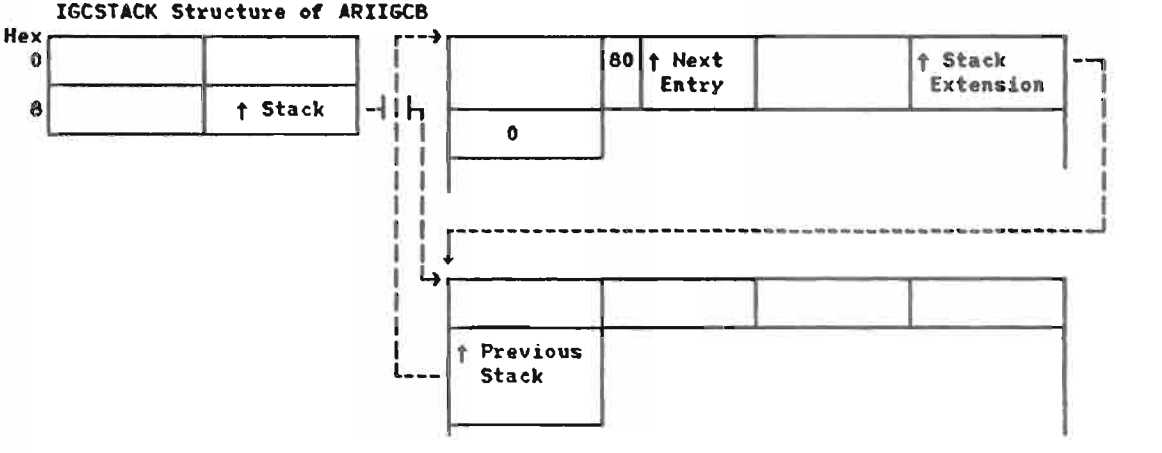
(Automatic Storage Only)



ARICGSE and ARICFSE can be called from any module when a new automatic storage stack is needed.

ARICGSE (Get Stack Extension)

- ARICGSE calls ARISYSDG to get storage to extend the stack. The setup of the extension is indicated in the next diagram ---------> ARISYSDG



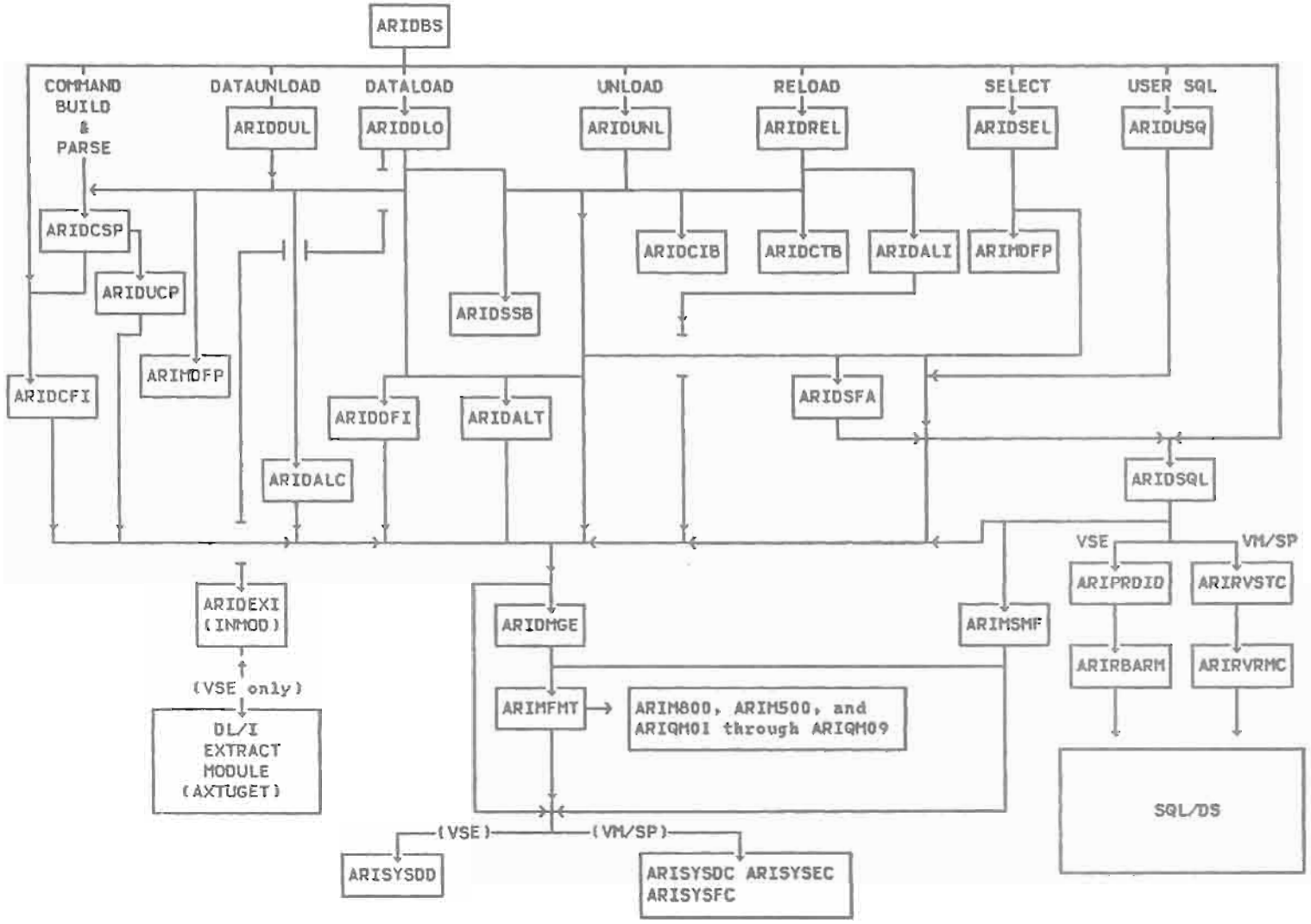
ARICFSE (Free Stack Extension)

- Get pointer to current stack from register 1.
- Free the current stack and rechain pointers back to previous stack. (This module utilizes pointers in IGCSTACK Structure of ARIIGCB and the Stack (see the diagram on the previous page).
- Return to caller ---------> RETURN

DBS (DATA BASE SERVICES) UTILITY

DBS UTILITY INTERCONNECTION DIAGRAM

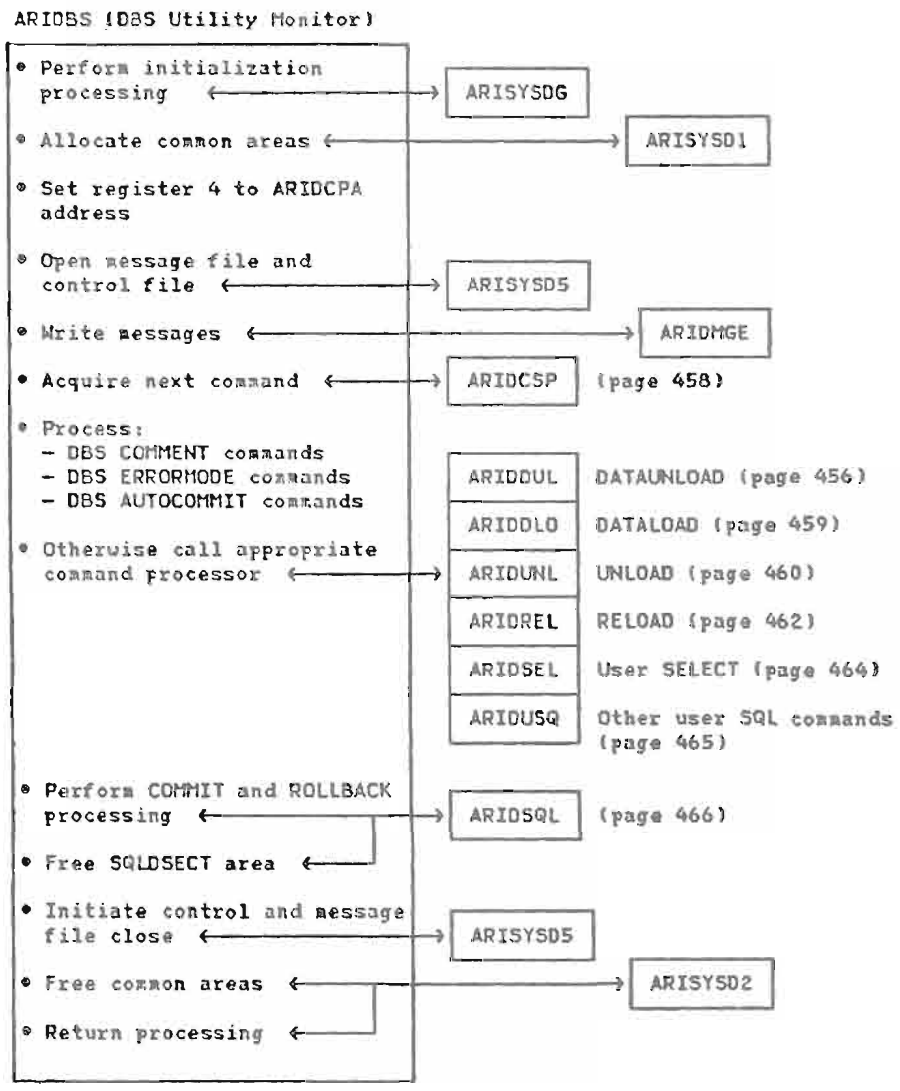
Read the module flow from top to bottom, left-to-right and right-to-left except where the arrows indicate otherwise. Upward flow is by RETURN only.

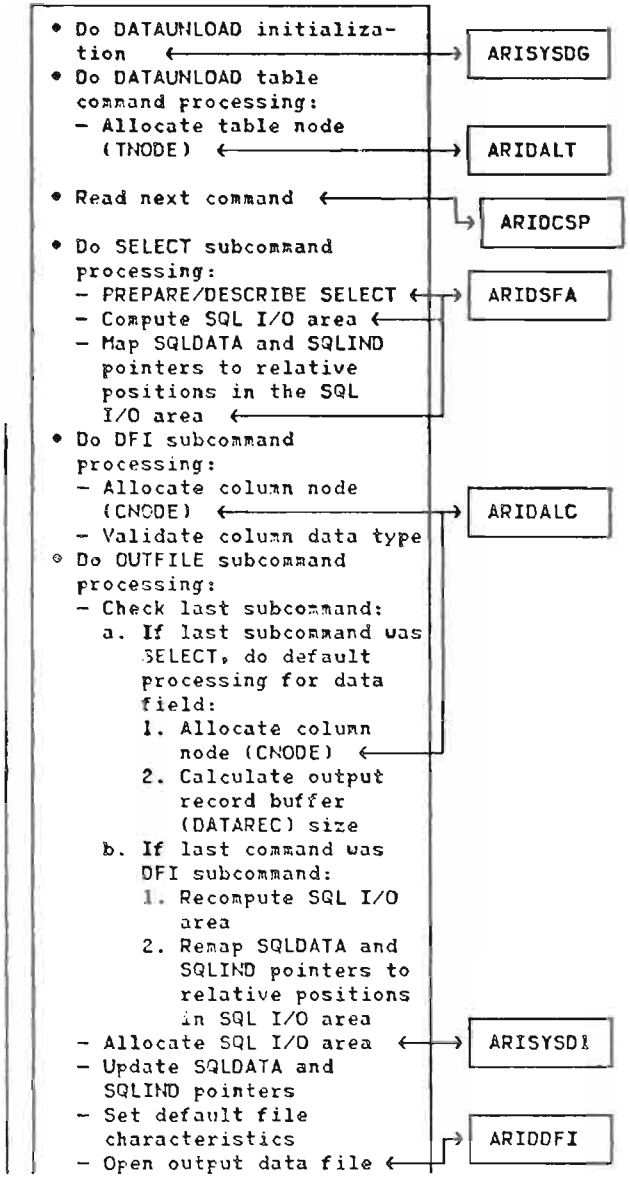


MAJOR DBS UTILITY MODULES AND THEIR FUNCTIONS

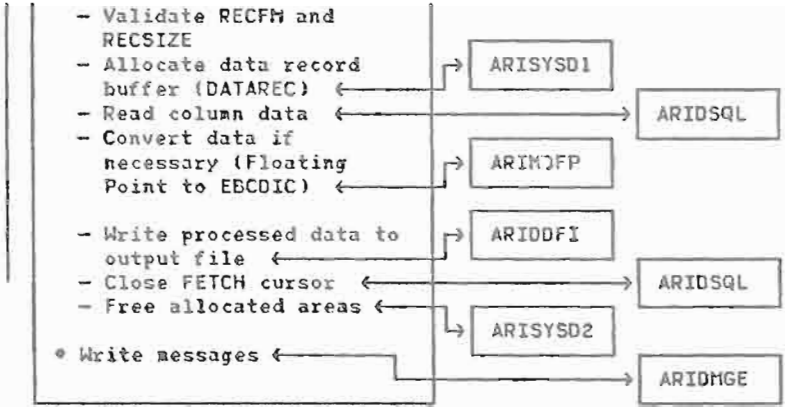
- ARIDBS – DBS Utility monitoring
- ARIDCSP – Control statement processing and command build
- ARIDUCP – DBS command parse
- ARIDDLO – DATALOAD command and sub-command processing
- ARIDDUL – DATAUNLOAD command and sub-command processing
- ARIDUNL – UNLOAD command processing
- ARIDREL – RELOAD command processing
- ARIDSEL – SQL SELECT command processing
- ARIDUSQ – User SQL command processing (except SELECT)
- ARIDSQL – SQL linkage processing

DBS UTILITY - DETAIL FLOW

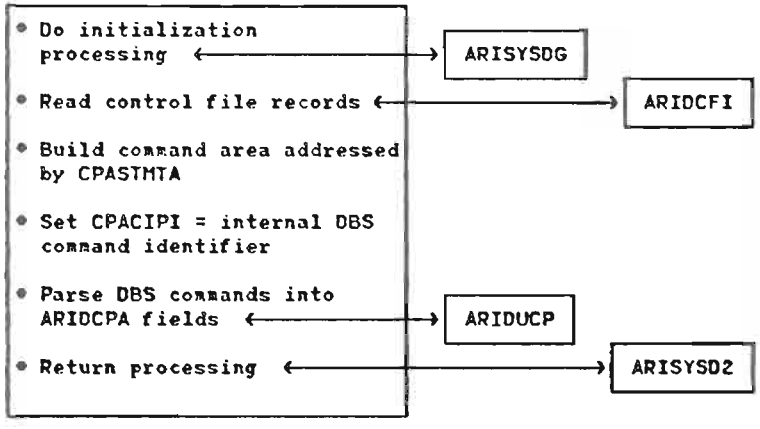


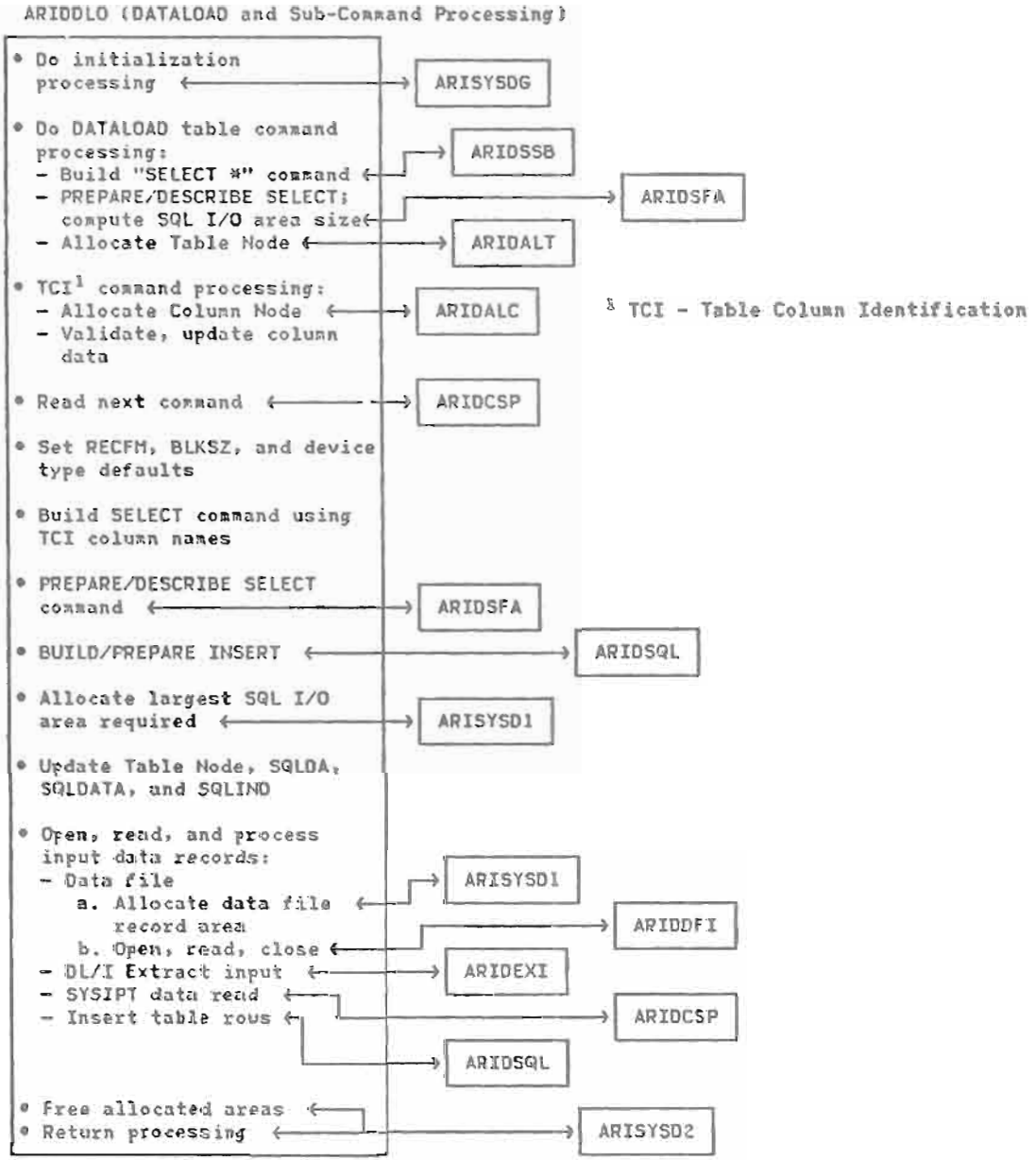


DBS Utility (continued)

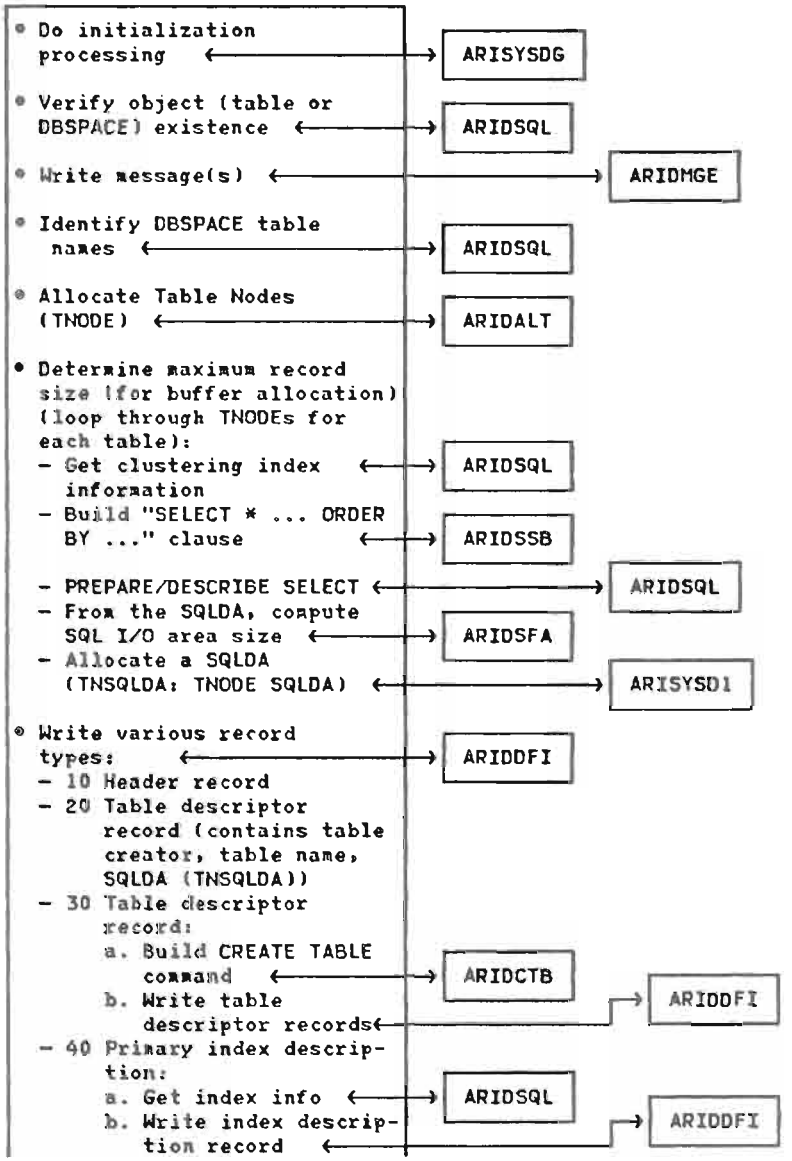


ARIDCSP (Control Statement Processing)

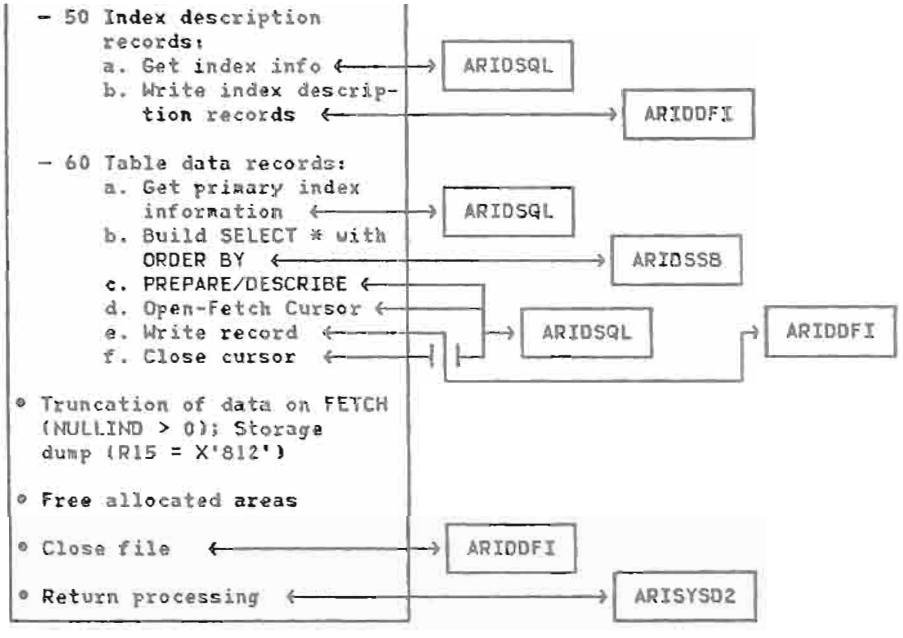



ARIDDLO (DATALOAD and Sub-Command Processing)
• Do initialization processing
ARISYSDG
• Do DATALOAD table command processing:
- Build "SELECT *" command
- PREPARE/DESCRIBE SELECT; compute SQL I/O area size
- Allocate Table Node
ARIDSSB
ARIDSFA
ARIDALT
• TCI[1] command processing:
- Allocate Column Node
- Validate, update column data
ARIDALC
[1] TCI - Table Column Identification
• Read next command
ARIDCSP
• Set RECFM, BLKSZ, and device type defaults
• Build SELECT command using TCI column names
• PREPARE/DESCRIBE SELECT command
ARIDSFA
• BUILD/PREPARE INSERT
ARIDSQL
• Allocate largest SQL I/O area required
ARISYSD1
• Update Table Node, SQLDA, SQLDATA, and SQLIND
• Open, read, and process input data records:
- Data file
a. Allocate data file record area
b. Open, read, close
- DL/I Extract input
- SYSIPT data read
- Insert table rows
ARISYSD1
ARIDDFI
ARIDEXI
ARIDCSP
ARIDSQL
• Free allocated areas
• Return processing
ARISYSD2

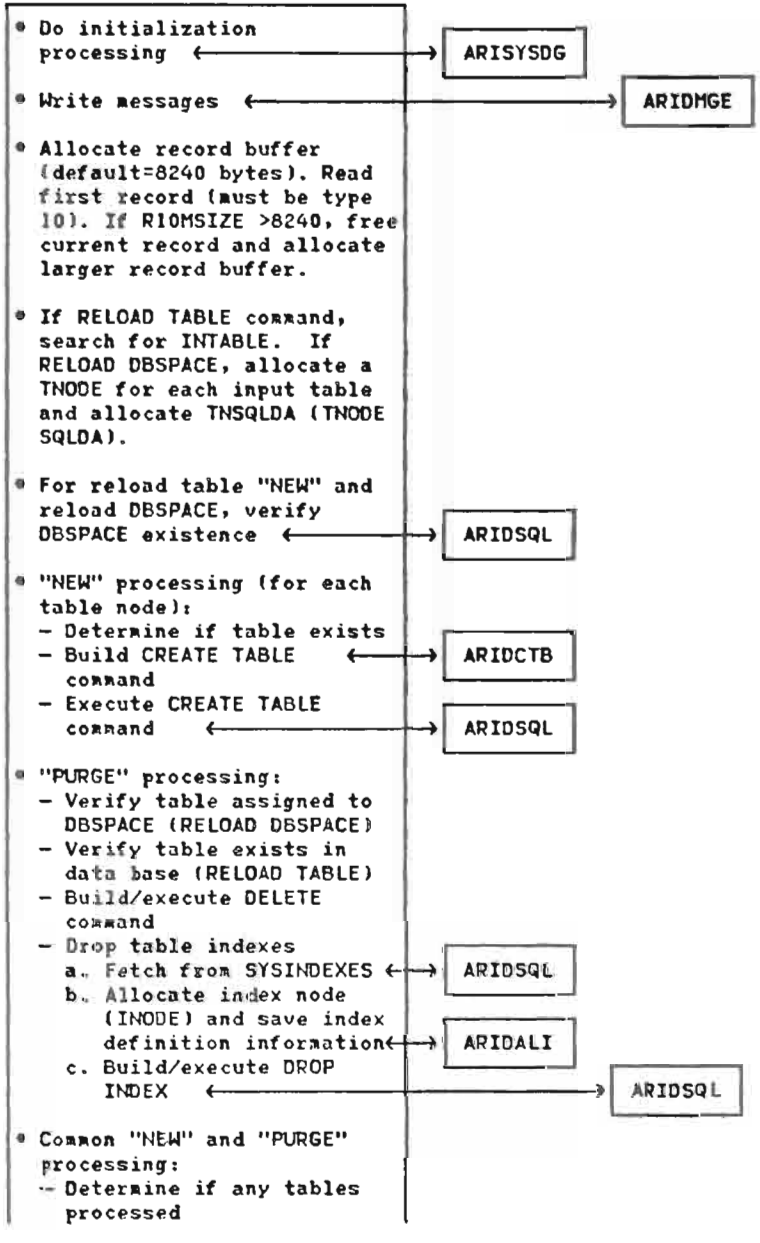
ARIDUNL (UNLOAD Table/DBSPACE Processing)



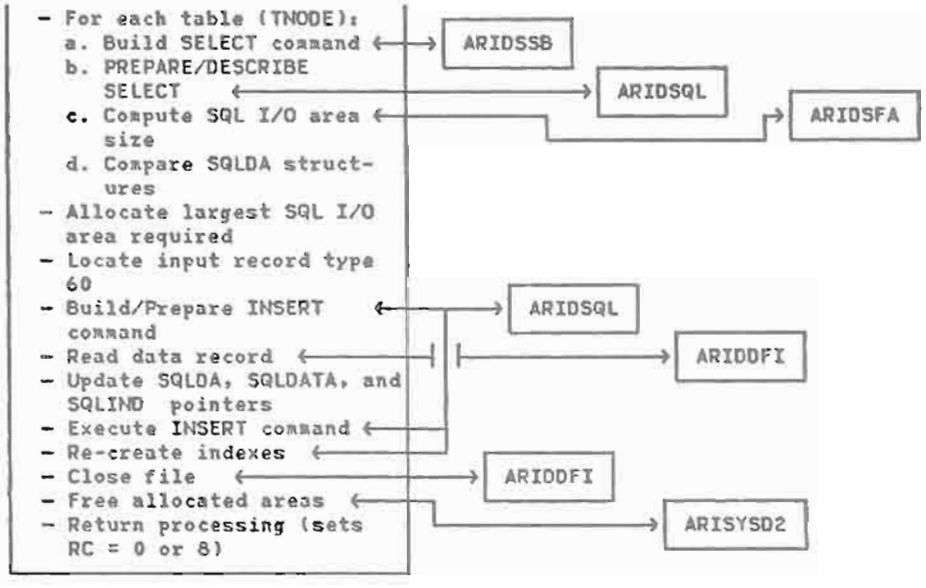
DBS Utility (continued)



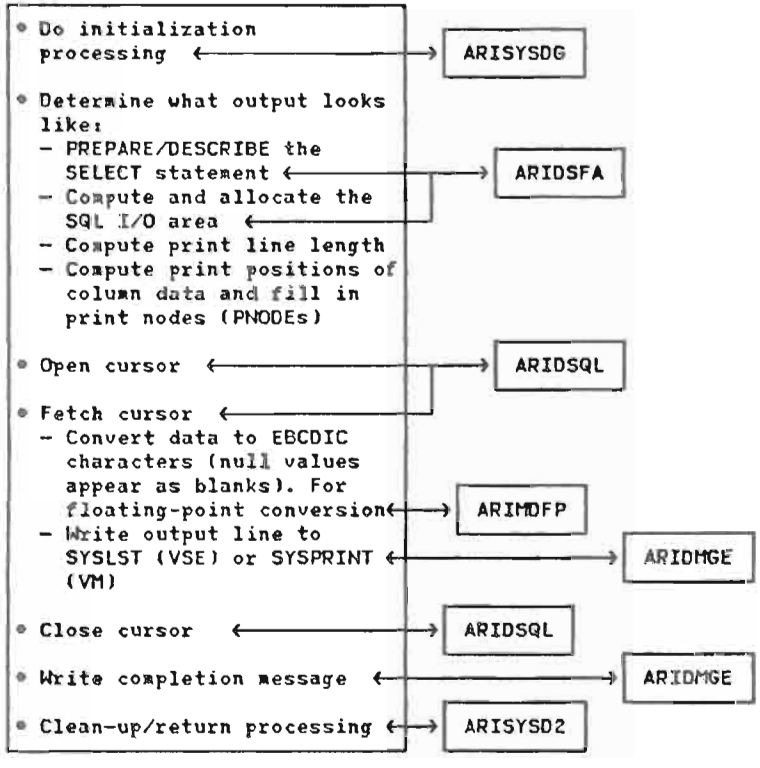
ARIDREL (RELOAD Table/DBSPACE Processing)



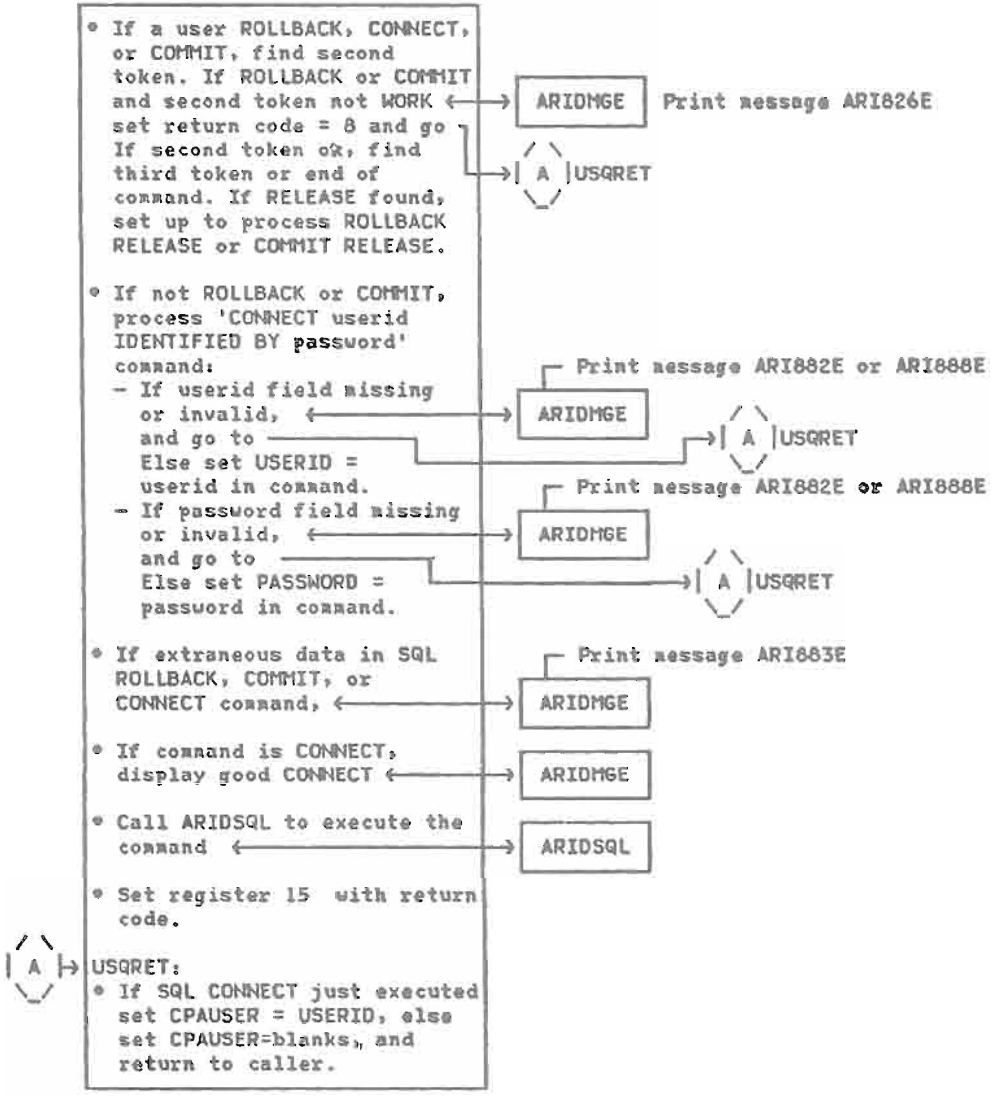
DBS Utility (continued)



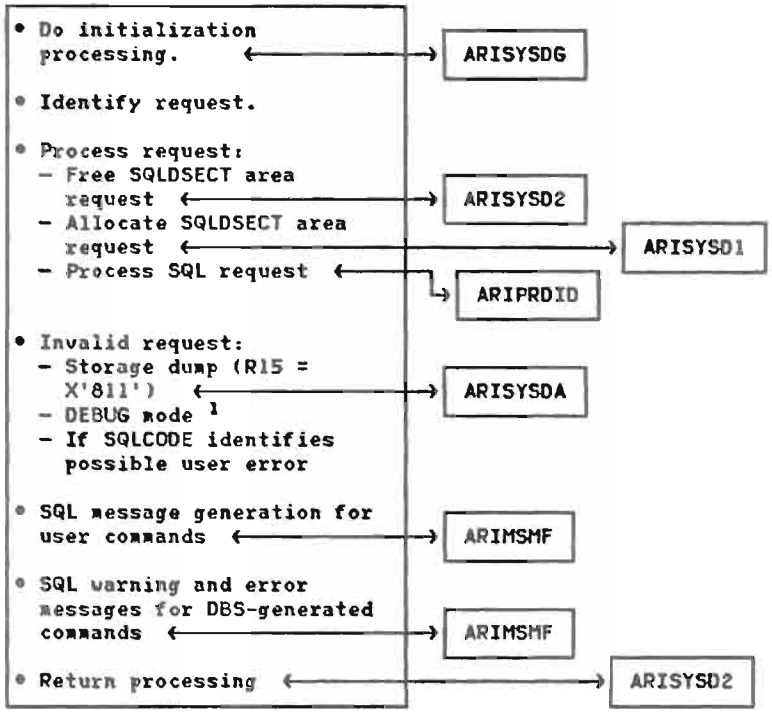
ARIDSEL (SELECT Statement Processing)



ARIDUSQ (User SQL Command Processing - Except SELECT)



ARIDSQL (SQL Linkage)



[1] ARIDSQL Debug Mode Processing:

a. To enter DBS Utility Debug Mode, supply the following commands:

```
.DEBUG
SET ERRORMODE OFF
```

(Note: ".DEBUG" must begin in command record column 1. Invalid command ID error message will result. "SET ERRORMODE OFF" must follow immediately.)

b. R13 + 4 = ARIDSQL save area at ARIPRDID call.

c. Referencing ARIDSQL save area address:
   + 12 = ARISYSDA (storage dump) return address within ARIDSQL
   + 16 = ARISYSDA entry point
   + 32 = Address of SQLTIE (R3)
   + 36 = DBS Utility ARIDCPA address (R4)
   (CPA + X'C' = R0 through R15 special save area)

d. Reference message ARI856E if SQL error occurs for a DBS-initiated SQL command execution.

SECTION 3: PROGRAM ORGANIZATION (EXEC AND MODULE DESCRIPTIONS)

This section includes an alphabetic listing of the EXECs with a brief description of their functions, and an alphabetic listing of the modules and a brief description of their functions. If a particularly complex algorithm is used within a module and it is necessary for the reader to understand it, this is also included. Some modules are not described in Section 2, "Method of Operation", which includes a more complete description of these modules.

For a cross-reference of the called and calling modules, see "Module Calls: To/By" in Section 6, "Diagnostic Aids."

For a cross-reference of the module and data areas, see "Data Areas to Modules Cross Reference" in Section 6, "Diagnostic Aids."

For the SQL code(s) and RDS code(s) detected by any module, see "Module Detecting SQL Codes and RDS Codes" or "SQL Code - RDS Codes, Detecting Modules" in Section 6, "Diagnostic Aids."

See Section 6 for the patch areas for each module.

ARICEMGC EXEC

The ARICEMGC EXEC issues messages as requested by the caller. ARICEMGC is called by other SQL/DS EXECs to have messages issued to the user's terminal.

ARICLOC XEDIT

The ARICLOC system product editor macro finds the data base name and SQL/DS virtual machine id in the ARISRMBT MODULE and stacks them for use by the SQLDBID EXEC.

ARISAVES EXEC

ARISAVES is the EXEC which is invoked as a command to perform a CP SAVESYS command for an SQL/DS component that can be defined as a discontiguous shared segment. For more information about ARISAVES or the installation process, refer to SQL/Data System Installation -- VM/SP, SH24-5044.

ARISDBMA EXEC

ARISDBMA is the SQL/DS data base maintenance EXEC, and supports the installation of service and other SQL/DS data base maintenance activities. ARISDBMA may perform the following:

- Prompt for and reset, if necessary, current password for userid SQLDBA
- Recreate SQL/DS DBS Utility, ISQL, and FORTRAN support access modules
- Load/reload SQL/DS English HELP text
- Load/reload SQL/DS sample tables
- Load/reload SQL/DS ISQL sample routines
- If necessary, restore current password for userid SQLDBA

For more information on ARISDBMA or the installation process, refer to SQL/Data System Installation -- VM/SP, SH24-5044.

ARISEMSG EXEC

ARISEMSG EXEC makes certain that the current CP error message function value is properly set (EMSG ON) to support DBS utility command processing. ARISEMSG EXEC must reside on the SQL/DS production minidisk. It is not intended for use by end users.

ARISESCP EXEC

ARISESCP EXEC makes certain that the current terminal LINEND, LINDEL, CHARDEL, and escape symbols are set properly to support SQL/DS command entry from a terminal. ARISESCP EXEC must reside on the SQL/DS production minidisk. It is not intended for use by end users.

ARISLKIT EXEC

ARISLKIT EXEC link edits a specific member (except $$$COSQL) of the SQL/DS production loadlib (ARISQLLD LOADLIB Q) on the SQL/DS production minidisk. ARISLKIT is invoked as a command by the user or by the SQL/DS system link edit EXEC (ARISSLKE). With the exception of the text file ARIRVSTC which resides on the SQL/DS production minidisk, all other text files must reside on the SQL/DS service minidisk.

ARISPDFC EXEC

ARISPDFC is the SQL/DS production minidisk copy EXEC. ARISPDFC performs the following:

- Acquires read access to the 295 minidisk, with filemode A (295 must be the SQLDBA machine's 195 minidisk).
- Copies IBM-supplied files from the SQL/DS production minidisk owned by the SQLDBA data base machine, to the production minidisk owned by the data base machine running this EXEC. (The files copied are determined by the contents of ARISPDEC MACRO A on SQLDBA 195 minidisk.)
- Erases and creates a file named ARISPIDC MACRO Q (identifies this machine as the one that owns the secondary production minidisk).
- Reaccesses the 195 (Q) minidisk in read mode.
- Reaccesses the 191 (A) minidisk.

ARISPDFC resides on the SQL/DS service minidisk, which must be explicitly accessed by the SQL/DS data base machine before ARISPDFC can be executed. Processing using the SQL/DS production minidisk should not be in progress when ARISPDFC is executed.

ARISPROD EXEC

The ARISPROD EXEC is called by other IBM-supplied SQL/DS EXECs to obtain read or write access to the SQL/DS production minidisk. ARISPROD obtains the userid of the machine which owns the SQL/DS production minidisk and its virtual address from the ARISPIDC MACRO file.

ARISQLLD EXEC

The ARISQLLD EXEC determines whether the SQL/DS load library exists on the minidisk as the EXEC. If it does not, a message is written and processing ends with return code 8. If the loadlib is found, a CMS FILEDEF command is issued. If ARISQLLD was invoked with a parameter STACKAM, a line containing the filenode and filename of the load library is placed LIFO in the current level of the CMS program stack.

ARISSERV EXEC

The ARISSERV EXEC is used to obtain read-only access to the SQL/DS service minidisk. The userid of the machine which owns the service minidisk and its virtual address are read from the ARISPIDC MACRO file.

ARISSLKE EXEC

ARISSLKE is invoked as a command to link edit all SQL/DS entries in the SQL/DS production minidisk load library. It performs the following:

- Link edits SQL/DS to the temporary loadlib
- Erases and recreates the SQL/DS production loadlib (All SQL/DS loadlib entries, except $$$COSQL, are recreated or replaced by ARISSLKE.)

The SQL/DS production minidisk cannot be accessed by another VM machine while ARISSLKE is executing.

ARISYSIN EXEC

ARISYSIN EXEC performs the following housekeeping functions which are necessary before invoking I5748XXJ:

- Checks for tape drive at virtual address 181
- Releases any minidisks already accessed as Q and/or V
- Positions distribution tape at file 3
- Invokes I5748XXJ
- Unloads tape from virtual address 181

For more information on ARISYSIN or the installation process, refer to SQL/Data System Installation -- VM/SP, SH24-5044.

ARI0MSGC EXEC

The ARI0MSGC EXEC contains the texts for messages to be issued by EXECs with numbers 000 through 099. ARI0MSGC cannot be executed.

ARI6MSGC EXEC

The ARI6MSGC EXEC contains the texts for messages to be issued by EXECs with numbers 600 - 699. ARI6MSGC cannot be executed.

ARI8MSGC EXEC

This EXEC contains the message text for messages unique to DBS utility processing and issued by the SQLDBSU EXEC. ARI8MSGC cannot be executed.

ARI9MSGC EXEC

This EXEC contains the message text for messages issued by EXEC SQLTRFMT (Trace Formatter EXEC). ARI9MSGC cannot be executed.

ISQL EXEC

The ISQL EXEC is used during ISQL initialization. It establishes the user's environment by setting PF keys and color settings. ISQL EXEC calls SQLISTRT to continue ISQL startup processing. When the ISQL session ends, ISQL restores the user settings for all settings that were altered.

I5748XXJ EXEC

I5748XXJ is the SQL/DS distribution library installation EXEC. I5748XXJ is invoked by the ARISYSIN EXEC to perform the following:

- Link to and format the SQL/DS service and production disks
- Skip files 3 and 4 of the distributin tape (assumes initial file tape position 3), loads file 5 to the

SQL/DS service disk and file 6 to the SQL/DS production disk.

For more information on the I5748XXJ EXEC or the installation process, refer to SQL/Data System Installation -- VM/SP, SH24-5044.

SQLADBEX EXEC

The SQLADBEX EXEC prompts the user for information, builds an input file, and invokes SQL/DS for the ADD DBEXTENT function.

SQLADBSP EXEC

The SQLADBSP EXEC prompts the user for information, builds an input file, and invokes SQL/DS for the ADD DBSPACE function.

SQLASMC EXEC

The SQLASMC EXEC is used at install time to preprocess, assemble, load, and execute the SQL/DS Assembler sample program.

SQLCBLC EXEC

The SQLCBLC EXEC is used at install time to compile, load, and execute the SQL/DS COBOL sample program.

SQLDBGEN EXEC

The SQLDBGEN EXEC prompts the user for information, builds an input file, and invokes SQL/DS for the data base generation function.

SQLDBID EXEC

The SQLDBID EXEC finds the data base name and virtual machine id in the module ARISRMBT. The name and machine id are written to the terminal or placed in the program stack (according to the invocation parameters).

SQLDBINS EXEC

SQLDBINS is the SQL/DS data base installation EXEC. SQLDBINS is invoked by the data base machine owner. The data base is generated using the SQL/DS-supplied starter data base or user-supplied data base generation (DBGEN) specifications. SQLDBINS calls EXECs SQLDBGEN and SQLSTART to create the SQL/DS production minidisk files with the DBGEN control statements, FILEDEFs, and startup information for the data base. SQLDBINS must reside on the SQL/DS Service minidisk. For more information on the SQLDBINS EXEC or the installation process refer to SQL/Data System Installation -- VM/SP, SH24-5044.

SQLDBSU EXEC

The SQLDBSU EXEC invokes the SQL/DS DBS utility in either SQL/DS single or multiple user mode. Its function includes processing the HELP request, querying and starting the console, spooling the virtual printer, writing the start message, and issuing/clearing SYSIN and SYSPRINT FILEDEFs. SQLDBSU is invoked as an EXEC by the user.

SQLFTN EXEC

The SQLFTN EXEC is used at install time to compile, load, and execute the SQL/DS FORTRAN sample program.

SQLGENLD EXEC

The SQLGENLD EXEC generates the bootstrap modules for a DCSSID. The user is prompted for DCSSID and for the bootstraps to be generated.

SQLINIT EXEC

The SQLINIT EXEC copies the Resource Manager and ISQL bootstrap modules to the user's 'A' disk, and completes the generation of the Resource Manager bootstrap.

SQLISTRT EXEC

The SQLISTRT EXEC parses user input parameters, does FILEDEFs for the SQL/DS load library and then invokes the ISQL bootstrap (ARISISBT) to load the ISQL module.

SQLLOG EXEC

The SQLLOG EXEC is invoked by the user to perform the SQL/DS cold log function. SQLLOG allows the user to change between single and dual logging or to change the virtual device addresses of his logs.

SQLPLI EXEC

The SQLPLI EXEC is used at install time to compile, load, and execute the SQL/DS PL/I sample program.

SQLPREP EXEC

The SQLPREP EXEC is used to preprocess and SQL/DS application program in both single- and multiple-user modes. In single-user mode, the SQLSTART EXEC is called to start SQL/DS and to execute the preprocessor. In multiple-user mode the preprocessor is loaded by a NUCXLOAD command. SQLPREP gives control to the preprocessor in order to preprocess the SQL/DS application program. For a more detailed description, see SQL/DATA System Application Programming, SH24-5018.

SQLSTART EXEC

The SQLSTART EXEC invokes SQL/DS for normal execution. SQLSTART may be invoked for either single virtual machine mode or multiple virtual machine mode.

SQLTRFMT EXEC

The SQLTRFMT EXEC is used to run the SQL/DS Trace Formatter utility program ARIMTRA. It invokes CMS XEDIT commands to allow the user to edit a CMS file for control statement input. It gives the user the option to direct Formatter output to a CMS file or to a virtual printer file. SQLTRFMT uses the CMS NUXCLOAD command to load program ARIMTRA.

5748XXJ EXEC

5748XXJ is invoked by the user for SQL/DS service installation. For more information on 5748XXJ or the installation process refer to SQL/Data System Installation -- VM/SP, SH24-5044.

ARICABE

ARICABE is the abnormal termination handler for SQL/DS.

ARICCLA

ARICCLA disables an agent structure. It does this by calling the SQL/DS communications manager (ARICCOM) to disconnect the cross-partition communication (XPTN) or Inter-User Communication Vehicle (IUCV) link. This routine is called only by the SQL/DS dispatcher (ARICDSP). Since the SQL/DS dispatcher has no automatic storage available, the parameter list for this routine (and for ARICENA and ARICRST) is in the DSCAREA of the agent being disconnected. The macro (DENADCL) maps this area.

After the communication link has been disconnected, the agent structure is cleaned up. (For example, the SQL/DS storage reset routine (ARICRST) resets the stacks (connected to the YTABLE1 and the RDAREA) and the working storage (connected to the DSCAREA) for the agent structure).

If VSE and if SQL/DS is in the process of shutting down (the operator shutdown command was issued and the SQL/DS quiesce indicator (YRSTRMQS) is on), then each DCE is made inactive and is not reconnected. In VM, shutdown cannot be completed until all pseudo-agents are disconnected.

Once all the agent structures are inactive, the checkpoint agent structure is posted to either perform a final checkpoint or to do an archive as indicated by the shutdown command.

If VSE and SQL/DS is not in the process of shutting down, the SQL/DS ready/recovery agent is reconnected without change. However, an in-doubt agent structure may be reconnected either as an in-doubt agent or a general purpose agent. This is determined by the setting of the in-doubt indicator (TPMNDOUT) in the TPMAP entry for the agent. A general purpose agent structure is reconnected in the same manner as an in-doubt agent structure.

If VM, and if SQL/DS is not quiescing or all users are not disconnected, turn on the indicator to request Enable Agent handling to remove pseudo agents from real agents.

ARICCOM - VSE

ARICCOM performs the SQL/DS cross-partition communication function. It is the linkage between SQL/DS and the operating system. ARICCOM is included with the DBSS, the RDS, and the Resource Manager (CICS and Batch). It provides subroutines to perform the following functions:

- Identify and remove SQL/DS as a subsystem (SUBSID)
- Identify SQL/DS or the Resource Manager as a user of CPC services (LOGON)
- Connect SQL/DS and the Resource Manager in either a specific or general connection (connect any and connect specific)
- Send (with reply) a message across the link (SENDR)
- Receive a message across the communication link (RECV)
- Reply with a message across the communication link (REPLY)
- Wait explicitly for a communication process to complete (WAIT)
- Clear or retract a message sent (RESETS)
  - Issued by SENDR side
- Clear to reject a message sent (RESETR)
  - Issued by receive/reply side
- Disconnect a communication link, either individually (DSC), unconditionally (DSCUN), or all links (DSCALL)
- Terminate SQL/DS or the Resource Manager as a user of CPC services through normal termination (LOGOFF), unconditional termination (LOGUN), or indicate SQL/DS is shutting down after all agent structures have terminated (SHTDN). Also, capability allows for an implicit

  wait (communication manager waits) or explicit wait (caller waits) and for a user exit (for an implicit wait only).

ARICCOM - VM

ARICCOM performs the SQL/DS Inter-User Communication Function. It is the linkage between SQL/DS and the operating system. ARICCOM is included with the DBSS, the RDS, and the Resource Manager. It provides subroutines to perform the following functions:

- Identify SQL/DS or the Resource Manager as a user of IUCV services (LOGON)
- Connect SQL/DS and the Resource Manager in either a specific or general connection (connect any and connect specific)
- Send (with reply) a message across the link (SENDR)
- Receive a message across the communication link (RECV)

- Reply with a message across the communication link (REPLY)
- Wait explicitly for a communication process to complete (WAIT)
- Clear to reject a message sent (RESETR)
  - Issued by receive/reply side
- Disconnect a communication link, either individually (DISCONNECT PURGE), or all links (DISCONNECT ALL)
- Terminate SQL/DS or the Resource Manager as a user of IUCV services through normal termination (LOGOFF). Also, capability allows for an implicit wait (communication manager waits) or explicit wait (caller waits) and for a user exit (for an implicit wait only).
- Connect to a DASD device (OPEN).
- Disconnect from a DASD device (CLOSE).
- Reset mask field (SETMASK) for IUCV interrupt control.

ARICCRA

ARIICRA creates the agent structures for SQL/DS. It gets and allocates the control blocks for the agent structure from the primary working storage obtained by ARICIP1. (For example, the working storage associated with prototype YTABLE1/DSCAREA.) ARICCRA gets and initializes the following control blocks.

- DSCAREA - (and adds it to the DSCAREA chain)
- YTABLE1 - (and adds it to the YTABLE1 chain)
- Scan table
- MODCB parameter list (VSE only)
- Working storage for the agent structure
  If multiple-user mode:
  - Communication manager parameter list (if VSE, and except for checkpoint agent)
  - Connect control block (if VSE, and except for operator and checkpoint agents)
  - Default input mailbox (except for operator and checkpoint agents)
  - Output mailbox (except for operator and checkpoint agents)
  - Sets the pointers to the connect and receive ECBs in the "multiple wait" list
- Output mailbox parameter list (except for operator and checkpoint agents)
  If STARTUP = WARM or RESTORE
  - RDAREA (and adds it to the RDAREA chain)
  - Stack for the agent structure (RDAREA)

Prior to setting up the RDAREA and its stack, a call is made to ARIYM00 to perform initialization of the transaction map (YTPMAP) (and the RPLs used for VSAM I/O - VSE) except for the operator agent structure. After the RDAREA (and its stack) is set up, ARIXEST is called to complete RDAREA initialization. At this point, a SQL/DS agent structure has been created and initialized.

ARICDMP

ARICDMP is the SQL/DS dump routine. It displays storage according to the parameters passed. The four parameters are:

- Title
- Title length
- Dump area
- Dump length

If the title length is zero (0), then no title is printed. The format of the dump is:

Address	Bits 1- 6
Offset	Bits 8-13
Dump data (hex)	Bits 16-50
Dump data (character)	Bits 53-72

The dump data is displayed 16 characters per line. Both the hexadecimal and the printable character are displayed. A period (.) is substituted for non-printable characters in the printable character part of the line. The offset is always from the start of the dump area. Also, duplicate lines are suppressed.

ARICDPT

ARICDPT posts the agent structure designated in the parameter passed via the call. It turns off either the general wait or lock/latch wait flag in the DCE. It sets the DS2DCESC flag (in the DS2CVT) on to ensure that the Dispatcher scans all DCEs in the chain.

ARICDSP

ARICDSP is the SQL/DS dispatcher. It dispatches the SQL/DS real agent structures in a round-robin fashion. If, while in multiple-user mode, ARICDSP detects that a communication link has been terminated by the user application partition or virtual machine, it invokes the agent structure clean-up process to make the agent available to any new users. If the application terminated normally, SQL/DS drives the implicit commit work process. If the application terminated abnormally, the implicit rollback work process is initiated. In VSE, if the user (CICS application) issues a CANCEL, the SQL/DS dispatcher starts up the rollback process and drives agent clean-up without disconnecting the communication link. In VM, an SQLHX issued by the user machine causes the communication link to be severed and reconnected.

ARICDWT

ARICDWT sets the various wait indicators in the DCE according to the status indicated by WAITTYPE. For a SUSPEND call, do not set any wait indicator. Instead, set the DS2DCESC indicator in the DS2CVT. This gives up control to any other SQL/DS task (agent) that is ready to run. If none are ready to run, then the suspended task (agent) regains control. The DS2DCESC indicator is always set regardless of WAITTYPE. Then call the Dispatcher Wait routine.

ARICENA

ARICENA enables an agent structure by setting the DCE flag (DCEPWF) to 0. It also sets the entry registers to the module to be invoked when the agent structure is called. Plus ARICENA sets a pointer to a save area reserved in the DSCAREA.

In VSE and multiple-user mode, it also performs the CPC CONNECT function for the ready/recovery, in-doubt, and general purpose agent structures. If no corresponding connect is done by the user application partition, then set the DCE flag (DCEXPTN) to indicate the agent structure is waiting for cross-partition communication. In VM and multiple-user mode it allocates and deallocates pseudo-agents to real agents (see discussion on page 160).

ARICFSE

ARICFSE is called when a stack extension is no longer needed. It frees the current stack extension and re-chains pointers back to the previous stack extension (or base stack).

ARICGSE

ARICGSE is called to allocate and initialize a new stack if the old one does not contain enough space for the current request.

ARICIMB

ARICIMB resides in the application side partition or virtual machine. Its function is to compact all the input data (from the user) and place it into a buffer (input Mailbox buffer) in the same sequence as requested. This module also manages the buffer allocation and deallocation (input Mailbox) when the original buffer is not big enough for the input request. The original (default) buffer, which exists until the end of SQL/DS operation, is allocated by the Resource Manager. ARICIMB builds the input Mailbox upon different opcodes (up to 13) passed to it. All the input data information exists in the RDS RDIIN control block. The pointer to RDIIN resides in IIFPARM (see "Mailbox Data Areas" in Volume 2, Section 5).

The following are the opcodes (call types) passed to ARICIMB:

CALL TYPE (RDICTYPE)	MEANING
30	AUX CALL
35	SET UP CALL
40	DESCRIBE CALL
45	CLOSE CALL
50	OPEN CALL
120	SCHEDULE OR CONNECT CALL
125	RECOVERY CALL
130	PREP INIT CALL
131	CREATE PROGRAM CALL
132	DROP STATEMENT CALL
135	LOOKUP CALL
140	SQL CALL
145	PREP FINISH CALL
155	OPERATOR COMMAND CALL
160	PREPARE-TO-COMMIT CALL
165	SET/RESET EXIT CALL
170	OPERATOR COMMAND CONTINUE CALL

ARICIMB has to know the size of input to Mailbox (RDIMBLEN in RDIIN) in order to use the right buffer for the input Mailbox (overflow or default buffer size). Also, it has to inform the Resource Manager which buffer has been used for Mailbox. There is a message size length at label IIFMSGLN in IIFPARM. If IIFMSGLN is greater than the default Mailbox size, the overflow buffer is used as the Mailbox; otherwise the default Mailbox is used. (Default buffer size is declared externally in ARICIMB). If RDIMBLEN has not been calculated, ARICIMB will find the size of input (first pass).

After building the input Mailbox, ARICIMB returns to the Resource Manager.

ARICINTC

ARICINTC handles external interrupts in the SQL/DS machine that occur as the result of IUCV functions. The only possible interrupt types in this case are CONNECT PENDING, MESSAGE PENDING, and SEVER PENDING. The occurrence of any other type is treated as a SYSTERR condition.

A secondary entry point in the module handles external interrupts arising from asynchronous I/O requests (SENDF).

ARICIP1

ARICIP1 is the first step in the SQL/DS initialization process (Phase 1). It gets and allocates storage for the primary SQL/DS control blocks and agent structures. This is based on the information obtained from the SQL/DS parameter processing routine (ARICPRM). The abend exit is also established.

If VM and single virtual machine mode, and if STARTUP = C, the GENCAT module (ARIGCAT) is loaded. If STARTUP = S, the ADDSEG module (ARISEGB) is loaded. If STARTUP = W or R, the user application is loaded.

If STARTUP = WARM or RESTORE:
- If VSE, RDS phase is loaded and its initialization routine is called.
- If VM, the RDS was previously loaded by the ARISDBBT bootstrap module. ARICIP1 needs only to call RDS initialization.

The operator agent structure is created, enabled, and initialized to dispatch the SQL/DS Initialization Phase 2 routine.

At this point, a dispatchable agent structure (operator) is built. The remainder of SQL/DS initialization can run under this agent structure and invoke the SQL/DS Dispatcher Wait routine. Finally, pointers to the DS2CVT (Register 11) and DCE (Register 13) are set and the SQL/DS Dispatcher is invoked.

After SQL/DS termination (ARICTRM) is complete, the address of CLEANUP (a label in ARICIP1) is obtained from the DS2CVT (DS2RETRG), and ARICTRM returns to this point. When the storage cleanup completes, control returns to the operating system.

ARICIP2

ARICIP2 completes the initialization of SQL/DS (Phase 2). It executes under the operator agent structure.

In VSE multiple-user mode, it calls the SQL/DS Communication Manager routine (ARICCOM) to identify (to VSE Supervisor) SQL/DS as a subsystem and to identify SQL/DS as a user of VSE CPC services. It initially performs 'IDENTIFY' for the general purpose (user) agent structures and for the in-doubt agent structures. After other agent structures are built, ARICIP2 also performs an 'IDENTIFY' for the SQL/DS ready/recovery agent structure.

If VM, the SQL/DS Communication Manager (ARICCOM) is called to perform an IUCV SETCMASK function to enable interrupts.

Regardless of SQL/DS mode (single/multi) or operating system, it calls ARIYT00 to complete the DBSS initialization. When STARTUP=L or E, no further processing is required. Then return to the operating system. When STARTUP=W or R, call ARIXER0 to complete RDS initialization.

The general purpose (user) agent structures are created based on the NCUSERS value minus the number of in-doubt agents. (When in-doubt agent processing is complete, the in-doubt agent structures are converted into general purpose agent structures.) For VM, the number of in-doubt agents is zero. In multiple-user mode, the general purpose agent structures are initialized to invoke RDS (ARIXERD). In single-user mode, there is only one general purpose agent structure (and no in-doubt agent structures). It is initialized to invoke ARICSPM. After SQL/DS initialization is complete, call ARIYM01. ARIYM01 establishes operator communication capability and passes control to the SQL/DS dispatcher. The SQL/DS dispatcher dispatches the first SQL/DS agent structure that is ready to run.

ARICMMB

ARICMMB resides in the application side partition or virtual machine. Its function is to decompact the data passed by SQL/DS to the input Mailbox and replace it in the user area.

The pointer to the input Mailbox is passed by the Resource Manager.

The Resource Manager is the linkage between the application programs and ARICMMB. ARICMMB communicates with the Resource Manager only.

ARICMOD (DS2MODE)

ARICMOD is the SQL/DS mode indicator for the Resource Managers. This module is loaded in both in single- and multiple-user modes. In single-user mode the indicator switch (DS2MDIND) is set to 'S' and the Resource Manager will test it to determine the mode. In multiple-user mode the indicator switch (DS2MDIND) is set to 'M' (pre-initialized at compile time). When the Resource Manager loads this module, it will find the indicator set to 'M' and then execute in a multiple-user mode environment. There are other fields that are also initialized for use by

the Resource Manager. The 'DS2MDSCP' field is initialized with the pointer to the 'DS2CVT'. There is also a pointer to the Resource Manager control block chain 'DS2MRMLP' (This is set in multiple-user mode only.)

ARICMUD

RDS uses ARICMUD to receive data via cross-partition communications or via the Inter-User Communication Vehicle (IUCV). ARICMUD calls the SQL/DS Dispatcher Wait routine (via ARICWAT) to wait for a message sent by the user partition or virtual machine. When the message is to be sent, ARICMUD checks if the SQL/DS buffer is large enough to contain it. If not, a larger buffer is obtained. If a larger buffer cannot be obtained, ARICMUD will PURGE (VSE), or REJECT (VM) the message. If the buffer is now large enough to contain the message, ARICMUD calls the SQL/DS Communication Manager (ARICCOM) to receive the message. If no error occurred, from the RECEIVE function, the message and its length are passed back to RDS. If a severe error occurs, then SQL/DS terminates (ARIYM02 is called). If it is not a severe error, call ARICWAT and the user is backed out.

ARICOMB

ARICOMB resides in the SQL/DS partition or virtual machine. Its function is to "get" the SQL/DS output to the user area. ARICOMB is sensitive to the running-mode environment. There is a flag (RDASYSM) in the RDAREA which states the running environment of the system ('S' = single user mode, 'M' = multiple-user mode). If SQL/DS is running in the same partition or virtual machine as a user is running, ARICOMB simply moves the data to the user area (no Mailbox). However, in a multiple-user mode environment, ARICOMB would use the Mailbox facility to pass the data to the user partition or virtual machine. The main functions when using the Mailbox facility in a multiple-user mode environment are:

- Build an output element in output Mailbox buffer
- Pass the output data (Mailbox) across partitions or machines (communication link) at the end of process.

Transfer of the output Mailbox happens in two cases:

1. The process is done and complete.
2. It is not the end of process but the output Mail box buffer is full (segmentation).

The following would cause the second case (segmentation):

- The output message length is greater than the output Mailbox buffer size.
- There is not enough room in the output Mailbox buffer.

In the above cases, ARICOMB segments the output data to the Mailbox capacity and then ships the Mailbox to the user partition or virtual machine. This action repeats until the whole output data is transferred. A WAIT is issued in the segmentation process until control is returned. This insures that output data is received in the other partition or virtual machine and placed in the user area before shipping the next segment.

OIFPARM, containing input parameters, is the linkage between the caller and ARICOMB when ARICOMB is called to build an output element in the output Mailbox or to do a simple move (single-user mode). The pointer to OIFPARM is in the DSCAREA at DSCOMBPP. (See "Mailbox Data Areas" in Volume 2, Section 5 for OIFPARM.)

ARICPDM

ARICPDM is the SQL/DS snap dump routine. It dumps according to the value specified in the SQL/DS DUMPTYPE initialization parameter.

If VSE and DUMPTYPE = F, ARICPDM dumps the entire partition. If some of the SQL/DS code is in the SVA, this code is also dumped.

If VM and DUMPTYPE = F, ARICPDM dumps the entire virtual machine. If some of the code is in the discontiguous saved segment area (beyond the end of the virtual machine), it is also dumped.

If VSE and DUMPTYPE = P, ARICPDM dumps the:

- Partition SAVEAREA
- Area between the start of the GETVIS area and the RDS (if loaded)
- Area between the RDS and the application program (if it is a SQL/DS phase and it is loaded)
- Area between the application program (for example, the SQL/DS code) and the Batch Resource Manager (if it is loaded)
- Area between the Batch Resource Manager and the end of the partition
- The DBSS/DSC and RDS link maps.

If VM and DUMPTYPE = P, ARICPDM dumps the:

- Area between the start of the virtual machine and the VM Resource Manager (if loaded).
- Area between the VM Resource Manager and the "user program" (if loaded, single-user mode only).

- Area between the "user program" (if loaded, single-user mode only) and the RDS/DBSS/DSC code.
- Area between the RDS/DBSS/DSC code and the end of the virtual machine.
- The DBSS/DSC and RDS link maps.

For VSE and VM, if DUMPTYPE = N, no dump is taken.

Note: If the application program is the user code and not a SQL/DS phase/program, the application program is dumped because it may not have been coded as re-entrant. This allows any 'data areas' to be displayed in the dump.

ARICPRM

ARICPRM is the SQL/DS parameter processor. It processes SQL/DS parameters entered on JCL (VSE) or SQLSTART EXEC (VM) when starting SQL/DS. For VSE, ARICPRM overrides the default SQL/DS parameters with parameters specified in source statement library member specified on 'PARMID=' parameter. For VM, ARICPRM overrides the default SQL/DS parameters with parameters specified in a CMS file with filename specified on 'PARMID=' parameter and filetype = "SQLPARM". ARICPRM also overrides that with parameters specified explicitly in the JCL (VSE) or SQLSTART EXEC (VM).

The basic flow is:

1. Initialize the default table with default SQL/DS parameters.
2. If no parameters are specified via the JCL (VSE) or SQLSTART EXEC (VM), then go to step 11.
3. If parameters are specified on the JCL (VSE) or SQLSTART EXEC (VM), then initialize the JCL parameter table (JCLTABLE) and the parameter data set table (PRMTABLE) to zero and blank values.
4. Scan the JCL (VSE) or SQLSTART EXEC (VM) parameter string for a '/', that indicates the beginning of the user parameters.
5. If a '/' is found, save the user parameters and the length of the user parameters (for example, the substring from the '/' to the end of the JCL or SQLSTART EXEC parameter string). Also, adjust the length for the JCL or SQLSTART EXEC parameter string so that it is scanned from the beginning of the JCL or SQLSTART EXEC parameter string to the '/', instead of being scanned over its total length. (You do not want to scan the user parameters.) If a '/' is not found, set up parameter linkage to reflect no user parameters. (DS2PARMP = 0, ULEN = 0) (If VM, set up an untokenized parameter list with the user parameter information (pointer to user application name, followed by the beginning address of the user parameters, followed by the ending address of the user parameters plus one byte, followed by a fullword '0'). The user application name and user parameters are moved into the areas indicated by this plist.)
6. Scan the JCL (VSE) or SQLSTART EXEC (VM) parameter string, getting one parameter at a time until the end of the JCL or SQLSTART EXEC parameter string. (It calls the tokenizer (see attached ARICTKN module.) Check each parameter against the table of valid parameters. If a match is found, call the corresponding routine to process it. If no match is found, display an error message and terminate the job.
7. After completing the scan, check to see if a name was saved as a result of 'PARMID=' being specified via the JCL (VSE) or SQLSTART EXEC (VM).
8. If there is a name saved (for example, if there is a parameter member data set to process), call PROCPDS routine to process the parameters in the parameter member data set. If no read error occurs in PROCPDS, merge the parameter member values (PRMTABLE) into the default table. If a read error does occur, then display an error message and go to step 12.
9. Merge the JCL parameter table (JCLTABLE) into the default parameter table.
10. Call CHKCONFL routine to check for conflicts according to defined hierarchy. If no conflicts exists, go to step 11. If conflicts exists, display the appropriate error message and go to step 12.
11. Display the SQL/DS parameter values in the default table.
12. Return to caller.

ARICRST

ARICRST resets the storage associated with an agent after an abnormal termination of the agent process. The RDS and DBSS stacks (automatic storage) are reset to their initial states (no overflow and empty). The input mailbox is reset to be the initial default buffer. All secondary (overflow) working storage is freed. Additionally, the miscellaneous status/state flags are reset in the DSCAREA.

ARICSHO

ARICSHO is a VM system-dependent module that enables the operator to display the VM userids and SQL/DS-ids of the users connected to SQL/DS via the IUCV. ARICSHO displays the users connected to real agents, the users waiting to connect to a real agent (in the order that they will be allocated a real agent), and the users that are connected to SQL/DS, but inactive.

ARICSHT

ARICSHT is the SQL/DS shutdown routine. It is invoked by the SQL/DS command processor when the system operator (VSE) or virtual machine terminal operator requests SQL/DS to shutdown. There are three types of shutdown:

1. Normal (operator issues 'SQLEND [NORMAL]' or 'SHUTDOWN [NORMAL]')
   - If VSE, SQL/DS issues a terminate quiesce to deactivate any inactive agent structures (the communication links are disconnected and no new users are allowed to connect to SQL/DS). Current users are allowed to complete their processing. When they disconnect from SQL/DS, their communication link is disconnected but not reconnected. If VM, external interrupts are disabled. No new users are allowed to connect. Once all agents become inactive, checkpoint is invoked (by clean-up agent - ARICCLA) to perform a final checkpoint (for VM, external interrupts must be re-enabled). Checkpoint also calls the SQL/DS termination routine (ARICTRM). If no agents are active when the SQLEND or SHUTDOWN operator command is issued, checkpoint is activated to perform a final checkpoint and it calls the SQL/DS termination routine.
2. Archive (operator issues 'SQLEND ARCHIVE' or 'SHUTDOWN ARCHIVE')
   - SQL/DS performs the same function as normal shutdown. In addition, it sets the 'archive requested' (YRSTRMAR) indicator on. When checkpoint is called, it performs the archive function.
3. Quick (operator issues 'SQLEND QUICK' or SHUTDOWN QUICK')
   - SQL/DS performs an immediate shutdown. It calls the SQL/DS termination routine (ARICTRM), which calls the SQL/DS trace close routine and goes to end-of-job. Then the operating system performs the communication disconnect function and the user application partition/virtual machine is posted with a 'SQL/DS disconnected and went to EOJ' indication. For VSE only, any active LUWs in prepare-to-commit status are resolved as in-doubt LUWs the next time SQL/DS is brought up. For VM, the IUCV LOGOFF is performed via a call to ARICCOM.

ARICSPM

ARICSPM is dispatched in single-user mode under the user agent structure.
- If VSE and if STARTUP = C, ARICSPM loads the GENCAT Module (ARIGCAT). If STARTUP = S, it loads the ADDSEG Module (ARISEGB). If STARTUP = W or R, it loads the user application program.
- If VM this loading has been done by ARICIP1 (DSC initialization phase 1).

The pointer to the user parameters (or a fullword of binary zeros) are passed on the call to the application program. Upon return from the application program, SQL/DS termination is called (ARICTRM).

ARICSTK

ARICSTK is called to allocate and initialize the initial stack for the DBSS and RDS of each agent.

ARICTIM

ARICTIM is called to convert the S/370 Store Clock instruction value from a 64-bit binary number to date and time in the format mm/dd/yy hh:mm:ss.

ARICTKN

ARICTKN is the SQL/DS parameter tokenizer routine. It obtains tokens from a character string that uses blanks or commas as delimiters. The input string, its length, and the starting position are input arguments. The pointer to the token, its length, and the index to the next byte following the token are returned as output. The input string is passed in a structure that contains the length of the input area and the pointer to its start. The starting position is an index value beginning with one. The pointer to the token and its length are returned in a structure similar to the input structure. In addition, the index value is updated to the next position after the token (for example, update to a comma, a blank, or else it contains the length of the string plus one).

ARICTRC

ARICTRC is the TRACE operator command processing routine.

Entry Point: ARICTRC
   Function: ARICTRC processes the SQL/DS TRACE operator command.

Entry Point: ARICTRC1
   Function: ARICTRC1 shuts down SQL/DS trace when the SQL/DS partition or virtual machine terminates normally or abnormally.

ARICTRI

ARICTRI is invoked during SQL/DS initialization if either the TRACDBSS or the TRACRDS parameter was specified as a SQL/DS initialization parameter. It supports SQL/DS trace activation via the SQL/DS initialization process (as opposed to the TRACE operator command).

ARICTRM

ARICTRM performs the termination function for SQL/DS. It prints a reason code for terminating SQL/DS. ARICTRM also prints a return code to be passed back to the operating system. ARICTRM also calls the data base close (if SHUTDOWN does not equal "QUICK") and trace close routines. If an error should occur during termination, ARICTRM sets a recursion indicator (YRSTRMST) in the DBSSCVT (YRSSCVT) to prevent it from attempting to retry termination. For VM, the IUCV LOGOFF is performed. ARICTRM returns to ARICIP1 to complete SQL/DS termination.

ARICWAT

ARICWAT is the RDS linkage to the SQL/DS Dispatcher Wait routine (ARICDSP1). It waits for the cross-partition communication or Inter-User Communication Vehicle (IUCV). In case of a cross-partition communication or IUCV failure, it initiates SQL/DS backout procedures (it calls ARIYT23). If communication failure occurs, it calls ARIYT15 to backout the user. After backout has been completed, RDS is redriven from the start (module ARIXERD) whenever the agent is re-dispatched.

ARICWFR

ARICWFR releases all attached extended pools. It does not check for any unallocated area in any extended pool.

ARICWSF

ARICWSF returns the space allocated (by ARICWSG) to the caller back to the storage pool(s).

ARICWSG

ARICWSG allocates the requested area from the storage pool(s) to the caller. (See the diagram on page 480) Upon successful operation, the pointer to the allocated area is returned to the caller (return code 0). If the storage pool(s) is(are) full or not big enough for the request, ARICWSG asks the system for more storage space. The system then chains the allocated space to the current pool(s). However, if the system itself runs out of space (return code 12), ARICWSG returns to the calling program (the caller is turned down for the space request).

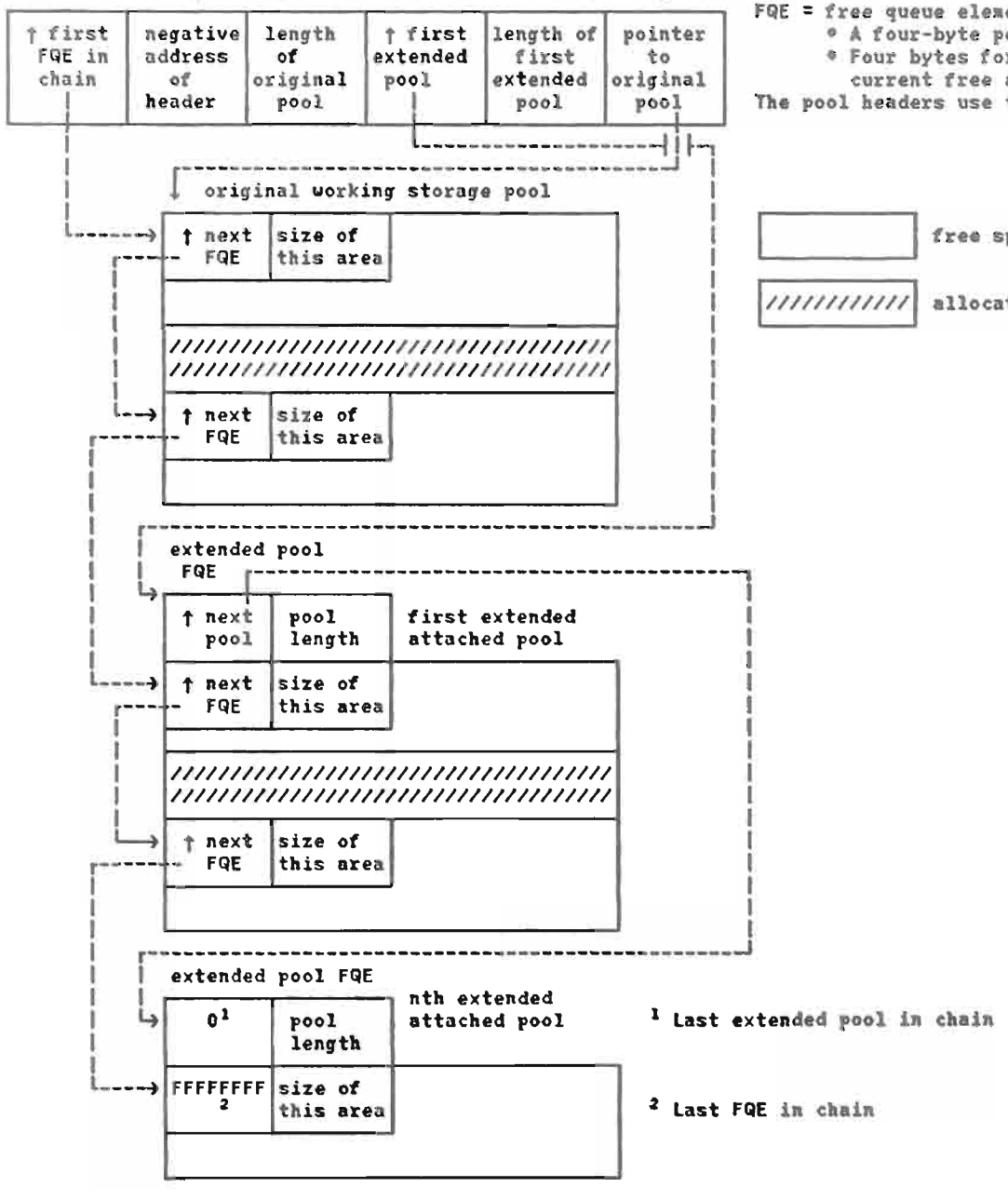
ARICWSI

ARICWSI provides an initial (original) working storage pool for an agent structure. It also initializes the working storage header.

Note: the working storage header is in DSCAREA beginning at offset 40(28).

FQE = free queue element. The FQE consists of two fields:
- A four-byte pointer to next FQE
- Four bytes for the size of the current free area

The pool headers use the same structure.



DATA BASE SERVICES (DBS) UTILITY MODULES

ARIDALC

ARIDALC is the DBS Utility column node (CNODE) storage allocation and free routine.

ARIDALI

ARIDALI is the DBS Utility index node (INODE) storage allocation and free routine.

ARIDALT

ARIDALT is the DBS Utility table node (TNODE) storage allocation and free routine.

ARIDBS

ARIDBS is the SQL/DS DBS Utility Monitor.
ARIDBS:

- Is the main DBS Utility entry point.
- Acquires virtual storage for the DBS Utility CPA (Common Processing Area), command build area, and SQLDA.
- Initiates a message file open request.
- Initiates a command file open request.
- Calls the command file read and parse module (ARIDCSP).
- Calls the various command processor routines while the message file and the command file are in an open state.
- Initiates a SQL logical unit of work COMMIT after the successful processing of most commands if the AUTOCOMMIT mode is ON.
- Initiates a SQL logical unit of work COMMIT if a valid command file EOF condition occurs during a logical unit of work and no previous errors have occurred.
- Initiates a SQL logical unit of work ROLLBACK if an error occurs.
- Initiates error processing (ERRORMODE ON) during which only general DBS Utility command parsing is performed and no SQL commands or data file operations are executed.
- Initiates a command file close request.
- Initiates a request to free ARIDSQL SQLDSECT area.
- Initiates a request to close the message file.
- Frees all real storage areas acquired at start of program.

ARIDCFI

ARIDCFI is the DBS Utility command file input routine. ARIDCFI performs three functions: OPEN command file processing, READ command file processing, and CLOSE command file processing.

ARIDCIB

ARIDCIB builds a SQL CREATE INDEX command in the area identified by and from the information provided by the calling program.

ARIDCSP

ARIDCSP is the DBS Utility command file record (control statement) processor.
ARIDCSP:

- Reads the command file records. It assumes that the command file is opened before the first call is executed.
- Constructs the command from command file records in the area addressed by CPASTMTA.
- Identifies the command and sets the command-in-process identifier field CPACIPI.
- Parses DBS Utility commands and updates command information in the CPA.

ARIDCTB

ARIDCTB builds a SQL CREATE TABLE command that is placed in the output file resulting from the DBS Utility unload processing. ARIDCTB is also called for subsequent RELOAD NEW processing.

ARIDDFI

ARIDDFI calls ARISYSD5 to open the data input or output file, to read from the data input file, to write to the data output file, and to close the data file.

ARIDDLO

ARIDDLO performs the DBS Utility DATALOAD command processing. It calls ARIDCSP to read and parse DATALOAD subcommands. If an INMOD subcommand is encountered, ARIDEXI

is called to communicate with the DL/I EXTRACT modules (VSE only).

ARIDDUL

ARIDDUL performs the DBS Utility DATAUNLOAD command and subcommand processing. It selects data from the data base. It does data conversions, if necessary, and writes the selected data to a sequential output file.

ARIDEXI

ARIDEXI is the SQL/DS DBS linkage to the user-supplied module identified as the PHASENAME1 parameter of the DATALOAD INMOD subcommand.

ARIDMGE

ARIDMGE calls ARISYSD5 to open the message file, to write to the message file (SYSLST for VSE or SYSPRINT for VM), to print to the message file, and to close the message file. It also calls ARISYSD3 to write a message to the operator console.

ARIDREL

ARIDREL performs the reload processing requested by a RELOAD TABLE command or a RELOAD DBSPACE command. Table(s) are either created new or are purged (all rows deleted). New rows are then inserted from the data in the input records. If the table(s) are purged, all tables indexes are dropped and then re-created after a table is reloaded. An UPDATE STATISTICS command is issued for each table reloaded.

ARIDSEL

ARIDSEL is the DBS Utility SELECT command processor. It writes the query output to the DBS message file in either a column or blocked format.

ARIDSFA

ARIDSFA is the DBS Utility SQL field analysis routine.

ARIDSFA performs the following analyses:

1. CALLTYPE = "ALLOCATE" (= 0):
   The user input SQL SELECT command (at address CPASTMTA) is prepared and described by calling ARIDSQL (CPAIPRDE). ARIDSFA obtains an I/O area of the computed size by calling ARISYSD1 and places the address in CPASFIOA.
2. CALLTYPE = "COMPUTE" (= 2):
   The user input SQL SELECT command (at address CPASTMTA) is prepared and described by calling ARIDSQL (CPAIPRDE). ARIDSFA does not obtain any I/O area for this calltype, but returns a length to the calling program.
3. CALLTYPE = "FREE" (= 4):
   ARIDSFA calls ARISYSD2 to free the area at the address in CPASFIOA.

ARIDSQLA

ARIDSQLA is the DBS Utility SQL linkage module. It executes SQL commands, interrogates the SQLCA, and requests SQL message generation. The SQL access module associated with the module is SQLDBA.ARIDSQL which is created as a result of the DBS Utility PREP during SQL/DS installation.

ARIDSSB

ARIDSSB builds a SQL SELECT * command with or without an "ORDER BY" clause depending on whether a clustering index exists for the table. The FROM clause uses the creator and table-name passed to it. The ORDER BY clause is built from the COLNUMBERS column of the SYSTEM.SYSINDEXES catalog entry for the table that has a CLUSTER column value = F or H.

ARIDUCP

ARIDUCP is the DBS Utility command parser. ARIDUCP further identifies DBS AUTOCOMMIT and ERRORMODE commands, parses the DBS commands, and updates the appropriate fields in the CPA (Common processing Area) with command data.

ARIDUNL

ARIDUNL unloads the data requested by a UNLOAD TABLE command or by a UNLOAD DBSPACE command. Along with the table data, other control data is unloaded. The control data includes an identification of the maximum input record length, table definitions, and table index definitions.

ARIDUSQ

ARIDUSQ is the DBS Utility user SQL command processor. It processes user entered SQL commands except SQL SELECT commands. ARIDUSQ parses the SQL ROLLBACK and COMMIT commands to determine if the RELEASE option is specified. It also parses the SQL CONNECT 'userid' IDENTIFIED BY 'password' command for the userid and password and initiates the special message file display for CONNECT commands. SQL command execution is initiated by calling the DBS Utility SQL linkage module ARIDSQL.

ARIIBIN

ARIIBIN processes the ISQL INPUT command. It also processes the SAVE, BACKOUT, CANCEL, and END input mode commands.

ARIIBIN allows the user to insert multiple rows into an existing SQL/DS table from data entered through the terminal.

ARIICAN

ARIICAN is the ISQL cancel processor. ARIICAN does the following:

- Cleans up a routine after a CANCEL
- Issues a ROLLBACK
- Deletes Temporary Storage
- Displays messages

ARIICICD - Terminal Control (VSE)

ARIICICD contains some of the CICS commands and system dependent functions used by the terminal control transaction for automatic storage GETMAIN and FREEMAIN. It gets control from CICS when the user enters the ISQL transaction-id, or when the second transaction issues an EXEC CICS START command to start the terminal control transaction.

Entry Point: ARIICIC
  Function: ARIICIC gets control from CICS and does ISQL initialization processing. If it is the beginning of the session, it does the welcome screen processing, starts the second transaction action, and calls ARIITRMD to set up the WAIT/POST linkage between the two transactions. If it is returning after "going away" then it does CANCEL request processing. If it was started from the terminal, and it does initialization and sets up the WAIT/POST linkage if it was started by the second transaction.

Entry Point: ARISYSDG
  Function: ARISYSDG gets storage from the CICS partition.

Entry Point: ARISYSD2
  Function: ARISYSD2 frees up storage from the CICS partition.

Entry Point: ARIIMGA
  Function: ARIIMGA is called by ARIIPFKD when a PF key error message must be displayed in the status area.

ARIICI2D - ISQL System Dependent Module (CICS/VSE)

ARIICI2D contains the CICS commands and system dependent functions used within ISQL in the nonterminal transaction. This routine is divided into several logical sections represented each by an external entry and will run within the CICS partition. Details of each entry point are provided in the prolog for that entry point.

The following CICS/VSE system dependent functions are provided by this routine:

- CICS HANDLE function
- CICS ADDRESS function
- CICS ASSIGN function
- CICS START function
- CICS RETRIEVE function
- CICS WAIT function
- CICS ENQ function
- CICS DEQ function
- CICS GETMAIN function (Macro level)
- CICS FREEMAIN function (Macro level)
- CICS RETURN function
- CICS READ Temp Storage Queue function
- CICS WRITE Temp Storage Queue function
- CICS DELETE Temp Storage Queue function
- CICS ROUTE function
- CICS SEND PAGE function
- CICS SEND TEXT function
- CICS DUMP function

Entry Points: ARIICI2D, ARISYSDG, ARISYSD2, ARIIGM, ARIIFM ARIIWAIT, ARIITS, ARIIENQ, ARIIDEQ, ARIIRTRN, ARIITIO, ARIITDP, ARIIROUT, ARIISTXT, ARIISPGE

Entry Point: ARIICI2D
  Function: ARIICI2D does ISQL initialization processing and control which consists of the following:
    - GETMAIN Automatic Storage
    - GETMAIN and initialize the Global Area
    - Retrieve ITRM Work Area pointer
    - Setup CICS Handle Conditions
    - GETMAIN Global Control Block Extension
    - GETMAIN Command Buffer
    - GETMAIN SQLCA
    - GETMAIN Current SQL Statement Buffer
    - GETMAIN Previous SQL Statement Buffer

- GETMAIN Normal Screen Buffer
- Initialize GCBPTR for ARIITRMD
- Post ITRM Startup ECB when Global Area is initialized
- Move Signon Routine to Command Buffer for ARIIPRF
- Post ARIITRMD to Display Initialization Status
- Call ARIISQL3 to issue a SQL Connect and verify it
- Call Profile Initialization Routine (ARIIPRF)
- Call ARIIDBS for Mainline Loop Processing
- Return to CICS when Exit is entered

Entry Point: ARISYSDG
Function: ARISYSDG does CICS GETMAIN(MACRO LEVEL) for automatic storage.

Entry Point: ARISYSD2
Function: ARISYSD2 does CICS FREEMAIN(MACRO LEVEL) for automatic storage.

Entry Point: ARIIGM
Function: ARIIGM does CICS GETMAIN(MACRO LEVEL).

Entry Point: ARIIFM
Function: ARIIFM does CICS FREEMAIN(MACRO LEVEL).

Entry Point: ARIIWAIT
Function: ARIIWAIT issues CICS WAIT to wait on a single event (ECB).

Entry Point: ARIITS
Function: ARIITS performs a WRITE to a CICS Temp Storage Queue.

Entry Point: ARIIENQ
Function: ARIIENQ does CICS ENQUEUE/ACCESS resource.

Entry Point: ARIIDEQ
Function: ARIIDEQ does CICS DEQUEUE/RELEASE resource.

Entry Point: ARIIRTRN
Function: ARIIRTRN is called to do cleanup and exit if ISQL encounters an unrecoverable error. It deletes the temporary storage queue, frees the global control block storage, and clears the extract XECBs.

Entry Point: ARIITIO
Function: ARIITIO starts the ISQL transaction if it is not already started. It POSTs the ISQL transaction to do some terminal I/O, and WAITs for the ISQL transaction to complete. ARIITIO then returns to ARIIRWI.

Entry Point: ARIITDP
Function: ARIITDP dumps the ISQL trace table and global control block for the ISQLTRACE DUMP command.

Entry Point: ARIIROUT
Function: ARIIROUT issues an EXEC CICS ROUTE command for terminal printers.

Entry Point: ARIISTXT
Function: ARIISTXT issues an EXEC CICS SEND TEXT command for terminal printers.

Entry Point: ARIISPGE
ARIISPGE issues an EXEC CICS SEND PAGE command for terminal printers.

ARIICMD

ARIICMD calls the command verification routine to determine if the command entered is a valid ISQL command. It also processes the EXIT, SHOW, and COUNTER commands or calls other modules to process the remaining ISQL commands.

ARIICNV

ARIICNV converts a halfword, fullword, or packed decimal number to an edited number with decimal places. Optionally leading zeros and punctuation insertion are performed. Punctuation may be American, European or French notation.

ARIIDBS

ARIIDBS is the mainline controller for ISQL. ARIIDBS reads user input by calling ARIIRWI. Then ARIIDBS calls ARIICMD to process ISQL commands, or calls ARIIPQL to process SQL statements, or calls ARIIHLD to process the ISQL HOLD command.

ARIIDQY

ARIIDQY is the query display module. It handles the forward/backward/column/tab/left/right commands. ARIIDQY also maintains the display buffer.

ARIIERS

ARIIERS erases stored SQL statements from the Stored Query Table (SQLDBA).

ARIIEXT

ARIIEXT receives control at AXTCMNDS or AXTCEXTS. It issues a message saying the extract facility is not installed, and returns to ISQL.

ARIIFCC

ARIIFCC is the FORMAT COLUMN command processor.

ARIIFCI

ARIIFCI processes the FORMAT EXCLUDE and INCLUDE commands.

Entry Point: ARIIFCI1
   Function: ARIIFCI1 is the FORMAT EXCLUDE command processor.

Entry Point: ARIIFCI2
   Function: ARIIFCI2 is the FORMAT INCLUDE command processor.

ARIIFCS

ARIIFCS processes the FORMAT GROUP, SUBTOTAL, and TOTAL commands.

Entry Point: ARIIFCS1
   Function: ARIIFCS1 is the FORMAT GROUP command processor.

Entry Point: ARIIFCS2
   Function: ARIIFCS2 is the FORMAT SUBTOTAL command processor.

Entry Point: ARIIFCS3
   Function: ARIIFCS3 is the FORMAT TOTAL command processor.

ARIIFET

ARIIFET is the ISQL fetch module. It fetches a row from the data base. ARIIFET also calls the report writer if totaling or subtotaling occurs.

ARIIFMC

ARIIFMC is the FORMAT command processor. It formats the display of a SELECT statement. ARIIFMC also checks the FORMAT command and stores the information in the Format Control Block.

ARIIFMT

ARIIFMT takes the row retrieved from SQL/DS in response to a query and formats the row so it can be displayed or printed.

ARIIFOL

ARIIFOL is the ISQL folding routine. It determines if input is a valid SQL statement and sets the IGCSQL bit. ARIIFOL folds all input from the terminal to upper case.

Entry Point: ARIIFOL
   Function: ARIIFOL folds all input to upper case and finds valid SQL statements on input.

ARIIFOR

ARIIFOR is the ISQL message formatter routine (twin module of ARIMFMT). The message formatter retrieves a message (or line of a multi-line message) and substitutes any caller-provided message variables into the message (or line). See ARIMFMT for more functional detail.

ARIIFULD - Full Screen Condition Handler

ARIIFULD displays the message prompting the user to PRESS THE CLEAR KEY OR ENTER CANCEL or displays the message prompting the user to PRESS THE CLEAR KEY. It sets the IGCINDEX and IGCERASE fields in the global control block so that the next display to the terminal will begin at the top of a cleared screen.

Entry Point: ARIIFUL
   Function: Checks the bit parameter passed to it. If it is on, display the message prompting for the CLEAR KEY or CANCEL. If it is off, displays the message prompting for the CLEAR KEY only.

ARIIHDR

ARIIHDR produces column headers to be displayed or printed.

ARIIHLD

ARIIHLD saves a SQL command in the current buffer without the statement being executed.

ARIIHLP

ARIIHLP processes the HELP command. ARIIHLP also:

- Verifies the topic entered on the HELP command exists
- Builds a SELECT statement to obtain the HELP documentation
- Calls ARIIPSQ to process the SELECT statement that was built.

ARIIIGN

ARIIIGN generates a message stating there is nothing to ignore.

ARIILST

ARIILST processes the LIST SQL command or calls ARIIQRY to process the LIST SET command.

ARIIMAP

ARIIMAP displays the address of the ISQL modules on the terminal (ISQLMAP command).

ARIIMSG

ARIIMSG is the ISQL message service routine.

Entry Point: ARIYM04
Function: ARIYM04 is invoked to write an ISQL message to the user terminal and optionally read a reply.

ARIINITC

ARIINITC is the ISQL initialization routine for CP/CMS.
ARIINITC performs the following:

- Saves pointer to sign-on routine
- Frees bootstrap storage
- Obtains and initializes the following work areas:
  - Global Control Block
  - SQLCA
  - Resource Manager area
  - Small command buffer
  - Large command buffer
  - Current SQL buffer
  - Previous SQL buffer
- Validates terminal size and type
- Calls ARIISYSC to obtain ARIISYSC's automatic storage
- Calls ARIISTK to obtain stack storage
- Obtains storage for trace table
- Establishes an ABEND exit
- Gets VM user-id
- Calls ARIISQL0 to establish CANCEL as an immediate command
- Initializes map table for ISQLMAP
- Obtains normal screen buffer
- Moves sign-on routine to command buffer for ARIIPRF
- Calls ARIIPRF to do profile processing
- Calls ARIIDBS for mainline loop processing
- Cleans up when exit is entered:
  - Stack NUCEXT drop
  - Remove immediate command extension for CANCEL
  - Reset ABEND exit
  - Free storage obtained for stack manager
  - Free storage for trace table
  - Free normal screen buffer
  - Free SQL DSECT storage that was obtained by ARIISQL
  - Free automatic storage obtained by ARIISYSC
  - Free work areas
    - Global Control Block
    - SQLCA
    - Resource Manager area
    - Small command buffer
    - Large command buffer
    - Current SQL buffer
    - Previous SQL buffer
- Returns to CMS, bypassing bootstrap.

ARIINME

ARIINME finds TABLE.COLUMN names. ARIINME returns the length of the name, pointer to the name, and an index into the name for the '.'.

ARIINSQ

ARIINSQ generates a SQL command not allowed at this time message.

ARIIOCI

ARIIOCI is the operator command linkage. It is the linkage between the ISQL mainline module and the operator command processors in the SQL/DS partition (VSE) or SQL/DS machine (VM).

ARIIOVD

ARIIOVD is the operand validation routine. It checks the operands of SET, QUERY, FORMAT or PRINT command and determines if they are valid. If they are valid, the address of the routine to handle the operand is returned to the calling module. If they are not valid, an address of 0 is returned.

ARIIPAT

ARIIPAT provides a common patch area for modules on VM and modules in the command processor transaction (CISQ).

ARIIPATD

ARIIPATD provides a common patch area for modules residing in the terminal transaction (ISQL) in VSE.

ARIIPFKD

ARIIPFKD is the VSE PF key processor. It is called to process all PF keys except the RETRIEVE key (PF12, PF24).

ARIIPQL

ARIIPQL is the buffer manager for the two SQL buffers. It copies the SQL statement from the current buffer to the previous buffer. ARIIPQL also copies SQL statement from the input buffer to the current buffer. ARIIPQL then calls ARIIPSQ to process SQL commands.

ARIIPQY

ARIIPQY is the SELECT SQL statement processor. It initializes the default formatting information and allocates the display buffers. ARIIPQY calls ARIIRWI to display the query results.

Entry Point: ARIIPQY1
Function: ARIIPQY1 processes the END command.

Entry Point: ARIIPQY2
Function: ARIIPQY2 processes the BACKWARD command.

Entry Point: ARIIPQY3
Function: ARIIPQY3 processes the navigation commands.

Entry Point: ARIIPQY4
Function: ARIIPQY4 processes the DISPLAY command.

Entry Point: ARIIPQY5
Function: ARIIPQY5 processes the FORMAT command.

Entry Point: ARIIPQY6
Function: ARIIPQY6 processes the PRINT command.

ARIIPRF

ARIIPRF sets up ISQL defaults and then issues an internal RUN command for each of the following (if they exist):

1. SQLDBA.PROFILE
2. userid.PROFILE
3. user-supplied signon routine.

ARIIPRTC

ARIIPRTC is the VM/CMS PRINT command routine (ISQL hardcopy routine). ARIIPRTC:

- Prints tables from display mode
- Centers top and bottom title
- Verifies the validity and range of any operands
- Provides date and page numbering

ARIIPRTD

ARIIPRTD is the VSE PRINT command routine (ISQL hardcopy routine). ARIIPRTD:

- Prints tables from display mode
- Centers top and bottom title
- Verifies the validity and range of any operands
- Provides date and page numbering

ARIIPSQ

ARIIPSQ handles all SQL commands. It checks whether the SQL statement is valid from ISQL. Then if statement is a SELECT statement, ARIIPSQ calls ARIIPQY. If statement is a 'normal' SQL statement, ARIIPSQ calls ARIISQL. ARIIPSQ also does the autocommit processing if it is necessary.

ARIIQRY

ARIIQRY processes the LIST SET command.

ARIIREAC

ARIIREAC reads input for VM ISQL. If the IGCTERM switch is on indicating that messages are forced to the terminal and input is read from the terminal, the RDTERM direct macro is used to bypass reading from the program and console stacks. If the IGCTERM switch is not on, a RDTERM macro is used without the DIRECT parameter. This first tries to read from the program stack. If the program stack is empty, it tries to read from the console stack. If both the program stack and console stack are empty, it reads from the terminal.

ARIIREAC also issues an SVC 202 to invoke the CMS SUBSET if the user enters CMS. ARIIREAC also calls ARIIVFYC if the user enters CANCEL.

ARIIREC

ARIIREC is the RECALL command processor. ARIIREC recalls a previously stored SQL statement and all related formatting information from the Stored Query Table (SQLDBA). ARIIREC also recalls the current or previous SQL statement.

ARIIRETD

ARIIRET is the VSE RETRIEVE processor. It is called to add new user input lines to the RETRIEVE buffer, and to retrieve user input lines from the RETRIEVE buffer.

ARIIRNM

ARIIRNM is the RENAME command processor. It renames a previously stored SQL statement in the Stored Query Table (SQLDBA).

ARIIRPT

ARIIRPT creates a total or subtotal row at the proper time.

Entry Point: ARIIRPT1
Function: ARIIRPT1 initializes the accumulators to subtotal and total the specified fields.

Entry Point: ARIIRPT2
Function: ARIIRPT2 inserts a subtotal pseudo-row into the row buffer, if it is required. ARIIRPT2 inserts the row that caused the control break after all the pseudo-rows are inserted.

Entry Point: ARIIRPT3
Function: ARIIRPT3 accumulates values according to the Format Control Block and sets a flag if at least one control break is encountered.

Entry Point: ARIIRPT4
Function: ARIIRPT4 frees storage gotten by the previous calls to ARIIRPT entry points.

ARIIRUN

ARIIRUN is the RUN command processor. It reads the routine into temporary storage. ARIIRUN also performs any RUN parameter substitution. Then ARIIRWI will execute the routine. The issuer of the RUN command can use a procedure of another user by using the other userid in the RUN command.

ARIIRWI

ARIIRWI is the SQL/DS ISQL Read/Write Linkage routine. It is the linkage between ARIITIO and modules requesting terminal I/O. ARIIRWI is also the linkage between ARIITS and modules requesting a read from a routine. It is called by any module that wants to write to the terminal and/or read from the terminal or a routine.

ARIISCC

ARIISCC processes the START, and CHANGE commands.

ARIISCMD - CICS COMMAND MODULE

ARIISCMD contains CICS command functions (SEND, RECEIVE, or WAIT) which are called by other modules in the terminal control transaction.

Entry Point: ARIISCM
Function: Checks the REQTYPE parameter passed to it, and performs the appropriate processing to handle the request indicated in the REQTYPE parameter.

ARIISET

ARIISET processes the SET command. It decides which operand to process. Then the correct routine is called to process the operand.

ARIISMG

ARIISMG is the ISQL/EXTRACT routine for writing SQL messages to the terminal. ARIISMG obtains the appropriate SQL message or condition code text. Then ARIISMG writes the message to the terminal.

ARIISQL

ARIISQL contains all ISQL SQL statements, except for specific EXTRACT commands. It is the only routine processed by the preprocessor.

Entry Point: ARIISQL
Function: Executes general purpose SQL statements.

Entry Point: ARIISQL1
Function: Manipulate stored SQL statements.

Entry Point: ARIISQL2
Function: Retrieves all stored SQL commands using a cursor.

Entry Point: ARIISQL3
Function: Performs CONNECT when user signs on using a userid and password.

Entry Point: ARIISQL4
Function: Insures the Help text topic is in table SQLDBA.SYSTEXT1.

Entry Point: ARIISQL5
Function: Tests for the existence of a table for a certain user.

Entry Point: ARIISQL6
Function: Insures the Help text topic is in table SQLDBA.SYSTEXT2.

Entry Point: ARIISQL7
Function: Obtains the userid currently connected to SQL/DS.

Entry Point: ARIISQL8
Function: Establishes the VM/SP cancel exit and makes CANCEL a valid name for the canceling process. The ARIRCAN macro is used for this.

Entry Point: ARIISQL9
Function: Insures that the Help text table SQLDBA.SYSTEXT1 is not empty.

ARIISTK

ARIISTK gets the storage for the initial stack for module automatic storage areas and initializes the stack header.

ARIISTR

ARIISTR is the STORE command processor. ARIISTR saves a SQL statement and all formatting information in the Stored Query Table (SQLDBA).

ARIISTX

ARIISTX is a message module for ISQL messages, and status messages that cannot be processed as normal messages by ARIYM04. There is no executable code in ARIISTX.

ARIIST2

ARIIST2 processes the AUTOCOMMIT, CASE, CONTINUE and DECIMAL operands of the SET command.

ARIIST3

ARIIST3 processes the NULL, PAGESIZE, SEPARATOR, and VARCHAR operands of the SET command.

ARIIST4D

ARIIST4D is a message module for ISQL terminal control transaction messages, and status messages that cannot be processed as normal messages by ARIYM04. There is no executable code in ARIIST4D.

ARIISUB

ARIISUB takes a string of tokens and substitutes them into the &1,&2,&3,...10 variables in the command buffer.

ARIISYSC - ISQL System Dependent Module (VM/SP)

Module ARIISYSC contains system-dependent subroutines for CMS (this is the VM/SP version of ARIICI2D). ARIISYSC contains the VM/SP command and system-dependent functions used within ISQL. This routine is divided into several logical sections, each represented by an external entry, and

will run in a user machine. Details of each entry point are provided in the prolog for that entry point.

The following VM/SP system-dependent functions are provided by this routine:

- DMSFREE to obtain storage
- DMSFRET to free storage
- LINEDIT to write out abend message
- MAKEBUF to create a program stack
- ATTN to write to the program
- CLOSE PRINT to close the stack virtual printer
- DUMP to dump the Global Control Block and trace table to the virtual printer
- SENTRIES to determine how many commands are in the program stack

Entry Points: ARIISYS, ARISYSDG, ARISYSD2, ARIIGM, ARIIFM, ARIITS, ARIITDP, ARIIABN, ARIISTC

Entry Point: ARIISYS
Function: Do a CMS DMSFREE to obtain the automatic storage for this module and save its address in IGCCI2D and its length in IGCSYSCL.

Entry Point: ARISYSDG
Function: Do a CMS DMSFREE to obtain automatic storage. An abend will occur if the storage is not available.

Entry Point: ARISYSD2
Function: Do a CMS DMSFRET to free automatic storage.

Entry Point: ARIIGM
Function: Do CMS DMSFREE to get buffer storage. Control returns to the calling module if the storage request is not available.

Entry Point: ARIIFM
Function: Do CMS DMSFRET to free buffer storage.

Entry Point: ARIITS
Function: Perform a routine I/O.

Entry Point: ARIITDP
Function: Dump the Global Control Block and trace table to the virtual printer.

Entry Point: ARIIABN
Function: Write out a user-friendly abend message when ISQL abends.

Entry Point: ARIISTC
Function: Determine how many commands are in the program stack.

ARIITIOC

ARIITIOC is called when VM/CMS terminal I/O is required. Input operations are performed by calling ARIIREAC. Line output operations are done using the WRTERM macro. Query result displays are done using DIAG X'58' CCW X'19'.

ARIITKN

ARIITKN gets the token after the start position for the scan, 'IPARM'. ARIITKN then sets up a pointer and length to the next token in the buffer passed to ARIITKN.

ARIITRC

ARIITRC is the ISQLTRACE command processor.

ARIITRMD

ARIITRMD is the VSE terminal control module. It is the linkage between the user terminal and the CISQ transaction. ARIITRMD waits on the CISQ transaction to post IGCTECB when a CICS service is required. It checks for the type of CICS service required. Then ARIITRMD either calls the appropriate service routine in ARIICICD, or ARIITRMD returns to ARIICICD to perform the service.

Linkage:

- Called by ARIICICD after initialization processing is complete.
- Posted by ARIIDBS (Gets control at next sequential instruction after WAIT.)
- Posted by ARIITIO (entry point in ARIICI2D). (Gets control at next sequential instruction after WAIT.)
- Posted by ARIIXIT (Gets control at next sequential instruction after WAIT.)
- Posted by EXTRACT (Gets control at next sequential instruction after WAIT.)

ARIIVFYC

ARIIVFYC is the cancel verification module for VM. If the autocommit bit is on, it sets IGCANCEL on; otherwise, it prompts the user to verify the cancel request. If the user verifies the request, it sets IGCANCEL on. If the user does not verify the request, it leaves IGCANCEL alone.

ARIIVFYD

ARIIVFYD is the cancel verification module for VSE. It checks the autocommit bit to see if CANCEL should be done. If the bit is off, it prompts the user to verify that the CANCEL should be done. If the users enters YES, or if the bit was on, the CANCEL request is done (IGCANCEL is turned on).

Entry Point: ARIIVFY
Function: Checks the contents of the buffer whose length and address are passed as parameters. If it is not CANCEL, return to the caller. If it is cancel, check the autocommit bit. If the bit is on, turn IGCANCEL on and return to the caller. If the bit is off, prompt the user to verify that the request should be done. If the user enters YES, turn IGCANCEL on. Otherwise, leave IGCANCEL off and return to the caller.

ARIIVLD

ARIIVLD is the ISQL command validation routine. It contains a table of all ISQL commands and extract commands and the addresses of the routines to process them. If the ISQL command is valid, the address of the routine is returned. A bit is also set, if the command is allowed, in the current environment (that is, during query display processing). If the ISQL command is not valid, 0 is returned. This routine is called by ARIICMD (Main Command Module), the bulk input module (ARIIBIN), and the query result manager (ARIIPQY).

ARIIWATD

Before VSE ISQL frees the terminal, ARIIWATD delays ISQL for 15 seconds. ARIIWATD delays ISQL to see if another I/O request will be issued.

ARIIWRTD - Line, Screen, and Status Write Module (VSE)

ARIIWRTD determines what type of the display is being requested, and then calls the corresponding internal entry point to process it.

Entry Point: ARIIWRT
Function: Checks the contents of the IGCWRIT field to determine which type of write operation (line, screen, or status) is to be done. For a line write, it calls ARIIDM. For a screen write, it calls ARIIDGS. For a status write, it calls ARIIDST.

Entry Point: ARIIDM
Function: Formats the 3270 data stream for a line write. Checks the IGCREAD bit in the global control block. If it is on, it sets up parameters to call ARIISCMD to do a SEND and a RECEIVE request. It calls ARIIPFKD to do PF key processing, if necessary. It calls ARIIVFYD to do CANCEL verification on the user input. If IGCREAD is off, it sets up parameters to call ARIISCMD to do a SEND request.

Entry Point: ARIIDGS
Function: Formats the 3270 data stream for a screen write. Checks the IGCREAD bit in the global control block. If it is on, it sets up parameters to call ARIISCMD to do a SEND and a RECEIVE request. It calls ARIIPFKD to do PF key processing, if necessary. It calls ARIIVFYD to do CANCEL verification on the user input. If IGCREAD is off, it sets up parameters to call ARIISCMD to do a SEND request.

Entry Point: ARIIDST
Function: Formats the 3270 data stream for a status write. Checks the IGCREAD bit in the global control block. If it is on, it sets up parameters to call ARIISCMD to do a SEND and a RECEIVE request. It calls ARIIPFKD to do PF key processing, if necessary. It calls ARIIVFYD to do CANCEL verification on the user input. If IGCREAD is off, it sets up parameters to call ARIISCMD to do a SEND request.

ARIIXITD - (VSE)

SQIP calls ARIIXITD to support a cancel in SQL/DS. It waits for the CANCEL ECB to be posted or the SQL/DS ECB to be posted.

ARISISBT - (VM)

See ARISISK.

ARISISK - (VM)

ARISISK is invoked in the VM environment by the SQLGENLD EXEC. It is a bootstrap module which is GENMODed under the name ARISISBT, and later invoked by the SQLISTRT EXEC. ARISISK loads the ISQL code and then branches to the ISQL initialization module.

COMMON SQL MESSAGE MODULES AND OP TREE AND SPACE BLOCK TRACE FORMATTER MODULES

(Modules ARIMPRT through ARIMPT5, ARIMSCF, and ARIMSCH belong to the Op Tree and Space Block Formatter.)

ARIMBIN

ARIMBIN converts a fixed(31) signed binary integer to an unpacked zoned decimal string. The first byte is blank if the number is positive and contains a minus sign if the number is negative. Leading zeros are not generated. The length of the string (including the sign byte) is returned.

ARIMCID

ARIMCID checks identifiers for valid contents. Length checking is up to the caller. The character string (XC) is checked left to right for the length indicated by the length field (XL). If unalloued characters are found embedded in the string, the module returns a return code of 12. If blanks are found and they continue to the end of the string according to the length specification (trailing blanks), then a return code of 4 is set. Embedded blanks are erroneous and cause a return code of 12. Valid identifiers will result in a return code of 0.

ARIMDEC

ARIMDEC converts an unsigned, unpacked decimal character string to a fullword binary integer. The decimal string can be zero to eight characters long. If the string is less than eight characters, it must be terminated by a blank character.

ARIMDFP

ARIMDFP is the SQL/DS floating point conversion routine. ARIMDFP is the twin module of the routine ARIMFLP. ARIMDFP is for non-RDS users and is currently used by the DBS Utility and ISQL. ARIMFLP is for RDS users. It is called by the optimizer routines: ARIXODF and ARIXOTF.

ARIMDFP uses the macros ARISGTM and ARISFM to perform its prolog get storage (by calling ARISYSD1) and epilog free storage (by calling ARISYSD2).

ARIMFLP uses the macros ARICRDG and ARICRDF to perform its prolog get storage (by using stack storage, the pointer is in RDAREA) and epilog free storage. ARIMFLP also has Register 10 restricted to contain the address of RDAREA (RDS convention).

Entry Point: ARIMDFP1
Function: ARIMDFP1 performs the conversion of an 8 byte floating point point number in IBM/370 internal representation to a character string that contains the external EBCDIC representation of the input floating point number.

FLTNGPT is the input parameter for this entry point. It contains a floating point number in IBM/370 internal representation. CHARACTR and LEN are output parameters for this entry point. CHARACTR is a character string containing the EBCDIC representation of the input floating point number, left justified. LEN is the length of the returned string. The maximum length is 20.

Exit Normal: Character representation in CHARACTR and its length returned in LEN.

Exit Error: None, no errors detected.

Entry Point: ARIMDFP2
Function: ARIMDFP2 performs the conversion of a literal string (EBCDIC characters) that represents a floating point number into an 8 byte floating point number in IBM/370 internal representation.

PARMSTR and ARGLEN are input parameters for this entry point. PARMSTR is a string containing the EBCDIC representation of a floating point number. ARGLEN is the length of the input character string PARMSTR. FLTNG is the output parameter for this entry point. It is a double-precision floating point number in the internal system 370 format.

Exit Normal: If Register 15 contains:
- Return code 0: Processing okay.
- Return code 4: Significant digits may be lost.

Exit Error: If Register 15 contains:
- Return code 8: Underflow - number is too small.
- Return code 12: Overflow - number is too large.
- Return code 16: Too many digits, (>2), in the exponent field.
- Return code 20: Length > 30.

- Return code 24: Syntax error.

ARIMFLP

ARIMFLP is the SQL/DS floating point conversion routine. ARIMFLP is the twin module of the routine ARIMDFP. ARIMDFP is for non-RDS users and is currently used by the DBS Utility and ISQL. ARIMFLP is for RDS users. It is called by the optimizer routines: ARIXODF and ARIXOTF.

ARIMDFP uses the macros ARISGTM and ARISFM to perform its prolog get storage (by calling ARISYSD1) and epilog free storage (by calling ARISYSD2).

ARIMFLP uses the macros ARICRDG and ARICRDF to perform its prolog get storage (by using stack storage, the pointer is in RDAREA) and epilog free storage. ARIMFLP also has Register 10 restricted to contain the address of RDAREA (RDS convention).

Entry Point: ARIMFLP1
Function: ARIMFLP1 performs the conversion of an 8 byte floating point point number in IBM/370 internal representation to a character string that contains the external EBCDIC representation of the input floating point number.

FLTNGPT is the input parameter for this entry point. It contains a floating point number in IBM/370 internal representation. CHARACTR and LEN are output parameters for this entry point. CHARACTR is a character string containing the EBCDIC representation of the input floating point number, left justified. LEN is the length of the returned string. The maximum length is 20.

Exit Normal: Character representation in CHARACTR and its length returned in LEN.

Exit Error: None, no errors detected.

Entry Point: ARIMFLP2
Function: ARIMFLP2 performs the conversion of a literal string (EBCDIC characters) that represents a floating point number into an 8 byte floating point number in IBM/370 internal representation.

PARMSTR and ARGLEN are input parameters for this entry point. PARMSTR is a string containing the EBCDIC representation of a floating point number. ARGLEN is the length of the input character string PARMSTR. FLTNG is the output parameter for this entry point. It is a double-precision floating point number in the internal system 370 format.

Exit Normal: If Register 15 contains:
- Return code 0: Processing okay.
- Return code 4: Significant digits may be lost.

Exit Error: If Register 15 contains:
- Return code 8: Underflow - number is too small.
- Return code 12: Overflow - number is too large.
- Return code 16: Too many digits, (>2), in the exponent field.
- Return code 20: Length > 30.
- Return code 24: Syntax error.

ARIMFMT

ARIMFMT is the message formatter routine. The message formatter retrieves a message (or line of a multi-line message) and substitutes any caller-provided message variables into the message (or line). It is the twin module of ARIYM05 and ARIIFOR and is used in load modules which do not have stack storage. Using the caller-provided message number or SQL code, the message index or SQL index table is searched to locate the message module containing the message.

MSGID, BUFFER, and VARLPT are input parameters for ARIMFMT. MSGID contains the requested message/SQL code number, a flag indicating the SQL code and for multi-line messages, and the line sequence number (line sequence number can be either 0 or 1 for first line of message; 0 is forced to 1). BUFFER contains the structure length, the unused flag fields, a field to return message line length, and a field to contain the returned message line. VARLPT is a pointer to the VARLIST structure (or null if no VARLIST). VARLIST tells how many variables are passed and contains the length and address of each variable. A special NOVAR flag requests *zero length* substitution.

The output for ARIMFMT is:
- Requested message or message line and length in buffer structure (return codes 0 - 12)
- Updated line sequence number in MSGID field SEQNO (return codes 0 and 8)
- Message ARI058E in buffer structure (return code 16)
- NONE (return code 20) (For ARIMSMF, also return code 24)
- Return code in Register 15 (see "Exit Normal" and "Exit Error")

Exit Normal:

- Return code 0: Message line in buffer, no errors, more lines to message
- Return code 8: Message line in buffer, no errors, no more lines to message

Exit Error:

- Return code 4: Message line in buffer, more lines to message, and message line has truncation error (too long for buffer) and/or substitution failure (&nnn. still appears in the line)
- Return code 12: Message line in buffer, no more lines to message, and message line has truncation and/or substitution error
- Return code 16: Error message in buffer that gives message/SQL code number and reason code identifying why message could not be retrieved. (ARIMFMT/ARIIFOR error) or error message in buffer gives a reason code identifying which MSGID fields contain invalid values (ARIMSMF detected error - results in 511 message).
- Return code 20: No message in buffer due to invalid caller parameter (null address or buffer less than minimum required size).

ARIMHEX

ARIMHEX converts a fixed(31) signed binary integer to a printable hexadecimal string. Leading zeros are not generated (thus a binary one produces a one byte string with the character one). If the binary input is zero, the output is a one byte string with the character zero.

ARIMPRT

ARIMPRT is the SQL/DS trace formatter print routine. It outputs a print line to SYSLST (VSE) or SYSPRINT (VM). ARIMPRT also updates the line counter in the ARIMFCT (formatter control table).

ARIMPTT

ARIMPTT is a trace formatter intermediate module in the module chain that outputs a formatted Op Tree block, its relocation directory, and subsequent space blocks(if present) as part of the off-line trace function of the SQL/DS system. ARIMPTT takes the trace blocks containing the OP Tree, relocation directories, and space blocks. Then it reconstructs them into the chained format (same format used by the RDS component) as follows:

Op tree block:
Relocation Directory - Pointer to this block's relocation directory.
Next Space Block - Pointer to the next space block in the chain.
Space Block:
Has the same type of pointers as the Op Tree block.

ARIMPTT receives a pointer to a block of data on each entry. Storage is obtained for this data and the block is moved into the storage. The storage address and length is saved in a table.

The blocks are trace output from the parser and optimization subcomponents during the PREP stage of a SQL statement in RDS. Trace output from the parser and post-view-composition (during optimization) consists of only one block and the Op Tree (or Block 0). After optimization, output consists of an even number of blocks. A block always has a relocation directory paired with it. The first block is always the Op Tree.

ARIMPT1

ARIMPT1 is the trace formatter entry module to the modules which will actually do the formatting of the Op Tree and the Space Block. It is the control formatter module calling the other formatter modules as required to perform specific formatting functions.

The specific format functions of this module are to format the headers and the ranges of the Op Tree and Space Blocks, to output the unformatted hex contents of the Op Tree along with the contents of the global structure, PARMS, used by the other formatter modules and passed as an argument to them, to initialize the tables which describes the Op Tree nodes and Op Tree descriptors, and to acquire and free the storage used for the vertex table.

The data to be formatted is organized as follows:

Op Tree Block:
Relocation Directory - Pointer to this block's relocation directory.
Next Space Block - Pointer to the next space block in the chain.
Space Block:
Has same type of pointers as the Op Tree block.

ARIMPT2

ARIMPT2 is the trace formatter main routine for walking the Op Tree and placing the result in VERNUMTB. ARIMPT2 is a recursive module because it calls itself in order to walk

the nodes of the Op Tree. ARIMPT2 walks the Op Tree top-down, left to right. For each vertex, a node number j is assigned. Also, the j-th entry of NODUMTB is filled with the block, index, class, and type (for nodes).

ARIMPT3

ARIMPT3 is the trace formatter Op Tree space block formatter module responsible for the formatting of:

- Relocation directory blocks of the Op Tree and space block
- The control block chain in the space block
- The SQLDA control block in the space block
- The cursor list in the Op Tree block
- The input variable list in the Op Tree block
- The output variable list in the Op Tree block

ARIMPT4

ARIMPT4 is a trace formatter module which formats the node descriptors found in the Op Tree block. The table DESCSTR contains the name and size of all the known node descriptors. When ARIMPT1 finds a descriptor in the vertex table VERNUMTB, it calls ARIMPT4 to format and output the contents of the descriptor.

ARIMPT5

ARIMPT5 is the trace formatter module of a DBSS scan control block and its associated structures. The formatting is kicked off by formatting the control block for a particular DBSS scan. As the formatting proceeds through the control block, other structures linked or chained off the control block are formatted. Internal procedures are used to format base structures, domains, key domains, and search arguments nested within a base structure.

ARIMSCF

ARIMSCF converts a fullword binary value to its character (EBCDIC) representation. The resultant character string is 16 bytes long and padded on the left with character zeros. If the value is negative, a minus sign is placed in the first byte of the string.

ARIMSCH

ARIMSCH converts a halfword binary value to its character (EBCDIC) representation. The resultant character string is eight characters in length, padded on the left by character zeros. If the input value was negative, the first character is a minus sign.

ARIMSMF

ARIMSMF is the SQL/DS SQL message formatter routine. It is the linkage module for the SQL user (PREP, ISQL, DBSU) and the SQL/DS common message formatter routine (ARIMFMT for PREP and DBS utility, ARIIFOR for ISQL. ARIIFOR and ARIMFMT use the entry point ARIMFMT). ARIMSMF formats the SQL messages for display by the user. System SQL messages are composed of system SQL message lines (contained in message text modules) and SQLCODE descriptive text (contained in SQLCODE text modules).

The ARIMSMF routine performs two major functions:

1. The execution of a SQL statement often results in more than one written message. ARIMSMF determines the correct messages, message call sequence, and message arguments. It then returns the caller a return code stating there are more or no more lines to be returned in the buffer structure.
2. Any SQLCA dependencies (as far as message processing is concerned) are isolated to the ARIMSMF routine. A SQLCA change does not affect the SQL user's call format.

The first time ARIMSMF is called, it examines the range of the SQLCODE (<, =, or > 0) and the SQLWARN flags in the SQLCA (the MSGID structure should be set to zero to determine which message to obtain). From this information, ARIMSMF fills in the VARLIST structure. Then it calls ARIMFMT (PREP, DBSS, DSC, and DBS Utility common system message formatter routine) or ARIIFOR (ISQL common message formatter routine, entry point is ARIMFMT). The return code is examined.

If there are more lines to the current message, ARIMSMF propagates the message number id on the front of the message text in the buffer. Then ARIMSMF returns to the caller with the return code = 0 or 4 (more lines).

If there are no more lines to the current message, ARIMSMF examines the last message number called for and the value of SQLCA and MSGID fields. Then ARIMSMF sets the return code and MSGID for either a subsequent call or for the end of message processing.

ARIMSMF formats the SQL message numbers 500 (SQL processing successful), 501 (SQL warning), 502 (SQL warning flags), 503 (SQL error), 504 (SQL supplemental error information), and 511 (ARIMSMF internal error) for display.

MSGID and BUFFER are input parameters for ARIMSMF. MSGID contains the requested message/SQL code number, a flag indicating the SQL code and for multi-line messages, and the line sequence number (line sequence number can be either 0 or 1 for first line of message; 0 is forced to 1). BUFFER contains the structure length, the unused flag fields, a field to return message line length, and a field to contain the returned message line.

The output for ARIMSMF is:

- Requested message or message line and length in buffer structure (return codes 0 - 12)
- Updated line sequence number in MSGID field SEQNO (return codes 0 and 4)
- Message ARI058E in buffer structure (return code 16)
- NONE (return code 20) (For ARIMSMF, also return code 24)
- Return code in Register 15 (see "Exit Normal" and "Exit Error")

Exit Normal:

- Return code 0: Message line in buffer, no errors, more lines to message
- Return code 8: Message line in buffer, no errors, no more lines to message

Exit Error:

- Return code 4: Message line in buffer, more lines to message, and message line has truncation error (too long for buffer) and/or substitution failure (&nnn. still appears in the line)
- Return code 12: Message line in buffer, no more lines to message, and message line has truncation and/or substitution error
- Return code 16: Error message in buffer that gives message/SQL code number and reason code identifying why message could not be retrieved. (ARIMFMT/ARIIFOR error) or error message in buffer gives a reason code identifying which MSGID fields contain invalid values (ARIMSMF detected error - results in 511 message).
- Return code 20: No message in buffer due to invalid caller parameter (null address or buffer less than minimum required size).
- Return code 24: Unrecognized return code from ARIMFMT.

ARIMTRA

ARIMTRA is the top module of the trace formatter ARIMTRA performs the following functions:

- Formats the SQL/DS generated trace tape (for example, decoding, data conversions, etc.).
- Allows wide range of output selectivity.
- Collects the data and passes it to other private routines (for example, RDS trace formatter).

ARIM000

ARIM000 is the message module for the SQL/DS control (DSC) component. It contains all messages issued by modules of the DSC as well as certain common messages issued by modules in or serving more than one SQL/DS component. All messages in this module have message numbers in the range 000 through 099.

ARIM200

ARIM200 is the message module for the DBSS component. It contains all messages issued by modules of the DBSS component. All messages in this module have message numbers in the range 200 through 299.

ARIM500

ARIM500 is the SQL message module for users of SQL. It contains all SQL messages (Note that this module does not contain SQLCODE descriptive text, just SQL messages. SQLCODE descriptive text is contained in the message modules ARIQM01 through ARIQM09). The message numbers reserved for use in this module lie in the range of 500 to 519.

ARIM520

ARIM520 is the message module for the Catalog Generation component. It contains all messages issued by modules of GENCAT, as well as the ADD DBSPACE routine, ARISE6C. All messages in this module have message numbers in the range 520 through 529.

ARIM700

ARIM700 is the message module for the ISQL component. It contains all messages issued by modules of the ISQL component. All messages in this module have message numbers in the range 7000 through 7999.

ARIM800

ARIM800 is the message text module for the SQL/DS DBS Utility component. It contains the text of all messages issued by the DBS modules. All messages defined in this module have message numbers in the range 800 through 899.

PREP MODULES

ARIPADR

ARIPADR performs the COBOL preprocessor address function. ARIPADR, along with ARIPRDI, is link-edited with the user's COBOL run-time routine.

Entry Point: ARIPADR
Function: ARIPADR provides an "ADDR" function to COBOL. The addresses of the variables are placed in the SQLDATA slots.

Entry Point: ARIPADR1
Function: ARIPADR1 provides the same function as ARIPADR, except that it handles the SQLIND field. This field follows the SQLDATA field.

Entry Point: ARIPADR2
Function: ARIPADR2 is a general purpose routine that puts the address (PARM2) into the PARM1 location.

ARIPARM

ARIPARM is a macro containing code and declarations that perform the parse of the PREP input parameters. This macro needs to be preceded by a module PROC statement along with any DECLAREs that will be used that are not in this copy code. ARIPARM needs to be followed by:

1) An internal procedure called MSGOUT that performs message handling for the module including this macro
2) Any prolog code that needs to follow the end of the internal procedure MSGOUT.

ARIPCID

ARIPCID checks identifiers for valid contents. Length checking is up to the caller. The character string (XC) is checked left to right for the length indicated by the length field (XL). If unallowed characters are found embedded in the string, the module returns a return code of 12. If blanks are found and they continue to the end of the string according to the length specification (trailing blanks), then a return code of 4 is set. Embedded blanks are erroneous and cause a return code of 12. Valid identifiers will result in a return code of 0.

ARIPCVF

ARIPCVF converts a fullword binary value to its character (EBCDIC) representation. It is called only by modules that do not have STACK storage. The resultant character string is 16 bytes long and padded on the left with character zeros. If the value is negative, a minus sign is placed in the first byte of the string.

ARIPCVH

ARIPCVH converts a halfword binary value to its character (EBCDIC) representation. It is called only by modules that do not have STACK storage. The resultant character string is eight characters in length, padded on the left by character zeros. If the input value is negative, a minus sign is placed in the first byte of the string.

ARIPCVP

ARIPCVP converts a pointer from hexadecimal to its character (EBCDIC) representation. It is called only by modules that do not have STACK storage. This resultant string is eight bytes long.

ARIPDCF

ARIPDCF is passed a FORTRAN specification statement that the source scanner (ARIPSQF) has identified as potentially containing host variable declarations. ARIPDCF identifies all valid host variables and places information about them (name, type, length) in a declare list (DCLLIST) structure. Variables that are not recognized as valid host variables are ignored.

ARIPDYA

ARIPDYA is the Assembler dummy entry routine. ARIPDYA contains (dummy) external entries only, which never have to be used. It has no meaning for PREP, and is used only as input to the linkage editor to obtain a clean link map of Assembler PREP.

Certain modules have been designed for use in different SQL/DS components. They sometimes refer to external entries of other modules. When those modules are not used in this component they will not be part of the link book.

Since the linkage editor tries to resolve all address constants, the entries of the non-linked modules will appear as unresolved, however, these unresolved constants are of no consequence.

To avoid misunderstanding and to obtain clean linkage editor output, all address constants which must appear as unresolved are placed in this module as external entries.

ARIPDYC

ARIPDYC is the COBOL dummy entry routine. ARIPDYC contains (dummy) external entries only, which never have to be used. It has no meaning for PREP, and is used only as input to the linkage editor to obtain a clean link map of Assembler PREP.

Certain modules have been designed for use in different SQL/DS components. They sometimes refer to external entries of other modules. When those modules are not used in this component they will not be part of the link book.

Since the linkage editor tries to resolve all address constants, the entries of the non-linked modules will appear as unresolved, however, these unresolved constants are of no consequence.

To avoid misunderstanding and to obtain clean linkage editor output, all address constants which must appear as unresolved are placed in this module as external entries.

ARIPDYP

ARIPDYP is the PL/I dummy entry routine. ARIPDYP contains (dummy) external entries only, which never have to be used. It has no meaning for PREP, and is used only as input to the linkage editor to obtain a clean link map of Assembler PREP.

Certain modules have been designed for use in different SQL/DS components. They sometimes refer to external entries of other modules. When those modules are not used in this component they will not be part of the link book.

Since the linkage editor tries to resolve all address constants, the entries of the non-linked modules will appear as unresolved, however, these unresolved constants are of no consequence.

To avoid misunderstanding and to obtain clean linkage editor output, all address constants which must appear as unresolved are placed in this module as external entries.

ARIPDYT

ARIPDYP is the FORTRAN dummy entry routine. ARIPDYT contains (dummy) external entries only, which never have to be used. It has no meaning for PREP, and is used only as input to the linkage editor to obtain a clean link map of Assembler PREP.

Certain modules have been designed for use in different SQL/DS components. They sometimes refer to external entries of other modules. When those modules are not used in this component they will not be part of the link book.

Since the linkage editor tries to resolve all address constants, the entries of the non-linked modules will appear as unresolved, however, these unresolved constants are of no consequence.

To avoid misunderstanding and to obtain clean linkage editor output, all address constants which must appear as unresolved are placed in this module as external entries.

ARIPEIF

ARIPEIF is the macro that handles SQL interface calls from FORTRAN at execution time.

ARIPEIFA

ARIPEIFA is the Assembler input (PREP output) that handles SQL interface calls at execution time.

ARIPEIFP

ARIPEIFP if the PREP input (PL/S output) that handles SQL interface calls at FORTRAN execution time.

ARIPERR

ARIPERR is called whenever the Assembler, Cobol, and PL/I PREP modules encounter an error. It builds the SQLCA structure using the input parameters passed by the PREP routines. It inserts the module name, the RDS code, the SQL/DS code, and message tokens (if any).

ARIPFIX

ARIPFIX is the first module entered in VM (it is not used at all in VSE). This module picks up a list of three pointers (obtained via register 0). The first points to a phase name indicating which PREP is to be invoked (command name), the second to the starting address of a character string containing PREP parameters specified by the user, and the third to the byte past the last character in the string (referred to as the ending address). The starting and ending addresses of the string of user-specified PREP parameters are compared to see that the end is larger than the start but not greater than the maximum string length allowed. Then the string of PREP parameters and its length are put into a structure which is passed to the invoked PREP. Finally, the correct PREP is invoked by checking the command name and calling the corresponding PREP.

ARIPLTP

ARIPLTP establishes the length and type information in the DCLLIST for declared host variables in PL/I PREP.

ARIPMSF

All messages are written from ARIPMSF, both ARI.... PREP messages resulting from PREP-detected errors and ARI.... SQL messages resulting from SQLCODEs returned in the SQLCA structure by SQL/DS.

ARIPMSH

ARIPMSH is the message handler for the SQL/DS COBOL, PL/I, and Assembler preprocessors. It determines the type of error (from MSGPARM) that the preprocessor encountered. Then ARIPMSH puts out the proper message to the SYSLST (VSE) or SYSPRINT (VM) file and, if desired, to the SYSPCH (VSE) or SYSPUNCH (VM), or SYS001 file. It calls ARIMSMF to handle all SQL error messages. It calls ARIMFMT to handle all other error messages.

ARIPMSM

ARIPMSM is the message module for the SQL/DS preprocessors. It contains all the messages that the preprocessors can issue except for the SQL error messages.

This module contains no executable code. It is never branched to from any module. ARIPMSM is only referenced by the message formatter module (ARIMFMT).

ARIPMVL

ARIPMVL moves characters (long) from the source string to the target string.

ARIPPIF

ARIPPIF is the PL/S source that handles SQL interface calls at FORTRAN PREP time. It is input for a PL/S compile.

ARIPPIFA

ARIPPIFA is the Assembler input (PREP output) that handles SQL interface calls at FORTRAN PREP time.

ARIPPIFP

ARIPPIFP is the PREP input (PL/S output) that handles SQL interface calls at FORTRAN PREP time.

ARIPPRA

ARIPPRA is the main routine of the Assembler preprocessor. ARIPPRA takes a user application program written in Assembler with embedded SQL statements in it and processes it. The program is executed so it accesses a data base according to the SQL statements in it.

ARIPPRC

ARIPPRC is the main routine of the COBOL preprocessor. ARIPPRC takes a user application program written in COBOL with embedded SQL statements in it and processes the program. It is executed so it accesses a data base according to the SQL statements in it.

ARIPPRF

ARIPPRF is the main routine of the FORTRAN preprocessor and performs the following control functions:

- Opens and closes files
- Calls parameter scanner to get PREP parameter values
- Calls SQL interface to connect to SQL/DS
- Calls SQL interface to create program
- Calls SQL statement scanner to identify declares

- Merges workfiles to final output
- Calls SQL interface to commit/rollback access module
- Writes final summary messages.

ARIPPRP

ARIPPRP is the main routine of the PL/I preprocessor. ARIPPRP takes a user application program written in PL/I with embedded SQL statements in it and processes the program. It is executed so it accesses a data base according to the SQL statements in it.

ARIPRDID (CSECT ARIPRDID) - Batch Resource Manager Stub

ARIPRDID is link-edited with the VSE application program. A call to ARIPRDID (CSECT name of module ARIPRDID) is generated by the PREP process. ARIPRDID determines if the Batch Resource Manager is loaded. If the Batch Resource Manager is not loaded, a CDLOAD is issued and ARIPRDID branches to the Resource Manager. If the Resource Manager is loaded by this task, ARIPRDID simply branches to it.

ARIPSCF

ARIPSCF scans through the PREP parameters which are supplied to the FORTRAN SQL/DS precompiler via the EXEC JCL card or via a VM/CMS EXEC. It scans for the required parameters 'USERID' or 'USER', 'PREPNAME' or 'PREP', 'CHECK', 'NOCHECK', 'KEEP', 'REVOKE', 'LINECOUNT' or 'LC', 'PRINT' or 'PR', 'NOPRINT' or 'NOPR', 'PUNCH' or 'PU', and 'NOPUNCH' or 'NOPU'. It also scans for unknown or misspelled keywords and flags them as errors causing precompilation to terminate.

ARIPSCN

ARIPSCN scans through the Assembler, COBOL, and PL/I PREP parameters that are supplied to the SQL/DS preprocessors via the JCL (VSE) or EXEC (VM). It scans for the required 'USERID=' and 'PREPNAME=' keywords and for the optional keywords 'CHECK', 'NOCHECK', 'KEEP', 'REVOKE', 'LINECOUNT' or 'LC', 'PRINT' or 'PR', 'NOPRINT' or 'NOPR', 'PUNCH' or 'PU', 'NOPUNCH' or 'NOPU', and for COBOL only, 'QUOTE' or 'Q' and 'APOST'. It also scans for unknown (or misspelled) keywords, that ARIPSCN flags as errors causing preprocessing to terminate. Defaults, if used, are set in this module and are 'KEEP', 'LC' = 60, 'PRINT', 'PUNCH', 'QUOTE', and 'NOCHECK'.

ARIPSQA

ARIPSQA is the ASSEMBLER SQL scanner. ARIPPRA calls ARIPSQA to scan a user's program, locate SQL statements and host variables, and send the SQL statements and host variables back to ARIPPRA. To do this, statements are read from the user's program one at a time. A scan is done for the SQL trigger characters 'EXEC SQL'.

This module is entered in one of two modes:
- First Pass - Scan only for BEGIN DECLARE SECTION and END DECLARE SECTION. Send back the host variables.
- Second Pass - Scan for SQL statements.

ARIPSQC

ARIPSQC is the COBOL prep scanner. ARIPPRC calls ARIPSQC to scan through a user source program looking for SQL statements and user-declared host variables. This module is entered in one of several modes:
- FRSTMODE: Initialize the scanner and go on to NEUTMODE.
- NEUTMODE: Not in working storage, linkage, or file section and not in PROCEDURE division.
- DATAMODE: In working storage, linkage, or data section. If WORKSTOR (switch) is on, it is working storage.
- DCLMODE: In a SQL DECLARE section.
- SQLMODE: In the PROCEDURE DIVISION.

Except for FRSTMODE, these modes are changed by the scanner itself. FRSTMODE is set by ARIPPRC as the initial value for 'mode'.

If a minor error is detected, SQLCA is set. For a terminal error, PREPERR is set instead. Upon normal return, SEQSTMT contains either a host variable or a SQL statement (depending on the mode).

ARIPSQF

ARIPSQF scans all statements in the user's source program looking for SQL statements to process and for various FORTRAN statements that require special consideration.

All statements are written to SYS001, SQL statements are processed, compressed, and passed back to the main line if further processing is required.

If present, the FORTRAN statements that must be located are:
- PROGRAM, IMPLICIT, SUBROUTINE, or FUNCTION for setting control information

- IF (expression) EXEC SQL or ELSE (expression) EXEC SQL. The FORTRAN portion of the statement must be separated from the SQL portion for processing.
- END to indicate the end of a program, subroutine, or function statement.

ARIPSQP

ARIPSQP is the PL/I SQL scanner. ARIPPRP calls ARIPSQP to scan a user's program, locate SQL statements and host variables and send the SQL statements and host variables back to ARIPPRP. To do this, statements are read from the user's program one at a time . A scan is done for the SQL trigger characters 'EXEC SQL'.

ARIPSSF

ARIPSSF performs a parse of the SQL statements located by the source scanner routine (ARIPSQF). The parse identifies the type of statement by referring to the STMNTFLG set by ARIPSQF. If a statement contains host variables this module locates them. The identified host variables are checked against those in the DCLLIST (declare list) structure to see if they have been declared. The host variables are removed from the statement and replaced with question mark variables ('?'). The statement is then prepared by calling ARIPPIF to perform an EDSF PREPARE. The returned statement identifier and any other values needed by the text generator are placed in the STMTINFO structure, and control is returned to the caller.

ARIPSXC

ARIPSXC is the macro for passing declares on calls from the main COBOL compiler module (ARIPPRC) to the COBOL scanner module (ARIPSQC). Declares remain global to all calls to the scanner.

ARIPSXG

ARIPSXG is the macro containing the declares for the common structure TXPRMSTR passed between Assembler, COBOL, and PL/I preprocessors and their respective text generating modules, their respective SQL statement scanning modules, and their common parameter scanning module. It also contains the declaration of several constants used within various modules.

ARIPS1C

The ARIPS1C macro contains common initialization code that is used by the PL/I, COBOL, and assembler preprocessors. It is included by ARIPPRA, ARIPPRP, and ARIPPRC.

ARIPS1D

This macro contains the common declares that are used by the PL/I, COBOL, and assembler preprocessors. It is included by ARIPPRA, ARIPPRP, and ARIPPRC.

ARIPS1S

This common code contains subroutines that are used by Assembler, COBOL, and PL/I SQL/DS preprocessors. It is included by ARIPPRA, ARIPPRP, and ARIPPRC.

ARIPS2

This is common code used by the Assembler, COBOL, and PL/I SQL/DS preprocessors. It is included by ARIPPRA, ARIPPRP, and ARIPPRC. It performs the initial parse of the SQL statement, looking for certain statements that need processing done on them before they are sent to SQL/DS for further processing. It also looks for statements that do not have to be sent to SQL/DS at all.

The statements that this code examines are:

- WHENEVER
- PREPARE
- OPEN cursor
- CLOSE cursor
- EXECUTE
- EXECUTE IMMEDIATE
- FETCH cursor
- DESCRIBE
- DECLARE cursor
- ROLLBACK WORK
- COMMIT WORK
- CONNECT userid

For other types of SQL statements, they are sent to SQL/DS and are marked as AUX (access module) calls.

ARIPTXA

ARIPTXA is the text module for the assembler preprocessor. It contains the majority of statements that could be generated at PREP time within a user application program

written in Assembler. The key parameter is the TXPARM parameter. It is passed by ARIPPRA. The parameter tells what kind of code is to be generated within the user program.

ARIPTXC

ARIPTXC is the text module for the COBOL preprocessor. It contains the majority of the statements that could be generated at PREP time within a user application program written in COBOL. The key parameter is the TXPARM parameter. It is passed by ARIPPRC. The parameter tells what kind of code is to be generated within the user program.

ARIPTXF

ARIPTXF is the text module for the FORTRAN preprocessor and generates the statements necessary to process an SQL statement. A call with TXTPARM = 1 generates the statements required for the SQLCA. A call with TXTPARM = 2 generates the statements necessary for processing SQL statements. A call with TXTPARM = 3 generates the statements necessary to initialize the common items (using FORTRAN block data).

ARIPTXP

ARIPTXP is the text module for the PL/I preprocessor. It contains the majority of the statements that could be generated at PREP time within a user application program written in PL/I. The key parameter is the TXPARM parameter. It is passed by ARIPPRP. The parameter tells what kind of code is to be generated within the user program.

SQL/DS SQLCODE DESCRIPTIVE MESSAGE TEXT

ARIQM01

ARIQM01 is the message module for the SQL codes -199 through +100. It is referenced by the message formatter module ARIIFDR or ARIMFMT.

ARIQM02

ARIQM02 is the message module for the SQL codes -299 through -200. It is referenced by the message formatter module ARIIFDR or ARIMFMT.

ARIQM03

ARIQM03 is the message module for the SQL codes -399 through -300. It is referenced by the message formatter module ARIIFDR or ARIMFMT.

ARIQM04

ARIQM04 is the message module for the SQL codes -499 through -400. It is referenced by the message formatter module ARIIFDR or ARIMFMT.

ARIQM05

ARIQM05 is the message module for the SQL codes -599 through -500. It is referenced by the message formatter module ARIIFDR or ARIMFMT.

ARIQM06

ARIQM06 is the message module for the SQL codes -699 through -600. It is referenced by the message formatter module ARIIFDR or ARIMFMT.

ARIQM07

ARIQM07 is the message module for the SQL codes -799 through -700. It is referenced by the message formatter module ARIIFDR or ARIMFMT.

ARIQM08

ARIQM08 is the message module for the SQL codes -899 through -800. It is referenced by the message formatter module ARIIFDR or ARIMFMT.

ARIQM09

ARIQM09 is the message module for the SQL codes -1000 through -900. It is referenced by the message formatter module ARIIFDR or ARIMFMT.

ARIRBRM

ARIRBRM is mainline of the Batch Resource manager. It processes batch application EXEC SQL requests.

ARIRB13

ARIRB13 is the Resource Manager message routine for writing to the operator with no reply. It has the entry point ARIYE13 and uses the DBSS message routines.

ARIRB51

ARIRB51 is the Batch Resource manager message routine for writing to the operator with a reply. It has the entry point ARIYM51 and uses the DBSS message routines.

ARIRCL1(C - VM Only

ARIRCL1(C is the VM Resource Manager cancel exit top level module. It is invoked by CMS as the result of the terminal operator issuing the SQLHX command or, for ISQL, the CANCEL command. ARIRCL1(C establishes addressability to the RMLO and calls the second level module ARIRCL2(C.

ARIRCL2(C - VM Only

ARIRCL2(C is the VM Resource Manager cancel exit second level module. It checks if the cancel ECB should be posted, and if so, posts it.

ARIRDIS

ARIRDIS is the Online Resource Manager termination and display transactions' information (CIRD) routine. The termination process disables online access to SQL/DS. The display information process displays all transactions' information and status.

ARIRECID - VSE Only

ARIRECID contains all of the EXEC CICS interfaces used by the CIRB and CIRT transactions.

ARIRENAD - VSE Only

ARIRENAD is the Resource Manager begin routine. This process enables online access to SQL/DS.

ARIRENTD - VSE Only

CICS/VS invokes ARIRENTD when a terminal operator enters the CIRB or CIRT transaction. ARIRENTD gets storage for RMLO and the stack by invoking DFHEAI0. It initializes the RMLO and stack. Based on EIBTRNID, ARIRENTD calls either ARIREWAD for Resource Manager Begin or ARIRDISD for Resource Manager terminate.

ARIRINTC

ARIRINTC handles external interrupts in the resource manager machine which occur as a result of IUCV functions. The only interrupt types which result are CONNECTION COMPLETE, REPLY PENDING, and SEVER PENDING. The occurrence of any other type of interrupt is treated as an error.

ARIRIRM

ARIRIRM is the initialization function of the Batch Resource manager. It allocates storage for control blocks, maintains the one link per task relationship, and passes control to ARIRBRM.

ARIRMI1D - VSE Only

ARIRMI1D is the level 1 module for online application EDSF (EXEC SQL) calls and CICS/VS synchronization calls. ARIRMI1D has the following process:

- Set Register 10 to point to RMLO. Set Register 12 to point to the CICS/VS TCA. Set Register 13 to point to a register save area in RMLO.
- Initialize RMLO, the stack, and the RMWL for an initial call from a transaction.
- Chain the RMLO to the RMLO chains.
- Call ARIRORM to process the request.

ARIRMSG

ARIRMSG is the message text module of the Resource Manager component. It contains the text of those messages issued by

the Resource Managers. All messages in this module have message numbers in the range 400 through 499.

ARIRORMD - VSE Only

ARIRORMD is the mainline of the Online Resource Manager. It processes online application EXEC SQL requests and synchronization requests. ARIRORMD does RMLO dechaining.

ARIRO13

ARIRO13 is the route Online Resource Manager message routine. ARIRO13 has ARIYE13 as its entry point. It calls ARIRECI to write a message to the terminal or transient queue. If there is no terminal or if the message requires system operator attention, ARIRO13 calls ARISYSD3 to write the message to the system operator console.

ARIRPAT

ARIRPAT is the common patch area module for the Resource Manager component. See Section 6, Patch Area Modules and Access, for the patch area modules.

ARIRRTED - VSE Only

ARIRRTED is the Online Resource Manager stub. It has the CSECT name ARIPRDI. ARIRRTED is an expansion of the CICS macro DFHRMCAL. This expansion locates the CICS router routine and branches to it passing this router routine the Online Resource Manager phase name.

ARIRSEND - VSE Only

ARIRSEND is the Resource Manager send message routine. It has ARIPRDI as its entry point. ARIRSEND uses DSC service to process the RDI as follows:

- Gather application's input into one continuous message (mailbox).
- Send the message.
- Use the user exit or CICS/VS to wait for the reply.
- Scatter the reply message into the application's output area.

ARIRSQLA

ARIRSQLA provides the following linkages for the Online Resource Manager:

- EXEC SQL CONNECT
- RDI SCHEDULE
- EXEC SQL COMMIT WORK
- EXEC SQL ROLLBACK WORK
- RDI PREPARE TO COMMIT
- RDI GET RECOVERY LIST

ARIRSQLP
(macro)

The ARIRSQLP is the source for the assembler PREP of ARIRSQL. It must be distributed as part of the source statement library so that the access module ARIRSQL may be inserted into the installation data base. The output of the SQL/DS Prep becomes the module ARIRSQLA.

ARIRTL1(C - VM Only

ARIRTL1(C is the VM Resource Manager implicit COMMIT/ROLLBACK exit top level module. It is invoked by CMS at end of command when control is being returned to the terminal operator. ARIRTL1(C establishes addressability to the RMLO and calls the second level module ARIRTL2(C.

ARIRTL2(C - VM Only

ARIRTL2(C is the VM Resource Manager implicit COMMIT/ROLLBACK exit second level module. It issues a SEVER with user data indicating COMMIT on normal CMS command end or ROLLBACK on abnormal CMS command end. It also clears up the nucleus extensions established by the Resource Manager and frees up storage allocated by the Resource Manager.

ARIRVIR(C - VM Only

ARIRVIR(C is the VM Resource Manager initialization routine. It acquires storage for the RMLO, RMLT, RMXC, the required parameter lists for IUCV communications, and the default mailbox. It initializes the control blocks and passes control to the main-line module, ARIRVRM(C.

ARIRVRM(C - VM Only

ARIRVRM(C is the VM Resource Manager main-line module. It establishes the IUCV paths for multiple virtual machine mode communication with the SQL/DS virtual machine, and transfers control to SQL/DS in single virtual machine mode.

ARIRVST(C - VM Only

ARIRVST(C is the VM Resource Manager stub. ARIRVST(C has the entry point name ARIPRDI. It is link edited with each application. It loads the VM Resource Manager according to the information passed from the bootstrap module ARISRMK(C.

ARISRMK(C - VM Only

ARISRMK(C is the VM Resource Manager bootstrap module. It contains information about the Resource Manager load characteristics (whether the Resource Manager is to be loaded into DMSFREE storage or as a shared segment). If the Resource Manager is to be loaded as a shared segment, ARISRMK(C will also contain the name of the shared segment.

SQL/DS SYSTEM DEPENDENT AND GENERATION MODULES

ARISCNV

During catalog generation, ARISCNV finds the columns containing integers or small integers in the subject relation. ARISCNV then converts them so they may be inserted in the data base.

ARISCOL

ARISCOL searches a given relation for a specific column. That is, it finds the appropriate column pointer during catalog generation processing. The column structure is part of GCGLOBWA (catalog generation global workarea).

ARISDBBT

See ARISDBK.

ARISDBK - VM Only

ARISDBK is a module which is invoked in the VM environment only, by the SQLGENLD EXEC. It is a bootstrap module which is GENMODed under the name ARISDBBT, and later invoked by the SQLSTART EXEC. It loads the DBSS/DSC and RDS code and then branches to Initialization Phase 1 (ARICIP1).

ARISDBM

ARISDBM contains the message texts for messages ARI900I through ARI999I which are issued by the data base generation process and by the Trace Formatter.

ARISDBR

ARISDBR opens SYSIPT (VSE) or SYSIN (VM) and reads the first set of data base generation control cards that describe the potential size of the data base which is expressed in keywords. Builds a table of values (either default or specified) for the keywords and returns the pointer to the table in a field passed as input.

ARISDFL

ARISDFL defines a link in the catalog and writes linkids into the SYSBOOT relation. It processes a DEFLINK command from the catalog generation input stream. It finds the relation IDs of the parent and child relations, uses DBSS to do the define, and updates the SYS structure that will later be inserted in the SYSBOOT catalog.

ARISDFM

ARISDFM processes the GENCAT DEFIM command and DEFUIM command during catalog generation. It defines an index on a system catalog and updates the SYS structure that will later be inserted in the SYSBOOT catalog.

ARISDFR

ARISDFR processes the DEFREL command during catalog generation. It defines a relation for a system catalog and updates the relation identifier in the SYS catalog structure that will later be inserted in the SYSBOOT catalog.

ARISDSK

ARISDSK defines additional DBEXTENTs to the data base using as input control information read from SYSIPT (VSE) or SYSIN (VM).

ARISD00D - VSE Only

ARISD00D is a generalized I/O routine for system and sequential disk/tape files.

ARISD01D - VSE Only

ARISD01D is a read access routine to the source statement library via VSE macro DTFSL.

ARISD10D - VSE Only

ARISD10D is a Disk file access routine for fixed-length unblocked records.

ARISD11D - VSE Only

ARISD11D is a Disk file access routine for fixed-length blocked records.

ARISD12D - VSE Only

ARISD12D is a Disk file access routine for variable-length unblocked records.

ARISD13D - VSE Only

ARISD13D is a Disk file access routine for variable-length blocked records.

ARISD14D - VSE Only

ARISD14D is a Disk file access routine for spanned unblocked records.

ARISD15D - VSE Only

ARISD15D is a Disk file access routine for spanned blocked records.

ARISD16D - VSE Only

ARISD16D is a Disk file access routine for undefined records.

ARISD20D - VSE Only

ARISD20D is a Tape file access routine for fixed-length unblocked records.

ARISD21D - VSE Only

ARISD21D is a Tape file access routine for fixed-length blocked records.

ARISD22D - VSE Only

ARISD22D is a Tape file access routine for variable-length unblocked records.

ARISD23D - VSE Only

ARISD23D is a Tape file access routine for variable-length blocked records.

ARISD24D - VSE Only

ARISD24D is a Tape file access routine for spanned unblocked records.

ARISD25D - VSE Only

ARISD25D is a Tape file access routine for spanned blocked records.

ARISD26D - VSE Only

ARISD26D is a Tape file access routine for undefined records.

ARISD30D - VSE Only

ARISD30D is a SYSLST Write routine used for internal (ARISYSDD-generated) messages and I/O trace records only.

ARISD31D - VSE Only

ARISD31D is a SYS001 Workfile I/O routine.

ARISD32D - VSE Only

ARISD32D is a SYS002 Workfile I/O routine.

ARISD33D - VSE Only

ARISD33D is a SYS003 Workfile I/O routine.

ARISEDI - VSE Only

ARISEDI breaks the catalog generation input record into tokens and places them in an array so the information may be processed by GENCAT.

ARISEGA

ARISEGA defines additional DBSPACEs to the data base using as input control information read from SYSIPT (VSE) or SYSIN (VM).

ARISEGB

ARISEGB modifies the SYSDBSPACES catalog as a result of an ADD DBSPACE command. It first issues a DBSI (ARIYM00) BEGIN WORK call to begin the LUW. Then internal procedure ARISYSB is called to open the SYSBOOT catalog. This is needed to get the RID (relation id) of the DBSPACE catalog so we may update the data base with the new DBSPACE information. A DBSI (ARIYM00) CNEXT command is issued to get the control information for each DBSPACE. If the return code from the CNEXT call is NFORMSEG(16) then the DBSPACE exists but has not been opened - these are the ones we must add to the DBSPACE catalog. A DBSI (ARIYM00) INSERT is done to update the catalog. Then a DBSI COMMIT WORK is issued.

ARISERR

ARISERR writes a GENCAT **or** ADD DBSPACE message to SYSLST (VSE) or SYSPRINT (VM). ARISERR builds the structures MSGID and BUFFER and invokes the message formatter for message retrieval and variable substitution. After the message is formatted, the system dependent routine is invoked to write the message. For multi-line messages, ARISERR loops printing each line of the message.

ARISFDB

ARISFDB calls the data base generation read keyword routine (ARISDBR) which returns a table of specified values for the keywords MAXEXTNTS, MAXDBSPC, and MAXPOOLS.

ARISFIL

ARISFIL sets up a row for each SYSDBSPACE in the data base ARISFIL then calls DBSI (ARIYM00) to insert them into the catalog SYSDBSPACE. ARISFIL fills the SYSDBSPACE catalog with the correct information about each DBSPACE defined at catalog generation time.

ARISFM1

ARISFM1 performs physical formatting of data base files.

ARISFM2

ARISFM2 defines DBSPACE(s) to SQL/DS by updating I/O component control blocks.

ARISIIO

ARISIIO computes the I/O table sizes from specified maximum values contained in a table built by ARISDBR.

ARISINI

ARISINI handles the INITIALIZE DBSPACE command of catalog generation. It decodes the input card and then does an insert DBSI (ARIYM00) CINSERT call to a DBSPACE control relation for this DBSPACE.

ARISISBT

See ARISISK under "ISQL Modules" on page 492.

ARISISK

See ARISISK under "ISQL Modules" on page 511.

ARISIST

ARISIST processes an INSERT command during catalog generation. The catalogs are upated with the information provided **on** the input record via a DBSI (ARIYM00) INSERT call.

ARISIST2

As a result of an INSERT command, ARISIST2 handles updating specific catalogs during insert processing for catalog generation. Special processing must occur for certain insert statements of GENCAT. These special processes are: SYSCATALOG, SYSCOLUMNS, SYSINDEXES, or SYSLINKS. A separate routine handles the processing for each of the subject catalogs.

ARISLAY

ARISLAY fills the SYSBOOT catalog with the row containing the catalog relation, index, and link IDs created during catalog generation. It uses the information set up in the SYS structure during GENCAT processing.

ARISMAI

ARISMAI is the main GENCAT module that reads the GENCAT control statements from the Source Statement Library (VSE) or from a CMS file (VM). It first obtains a data storage area for GENCAT processing. It calls DBSI (ARIYM00) with a BEGIN WORK request to begin the LUW. It then reads each record and determines the function required. The appropriate GENCAT routine is called to process the control statement. After each record is read and the catalogs are built, any user supplied records are read and processed. ARISLAY is then called to update the SYSBOOT catalog with the IDs of the catalogs. Then DBSI (ARIYM00) is called again to end the LUW (COMMIT WORK), and the GENCAT storage is freed.

ARISOCM

ARISOCM provides the linkage between the VSE operator console and the SQL/DS operator command linkage module (ARIYM10) when the operator wants to enter a SQL/DS operator command. The module is entered when the system operator enters the VSE attention command MSG that notifies the the VSE attention routine that he wants to communicate with the SQL/DS partition. The operator agent structure is posted to 'wake up' the operator agent causing module ARIYM10 to be dispatched.

ARISPFM

ARISPFM is the first routine called by ARIYT00 (INITRSS) for startup = C meaning DB GEN. This routine calls ARISFM1 to format the directory, DBEXTENTs, and log(s). In addition, it reads the DBEXTENT/storage pool and DBSPACE definition control records from SYSIPT (VSE) or SYSIN (VM). Using this information it initializes the extent table, pool table and summary counters. It calls ARISFM2 passing the DBSPACE information read to build the SEGTABLE and SEG attribute table.

ARISREL

ARISREL finds a relation pointer during catalog generation for a system catalog. The relation structure is part of GCGLOBWA (catalog generation global work structure).

ARISSCB

ARISSCB displays the contents of the data base control block.

ARISSET

ARISSET verifies the SET DEFAULT catalog generation command, ensuring the default value is a valid integer. Then ARISSET sets the default in the SYS structure that will later be inserted into the SYSBOOT catalog.

ARISTIM

For a functional description of ARISTIM see ARICTIM on page 478

ARISTLD

ARISTLD uses SQL DROP to drop old HELP tables, HELP.SYSTEXT1 and HELP.SYSTEXT2, if they exist. ARISTLD uses SQL CREATE to create new tables. It also creates indexes on these two tables.

ARISUPD

ARISUPD handles the UPDATE command during catalog generation. The information on the control record is used to update the appropriate catalog row in the data base.

ARISYSD - VM/SP System-Dependent Module

Entry Point: ARISYSD
Function: ARISYSD returns a three-character value into a user-supplied area designating the operating system environment ('CMS' = VM/SP Operating System).

Entry Point: ARISYSD1
ARISYSDI
Function: ARISYSD1/ARISYSDI gets storage (conditionally) via CMS DMSFREE macro.

Entry Point: ARISYSD2
Function: ARISYSD2 frees storage via CMS DMSFRET macro.

Entry Point: ARISYSD3
Function: ARISYSD3 writes to the virtual machine operator console. The maximum message length is 80 bytes.

Entry Point: ARISYSD4
Function: ARISYSD4 writes to the virtual machine operator's console and waits for a reply. The maximum message length is 80 bytes. The maximum reply message length is 68 bytes. Replies longer than 68 bytes will be truncated.

Entry Point: ARISYSD5 - None (See module ARISYSE)

Entry Point: ARISYSD6
Function: ARISYSD6 (in module ARISYSD) performs a 'LOAD' function in the VSE/Advanced Function environment. It will not be used in the VM/SP environment. If invoked, it returns a return code 255 (function not supported) to the caller.

Entry Point: ARISYSD7
Function: ARISYSD7 performs a system-dependent abend and, optionally, a dump to the virtual printer. The abend code is inserted into register 15.

Entry Point: ARISYSD8 - None (See module ARISYSF)

Entry Point: ARISYSD9
Function: ARISYSD9 performs a system-dependent load into, or drop from, the area requested by the caller. The load address, entry point, and end address are returned to the caller when possible. The system return code is returned to the caller in LOADRETC.

Entry Point: ARISYSDA
Function: ARISYSDA performs a dump to the virtual printer and returns control to the caller.

Entry Point: ARISYSDB
Function: ARISYSDB is not used in the VM/SP environment. A return code of 255 (function not supported) is passed to the caller.

Entry Point: ARISYSDC
Function: ARISYSDC is not used in the VM/SP environment. A return code of 255 (function not supported) is passed to the caller.

Entry Point: ARISYSDE
Function: ARISYSDE provides the VM user id. It also provides the 'time zone' as a geographical position relative to Greenwich Mean Time, as a two-byte binary value, indicating the difference in minutes between local time and Greenwich Mean Time. A negative value means 'west', and a positive value means 'east'.

Entry Point: ARISYSDF
Function: ARISYSDF is NOPed. It returns a return code 0 to the caller.

Entry Point: ARISYSDG
Function: ARISYSDG provides an unconditional get free storage function via CMS DMSFREE macro. If the storage cannot be obtained, an error message is issued to the virtual machine console. Return is made to the operating system (CMS).

Entry Point: ARISYSDH - None (See module ARISYSG)

Entry Point: ARISYSDI (this is the same entry point as ARISYSD1 - see that description)

Entry Point: ARISYSDJ
Function: ARISYSDJ reads message replies and operator commands from the virtual machine operator console. Maximum message reply/operator command is 130 bytes.

Entry Point: ARISYSDK
Function: ARISYSDK provides a CMS abend exit capability, allowing programs to intercept abend situations before abend recovery begins. The exit is specified via the CMS ABNEXIT macro.

ARISYSD - VSE/Advanced Functions System-Dependent Module

Entry Point: ARISYSD
Function: ARISYSD returns a three-character value into a user-supplied area designating the operating system environment ('DOS' = VSE (DOS) Operating System).

Entry Point: ARISYSD1
Function: ARISYSD1 gets storage (conditionally) via VSE GETVIS macro.

Entry Point: ARISYSD2
Function: ARISYSD2 frees storage via VSE FREEVIS macro.

Entry Point: ARISYSD3
Function: ARISYSD3 writes to the operator. The maximum message length is 80 bytes.

Entry Point: ARISYSD4
Function: ARISYSD4 writes to the operator and waits for a reply. The maximum message length is 80 bytes. However, the recommended maximum size is 68 bytes.

Entry Point: ARISYSD5
Function: ARISYSD5 performs general input/output functions to:

SYSIPT	SYS001	Disk sequential files
SYSPCH	SYS002	Tape sequential files
SYSLST	SYS003	

Source Statement Library (simulated SYSIPT input)

Each I/O request is described by means of the File Descriptor Block (macro ARIGENIO).

ARISYSD5 loads the generalized I/O routine ARISD00D into the GETVIS area (if not already loaded) and provides a work area of 4K bytes. The address of the work area is passed in Register 13. The work area used is requested only for OPEN processing. It is kept until CLOSE processing is finished. The work area address is saved in the File Descriptor Block (field name SYSDDWA).

All necessary information is stored within the save area (part of file descriptor) provided by the caller. Therefore, it is essential for this program that all following requests after OPEN refer to the original File Descriptor Block (OPEN/GET/PUT/CLOSE requests).

ARISYSD and ARISD00D use the save area. The save area contains the link address, the DTF address, the alternating I/O register (IOREG2), and counter and initialization information.

The link to ARISD00D is established on an OPEN request and disconnected after CLOSE. The link address is saved in the save area together with the address of the work area established via GETVIS during processing of the OPEN request. After a return from a CLOSE request, the work area is freed.

Entry Point: ARISYSD6
Function: ARISYSD6 performs an operating system dependent load (The VSE CDLOAD macro loads the requested phase into the partition GETVIS area.)

Entry Point: ARISYSD7
Function: ARISYSD7 provides for abnormal cancellation of a task and inserts the completion code into register 15. A partition dump is optionally provided.

Entry Point: ARISYSD8
Function: ARISYSD8 provides read access to the parameter library using 'FIND' for the member entry and 'GET' for reading 80 byte records. (The source statement library will be accessed.) The caller supplies the sublibrary (with 'A' as default).

Entry Point: ARISYSD9
Function: ARISYSD9 is not used in the VM/SP environment. A return code of 255 (function not supported) is passed to the caller.

Entry Point: ARISYSDA
Function: ARISYSDA performs a dump and returns control to the calling function.

Entry Point: ARISYSDB
Function: ARISYSDB establishes an 'OPERATOR COMMUNICATION EXIT', which is invoked when an operator console interrupt has been issued. It assumes that Register 10 is pointing to 'YRSSCVT' or (RDAREA).

Entry Point: ARISYSDC
Function: ARISYSDC provides storage related information (partition start and end addresses).

Entry Point: ARISYSDE
Function: ARISYSDE provides the 'time zone' as a geographical position relative to Greenwich as a decimal value, indicating the difference in minutes between local time and Greenwich mean time. The value given is a two-byte value. A negative value means 'west', and a positive value means 'east'. The zone value is set via the IPL SET command.

Entry Point: ARISYSDF
Function: ARISYSDF pages out a requested area of virtual storage. This code is NOP'ed currently.

Entry Point: ARISYSDG
Function: ARISYSDG provides an unconditional GETVIS function. In case the request is not completed successfully, it issues an error message on the operator console and returns to the operating system via CANCEL.

Entry Point: ARISYSDH
Function: ARISYSDH writes a message to the operator console, SYSLST, or both, depending on the information contained in 'PUTSWIT'. This function communicates with the already existing entries ARISYSD3 and ARISYSD5 and also returns the return code given by these entries.

Entry Point: ARISYSDI
Function: ARISYSDI is a special entry point to satisfy storage requests originating from ARICWSG, ARIXEAB, and ARIXELX. All other requesters are serviced by entry point ARISYSD1. The difference is that the storage search by ARISYSDI begins at a special address contained in the DS2CVT pointer called DS2GETVP, which is set up by ARICIP1 and ARICCRA. This method of search results in a performance enhancement due to reduced paging.

Entry Point: ARISYSDJ
ARISYSDJ is not used in the VSE environment. A return code of 255 (function not supported) is passed to the caller.

Entry Point: ARISYSDK
Function: ARISYSDK provides a VSE abend exit capability to allow programs to intercept abend situations before abend recovery begins. The exit routine is specified via the STXIT AB macro.

ARISYSE - VM Only

ARISYSE is always invoked via entry point ARISYSD5 for sequential input/output functions including the system files (SYSIN, SYSPRINT, SYSPUNCH), work files for PREP, and general tape and DASD (normally CMS) files. It processes OPEN calls (in or out), CLOSE calls, and GET and PUT calls. Status and control information about the request is passed to entry point ARISYSD5 and to the caller via the File Descriptor Block (ARIFDESC, generated by the ARIGENIO macro).

CMS/OS QSAM is used to access system files and general files. The CMS FS macros (FSCB, FSREAD, FSWRITE, etc.) are used to access PREP work files. For standard label tape files, ARISYSE provides support for multi-volume files (using the CMS TEOVEXIT, RDTAPE, and TAPESL macros) since CMS/OS QSAM does not support multi-volume tape files.

The first OPEN call obtains automatic storage for the module for processing the specified file. The automatic storage is retained (unless the OPEN fails) until the file is closed. Unlike VSE/Advanced Functions ARISYSD5, ARISYSE does not load any auxiliary modules to perform its function.

ARISYSF - VM Only

ARISYSF is always invoked via entry point ARISYSD8 to find and read a card image CMS file. In addition to invocation by various SQL/DS programs, this function is invoked by ARISYSE when a READ FILE statement is encountered on SYSIN processing.

The 'FIND' call obtains automatic storage for the module for processing the file. The automatic storage is retained until EOF is encountered or until an error occurs.

ARISYSG - VM Only

ARISYSG is always invoked via entry point ARISYSDH to display (for the DSC) a message or other line to the virtual machine terminal. Note that unlike the VSE/Advanced Functions ARISYSDH entry point, it never outputs to a print file. It calls entry point ARISYSD3 in module ARISYSD for the output display.

RDS AUTHORITY

ARIXA01 - CONNECT

ARIXA01 checks the password by looking into the SYSUSERAUTH catalog. It then calls DBSS with SCHEDULE call and sets RDAUSER to the passed userid. ARIXA0 sets RDASPEC from the SYSUSERAUTH catalog and turns on the RDASCHED bit.

ARIXA02

ARIXA02 checks whether a person is authorized to perform a specific operation on an arbitrary relation. ARIXA02 also checks whether a person is allowed to run a program.

ARIXA03

ARIXA03 deletes the columns from SYSCOLAUTH associated with the target of a deleted TABAUTH row.

ARIXA04 - DROP/REVOKE

ARIXA04 is responsible for the deletion of rows from the tables:

- SYSTEM.SYSTABAUTH
- SYSTEM.SYSCOLAUTH
- SYSTEM.SYSUSERAUTH
- SYSTEM.SYSPROGAUTH

ARIXA06 - Grant Authority

ARIXA06 is responsible for all entries in the following tables:

- SYSTEM.SYSTABAUTH
- SYSTEM.SYSCOLAUTH
- SYSTEM.SYSUSERAUTH
- SYSTEM.SYSPROGAUTH

ARIXA07 - GRANT Privilege

ARIXA07 operates on the parse tree. Authorizations are checked by calling ARIXA02. Object existence is checked in the catalogs. ARIXA06 is then called for each user in the usernode list.

ARIXA08 - REVOKE Authority

ARIXA08 searches along a chain of grantor/grantee combinations. ARIXA09 initially calls ARIXA08. ARIXA08 calls itself if it finds a grantee that has granted his rights to somebody else - a grantee that has used a table in a program or a grantee who has defined a view.

ARIXA09

ARIXA09 is responsible for the revocation of row(s) corresponding to the GRANT command issued from the revoker to the revokee at an earlier time. Only this initial row is handled by ARIXA09. ARIXA08 handles the remaining rows in the authorization table that may be affected by this REVOKE command.

ARIXA10

ARIXA10 calculates the minimum time, when a specific grantee has received his authorization rights.

ARIXA11

ARIXA11 calculates the minimum AUTH for a view. When a view is created or an update is made on a view AUTH, the minimum AUTH on the view from all the underlying tables is used or granted.

ARIXCOT - SQL/DS Code Generator table that defines operator nodes in the ASL tree. It is used by ARIXC37 to determine which routine in ARIXC38 is to be called to process an operator in the optimized ASL tree.

ARIXCRA

ARIXCRA is a Code Generator routine to call the DBSS to perform a sort. It also makes sure the TID, as well as the SCANID, is zeroed when we close the scan.

ARIXCRB

ARIXCRB is a Code Generator routine that calls DBSS to acquire a temporary DBSPACE.

ARIXCRC

ARIXCRC is a Code Generator run-time routine to call the DBSS to initialize a temporary DBSPACE and to create a relation or list.

ARIXCRD

ARIXDRD is a Code Generator run-time routine to delete rows with long fields.

The input variable YBASEPTR points to a YBASE, that describes a parent whose children are to be deleted. This YBASE may be obtained in one of the following ways:

1. After an INSERT -- the long-field children inserted first, but the insertion of the parent failed (for example, due to unique key violation). All necessary information to find and delete the children may be found from this YBASE.
2. Before a DELETE -- before deleting a row with long-field children, fetch that row and return data for at least the long fields. When that is done, you have all the information that is in the INSERT case.
3. After an UPDATE -- if no long fields were updated, then no long-field code is invoked. If a long field was updated, this update was accomplished by inserting the long fields first. Then update the parent row. If the update succeeds, delete the old long field and leave the new long field. If the update fails, leave the old fields and delete the new fields. Therefore, the update code must have fetched the row to be updated (otherwise, you lose the pointers to the long fields). In order to preserve this information across the UPDATE command, the update must be done through a separate YBASE. Therefore, after success or failure of the update, you are passed a YBASE that points either to the old or new TIDs. On a success, we receive the YBASE for the fetch. On a failure, we receive the YBASE for the update. We do not need to know which is which. Either way, the auxiliary information hanging from the end of the YBASE, together with the data pointed to by the domain structure, tells us all you need to know.

The algorithm:

1. Build a YBASE2 using the SEGMENT / RID / LID information pointed to by the tail end of YBASE. This YBASE2 is used to open a scan on the children, one by one.
2. For each long field in the LFINFO:
   A. Find the TID of the list head. LFCOLNR is used to index domains, whose FLDPTR points to the datum from the data base, which contains the desired TID.
   B. The datum also contains the null indicator (if any) and the length of the long field. If the long field is null or is of zero length, then there is nothing to do -- advance to next long field.
   C. Install the TID in the YBASE2 and open a scan.
   D. Delete the row returned by the OPEN (or NEXT).
   E. Subtract 4000 from the length of the long field. When that number goes zero or negative, we have deleted all the rows comprising the long field. Now we can close the scan and advance to the next long field. This length-counting trick saves us the RSI call which would return not found.
   F. Otherwise, do a NEXT on the scan and loop by returning to (D).
3. We do nothing with the parent row -- that is the responsibility of the caller. Return to him.

ARIXCRE

ARIXCRE is a Code Generator run-routine to bind an input variable to a floating-point slot. It performs the copy, taking care of the conversion from integer or halfword type if necessary. ARIXCRE updates the pointers in Registers (1) and (4).

ARIXCRF

ARIXCRF is a Code Generator run-time routine to determine whether a given string is "LIKE" a given pattern. The pattern may contain the characters "_" and " " ("_" matches any single character, while " " matches an arbitrary string of zero or more characters).

ARIXCRG

ARIXCRG is a Code Generator run-routine that updates a row containing long fields being modified. (If the row contains long fields, but none of the long fields are modified, then the normal update processing handles that case.)

ARIXCRG is passed two pointers:

NEXTPTR - Points to a YBASE that fetches the row to be updated.

MODPTR - Points to a YBASE that performs the update.

ARIXCRH

ARIXCRH is a Code Generator run-time routine to bind an input variable to a decimal slot. It performs the copy, taking care of the conversion from halfword, integer, decimal, or float type if necessary. ARIXCRH updates the pointers in Registers (1) and (4).

ARIXCRI

ARIXCRI is the Code Generator run-time routine to convert a decimal number to a floating point number.

ARIXCRJ

ARIXCRJ is a Code Generator run-time routine to convert a floating-point number to a decimal number.

ARIXCRL

ARIXCRL is a Code Generator run-time routine to determine whether a given string is "like" a given pattern. The pattern may contain the characters "426D" and "426C" (in hexadecimal) ("426D" matches any single character, while "426C" matches an arbitrary string of zero or more characters). This routine is similar to ARIXCRF, except that it only checks for two bytes at a time.

ARIXCR0

ARIXCR0 is a Code Generator run-time routine to close the currently open scan.

ARIXCR1

ARIXCR1 is the Code Generator run-time routine to handle run-time errors.

ARIXCR2

ARIXCR2 is the Code Generator run-time routine to put out data to the select list. ARIXCR2 returns the values in a select list to the user's data space.

ARIXCR3

ARIXCR3 is the Code Generator run-time routine to insert a literal row with long fields.

ARIXCR4

ARIXCR4 is the Code Generator run-time routine to find a cursor.

ARIXCR5

ARIXCR5 is the run-time routine to bind an input variable to an integer slot. It performs the copy, taking care of the conversion from halfword, integer, decimal, or float type if necessary, and updates the pointers in Registers (1) and (4).

ARIXCR6

ARIXCR6 is the run-time routine to bind an input variable to a halfword slot. It performs the copy, taking care of the conversion from halfword, integer, decimal, or float type if necessary. ARIXCR6 updates the pointers in Registers (1) and (4).

ARIXCR7

ARIXCR7 is the run-time routine to bind an input variable to a fixed-length character, or fixed-length DBCS character slot. It performs the copy, taking care of the conversion from variable-length character type, if necessary. ARIXCR7 updates the pointers in Registers (1) and (4).

ARIXCR8

ARIXCR8 is the run-time routine to bind an input variable to a variable-length character, or variable-length DBCS character slot. It performs the copy, taking care of the conversion from fixed-length character type, if necessary. ARIXCR8 updates the pointers in Registers (1) and (4).

ARIXC01

ARIXC01 allocates space in BLK10 (data block 10) for the caller. It passes back the displacement (in INDX10) into the data block from which the space was allocated.

ARIXC02

ARIXC02 puts out code in Block 2 to perform arithmetic operations. The basic steps are:

1. Build a node on the semantic stack that describes the result.
2. If necessary, test whether either operand is null. Store a 0 or -1 in the NULLINFO slot of the result.
3. Load the left-hand operand into a register. This may involve moving it to the appropriate boundary and/or flipping the sign bit.
4. If the operand cannot be performed on the right-hand operand as it exists in main storage, load it into a register.
5. Perform the operation and store the result in the dynamic storage area.

ARIXC03

ARIXC03 puts out code in Block 2 to bind input variables at OPEN. It finds the list of input variables. Then ARIXC03 puts out code that binds them at open time.

ARIXC05

ARIXC05 puts out code in Block 2 to perform compares. It:

1. Generates the code for a comparison predicate.
2. Handles the "BETWEEN" predicate.
3. Handles the "IN" predicate.
4. Also tries to handle the "EXISTS" predicate.

ARIXC06

ARIXC06 puts out code to evaluate a described SET function. It takes a SET function node, pointed to by CGCMPSTK(STKPTR).CGCSNODE, generates code that computes the value of the SET function, and leaves on the semantic stack a pseudo-parse tree node describing the result. Its action is similar to that of ARIXC02.

ARIXC06 now computes the average and stores it. The average is stored on top of the sum. This is done so that the ASL tree for the HAVING clause can be created during optimization.

ARIXC07

ARIXC07 resolves undefined labels in Block 2. It resolves previously undefined symbols in the fragments that exist in the output code block.

ARIXC08

ARIXC08 determines the displacement of a data slot's data, null indicator, and length field.

ARIXC10

ARIXC10 processes the GROUP BY command. It generates code for saving the data identifying a group (move from column into a slot, as indicated in the bind table). It also generates code for comparing group domains to identify a new group in the bind table.

ARIXC11

ARIXC11 puts out code to load a register with the address of a block of data.

ARIXC13

ARIXC13 is the SQL/DS Code Generator initialization routine. It:

1. Allocates another data block.
2. Runs down the pre-existing data blocks, counting how many there are. It fills in the NEXTSPACE PTROF, the last block with the address of the newly allocated block.
3. Fills in the new data block.
4. Retraces the chain of data blocks.
5. Fixes up our new block.
6. Fixes up the block to receive our generated code.
7. Initializes CODESTR to no error detected.

ARIXC14

ARIXC14 cleans up loose ends at the end of code generation.

ARIXC15

ARIXC15 is the SQL/DS Code Generator routine that assists in using WHERE <expr-1> <= ANY SELECT expr-2> .

<expr-2> is located by BQINDX. BQINDX is an index into the syntactic stack. At the location, we find a BQUERYNODE and its operand(s). <expr-1> is located directly. PLHS is a pointer to the parse tree node that describes the value of <expr-1>.

If we are in the case of a GROUP BY query, then ARIXC38 calls out a different fragment. ARIXC38 is also responsible for calling out ARIXF05 in the usual (non-GROUP-BY) case.

ARIXC16

ARIXC16 is the SQL/DS Code Generator routine that puts out code to test the null byte of value. It uses the "IS(IS NOT) NULL" predicate.

ARIXC17

ARIXC17 is the SQL/DS Code Generator routine that puts out code to load a register with the address of a data slot.

ARIXC18

ARIXC18 is the SQL/DS Code Generator routine that moves the named fragment of machine code into Block 2 where we are building a section. To fix it up, relocate internal pointers and take note of external references.

ARIXC19

ARIXC19 is the SQL/DS Code Generator routine that allocates a new space block and relocation directory. This holds the compiled code for a compiler generated subroutine.

ARIXC20

ARIXC20 is the SQL/DS Code Generator routine that puts out code to load a block address into a register. Then ARIXC20 returns a block-number/displacement of data to the caller. It sets the block number and displacement into the block for the data associated with a parse tree entry.

ARIXC21

ARIXC21 is the SQL/DS Code Generator routine that puts out code to move data items to the user area.

ARIXC22

ARIXC22 is the SQL/DS Code Generator routine that puts out code to move a returned subquery to a slot in Block 1. This is where the subqueries are used as input values for an outer query.

ARIXC23

ARIXC23 processes the BINDTABLE. Call this routine when data is to be moved around. Follow the PTREEP1 pointer to a BINDTABLE in the OP. Put out the code that performs the actions indicated in the bind table.

ARIXC24

ARIXC24 is the SQL/DS Code Generator routine that puts out code to call subroutines for subqueries.

ARIXC25

ARIXC25 is the SQL/DS Code Generator routine that handles the processing of the binding operation for a re-open on the cursor situation. It re-opens the bind tables.

ARIXC27

ARIXC27 is the SQL/DS Code Generator routine that puts out code to initialize the subquery output slots to null.

ARIXC28

ARIXC28 puts out code to initialize for a SET function.

ARIXC29

ARIXC29 is the SQL/DS Code Generator routine that sets up tables and code blocks. The tables and code blocks place machine code into a new block for a subroutine.

ARIXC30

ARIXC30 is the SQL/DS Code Generator routine that puts out code to test if "end-of-group" was encountered for the RHS of a Type-2 join. (It tests the "ADVANCE" bit.)

ARIXC31

ARIXC31 is the SQL/DS Code Generator routine that puts out code to test the relation of a RHS row to a LHS row. Then ARIXC31 takes appropriate action. It tests the end-of-group (for Method-2 joins). JOINDESP points to a JOINDESCRIP, that leads us to two parse-tree nodes:

1. A COLUMNNODE that describes the slot containing the value of the join column on the left-hand side.
2. A COLUMNNODE describing a slot which contains the value of the join column in this row (the RHS).

Our job is to produce code that tests the relation of the RHS to the LHS. There are three possibilities:

1. The RHS is less than the LHS. Go back and fetch another.
2. The RHS is equal to the LHS. We have a winner: return to the caller after checking the WHERE clause.
3. The RHS is greater than the LHS. Return "not found" to the caller.

ARIXC32

ARIXC32 is the SQL/DS Code Generator routine that puts out code to test for a new group. Then ARIXC32 takes appropriate action. It tests the new group (for Method-2 joins). JOINDESP points to a JOINDESCRIP, that leads us to two parse-tree nodes:

1. A LITNODE describing a slot containing the value of the join column in the previous row
2. A COLUMNNODE describing a slot containing the value of the join column in this row.

Our job is to produce code that:

- Skips the test, if we are currently sitting on the first row.
- Tests whether the scan has entered a new group (i.e., whether (LITNODE)=(COLUMNNODE)).
- Sets Register (9) to 0, if the scan has entered a new group.
- Copies the first row value into the memory slot (the LITNODE), if the scan has entered a new group, or if you are pointing to the first row (i.e., if you are sitting on the first row of a new group).
- Sets Register (9) to 1, if it is still in the same group.

ARIXC33

ARIXC33 is the SQL/DS Code Generator routine that extracts information about a passed parse tree node's data type:

TYPE - Basic flavor (INTTYPE, HWTYPE, or CHARTYPE)
NUL - 1 if its an N (null) type
RS - 1 if its an RS type

It returns the DATATYPE of the datum described by the parse tree node, plus certain data obtained from the type.

ARIXC34

ARIXC34 is the SQL/DS Code Generator routine that puts out code to complement a number.

ARIXC35

ARIXC35 is the SQL/DS Code Generator routine that puts out code to fill a column with its new value for the UPDATE function.

ARIXC36

ARIXC36 generates code to move a value from one slot in real storage (the source) to another slot in real storage (the target). A parse tree describes the source and the target. It may be a real node in the parse tree or a pseudo-node constructed solely for the benefit of this routine.

ARIXC37

ARIXC37 is the SQL/DS Code Generator main (control) routine. It steps through the optimized parse tree and builds the syntactic stack entries. Then the entries are passed to ARIXC38 to be processed.

ARIXC38

ARIXC38 is the SQL/DS Code Generator routine that controls the generation of access module code based upon the operator in the syntactic stack passed by ARIXC37. It creates access module code from optimized data.

ARIXC39

ARIXC39 is the SQL/DS Code Generator routine that resolves references for branch instructions in generated code. It inserts into the CODEBLK the base and displacement portion of an instruction that references a symbol.

ARIXC40

ARIXC40 is the SQL/DS Code Generator routine that puts out a passed RR type instruction. It moves the RR instruction in INST into the next availalbe slot in CODEBLK.

ARIXC41

ARIXC41 is the SQL/DS Code Generator routine that puts out the passed RX type instruction. It moves the RX instruction in INST into the next available slot in CODEBLK.

ARIXC42

ARIXC42 is the SQL/DS Code Generator routine that puts out the passed SI type instruction. It moves the SI instruction in last two bytes of HW1, HW2 into an available CODEBLK slot.

ARIXC43

ARIXC43 is the SQL/DS Code Generator routine that puts out the passed SS type instruction. It moves the SS instruction contained in the three fullword parameters into the open slot in the CODEBLK.

ARIXC45

ARIXC45 is the SQL/DS Code Generator routine that sets up addressability to the length of a varying character string. P points to a parse tree node that describes a varying-length string. R is the register that points to the block containing the length. LOISP is the (byte) displacement from the beginning of where the length is to be found.

ARIXC47

ARIXC47 is the SQL/DS Code Generator routine that converts a pointer in block-number/displacement form to a real address.

ARIXC48

ARIXC48 is the SQL/DS Code Generator routine that converts block-number and displacement to a real storage address via the BLKADDR table.

ARIXC49

ARIXC49 is the SQL/DS Code Generator routine that breaks a pointer in block number/displacement form into two halfword fields (BLKNR,DISP). Then it returns them to the caller.

ARIXC51

ARIXC51 is the SQL/DS Code Generator routine that extracts descriptive data on a LITNODE. It supplies ARIXC03 with information about a LITNODE.

ARIXC53

ARIXC53 is the SQL/DS Code Generator routine that puts out code to convert a floating-point number from RSI format to standard S/370 format. Then ARIXC53 places the resulting number in the DSA.

ARIXC54

ARIXC54 is the SQL/DS Code Generator routine that puts out code to perform test and branch instructions.

This code works with LAB(25) and LAB(26) in ARIXC38.

ARIXEAB

ARIXEAB allocates the OP, space blocks, and the CODE BLOCK. It allocates the relocation directory for the block if the caller requests it.

ARIXEAD

ARIXEAD is the module called by ARIXERD to handle an EDSF (Extended Dynamic Statements Facility) PREPARE statement against a modifiable AUX. The functions performed vary slightly depending on whether or not the EDSF PREPARE follows a CREATE PROGRAM statement in its Logical Unit of Work (LUW).

When the EDSF PREPARE follows a CREATE PROGRAM statement, ARIXEAD initiates statement compilation and AUX section storing similar to ARIXEDS. The only difference is that each section's gencode or parse tree (in the case of interpretive sections) is kept around for the duration of the LUW in case it will be executed.

When not following a CREATE PROGRAM, ARIXEAD has an additional function which applies specifically to the first EDSF PREPARE of the LUW. The referenced AUX must be validity checked, loaded, and reprepped if necessary, and have needed storage areas allocated. ARIXEAD calls ARIXEPP to perform statement compilation.

ARIXEPP returns to ARIXEAD without storing the AUX section. ARIXEAD determines where the AUX section and its SQL statement will be stored in the AUX, then calls ARIXESX4 to do the store.

ARIXEAD then does address relocation in code blocks in case the section is executed during this LUW.

There are two additional entry points in ARIXEAD.

ARIXEAD1 is called at LUW COMMIT time. ARIXEAD1 calls ARIXESX5 to update the AUX length row and the AUX SLT rows. ARIXA06 is called to degrade run authorization, if necessary. ARIXEPH is called to mark all occurrences of the updated AUX invalid in the AUX cache. Finally, all storage areas are freed to include the code blocks (or parse trees) left around for possible execution.

ARIXEAD2 is called at LUW ROLLBACK time. It performs the same free storage functions as ARIXEAD1.

ARIXEBR

ARIXEBR is the linkage module between the RDS and an access module (AUX). ARIXERD calls ARIXEBR to give control to an access module (AUX). The AUX has already been loaded and resides in real storage.

ARIXECK

ARIXECK builds the SQLCA structure for DBSS errors. It is called whenever any RDS module has made a DBSI (ARIYM00) call to DBSS and has been passed back an error by DBSS.

ARIXECL

ARIXECL closes a set of cursors. It follows a chain of control blocks anchored in the OP pointed to by OPLOCPTR. If the control block status is not closed, and the SCANOPEN bit in the status bits structure indicates that the RSI scan is still open, DBSS closes the scan. The control block status is set to "CLOSEDCODE" in any event.

ARIXECW

ARIXECW is called by ARIXECK or by any RDS module that detects or causes a COMMIT WORK, ROLLBACK WORK, or a LINK BREAK. ARIXECW cleans up transactions and certain control blocks that the RDS uses, such as PROGS (list of loaded access modules), ACQUIRE blocks, and SORT blocks.

ARIXEDB

ARIXEDB is the RDS to DBSS linkage. It is called by many RDS modules whenever they wish to make a DBSI (ARIYM00) call to DBSS. ARIXEDB locates the address of the DBSS (ARIYM00) module. It sets Register 10 to point to YTABLE1 and BALR to DBSI (ARIYM00).

ARIXEDC

ARIXEDC is called by ARIXERR when ARIXERD discovers that the user issued a DESCRIBE call. This module copies the SQLDA from the SPACEBLK into the users SQLDA space. If the actual SQLDA is larger than the space, it only returns the actual

number of items. Otherwise, the entire SQLDA is copied to the user.

ARIXEDP - Dynamic Parse, Optimize, and Code Generate

This module is called from ARIXERD for three reasons:

1. To handle a set-up call created to handle a "PREPARE" SQL statement. This module causes the SQL statement, that is sent over at run-time, to be processed through the Parser, Optimizer, and Code Generator. ARIXEDP causes a section of an access module to be produced in real storage but not stored in the data base.
2. To handle indefinite sections produced by an "EXECUTE IMMEDIATE" SQL statement. This module processes the SQL statement sent over at run-time in the same manner as for a set-up call above. However, once processing is complete, the section is immediately given control and executed.
3. To handle parsed sections produced at PREP time. This occurs if a SQL statement could only be parsed and further processing is delayed until run time because a table or view did not yet exist or a user was not yet authorized. This module causes the statement to complete optimization and code generation and then executes the completed section.

ARIXEDR

ARIXEDR is the entry module for handling the deletion of an existing section in an access module. The section deleted is specified in the RDIIN field RDISECT# generated as the result if a DROP STATEMENT. ARIXEDR deletes the section and SQL statement rows of the specified section in the access module, and marks the SLT entry for the section as reusable (meaning it may be used by a subsequent add section request, that is, an extended PREPARE). TID (tuple ID) information is stored in each SLT entry which enables ARIXEDR to perform the row deletions. The TIDs of the first section and SQL statement rows are stored in the SLT fields used for IVAR and OVAR pointers, respectively. These fields may be used since IVARs and OVARs are not permitted in SQL statements prepared using extended PREPAREs. The row organization of an access module is:

1. AUX length row
2. SLT rows
3. Section rows
4. SQL statement rows.

Deletion is done in the following manner for both section row deletion and SQL statement row deletion:

1. A DBSS scan is opened to position on first row using supplied TID.
2. Row is deleted using DBSS delete.
3. DBSS NEXT is issued.
4. If TID of returned row is terminating TID, or if end of file return code is received, go to Step 5. Else, row is deleted using DBSS delete. Then go back to Step 3 to get next row.
5. Close scan using DBSS close.

Since AUX rows are stored using a unary link, row TIDs may be used to position on a specific row. When deleting the last section, stop at the TID of the first SQL statement row for section rows, and a DBSS EOF for SQL statement rows. The TIDs originally stored in the SLT entry of the deleted section are replaced by the first section/statement row TIDs of the next section. If section 2 were deleted its TIDs would be replaced by those of section 3. Updates to TIDS of any immediately preceding SLT entries marked as reusable must also be made.

ARIXEDS

ARIXEDS acts as the linkage between the Precompiler and SQL/DS. ARIXERD calls ARIXEDS whenever PREP issues one of its special SQL/DS calls - PREP INIT call, SQL call, LOOKUP call, and PREP FINISH call.

- The PREPINIT call causes control blocks to be allocated and initialized. These control blocks are used by other RDS modules during the PREP process. Also, call ARIXESX1 to initialize the access module.
- The LOOKUP call causes a search of the SLT (Section Location Table) to see if a section of the access module was previously created. It is the section used by the SQL statement currently being processed by PREP.
- The SQL call causes a call to ARIXEPP.
- The PREP FINISH call basically frees all areas created by a PREP INIT call. It also calls a routine (ARIXESX3) that finalizes the access module and causes the access module to be stored in the data base.

Modifications have been made that enable EDSF (Extended Dynamic Statements Facility) to handle CREATE PROGRAM calls, SETUP calls (extended PREPARE) and commit or rollback of a CREATE PROGRAM LUW. Output mailbox adjustments have also been made.

ARIXEER

ARIXEER is called whenever any RDS module encounters an error. It builds the SQLCA using the input parameters passed by the RDS module.

ARIXEFB

ARIXEFB frees an OP, space block, or code block along with their associated relocation directories. It frees either a single block or an entire chain of blocks.

ARIXELK

ARIXELK is the entry point to the RDS load module. It is called only once at SQL/DS startup time. ARIXELK finds the addresses of the key RDS modules. Then ARIXELK puts these addresses into the RDS CVT so SQL/DS control can use them later.

ARIXELX

ARIXELX loads an access module from the data base into storage. ARIXERD calls ARIXELX when it finds that the load module is not yet in storage. ARIXERD also calls ARIXELX if the load module was deleted from storage since it was last used.

Entry Point: ARIXELX1
Function: ARIXERP calls ARIXELX1 to find SQL statements on the data base.

Entry Point: ARIXELX2
Function: ARIXELX2 is required for UPDATE LUW initialization. It enables the SLT to be loaded into the PSLT extendable blocks, the AUX sections to be loaded and to point to each section out of PSLTLOCP of its PSLT entry.

When fetching the first row of an AUX, the possibility that it might be a CREATE PROGRAM AUX and have a longer first row must be handled. The information in the first row must be saved in the store load structure and the RDAREA.

When loading the AUX SLT, new information for CREATE PROGRAM access modules must be transferred from the stored SLT to the AUX SLT.

Entry Point: ARIXELX3
Function: ARIXELX3 is called by ARIXEAD at start of AUX update activity to see if the AUX exists, if it is modifiable, and if it needs reprepping.

ARIXEOC

ARIXEOC provides the linkage between the RDS module that receives control when an operator command is received from the CICS partition (ARIXERD) and the module that makes the call to the DBSI (DBSS linkage module ARIYM00). It converts the RDIIN structure that contains the operator command to OBASE. OBASE is the linkage structure between RDS and DBSS.

ARIXEPH

ARIXEPH performs the following operations on PROGS (list of loaded access modules):

- Moves all entries from RDAREA to the free list, blank list, or extension list
- Invalidates/deletes all entries from PROGS for a specific program name
- Moves entry from anywhere to the blank or extension list
- Moves lsat entry from the free list to the blank or extension list
- Moves all entries from the free list to the blank or extension list
- Disconnects PROG from the user list directory for a specific PROG and removes user list if needed
- Moves invalid PROGS to blank or extension list.

ARIXEPP

ARIXEPP acts as a router to the major RDS components: Parser, Optimizer and Code Generator. It receives control from ARIXERD, allocates the IVNAMES, IVIND, OVNAMES and OVIND data areas, and calls ARIXPA1 (Parser). Upon return from ARIXPA1 with a good return code, ARIXEPP allocates the NAMELIST data areas and builds them using the information passed back from ARIXPA1. It then calls ARIXOOP (Optimizer). If the Optimizer passes back a valid return code, ARIKEPP calls ARIXC37 (Code Generator). If everything is o.k., it calls ARIXESX2 to store the section of the access module that was created.

At appropriate points in the program, checks are made to determine if an extended PREPARE is currently being handled (RDACPLUW is on). An SQL statement cannot have input or output host variables; only question mark variables are allowed. All SELECTs are treated as cursor queries. For question mark variables, data type and length are obtained from an SQLDA, if present. The DESCRIBE argument (3D) is set ON for calls to the optimizer for queries so that an SQLDA will be created in the space block. After storing a

section the chain of space blocks for a compile section of a modifiable AUX is not freed.

ARIXERD

ARIXERD is the topmost module in the RDS. It is the first module to be called from the SQL/DS Dispatcher for all normal SQL/DS calls from a user. Its main purpose is to act as a router between the call type in the RDIIN and the RDS module which handles the particular call type. It also checks the users authorization, insures the user is properly scheduled, and causes the access module to be loaded during execution time. ARIXERD maintains a list of access modules that have been loaded and a list of userids authorized to use these access modules. If the access module is on the list and the user is on the authorized list for the access module, no further checking is done.

Additionally, EDSF can handle the extended SQL statements which are CREATE PROGRAM, extended PREPARE, extended DECLARE CURSOR, extended OPEN, extended FETCH, extended CLOSE, extended DESCRIBE, and extended EXECUTE. These statements may be handled prior to, and/or after loading an AUX. Input mailbox handling has been modified to accommodate changes required for support of the extended statements. Code has been inserted to detect and handle commits and rollbacks of CREATE PROGRAM LUWs and LUWs that update an existing AUX (update LUW). In addition, code has been inserted to detect implicit rollbacks within one pass through RDS (between entry and exit of ARIXERD).

A new entry point, ARIXERD1, has been added to handle normal and abnormal application termination. For the normal termination case, entry is made if the application did not end the LUW by issuing a commit or rollback.

ARIXERO

SQL/DS control calls ARIXERO when SQL/DS is being brought up. ARIXERO reads the catalog addresses of all the system catalogs. These addresses are located in the first relation of Segment 1 of the data base.

ARIXERP

ARIXERD calls ARIXERP for REPREP if the access module for the program being executed is marked invalid. ARIXERP then calls ARIXELX1 to retrieve the SQL statements from the data base where they were stored during PREP time. ARIXERP then calls ARIXEPP, that calls the Parser, Optimizer, and Code Generator. This causes a new section to be stored for each SQL statement. It calls ARIXESX3 to finalize the new access module.

ARIXESP

ARIXESP issues an ENDRSS DBSI (ARIYM00) call to stop an agent.

ARIXEST

ARIXEST is called one time for each agent created by SQL/DS Control. ARIXEST allocates RDS SQLCA and TBASE, and initializes RDAREA.

ARIXESX

ARIXESX is the store access module routine.

Entry Point: ARIXESX1
Function: ARIXESX1 initializes the AUX. It is called when a program is first being prepped. ARIXESX1 issues a BEGIN TRANSACTION, that covers the entire PREP process. ARIXESX1 also causes any old access module by the same name to be dropped. Then ARIXESX1 allocates a new access module.

Entry Point: ARIXESX2
Function: ARIXESX2 adds section and SQL statement rows to an AUX which is not modifiable.

Entry Point: ARIXESX3
Function: ARIXESX3 finalizes an AUX which is not modifiable or finalizes a null modifiable AUX (contains no sections). The SLT rows of the AUX are added, then the AUX length row (first row) is updated to reflect the number of sections in the AUX. An entry for this AUX is inserted into the SYSACCESS catalog. The AUX is then committed to the data base. No finalization occurs if an error was encountered during preprocessing; the prep LUW is rolled back.

Entry Point: ARIXESX4
Function: ARIXESX4 adds section and SQL statement rows to an AUX which is modifiable.

Entry Point: ARIXESX5
Function: ARIXESX5 finalizes an AUX which is modifiable. Existing AUX SLT rows are updated or inserted,

as necessary. The AUX length row is updated. An entry is made for this AUX in the SYSACCESS catalog, if necessary.

ARIXEUH

ARIXEQU performs the following operations on user list and user list directory:

- Deletes user from a user list for a specific program
- Destroys old user list and creates a new one, leaving count in user directory unchanged.

ARIXETR

ARIXETR is the RDS trace service routine.

Entry Point: ARIXETR
Function: ARIXETR causes trace point data to be collected and written to the SQL/DS trace output file. This function is invoked by the XTRACE macro with the LOCK option omitted.

Entry Point: ARIXETR1
Function: ARIXETR1 causes trace point data to be collected and written to the SQL/DS trace output file. It also leaves the trace output file 'locked' (see below) so that the caller, itself, can create additional trace point data records and cause them to be written to the trace output file. This function is invoked by the XTRACE macro with the LOCK option specified.

Entry Point: ARIXETR2
Function: ARIXETR2 causes the SQL/DS trace output file to be 'unlocked'. This function is invoked by the XTRACE macro with the UNLOCK option specified.

Entry Point: ARIXETR3
Function: ARIXETR3 causes caller-created trace point data records to be written to the trace output file. See entry point ARIXETR1 above. This function is invoked by a call with the address of the record as the parameter.

On normal entry, entry point ARIXETR and ARIXETR1, tests are made to determine if trace is:

- Active at all
- Active for the current user-id
- Active for the invoking subcomponent or function

If the trace is not active, return with no output.

If a snap dump was requested (by the TRACE command) for this trace point, the SQL/DS partition snap dump routine is called and the snap dump request is disabled (dump first time only).

RDS CODE FRAGMENTS

ARIXFnn

The ARIXFnn modules are code fragments that may be inserted into access module sections by the Code Generator, depending on which SQL functions are requested.

ARIXFA0 - Calls the run-time routine ARIXCR3 to insert a literal row containing long fields.

ARIXFTB - Contains the vector table for fragments and run-time routines. These fragments are used by the Code Generator when building an access module.

ARIXF01 - Does the initialization for the outer query subroutine. Set up so this section looks like compiled code from an RDS program.

ARIXF02 - Tests for OPEN or a NEXT.

ARIXF03 - Does an OPEN.

ARIXF04 - Issue the NEXT.

ARIXF05 - Tests the row against the WHERE clause. If the row did not satisfy the WHERE clause, we go back to AGAIN to issue another NEXT.

ARIXF06 - Sets register 2 with first PVAR in user's PVAR structure. This block of code precedes the outputting of values from the select-list. It fills register 2 with the address of the first PVAR in the user's PVARS_STRUCT, and calls ARIXCR2.

ARIXF07 - Returns control to the user. This is the last fragment. The Code Generator is all done and ready to return control to the user. Gives him a zero return code if control floued into here, and then uses the RDS standard return sequence. This fragment also contains the error exit.

ARIXF08 - Checks the truth value for "FALSE or NULL".

ARIXF09 - Checks if the truth value was a "TRUE".

ARIXF11 - Exits for UPDATE, DELETE WHERE <predicate>. Calls run-time routine ARIXCR1.

ARIXF12 - Obtains a userid. Finds the address of the userid slot in the RDAREA, and leaves that address in register 6.

ARIXF13 - The entry for an inner-level query routine.

ARIXF14 - Returns "not found" to the caller immediately. This fragment is output in a "BINDTABLE" situation (for example, for join).

ARIXF15 - Uses ARIB016 to set the loop function.

ARIXF16 - Initializes the set function's INITBYTE. This fragment is output after the processing of the select-list of the query that includes set functions.

For each query involving set functions, there is a byte allocated in "BLOCK 10", our static CSECT data block, which indicates the status of the immediate environment: whether the SETFNDATA slots have been initialized, and whether a row has been fetched and accumulated.

ARIXF17 - Fragment for "OPEN" the cursor query with set function.

ARIXF18 - Fragment for "NEXT" on cursor query with set function. This fragment is entered when we have a cursor query involving a set function, and the user has issued a $FETCH against the cursor.

ARIXF19 - Exits from cursor query with set function. This is the last fragment to be output for a cursor query that involves a set function. We flow into here when all of the rows have have been fetched, EOF has been reached, and the set function has been computed. Because everything is OK at this point, and ARIXEBR initialized the return code structure to "OK", we simply return to the user.

WINDUP is another valid entry, which is reached when we find the flag that indicates that we should return the "not found" condition to the the user. Close the scan (if it is open), and close the user's cursor.

ERROR is entered in response to a severe error anywhere in the section.

ARIXF20 - This fragment issues the UPDATE command for a row identified by the "current of cursor" predicate. Control is returned to the caller if the UPDATE command succeeds; otherwise, control is transferred to ARIXCR1.

ARIXF21 - Fragment for WHERE clause of "DELETE ... WHERE CURRENT OF CURSOR"

ARIXF22 - Issues an RSI FETCH when everything is already known.

ARIXF23 - Fragment that moves the SCANID from one TBASE to another.

ARIXF24 - The exit for select one row for a non-cursor SELECT.

ARIXF25 - This fragment issues the RSI "CLOSE" command against the TBASE pointed to by the "NEXTPTR" of the control block.

ARIXF26 - Tests the RSI return code. If the return code is not zero, this is an unexpected error and ARIXF26 calls ARIXCR1 to handle it.

ARIXF27 - Issues the RSI INSERT command to insert a literal row.

ARIXF28 - Fragment for "OPEN" and "NEXT" in support of INSERT from subquery and UPDATE ... WHERE ... and ... DELETE ... WHERE.

This fragment is to open a scan, and then return rows on demand. The difference between ARIXF28 and a combination of ARIXF03 and ARIXF04 is that the fragment does not return to the user after the open, but plunges right in with a fetch of the first row.

ARIXF29 - Fragment for INSERT from subquery with clustering by default. This fragment issues the RSI INSERT for a row that has been returned from a subquery. Control is returned to the caller only when the subquery reports an EOF condition.

The ARIXF29 fragment is used, instead of ARIXF49, in the case where there is no clustering image, and clustering is by defalt. The difference is that here we leave the TID returned by the last insert as the candidate TID for the next.

ARIXF30 - Fragment for DELETE ... WHERE. Control is returned to the caller only when the subquery reports an EOF condition.

ARIXF31 - Fragment for processing we're all done.

ARIXF32 - Calls ARIXCRH to bind a decimal number.

ARIXF33 - Calls ARIXCRI for decimal to floating-point conversion.

ARIXF34 - Calls ARIXCRJ for floating-point to decimal conversion.

ARIXF35 - Fragment to process a "NOT" node. register 9 currently contains either the value "1"("TRUE"), "0"("NULL"), or "-1"("FALSE").

ARIXF36 - Fragment to reset "NO ROWS" flag for set function.

ARIXF37 - Fragment to set "have seen a row" flag. This fragment is output after the processing of the select-list of a query that incldes set functions.

ARIXF38 - Fragment to accumulate the minimum varying length character string that has been seen. If the datum in hand is less than the current minimum value, the datum becomes the new current minimum.

ARIXF39 - Fragment to accumulate the maximum varying-length character string that has been seen. If the datum in hand is less than the current maximum value, the datum becomes the new current maximum.

ARIXF40 - Fragment to accumulate COUNT and SUM for integers. Called from ARIXC28.

ARIXF41 - Fragment to compute average for SUM and COUNT.

ARIXF42 - Test bit 0 of SETSTATE to see whether the set function ignored nulls; if it did, set the NULLSIG bit of RDSCODE.

ARIXF43 - Set null indicator for AVG, MIN, MAX.

ARIXF44 - Fragment to call run-time routine ARIXCRL which tests a string to determine whether it is a "LIKE" pattern for DBCS.

ARIXF45 - Fragment to output a character string of either fixed- or varying-length. It calls upon the run-time routine ARIXCR3 to actually do the transfer to the user's variable location.

ARIXF46 - Fragment to call the run-time error routine ARIXCR1. Load register 0 with an error code. Load register 1 with additional information. Find the address of ARIXCR1 and branch to it.

ARIXF47 - Fragment for WHERE clause of "UPDATE ... WHERE CURRENT OF CURSOR".

ARIXF48 - Issues the RSI UPDATE command for a row that has been returned from a subquery. Control is returned to the caller only when the subquery reports an EOF condition.

ARIXF49 - Issues the RSI INSERT command for a row that has been returned from a subquery. Control is returned to the caller only when the subquery reports an EOF condition. ARIXF49 is used when there is a clustering image. It differs from ARIXF29 in that the candidate TID is made zero so that clustering by image will take place.

ARIXF50 - Binds an input variable to a fullword slot. It calls the run-time routine ARIXCR5.

ARIXF51 - Binds an input variable to a halfword slot. It calls the run-time routine ARIXCR6.

ARIXF52 - Binds an input variable to a character slot. It calls the run-time routine ARIXCR7.

ARIXF53 - Binds an input variable to a varying character slot It calls the run-time routine ARIXCR8.

ARIXF54 - Moves TID from one TBASE to another.

ARIXF55 - Fragment for Step1 query -- simple branch to AGAIN.

ARIXF56 - Defines label "INITSTFN".

ARIXF57 - Fragment to close scan(s) for non-cursor join.

ARIXF58 - Re-initializes the SYR structure if it contains a "not found" code. It is invoked in cases where a subquery has reported "not found" in both the SYR structure and register 15, and we don't really want to report "not found" to the user.

ARIXF59 - Calls the run-time routine ARIXCRB to get temporary storage.

ARIXF60 - Sets up register 15 for OPEN check list procedure.

ARIXF61 - Calls ARIXCRA to call DBSS to perform a sort.

ARIXF62 - Fragment for a join: test call type. This is the first fragment after the prologue in a compiled subroutine. It decodes the command (a single character, pointed to by register 1, and branches accordingly.

ARIXF63 - Calls the run-time routine ARIXCRC, which calls the DBSS to initialize a temporary segment and to create a list or relation in it.

ARIXF64 - Fragment to assist evaluation of <expr> in <subquery>.

ARIXF65 - Fragment to call a compiled subroutine to perform a join via method 1.

ARIXF66 - Define the external label "GREXIT".

ARIXF67 - Defines the label "INITSTFN" for "GROUP BY".

ARIXF68 - Uses macro ARIB014 to issue the "NEXT" for "GROUP BY".

ARIXF69 - Uses the macro ARIB015 to issue the "OPEN" for the "GROUP BY".

ARIXF70 - This is the last fragment to be output for a "GROUP BY". We flow into here when all of the rows have been fetched (EOF has been reached). Because everything is OK at this point, and ARIXEBR initialized the return code structure to "OK", we simply return to the user.

ARIXF71 - Uses macro ARIB016 to generate code for the loop function with "GROUP BY".

ARIXF72 - This fragment's sole job in life is to transfer to the label "AGAIN." This is done when we are processing an INSERTNODE that is not at the highest level (i.e., we are building a temporary relation), and we discover that the source value is NULL and the corresponding column in the temporary relation does not permit nulls.

ARIXF73 - This fragment follows the ARIXF29/ARIXF49 fragment. It branches back to get another group. This branch has been separated from ARIXF29/ARIXF49 because it is different for INSERT a GROUP BY.

ARIXF74 - This fragment follows the ARIXF29/ARIXF49 fragment. It branches back to read another row. This branch has been separated from ARIXF29/ARIXF49 because it is different for INSERT from a GROUP BY.

ARIXF75 - Fragment to advance (close, re-open) the PLACEHOLDER scan.

ARIXF76 - Fragment to return "not found" code to user, both in SYR.RDSCODE and in register 15.

ARIXF77 - Fragment that will perform a "NEXT" on the actual scan.

ARIXF78 - Fragment that will return "not found" if RHS > LHS join column.

ARIXF79 - Initializes switch for Type 2 joins.

ARIXF80 - Calls RHS to open, fetch, or advance.

ARIXF81 - Test call type, entry via join Method-2. This is the first fragment after the prologue in a join subroutine. There are five possible single-character command codes that register 1 may point to: O (OPEN), F (FETCH), C (CLOSE), R (REOPEN), and A (ADVANCE).

ARIXF82 - Open two scans on a relation that is on the right-hand side (lower level) of a Type-2 join.

ARIXF83 - Fragment to do a "NEXT" on the LHS relation in a Type-2 join.

ARIXF84 - Loop for HAVING ... in INSERT. This fragment follows the computation of HAVING. It branches back to get another group if the last one does not satisfy the HAVING clause.

ARIXF85 - Loop back after HAVING computation. This fragment follows the computation of HAVING. It branches back to get another group if the last one does not satisfy the HAVING clause.

ARIXF86 - Call ARIXF28 for LHS of Type-2 join. Fragment opens a scan, and then returns rows on demand. The difference between ARIXF86 and a combination of ARIXF03 and ARIXF04 is that the fragment does not return to the user after the open, but plunges right in with a fetch of the first row.

ARIXF87 - Sets loop function (uses macro ARIB016) for join Method-2.

ARIXF88 - Sets loop function (issues macro ARIB016) for join Method-2 with "GROUP BY".

ARIXF89 - Initialize "USQFLAG" (for $SELECT), the halfword slot that we will use to tell whether a select-without-cursor returned 0, 1, or more-than-1 row.

ARIXF90 - Increment "USQFLAG". Add 1 to the slot that counts how many rows we have seen (in the case of a select-without-cursor), and to branch to the exit code if the result exceeds 1.

ARIXF91 - Set to 1 the slot ("USQFLAG") that counts how many rows we have seen (in the case of select-without-cursor, and the query is a sub-query of an IS-IN or IS-NOT-IN predicate). When we reach ARIXF24, the slot will contain 0 if no rows have qualified, and 1 if 1 or more have qualified.

ARIXF92 - Set NULL byte for MIN and MAX set functions.

ARIXF93 - Output a floating-point number. It calls upon the run-time routine ARIXCR2 to actually do the transfer to the user's variable location.

ARIXF94 - Call the run-time routine ARIXCRE to bind an input variable to a floating-point slot.

ARIXF95 - Convert a fullword integer to floating-point format. The integer to be converted is in GPR (0); the resulting floating-point equivalent is left in (XXX) and in FPR (0).

ARIXF96 - Call the run-time routine ARIXCRF, which tests a string to determine whether it is "LIKE" a pattern.

ARIXF97 - At Bind time, checks to be sure that the user has supplied exactly as many input program variables in his PVARS_STRUCT as are needed - if not, generate an error code.

ARIXF98 - We output this fragment after testing the ISIN / ISNOTIN / EXIST part of the predicate. Reg (9) is the TRUE / FALSE indicator: 1 => TRUE, 0 => null, -1 => FALSE. If the row did not satisfy the WHERE clause, we go back to TSTQUALF to get another row. We do not go back to AGAIN because we may be in a join situation. AGAIN would get us another row from the left hand side, and if there is another row on the right hand side then we need that one.

ARIXF99 - Turn on the "GROUP1" status bit in the control block for a scan. This bit is turned on to indicate that the DBSS scan needs to be opened; normally this is done by ARIXF17, but that fragment is produced only for cursor queries. For non-cursor queries, we call out this fragment before ARIXF15 (or its Method-2 join cousin).

RDS INTERPRETER MODULES

ARIXISH

ARIXISH is called to calculate hash values for non-indexed columns during update statistics.

ARIXIST

ARIXIST collects statistics for non-indexed columns.

ARIXI01

ARIXI01 acquires DBSPACEs. The basic steps are:

1. Check that the DBSPACE name does not already exist for user.
2. Find a available DBSPACE in SYSDBSPACES.
3. Do a CINSERT to initialize the DBSPACE. Get NHEADER, PCTFREE, PCTIMAGE, and LOCKMODE from SEGSPECNODE.
4. Set owner = 'PUBLIC ' for Type-1 DBSPACEs. Set NRELS = 0. Update SYSDBSPACEs.

ARIXI03

ARIXC03 changes LOCKMODE and/or PCTFREE for specified DBSPACE. The basic steps are:

1. Fetch from SYSDBSPACES for appropriate DBSPACE.
2. Do a CFETCH to get SCR row. This is needed because you can only specify all fields on an update. Fix up the PCTFREE field or the ENTLOCK field. Do a CUPDATE.
3. Update the PCTFREE and lock the fields in the SYSDBSPACES.

ARIXI04

ARIXI04 inserts comments in SYSCATALOG or SYSCOLUMNS. The basic steps are:

1. Fetch the comments from the appropriate catalog.
2. Update the row. The old comment, if any, is erased.

Entry Point: ARIXI03

ARIXI06

ARIXI06 processes a CREATE INDEX. The basic steps are:

1. Check if the index name exists for this user.
2. Do a fetch from SYSCATALOG for the table. Pick up the SEGNO and RID.
3. Build an ICT for the new index by scanning the column nodes. Also build parts of the row for SYSINDEXES.
4. Finish building the row for SYSINDEXES. Then insert it as the tail child on INDXLINK.

ARIXI07

ARIXI07 creates a relation. The basic steps in creating a relation are:

1. Check if the table name exists for user.
2. If the DBSPACE is specified, check that it exists. Otherwise, get (first) existing Type-2 DBSPACE of the user.
3. Insert the master control row (MCT) for the new relation. This involves scanning COL NODE LISR to get columns lengths. During scan, build the structure COLUMNINFO that remembers if nulls are allowed and stores other items used later to build entries for SYSCOLUMNS. Increment NRELS in the SYSDBSPACES row by one and update. Create companion relation and unary link of long fields.
4. Insert the entry for a relation in SYSCATALOG.
5. Insert the entries for columns in SYSCOLUMNS. Connect them on COLLINK.

ARIXI08

ARIXI08 creates a synonym. The basic steps in creating a synonym are:

1. Do a fetch from SYSCATALOG for the synonym to insure it is not already in use as a table (or view) name.
2. Insert the new synonym in SYSSYNONYMS.

Entry Point: ARIXI03

ARIXI09

ARIXI09 creates a view. The basic steps in creating (defining) a view are:

1. Check if the table name exists for user.
2. Insert the entries for authorization.
3. Build rows for SYSVIEWS catalog.
4. Insert the entry in SYSCATALOG for view.
5. Build and insert the entries in SYSCOLUMNS.
6. Insert the entries in SYSUSAGE for dependencies.
7. Create and store AUX for the view. Use the store program that is used for regular access modules.

ARIXI10

ARIXI10 is the top module for dropping (depending on value of DROPNODE) one of the following objects: index, table, view, DBSPACE, synonym, program, and AUX (access module). This module will call the appropriate module for the object to be dropped.

Entry Point: ARIXI10
Function: ARIXI10 is the main entry.

Entry Point: ARIXI101
Function: ARIXI101 is the entry to drop AUX.

Entry Point: ARIXI102
Function: ARIXI102 is the entry from the REVOKE routine. It drops or invalidates AUX or a view.

ARIXI12

ARIXI12 executes COMMIT WORK.

ARIXI13

ARIXI13 alters a table. The basic steps to alter a table are:

1. Do a fetch from SYSCATALOG for the table. Pick up the SEGNO and RID. Increment the NCOLS field by one.
2. Do a CFETCH for a MCT. Add the length for the new field and do a CUPDATE. Create the companion relation for long fields if needed. Update SYSCATALOG row.
3. Open the link scan on COLLINK using the row from (1) as the parent. Do a NEXT with a SARG of COLNO: number of fields previously in the table to position on the entry for the last field in SYSCOLUMNS.
4. Build the new SYSCOLUMNS entry and insert it (also connect it to COLLINK). Grant update to userid if the catalog expanded.

ARIXI14

ARIXI14 is the top module for the Interpreter component. Depending on NODETYPE (operation), ARIXI14 calls corresponding module.

ARIXI15

ARIXI15 locks the DBSPACE or a table. The basic steps in locking a DBSPACE or table are:

1. Fetch id's from SYSCATALOG or SYSDBSPACES. Make sure the table is not a view.
2. Lock the call.

ARIXI16

ARIXI16 executes ROLLBACK WORK.

ARIXI18

ARIXI18 scans SYSDROP and CDELETES all the entries. It also resets the RDADROP bit.

ARIXI19

ARIXI19 updates statistics. It first does a DBSS unload cycle on the given table. Then ARIXI19 does an additional unload cycle for each extra index on that table.

ARIXI20

ARIXI20 updates statistics for the RDS system tables. It also updates the clustering-related entries in the catalogs.

ARIXI22

ARIXI22 drops an index.

ARIXI23

ARIXI23 drops a table.

ARIXI24

ARIXI24 drops a view.

ARIXI25

ARIXI25 drops the DBSPACE.

ARIXI26

ARIXI26 drops a synonym.

ARIXI27

ARIXI27 executes a CDELETE for index. For other objects, an entry is made in SYSDROP.

ARIXI28

ARIXI28 deletes all rows on link RBLID from SYSCATALOG to CATNAME.

ARIXI29

ARIXI29 scans SYSUSAGE for the base object using IMUSAGE. It deletes or invalidates all dependent objects except synonyms.

ARIXI30

ARIXI30 drops all rows for the AUX. If AUXSTT1 is 'I', then invalidate the AUX. If it is 'D' or 'K', then delete the AUX. For AUXSTT1 = 'I' or 'K', call AUTH BOX with code 'Q' to indicate that RUNAUTH is not to be erased.

ARIXI31

ARIXI31 drops all SYSUSAGE rows where an argument is dependent.

ARIXI32

ARIXI32 removes all catalog rows for specified tables, except those in SYSACCESS and SYSSYNONYMS. It calls ARIXI28 to execute it.

ARIXI33

ARIXI33 performs update statistics for a table or a DBSPACE. For a table, ARIXI19 is called once to update statistics for the table. For a DBSPACE, ARIXI19 is called to update statistics once for each table in the DBSPACE.

RDS LINK MAP MODULES

ARIXLNK

ARIXLNK is the RDS link map module for both VSE and VM. ARIXLNK may be found in a dump, and shows the module name and corresponding link address of each RDS module.

For VM only: ARIXLNK contains the first 200 RDS module names and link addresses. The remaining entries are in ARIXLN1 (255 is the maximum allowable number of VCONS in one text module).

ARIXLN1 (VM Only)

ARIXLN1 is part 2 of the RDS link map module, and contains entries after the first 200 that were found in ARIXLNK. ARIXLN1 may be found in a dump, as can ARIXLNK.

Optimizer Data Conversion

The Optimizer converts literals in SQL statements to necessary data type for optimal path selection as well as providing slots in which data will reside at run time.

The macro ARIBRDS contains the format of the various data types. Optimizer module ARIXOTF handles actual data conversion and building of data slots in the space block.

Certain peculiarities should be noted in the DBSS representation of certain data types.

- Long VARCHAR - A long field resides in a row as a 6 or 7 byte field consisting of:
  - 8 bit null indicator if null-accepting
  - A halfword containing the length of the long field
  - A TID (Row ID) of the first row of the auxiliary relation which contains the data. Each row in the auxiliary relation consists of 16 fields of 250 bytes each. Therefore, one row may represent a maximum of 4000 bytes of data. Rows are chained together on a unary link.
- Halfwords and Fullwords - The high order bit of the numeric representation is inverted. For example, a HW value of one would be X'8001', and the HW value of a negative one would be X'7FFF'.
- Decimal - The length of a decimal field is (precision / 2) + 1. The high order four bits contain a complement representation of the sign as it existed in S/370 packed format. The remaining bytes are the actual number in packed format. A negative number is represented as the complement of its packed format.
- Float - A positive float value is represented in its internal S/370 format with the high order bit set on. A negative floating number is stored as the complement of its S/370 format.

Decimal Arithmetic Operation Formulas

The following describes the formulas used in ARIXOOS and ARIXOEX to calculate the precision and scale of the result of a decimal arithmetic operation.

If the arithmetic result of a select-list or predicate arithmetic expression is to be decimal, then the precision and scale of the result is calculated according to the following formulas and placed in PTREEP2 of the ARITHNODE for later use by the Code Generator. (Note: (P,S) is precision and scale of the left-hand side of an arithmetic operator; (P',S') is the precision and scale on the right.)

- Addition and subtraction
  Precision:
  MIN(15,MAX(P-S,P'-S')+MAX(S,S') + 1
  Scale:
  MAX(S,S')

- Multiplication
  Precision:
  MIN(15,P+P'+1)
  Scale:
  S+S'

- Division
  Precision:
  15
  Scale:
  15-P+S-S'

- Sum of decimal expression (P,S)
  Precision:
  15
  Scale:
  S

- Average of decimal expression (P,S)
  Precision:
  15
  Scale:
  15-P+S

Basic Optimizer Cost Formula

The objective of the optimizer is to minimize the cost of accessing data to answer the SQL request. The cost is a combination of I/O cost plus CPU cost.

Access Cost = I/O Cost + CPU Cost * Weight Factor

I/O Cost is expressed as the expected number of page fetches required to handle the request.

CPU Cost is expressed as the number of calls to the DBSS that are required to handle the request.

The Weight Factor relates the relative importance of I/O Cost to CPU Cost for the installation. A high Weight Factor says CPU is the critical resource. A low Weight Factor says I/O is the critical resource. The current Weight Factor used by the Optimizer is 1/3 (obtained from RDCDPATH * RDCRATIO).

Input to Optimizer: Path Selection/Cost Formulas from Catalogs

Catalog	Catalog Column Name	Optimizer Table Taken From	Variable Name in ARIXOGC, if different	Default Value
SYSCATALOG				
	ROWCOUNT	TBACARD	R	10 000
	NPAGES	TBANPAGS	T	CEIL (R/20 + 1)
	PCTPAGES	TBAPCTPG	---	---
	NCOLS	CATNUMCL	---	---
SYSINDEXES				
	INDEXTYPE	IDXUNQIN	---	---
	CLUSTER	IDXCLIDX	---	---
	FULLKEYCOUNT	IDXFULKC	ICARD	10
	NLEAF	IDXNLEAF	L	---
	NLEVELS	IDXNLVLS	---	---
	FIRSTKEYCOUNT	IDXFSTKC	ICARD	10
	COLNUMBERS	IDXCLNOS	INDXCOLS	---
SYSDBSPACES				
	PCTINDX	TBAPCTIX	----	---
	NPAGES/128	TBANBLKS	---	---
	NTABS	TBANRELS	---	---
SYSCOLUMNS				
	COLCOUNT	CATFLDCD	---	---
	COLNO	CATCOLNO	---	---
	HIGH2KEY	CATHIGH2	---	---
	LOW2KEY	CATLOW2	---	---

Other assumptions:

- For segment scan cost (SEGSNCST), assume Data from this table occupies 1/100 of pages in DBSPACE if statistics don't exist (exclude Data from other tables, indexes, blocks, etc)
- Assume multi-programming level of 3 for sharing the buffer pool.

Set-up Formulas for Optimizer Cost Calculations (in ARIXOGC)

The following calculations are done as initialization and are used in the cost formulas listed below.

- Calculating the number of block pages for the DBSPACE not counting index blocks:

  RB = CEIL ((100 - TBAPCTIX) * TBANBLKS)) / 100)

  (TBANBLKS is the number of blocks pages for the entire DBSPACE = (total pages in DBSPACE)/128)

- Calculating the number of non-leaf pages (intermediate index pages):

  NONL = FLOOR L/200

- Calculating the number of intermediate levels in an index between root and leaf pages:

  NLEVD = MAX (0,INDXLVLS - 2)

- Calculating the number of index block pages (block pages to describe the index pages):

  IB = (L + NONL) / 128

- Calculating the number of data block pages if data is clustered:

  IDB = CEIL T/128

- Calculating the number of data block pages for non-clustered data:

  NIDB = ICARD + CEIL (fpages (R/ICARD,IDB))

  (the fpages function is described later)

Optimizer Cost Formulas (in ARIXOGC)

The following formulas are used by the Optimizer to estimate the I/O costs to scan a table using each type of access path. The resulting costs are added to the CPU cost and compared to find the cheapest overall path and the cheapest path that the returns the answer set in an order that matches an ORDER BY, GROUP BY, or join predicate.

Segment Scan (SEGSNCST - variable name in ARIXOGC):

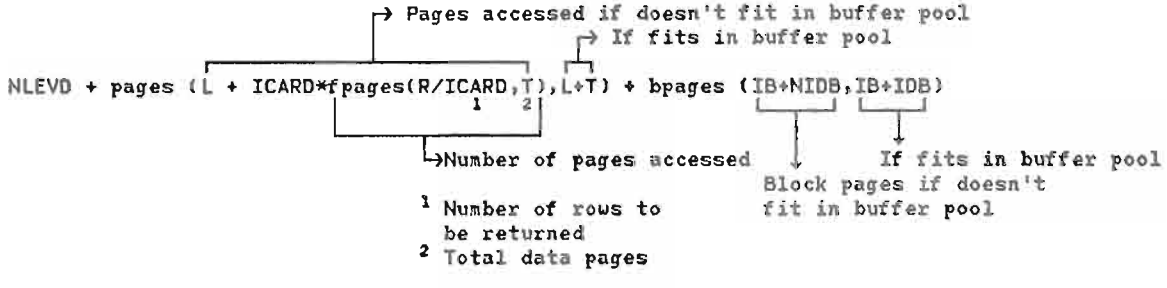
$$\frac{T * 100}{TBAPCTPG} + RB$$

(number of data pages for DBSPACE + data block pages for DBSPACE)

Clustered Index Scan of Whole Table (CLUSCOST - variable name in ARIXOGC)

L + NLEVD + IB + T + IDB (leaf + intermediate index levels + index blocks + data pages + data blocks)

Non-clustered Index Scan of Whole Table (NCLUSCST) (Only keep the best unclustered index path)



(fpages, pages, bpages functions are described below)

Unique Index and Matching Predicate (BESTCOST)

4 + NLEVD
(leaf + data + 2 block pages + intermediate index pages)

If all selected fields are in the index and no SARGs, then get data from index without reading data/data block: 2 + NLEVD

Clustered Non-unique Index Matching Predicate (ACPTHCST)

CEIL (L*FF) + CEIL (T*FF) + CEIL (IDB*FF) + CEIL (IB*FF) + NLEVD

(leaf pages + data pages + block data pages + index block pages + intermediate index pages adjusted for selectivity (FF) of predicate(s) - see section on "Selectivity of Predicates")

Non-clustered Index Matching Predicates (ACPTHCST)

NLEVD + pages [CEIL(L*FF) + CEIL(FF*ICARD*fpages(R/ICARD,T)),CEIL(L*FF + T)]
+ bpages[CEIL(IB*FF) + CEIL(FF*ICARD*fpages(R/ICARD,IDB)), CEIL (IB*FF) + IDB)]

(intermediate index pages + data + index + data block + index block adjusted for buffer size and selectivity (FF))

Functions Used in Optimizer Formulas

pages - Estimate number of data pages accessed, given the size of the buffer pool

bpages - Estimate number of block (data block) pages accessed, given the size of block buffer pool

fpages - Input: X = number of rows to be returned from
Y = total number of pages selected from
Output: Expected number of pages accessed based on a normal distribution and probability formula
Y*(1-(1-1/Y)**X)

Sort Cost Formulas (ARIXOSS)

Sorting may be done to produce a result for an ORDER BY or GROUP BY, or to match a join predicate if no index is available, or if using the index is relatively expensive. The cost of sorting is calculated using the following formulas.

Input

NSORTIN - Number of items to be selected
COSTIN - Cost of accessing input
SRTDIRCT - ON if input is from a permanent table
OFF if input must be put in an internal DBSPACE before use

Calculations

TEMPAGES = NSORTIN/(4000/W)
TEMPAGES is the number of temporary pages required for Sort
W is the width of Sort records

$$MRGPAGES = \log_2(TEMPAGES)/3$$

MRGPAGES is the number of times that the DBSS Sort algorithm must perform a merge operation (merge passes)

SORTCOST = (4 + 2*MRGPASES)*TEMPAGES + COSTIN
(for I/O)

This includes the cost of input scan assuming no temporaries are needed (COSTIN), writing TEMPAGES of data to encode, reading from internal table and writing to output table to decode, a read and write for each merge pass, and a read from the output table to return data to the user.

This assumes that the input is taken directly from a base table (SRTDIRCT=BITON). If not (taken from a temporary), we must add 2*TEMPAGES to the I/O cost of sorting to account for another write and read of data.

Adding in the cost of CPU time to do the encode and decode in the DBSS, we assume encode costs 200 instructions/field and decode is 100 instructions/field. RDCRATIO is the ratio of instructions weighted against I/O cost. The formula is:

SORTCOST (Total) = SORTCOST (I/O) + RDCRATIO * (200 + 100) * NSORTIN * NFIELD

Total SORTCOST is returned to the caller in SORTOUT. TEMPAGES is returned in TEMPGOUT.

Selectivity of Predicates (Filter Factor Formula) - Calculated in ARIXODF)

Predicates (WHERE-clause search conditions) are applied against a table to limit the number of rows returned. The Optimizer is interested in how many rows will be returned (cardinality) in calculating alternative access path costs. The following formulas calculate a filter factor or selectivity factor, which estimates the proportion of rows that will satisfy each type of predicate.

Filter factor (FF) for multi-column indexes with equal predicates is 1/IDXFULKC (done in ARIXOGC)

- Column = Value form (JOB='CLERK')

  FF = 1/CATFLDCD Default (no stats) = 1/10
  (CATFLDCD is the number of unique values for the column)

- Column1 = Column2 form (NAME=JOB)

$$FF = \frac{1}{MAX\ (\ CATFLDCD(COL1), CATFLDCD(COL2))} \qquad Default = 1/10$$

  (use the more selective column)

- COLUMN [IN| NOT IN | =ANY | =ALL] [LIST | SUBQUERY] form
  (for example, DNO IN (23,30,40) , DNO = ANY (SELECT ...)

  For LIST:

$$FF = MIN\left(\frac{1}{2}, LISTSIZE * \frac{1}{CATFLDCD}\right)$$

  where LISTSIZE is the number of items in the list.
  Default is 1/10 * LISTSIZE

  For SUBQUERY:

$$FF = \frac{QARESCAR}{TOTSUBQC} \qquad Default\ is\ 1/10$$

QARESCAR is the expected number of rows to be returned by the subquery.
TOTSUBQC is the total number of rows that exist for the subquery tables (default is 10 000)

- RANGE form SAL [>,<,≥,≤] [COL | constant]

If form is COL OP literal, and the literal is not an IVAR (is a constant), and we have column statistics, a linear interpolation is done between the second highest and the second lowest key values for the column. Otherwise, the default filter factor of 1/3 is assumed.

For > or ≥

$$FF = \frac{(CATFLDCD - 3) * (CATHIGH2 - LITVALUE) + 2(CATHIGH2 - CATLOW2)}{CATFLDCD\ (CATHIGH2 - CATLOW2)}$$

For < or ≤

Use FF = 1 - FF(> or ≥) Default 1/3

- BETWEEN SAL BETWEEN x AND y

  If form is COL BETWEEN literal and literal, and literals are not IVARs, and we have column statistics, do interpolation for BETWEEN range; if not, use default = 1/4.

$$FF = \frac{(CATFLDCD - 3) * (HIGHLITVALUE - LOWLITVALUE)}{CATFLDCD * (CATHIGH2 - CATLOW2)}$$ Default = 1/4

- OR of Two Basic Predicates For example, SAL < 30000 or SAL = 50000

  Recursively calculate the filter factors of the left-hand and right-hand sides of the OR. The filter factor of the combination is:

  FF = LEFTFF + RIGHTFF - LEFTFF*RIGHTFF

  (Sum of filter factors less the probability that both are true)

- Other Types of Predicates

  Default filter factors are:

Complex OR expression	=	1/2
> ANY, < ANY, ≥ ANY, ≤ ANY	=	5/6
> ALL, < ALL, ≥ ALL, ≤ ALL	=	1/6
= ALL	=	1/10
¬= ALL	=	9/10

- NOT (For example, NOT IN, NOT BETWEEN, ¬=)

  Use the complement of the FF for the positive case:

  FF = 1 - FF

ARIXOAF

ARIXOAF allocates/frees common optimizer storage.

ARIXOBY

ARIXOBY increments the index of the next available word in the space block header.

ARIXOB1

ARIXOB1 is the BQUERY node processor, first pass. It calls ARIXOSL and ARIXOW1 to collect names in the select list and where tree.

ARIXOB2

ARIXOB2 iterates through Query Array, calling ARIXOSL and ARIXOW2 (if there is a WHERE tree) to do type distribution and make entries into predicate array.

ARIXOCA

ARIXOCA looks up table information in SYSCATALOG and SYSDBSPACES for each table in the Table Array. ARIXOCA looks up all the index entries in SYSINDEXES for each table. It then places them into the Index Array. ARIXOCA calls ARIXOIU to make dependency entries into SYSUSAGE.

ARIXOCK

ARIXOCK looks up each column in SYSCOLUMNS and associates the columns with their tables.

ARIXOCS

ARIXOCS evaluates the costs of access paths calculated by ARIXOGC and fills in the costs of join possibilities.

ARIXOCT

ARIXOCT sets up a control block. The control block contains pointers to the DBSS interface structures. The structures are used in the generated code to perform DBSS SCAN or row operations against a table.

ARIXOCU

ARIXOCU is the cleanup routine. It frees CATCOL storage associated with each table. ARIXOCU then calls ARIXOAF to free the five arrays used by the optimizer: TBA, CLA, QAR, PDA, and IDX.

ARIXOCY

ARIXOCY copies source BINDTABLE into Op Tree.

ARIXODF

ARIXODF calculates filter factor (selectivity factor) for each predicate in Predicate Array. The formulas used are described above on page 543

ARIXODM

ARIXODM acquires the DOMAINS DBSS structure. The structure is used to describe the columns retrieved (on query) or submitted (update/insert/delete) in a DBSS operation on a table.

ARIXOD1

ARIXOD1 is called during optimizer processing of an EXPLAIN statement. EXPLAIN can have any subset of the options reference, structure, cost, and plan. Tuples are inserted into the corresponding table(s) owned by the current user (REFERENCE_TABLE, STRUCTURE_TABLE, COST_TABLE, or PLAN_TABLE).

ARIXOD1 checks for the existence of necessary tables, and performs EXPLAIN COST. ARIXOD1 calls ARIXOD2 to perform EXPLAIN REFERENCE, EXPLAIN STRUCTURE, or EXPLAIN PLAN, if necessary. If a necessary table does not exist, an error condition (-204) is returned to the user, and no tuples are put in any table.

ARIXOD2

ARIXOD2 is called during optimizer processing of an EXPLAIN statement by ARIXOD1 to perform EXPLAIN REFERENCE, EXPLAIN STRUCTURE, and EXPLAIN PLAN, if necessary.

ARIXOEX

WHERE or HAVING Expression Walkthrough. During Pass 1, ARIXOFC is called for each column found and places it in the Column Array. During Pass 2, each predicate is checked for a column. Also a type conversion is done on literals.

ARIXOFC

ARIXOFC handles column array entries.

ARIXOFE

ARIXOFE is called on data definition statements ACQUIRE, ALTER DBSPACE, CREATE TABLE, CREATE INDEX, ALTER TABLE, DROP, COMMENT, DEFINE SYNONYM, RESTORE, and SAVE. It places creator id in the entity descriptor.

ARIXOFF

ARIXOFF fills in pointer values in the DBSS DOMAINS structure. The structure points to the I/O areas of the submitted columns.

ARIXOFP

ARIXOFP makes predicate entries into the Predicate Array.

ARIXOFQ

ARIXOFQ is called by ARIXOB1 to make entries into the Query Array for each BQUERY node found in the Op Tree scan.

ARIXOFR

ARIXOFR creates a DBSS interface structure in SPACEBLK pointed to out of the control block. This structure is used in generated code to accompany DBSS calls to perform SCAN or ROW operations.

ARIXOFT

ARIXOFT makes Table Array entries.

ARIXOGA

ARIXOGA is the ASL Generation Entry module. It interprets the chained miniplans output by ARIXOGP. ARIXOGA represents the miniplans as a modified Op Tree and DBSS operation interface structure.

ARIXOGB

ARIXOGB scans the Query Array entries for GROUPBY clauses and/or built-in functions (SETFN node). It changes GROUPBY columns in the select list to objects of the MIN builtin function.

ARIXOGC

ARIXOGC gets the cost of accessing this table using information from the system catalogs and a series of cost analysis formulas. (See the introduction to this section for the appropriate formula.)

ARIXOGP

ARIXOGP is the entry module for selection of the optimal access path. An access path is created for each entry in the Query Array. It processes the entries from bottom to top. An access path is selected by modules under ARIXOTS and the plan is translated to ASL by modules under ARIXOGA.

ARIXOIN

ARIXOIN handles INSERT SQL requests.

ARIXOIU

ARIXOIU enters usage information in SYSUSAGE.

ARIXOLC

ARIXOLC builds a LOCDESCRIP in the Op Tree.

ARIXOLF

ARIXOLF scans the Column Array for long fields.

ARIXOMA

ARIXOMA acquires temporary DBSPACE in the ASL by creating an acquire node in the Op Tree and an acquire block in the space block.

ARIXOMB

ARIXOMB makes COL1 OP COL2 appear as COL1 OP LIT in predicates.

ARIXOMD

ARIXOMD constructs a SQLDA for DESCRIBE in the space block.

ARIXOMF

ARIXOMF builds a DBSS Interface structure that is used with a DBSS Fetch operator.

ARIXOML

ARIXOML generates a list in the ASL. This implies creating acquire temp, create temp, and insert ASL nodes in the Op Tree and the acquire block structure in the space block.

ARIXOMS

ARIXOMS sets up the DBSS Scan Interface structure for the scan DBSS operations on a specific table. It initiates creation of any DOMAINS, KDOMAINS, or SARG structures that will be associated with the BASE structure.

ARIXONV

ARIXONV expands a SELECT *.

ARIXONW

ARIXONW extends the size of the existing Op Tree.

ARIXOOP

ARIXOOP is the entry module for RDS Optimizer functions. It identifies the types of root nodes. ARIXOOP then decides what modules to call to handle the root functions.

ARIXOOS

ARIXOOS is the select list expression tracer for passes 1 and 2 of the select list.

ARIXOQ1

ARIXOQ1 is the First-Pass Query processor.

ARIXOQ2

ARIXOQ2 is the Second-Pass Query processor.

ARIXORM

ARIXORM removes a sargable predicate from the Op Tree.

ARIXOSC

ARIXOSC scans the Table Array. It initiates check for SELECT authority on each table. If select authority exists, ARIXOSC calls ARIXA06 to make an entry in SYSTABAUTH.

ARIXOSG

ARIXOSG converts a sargable predicate into disjunctive normal form (DNF).

ARIXOSL

ARIXOSL is a procedure to process the SELECT list. It is called on two passes.

ARIXOSO

ARIXOSO handles SQL SELECT UNION function.

ARIXOSR

ARIXOSR implements a SORT in the ASL by creating a SORT node in the Op Tree and producing a sort block in the space block.

ARIXOSS

ARIXOOS computes the sort costs for weighing alternatives to handle ORDER BY, GROUP BY, or JOIN.

ARIXOST

ARIXOST processes the columns of the SET clause of an UPDATE SQL statement.

ARIXOSU

ARIXOSU examines each block's relocation directory. Then it changes the pointers, whose index location is the directory, to a block number offset format.

ARIXOS1

ARIXOS1 marks nodes reachable from the specified node. It is an iterative procedure called from ARIXOSG.

ARIXOS2

ARIXOS2 removes the 'AND' descendent. It is an iterative procedure called from ARIXOSG.

ARIXOS3

ARIXOS3 copies a subtree from the WHERE tree into the Tree Array. It is an iterative procedure called from ARIXOSG.

ARIXOS4

ARIXOS4 is an iterative procedure in the search argument generation process. It copies a subtree starting at the input parameter OUTIND and connects it to the child or sibling passed in the OUTIND entry. This guards against the possibility that the internal procedure transform (in ARIXOSG) has left the subtree as the child of more than one parent node.

ARIXOTF

ARIXOTF transforms literals to proper data type.For a literal string, ARIXOTF creates a slot in the space block to place the transformed result.

ARIXOTS

ARIXOTS generates the tree of possible access paths, uses ARIXOGC and ARIXOCS to calculate costs, and selects the best path for the query.

ARIXOUS

ARIXOUS checks for synonyms when no creator name is specified in an entity descriptor.

ARIXOVC

ARIXOVC composes a view into the Op Tree.

ARIXOVD

ARIXOVD handles the view definition statements.

ARIXOV1

ARIXOV1 copies the subtree into Op Tree as part of view composition. It is an iterative procedure called from ARIXOVC.

ARIXOW1

ARIXOW1 examines the WHERE predicates on the first pass.

ARIXOW2

ARIXOW2 examines the WHERE predicate tree on the second pass.

ARIXO1S

ARIXO1S makes SARGs (DBSS search arguments) out of expressions that are in DNF (Disjunctive Normal Form).

RDS PARSER MODULES

The Parser modules are ARIXPA1, ARIXPA2, and ARIXPA3.

ARIXPAT

ARIXPAT is the SQL/DS common patch area for the RDS component.

ARIXPA0

ARIXPA0 uses ARIBPRD to map production numbers to production names. ARIXPA0 also uses ARIBPD1 macros to generate Parser semantic routine linkage to call the proper entry points in the semantic routines contained in modules ARIXPA2 and ARIXPA3. The ARIBPD2 macro is used to generate entry point tables. The proper entry points in ARIXPA2 or ARIXPA3 are called based upon the above resolution.

ARIXPA1

ARIXPA1 is the Parser main line module and scanner. The main line is driven by the parse tables (ARIBPAM). ARIXPA1 calls ARIXPA4 to get addressability to the parse tables. Parse tables are based on BNF-defined tokens. When the equivalent of a BNF line is isolated by the searcher, the corresponding semantic routine is invoked through ARIXPA0 to process those tokens. Semantic routines are identified by entry points contained in modules ARIXPA0, ARIXPA2, and ARIXPA3. ARIXPA0 also contains distributive code (through ARIBPD1 and ARIBPD2 macros) to route processing to the semantic routine entry points. The four status names for main line functions are GET status, LA status, BUMP status, and SEMANT status. The status indicator is used as an index into the parse tables which are themselves tables containing numbers used to control the logic flow in the Parser. The parse tables are generated off-line from BNF statements.

ARIXPA2

ARIXPA2 and ARIXPA3 contain the Parser semantic routines. Each time ARIXPA2 is invoked a single semantic routine (called a production) is executed.

ARIXPA3

ARIXPA2 and ARIXPA3 contain the Parser semantic routines. Each time ARIXPA2 is invoked a single semantic routine (called a production) is executed.

ARIXPA4

ARIXPA4 contains parse tables that drive the Parser. When called by ARIXPA1, it returns a pointer that establishes addressability to these tables.

ARIXSCF

ARIXSCF converts a fullword binary value to its character (EBCDIC) representation. It is for use only in the SQL/DS partition. The resultant character string is sixteen bytes long and padded on the left with character zeros. If the value was negative, a minus sign is placed in the first byte of the string.

ARIXSCH

ARIXSCH converts a halfword binary value to its character (EBCDIC) representation. It is for use only in the SQL/DS partition. The resultant character string is eight characters long, padded on the left by character zeros. If the input value was negative, the first character is a minus sign.

ARIXSCN

ARIXSCN handles concatenation. The input parameter is a pointer to a structure that contains a list of pointers to the character variables to be concatenated. This structure also contains the lengths of the character variables and a pointer to the target string. There is also a pointer to a field where the length of the resulting target is placed if the address contained therein is not null.

ARIXSCP

ARIXSCP converts a pointer from hexadecimal to its character (EBCDIC) representation. It is for use only in the SQL/DS partition. The resultant string is eight bytes long.

ARIXSDB

ARIXSDB is the top module to convert a halfword into character form. All leading zeroes are substituted by blanks.

ARIXSDT

ARIXSDT calculates the date (the DOS/VSE date in the Supervisor Communication Area must be in the form "mm/dd/yy").

ARIXSLN

ARIXSLN looks for the last nonblank position in a particular character string to determine the length of the character string.

ARIXSRT

ARIXSRT performs a ROLLBACK WORK and frees allocated storage. You reach this point because an error occurred after system changes were made. If an automatic RESTORE has not been done and a fatal DBSS error has not occurred, this code restores to save point 0. If this fails, put the current SYR.CODE at the end of SYR.MESSAGE and set SYR.CODE to fatal error (-902).

ARIXSTM

ARIXSTM calculates time of the day.

ARIXSUT

ARIXSUT returns a guaranteed unique character string of 12 characters (A-Z, 1-9). ARIXSUT first performs a System/370 "STORE CLOCK" instruction. It then turns off the high-order bit, thus giving a 63-bit integer. This integer is interpreted in base 35, with the base-35 digits being A-Z and 1-9. Therefore, successive calls to ARIXSUT return a strictly monotone increasing sequence of 12-byte character strings.

RDS TRACE DESCRIPTOR MODULES

ARIXTR0

ARIXTR0 is the trace point descriptor module for the RDS Executive component. It contains the trace point descriptors for all trace points in modules of the RDS Executives component. The trace point numbers are in the range 4000 through 4079. See "Trace Points" in Section 6, Diagnostic Aids.

ARIXTR1

ARIXTR1 is the trace point descriptor module for the Parser and Optimizer components. It contains the trace point descriptors for all the trace points contained in modules of the Parser and Optimizer components of RDS and for Working Storage (analysis - modules ARICWSG and ARICWSF). The trace point descriptors in this module have trace point numbers in the range 4400 through 4935. See "Trace Points" in Section 6, Diagnostic Aids.

ARIXTR2

This module is Part 1 of the trace point descriptor modules for the Interpreter component. It contains trace point descriptors for trace points contained in modules of the RDS Interpreter component. All trace point descriptors in this module have trace point numbers in the range 5800 through 5944. See "Trace Points" in Section 6, Diagnostic Aids.

ARIXTR3

This module is Part 2 of the trace point descriptor modules for the Interpreter component. It contains trace point descriptors for trace points contained in modules of the RDS Interpreter component. All trace point descriptors in this module have trace point numbers in the range 5945 through 6099. See "Trace Points" in Section 6, Diagnostic Aids.

ARIXTR4

ARIXTR4 is the trace point descriptor module for the authorization component. It is also the trace point descriptor module for the interpreter and authorizaton subroutines. All trace point descriptors in ARIXTR4 have trace point numbers in the range 6150 through 6410. See "Trace Points" in Section 6, "Diagnostic Aids". Section 6 also contains Security/Audit trace points.

ARIXTR5

ARIXTR5 is the trace point descriptor module for the code generator component. It contains the trace point descriptors for all trace points in modules of the RDS Code Generator component. It also contains trace point descriptors for Service Temporary trace points for all RDS modules. All RDS Code Generator trace point descriptors in this module have trace point numbers in the range 5200 through 5278. All Service Temporary trace point descriptors in this module (for all RDS modules) have trace points in the range 9900 through 9999.

ARIXTR6

ARIXTR6 is Part 3 of the trace point descriptor module for the interpreter component. It contains trace point descriptors for trace points contained in modules of the interpreter RDS component. All trace point descriptors in this module have trace point numbers in the range 6500 through 6599.

DBSS DATA CONTROL

ARIYC00

ARIYC00 is the DBSS module that fetches the next control row from a control relation. The control relation is a set of the entire MCR1 or subsets of ICR, LCR, PLCR. The control relation and the row being accessed are identified by the qualf segment, relation-id, image-id, and link-id.

ARIYC01

ARIYC01 is the DBSS fetch control row routine. The control relation and the row being accessed are identified by the qualf segment, relation-id, image-id, and link-id.

ARIYC02

ARIYC02 is a DBSS module that inserts control rows.

ARIYC03

Entry Point: ARIYC03
Function: ARIYC03 is the control row creater. ARIYC03 searches the block enties for a header page that has enough space to accommodate the row length. If there is not enough contiguous free space, ARIYD14 eliminates the holes and updates the page header.

Entry Point: ARIYC031
Function: ARIYC031 updates the pointer, free space, and the slot in a header page.

ARIYC05

ARIYC05 inserts a new link control row in accordance with the information the caller supplies.

ARIYC06

ARIYC06 moves the submitted return code to CRCODE in CBASE. ARIYC06 also unfixes all directory block and page buffers and releases all short locks.

ARIYC07

ARIYC07 is the DBSS/DC initializer. ARIYC07 is called at the beginning of all data control calls. ARIYC07 does the following:

- Checks for boundary alignment
- Checks that BEGIN WORK was issued
- Acquires SI-lock
- Initializes a number of YTABLE1 variables

ARIYC08

Entry Point: ARIYC08
Function: ARIYC08 traces for level 2 a data control call on DBSS entry. ARIYC08 displays the base control information and auxiliary structures. The auxiliary structures are SCR, MCR, ICR, LCR, PLCR. See 'Control Header Information' in Section 5 for details or the auxiliary structures.

Entry Point: ARIYC081
Function: ARIYC081 is a secondary entry point to trace for level 2 a data control call on DBSS exit. ARIYC081 displays information from the same structures used above.

ARIYC09

ARIYC09 is the header page formatter. It formats the page as a header page and updates the related entry in the directory block.

ARIYC10

ARIYC10 updates the control rows according to the information supplied by the RDS caller.

ARIYC11

ARIYC11 deletes a control row.

ARIYC13

ARIYC13 deletes a link control row (LCR or PLCR) of a unary or binary link.

ARIYC14

ARIYC14 deletes an index control row (ICR).

ARIYC15

ARIYC15 returns space occupied by a header row.

ARIYC18

ARIYC18 inserts the DBSPACE header row (SCR) in re-do and deletes the row in un-do.

ARIYC19

ARIYC19 inserts a table control row (MCR1) in re-do and deletes the row in un-do or backup.

ARIYC20

ARIYC20 inserts an image control row in re-do and deletes the row in un-do or backup.

ARIYC21

ARIYC21 inserts child and parent link control rows (LCRs and PLCRs) in re-do. It deletes the rows in un-do and backup.

ARIYC22

ARIYC22 updates the control rows in re-do, un-do, and backup.

ARIYC23

ARIYC23 deletes the DBSPACE header row (SCR) in re-do.

ARIYC24

ARIYC24 deletes a table control row (MCR) and domain row (DCR) in re-do.

ARIYC25

ARIYC25 deletes an index control row (ICR) in re-do and inserts an index control row in un-do or backup.

ARIYC26

ARIYC26 deletes child link (LCR) and parent link (PLCR) record rows in re-do.

ARIYC27

ARIYC27 is the trace point descriptor module for the Data Control component. It contains all trace point descriptors for all trace points contained in modules of the Data Control component. The trace point numbers are in the range 1200 through 1499.

ARIYD00

ARIYD00 performs the DBSI call to close a scan identified by the submitted SCANID. The SCB (scan control block) is invalidated (marked closed) and made available for reuse within the agent scan area. Scan holds automatically maintained by the DBSS are released.

ARIYD01

ARIYD01 performs the DBSI call to connect a child row T1 before or after another row T2 in a link.

ARIYD02

ARIYD02 performs the DBSI call to delete a row from a table and (optionally) disconnect it from the link.

ARIYD03

ARIYD03 performs the DBSI call to disconnect a child row from a link.

ARIYD04

ARIYD04 performs the DBSI call to fetch a row from a table.

ARIYD05

ARIYD05 performs the DBSI call to:

- Insert a row in a table and (optionally) connect it in a link.
- Insert a row at the end of a list.

ARIYD06

ARIYD06 performs the DBSI call to get the next row. GET NEXT is the DBSI call to search and retrieve a row from a table or a sequential list using an opened scan.

ARIYD08

ARIYD08 performs the DBSI call to open a scan on an index or a link of a table or on a table or a sequential list. Open scan also returns the domains of the row on which the opened scan is positioned as specified in the DBSI caller DOMAINS structure.

ARIYD09

ARIYD09 performs the DBSI call to retrieve a parent row of a submitted child row in a specified binary link.

ARIYD10

ARIYD10 performs the DBSI call to update the domains of a row of a table.

ARIYD11

ARIYD11 is executed at the end of a DBSI call that has to back up. It assumes that all updates from the call have been restored (undone). It unfixes all buffers and releases all the short locks. ARIYD11 then tests the lock flags and does one of the following:

1. Indicates that the DBSI call should be retried.
2. Indicates that the DBSI call should not be retried. (It also moves the DBSS return code, that caused the DBSI call failure, to the caller parameter field RECODE.)
3. Calls the LUW backup (ROLLBACK) routine to rollback the LUW. Then it moves the DBSS return code, that caused the LUW ROLLBACK, to the caller.

ARIYD13

ARIYD13 checks the addressing boundaries of DOMAINS and of requested data fields. It returns in NDOM the number of domains effectively requested by the user (entries with FACLTH > 0). ARIYD13 is used for row fetch type DM calls (FETCH ROW, GET NEXT, GET PARENT, OPEN SCAN).

ARIYD14

ARIYD14 compacts an entity page by eliminating holes and updating the page header. If TID locks are outstanding, it checks whether the locked holes are still held by an active LUW. If the locked holes are not held, ARIYD14 then recovers their space.

ARIYD15

ARIYD15 connects child row T1 before or after row T2 in the link identified by LID. T1 is identified by fields: SEGMENT, RID, and TID in BASE. T2 is identified by fields: PSEGMENT, PRID, and PTID or by SCANID in BASE. ARIYD15 updates the prefixes in entity space of T2 and T0. T0 is the other row T1 is being connected to. ARIYD15 also flags physically dirty link SCBs whose current TIDs are T1's left or right twins. It changes to 'ON' T1 the state of a scan that:

1. Is on link LID, and
2a. Is 'BEFORE' T2 and QUALF = 'B', or
2b. Is 'BEFORE' T1 or 'EOF' T2 and QUALF = 'A'.

ARIYD15 also writes the link prefix part of the log-record (present in LCONNECT and LINSERT) that is passed as input in LOGB.

ARIYD16

ARIYD16 checks whether the page, whose number is passed in INPAGE, was damaged.

Return code = 0 if page was damaged
Return code = -1 if I/O error occurred in accessing the list of damaged pages
Return code = 1 if the page was not damaged.

ARIYD17

ARIYD17 deletes base rows and possibly overflow rows given by ETID(1,2) and EPPTR(1,2) in YTABLE1. All needed locks are acquired. The created holes are marked as locked if TID locks are being used. It updates the row header and VPTRMAP entry. ARIYD17 also updates the page header and the page map block free space entry for the page(s).

ARIYD18

ARIYD18 disconnects the child row T from the link described in YLINKREC. The child row T is identified by SEGMENT and TID. The link is identified by SEGMENT and LID. ARIYD18 updates the prefixes in entity space of the rows connected to T. It also physically dirties all the link SCBs referring to these rows. ARIYD18 changes the state of the SCBs on link LID that refer to T. It then physically and logically dirties these SCBs. In LOGB, ARIYD18 also completes the data referring to the disconnection of row T from link LID. It then moves the DBSPACE, table-ID, and TID of the parent of row T to PSEGMENT, PRID, PTID (fields in BASE)

ARIYD19

ARIYD19 formats a page as a free entity page. Then ARIYD19 updates the related page map entry in the directory block.

ARIYD20

ARIYD20 searches a page for a row belonging to the current table. The search begins with the row identified by INTID. If search arguments are used and the DBSI call is SORT or GET NEXT, then the arguments are checked. When search arguments are used, searching continues until there is a match or until there are no more rows on the page.

ARIYD23

ARIYD23 retrieves a row and its page (and directory block if applicable) for read or write.

Entry Point: ARIYD23
Function: For a header row (read only).

Entry Point: ARIYD231
Function: For a base entity row.

Entry Point: ARIYD232
Function: For an overflow entity row.

ARIYD26

ARIYD26 fetches for read or write a header, base, or overflow row and its page (and its directory block). Input is an internally generated TID, read/write code, and row type. It returns the requested page and row pointer in Register 2 and Register 3. TID validity is not checked.

ARIYD27

ARIYD27 builds an index key from the key information submitted by a DBSI caller. It checks the acceptability of the submitted NKEYDOM, KDOMAINS, and key-domain values. Then ARIYD27 assembles IKEY from them. IKEY is a YTABLE1 substructure that describes a submitted index key to the DBSS Index Component.

ARIYD28

ARIYD28 builds an index key from a stored row using the index control record and the domain control record. The index key is assembled in IKEY. IKEY is a YTABLE1 substructure that describes a submitted index key to the Index Component.

ARIYD29

ARIYD29 builds an index key from the domain information submitted by the DBSI caller. The index key is assembled in IKEY. IKEY is a YTABLE1 substructure that describes a submitted index key to the DBSS Index Component.

ARIYD30

ARIYD30 builds an index key from the DBSI caller submitted domains and the stored row using the index control record and the domain control record. The index key is assembled in IKEY. IKEY is a YTABLE1 substructure that describes a submitted index key to the DBSS Index Component.

ARIYD32

ARIYD32 obtains the MCR (master control record) and the table lock for the table whose RID is specified by parameter INRID. Note: The table can be a sequential list.

ARIYD33

ARIYD33 obtains space for a data row that has to be moved. The row is moved because an update is not in place or there is an increase in the link prefix size. When the update is not in place, ARIYD33:

- Moves the row prefix to the newly allocated row.
- Moves the DBSI caller replaced fields (DOMAINS input) to the newly allocated row.
- Moves the unupdated fields from the old row to the newly allocated row.
- Logs the row update.

When there is an increase in the link prefix size, ARIYD33 copies the old row to the newly allocated row. Then it increases the link prefix to the maximum size recorded in the table MCR (master control record). ARIYD33 then pads the added prefix area with binary zeros.

ARIYD34

ARIYD34 allocates a new SCB (scan control block) in the agent scan table. A set of map entries at the end of the scan table point to each allocated SCB in the table. The map entry ordinal becomes the scan-id. The scan-id is stored in the allocated SCB whose address is returned.

ARIYD38

ARIYD38 obtains DBSPACE attributes, a required DBSPACE lock, and the DBSPACE header record (SCR) for an input DBSPACE. ARIYD38 also updates the DBSPACE data in YTABLE1. ARIYD38 can be invoked for a public, private, or an internal DBSPACE.

ARIYD39

ARIYD39 obtains a TID and an entity page close to TPAGE where a new row of length ELTH can be inserted into a table. ARIYD39 is called in INSERT to obtain a base TID (TIDTYPE = 1). ARIYD39 is called in UPDATE, CONNECT, INSERT (LID ≠ 0) to obtain an overflow TID (TIDTYPE = 2).
The selected page and TID are locked if page and/or TID locking are required. The selected page is not updated (other than possible page compacting) except where a new VPTRMAP slot is needed for the selected TID (the VPTRMAP slot is set to 0).
When searching for a page to select for the new row, the DBSPACE free space percentage is used to select the page. This is used so the insert will not exceed the free space percentage limit.

ARIYD40

ARIYD40 accepts a DBSI caller provided TID, a read/write code, and a TID type (type of header row or base entity row) as input. ARIYD40 returns the corresponding page and row pointers in Register 2 and Register 3. It also saves the fixed buffer pointers in YTABLE1. TID validity is checked.

ARIYD41

ARIYD41 obtains a user or system share mode lock on a row or a page.

Entry Point: ARIYD41
Function: ARIYD41 obtains a user share mode lock on a row (or its page) if the DBSPACE and table lock states are less than share mode.

Entry Point: ARIYD411
Function: ARIYD411 obtains a system share mode lock on a row (or its page) if required.

ARIYD43

ARIYD43 copies BASE (DBSI call linkage parameter block) into YTABLE1. It also checks that a BEGIN WORK (LUW) was issued. ARIYD43 acquires a DBSI-lock, initializes a number of YTABLE1 variables, and clears the general feedback fields in BASE. It is called at the beginning of all data manipulation calls.

ARIYD44

ARIYD44 updates the scan table caused by a CDELETE, a CINSERT, or the release of an internal DBSPACE. ARIYD44 can be called in 6 circumstances:

1. Deletion of a DBSPACE (caused by CDELETE of an SCR). All scans open on objects in the DBSPACE are closed. All scans for links having a child and/or a parent in DBSPACE are closed. All link scans are physically dirtied.
2. Deletion of a table (caused by CDELETE of an MCR). All the scans open on objects in the table are closed. All scans for links having the table as a parent or a child are closed. All the link scans are physically dirtied.
3. Deletion of a link (caused by a CDELETE of a link). All the link scans for that link are closed. All link scans are physically dirtied.
4. Deletion of an index (caused by a CDELETE of an ICR). All the index scans on that index are closed.
5. Insertion of a link (caused by a CINSERT of a link). All the link scans are physically dirtied.
6. Release of an internal DBSPACE.

ARIYD45

ARIYD45 checks that all DBSI caller requested domains are included in the domains of the current index. ARIYD45 then moves the requested domains from the index key to the DBSI caller.

ARIYD46

ARIYD46 validates the DBSI caller submitted key domains information and assembles (in IKEY in YTABLE1) the index key value to be submitted to the Index component. ARIYD46 computes the lock requirements for the index component from the DBSPACE, table, and gross lock requirement lock states. ARIYD46 also retrieves the TID of the row that matches the assembled index key and the DBSI caller submitted ICOMP value from the Index component.

ARIYD47

ARIYD47 acquires the page and TID locks for an entity row INTID in state INSTATE for a duration DURCODES in YTABLE1.

ARIYD48

ARIYD48 is called by CDELETE (control row delete) to perform one of the following operations:

- Drop all the rows of a table from a DBSPACE entity space.
- Erase (and compress) the child and/or parent TID pointers of the link prefix from the child and/or parent rows in the link that is being dropped.

ARIYD49

ARIYD49 moves the requested row field values (domains) from the stored row to DM caller DOMAINS parameter structure. If a domain is longer than the caller provided field for the domain, then the domain is moved in but it is truncated. If a requested field is not stored in the row, then a null character string is returned to the caller. The caller specifies which domains are to be retrieved.

ARIYD50

ARIYD50 searches the rows in the present DBSPACE looking for a row belonging to the current table. If search arguments (DM call structure SARGS) are present, they become criteria in selecting the row. The search is forward, starting with the row after the one identified by the input parameter INTID.

ARIYD52

If the input parameter INCODE is:
- 'H' - ARIYD52 unfixes the DBSPACE header directory and page buffers.
- 'E' - ARIYD52 unfixes the entity page buffers and their directory buffers.

ARIYD53

Entry Point: ARIYD53
Function: ARIYD53 unfixes the DBSPACE header directory and page buffers.

Entry Point: ARIYD531
Function: ARIYD531 unfixes the entity page buffers and their directory buffers.

ARIYD55

ARIYD55 performs the cleanup functions for DM and Sort operations.

Entry Point: ARIYD55
Function: ARIYD55 unfixes the DBSPACE header directory and page buffers and performs the functions listed under entry point ARIYD551.

Entry Point: ARIYD551
Function: ARIYD551
- Unfixes the entity page buffers and their directory buffers.
- Unfixes the first index page buffer and its directory buffer.
- Releases all short term locks acquired during execution of the DM or Sort DBSI call.
- Moves YTABLE1 copy of the DBSI caller structure BASE to the caller copy of BASE if the operation is not SORT and the DBSS return code is ok or is a warning.
- Stores the DBSS return code in the DBSI caller structure BASE (field URCODE) if the operation is not SORT.

ARIYD57

ARIYD57 checks the validity of the DBSI caller search arguments (SARGS structure). ARIYD57 checks for:
- Number of search arguments specified.
- Valid domain number in each search argument.
- Valid domain length in each search argument.
- Valid domain comparison operator in each search argument.
- Valid Boolean operator in each search argument except the last one.

ARIYD58

ARIYD58 checks whether the submitted 'search clause' is satisfied by the row pointed to by Register 3. The 'search clause' refers to those search arguments specified in the structure SARGS. The specified search arguments are compared to the corresponding domains in the row.

ARIYD59

ARIYD59 checks for data manipulation calls that a SCB (scan control block), identified by SCANID from BASE, is legal and in 'on' state (SCB must be open, 'on', and not list type). If it is open and 'on', the DBSPACE ID and table ID are moved to the copy of BASE in YTABLE1. If the SCB is not list type, the row ID is moved to the copy of BASE in YTABLE1. The SCB pointer is also inserted into the copy of BASE in YTABLE1. If the SCB is flagged dirty, a warning return code is given.

ARIYD60

ARIYD60 checks for data manipulation link calls that a SCB (scan control block), identified by SCANID in YTABLE1, is legal and 'on' for a row in a table (SCB must be open, in 'on' state, not be list type). If check succeeds, the row still exists. It then moves the DBSPACE ID, table ID, and row ID from the SCB into YTABLE1. The SCB pointer is also inserted into YTABLE1. If the SCB is flagged 'dirty', a warning return code is given.

ARIYD61

ARIYD61 compacts the scan table by eliminating holes (closed SCBs) and updating the scan table header. The scan table is located in the 'SCANS' area pointed to by YT1SCANP in YTABLE1. The format of the scan table (SCANS) is similar to a data page. It contains a:

Header - Used when allocating new entries or maintaining the scan table.
Space for SCBs.
Map Entries - Located at the end of the scan table. A set of pointers to the SCBs within the scan table. The ordinal of the map entry corresponds to the SCANID.

ARIYD62

ARIYD62 fetches a row from the data base for data manipulation 'read only' calls with lock protocol specified in the LOCKMODE parameter. Pointers to the retrieved page and row are returned in Register 2 and Register 3. If the base row is a pointer row, the overflow row is retrieved and returned to the caller. The lock protocol is analysed. Then the proper locks are requested so that the row can be fetched.

ARIYD63

ARIYD63 updates the free-space entry in the directory block corresponding to the page pointed to by PAGEPTR.

ARIYD64

ARIYD64 fetches a row from the data base for data manipulation UPDATE or DELETE calls. The current lock state is analyzed to determine if any locking is required for the update or delete. If yes, the proper entity locks are requested.

ARIYD65

ARIYD65 connects row T in REDO or disconnects row T in UNDO and ROLLBACK using the logged information.

ARIYD66

ARIYD66 re-deletes row T for REDO (recovers delete operation). If the DELETE was with DISCONNECT, ARIYD66 re-disconnects any parent and twin links. ARIYD66 inserts row T for ROLLBACK and UNDO (recovers delete operation). If DELETE was with DISCONNECT, ARIYD66 connects row T to any parent and twin links.

ARIYD67

ARIYD67 disconnects row T in REDO or reconnects row T in UNDO and ROLLBACK using the logged information. (Recovers disconnect operation.)

ARIYD68

ARIYD68 re-inserts row T for REDO (recovers insert operation). If INSERT was with CONNECT, ARIYD68 re-connects any parent and twin links. ARIYD68 deletes row T for ROLLBACK and UNDO (recovers insert operation). If INSERT was with CONNECT, ARIYD68 disconnects row T from any parent and twin links.

ARIYD69

If REDO, ARIYD69 updates the data row using the logged information. If UNDO or ROLLBACK, ARIYD69 'un-updates' the data row using the logged information. Any indexes affected by the update/un-update are updated to reflect the update/un-update. (Recovers update row operation.)

ARIYD70

ARIYD70 is used in REDO, UNDO, and ROLLBACK. It compacts a page pointed to by PAGEPT. When ARIYD70 is called (in ROLLBACK) with the TID locks being used (EPAGELKU = 'P'), it recovers holes that are unlocked or not held by any active LUW. Also, ARIYD70 recovers exactly one locked hole of length LTH and slot-ID IDN in the page. It does not recover the hole if LTH is not equal to the hole length. If after compacting, contiguous free space is less than LTH, compacting is done again recovering all locked holes. When called with EPAGELKU ≠ 'P', it recovers all the holes. In addition to holes, surplus VPTRMAP entries are converted to free space.

ARIYD71

ARIYD71 connects or disconnects a row from parent and/or twin links during REDO, UNDO, or ROLLBACK of INSERT or CONNECT data row operation (i.e. when called by ARIYD68 for an INSERT UNDO or REDO or by ARIYD65 for a CONNECT UNDO or REDO) by updating the link prefixes of the parent and twin rows connected (DISCONNECT) or to be connected (CONNECT) to row T.

ARIYD72

ARIYD72 disconnects or connects a row T from parent and/or twin links during REDO or UNDO/ROLLBACK of DELETE or DISCONNECT data row operation (i.e. when called by ARIYD66 for a DELETEU (DELETE UNDO) or DELETE REDO or when called by ARIYD67 for a DISCONNU (DISCONNECT UNDO) or DISCONNECT REDO). This is done by updating the link prefixes of the

parent and twin rows disconnected (CONNECT) or to be disconnected (DISCONNECT) from row T.

ARIYD73

ARIYD73 changes the size of a link-prefix of a row for ARIYD68 for an INSERTU (INSERT UNDO/REDO) or for ARIYD65 for a CONNECTU (CONNECT UNDO/REDO) call. In REDO, the link prefix is expanded and padded with 0's. In UNDO/ROLLBACK, the link-prefix is shrunk.

ARIYD74

ARIYD74 initializes YTABLE1U in recovery calls. It acquires a backup lock if any request for a page lock is anticipated. If any index has to be updated, it retrieves SCR, MCR, and DCR rows and stores pointers (and rows for SCR and MCR) in YTABLE1.

ARIYD75

ARIYD75 inserts or deletes all key entries in all indexes that are tied to a data row being inserted or deleted as a result of recovery REDO, UNDO, or ROLLBACK. The caller identifies the row and whether deleting or inserting.

ARIYD76

ARIYD76 unfixes the last entity page and block buffers (if they exist) for recovery.

ARIYD77

ARIYD77 retrieves and partially updates the non-index page to be updated during recovery operations. At entry, the ID of the row to be processed in the page is supplied in INTID. INLTH describes the nature of the required operation:

- INLTH ≠ 2 causes the row to be deleted. A complete page and block update is then performed.
- INLTH ≠ 1 causes the row to be shortened. All page and block processing is done with the exception of entering the TID-PTR in the new pointer row.
- INLTH = 0 causes the page to be prepared for update. No page or block processing is done.
- INLTH > 0 causes the page to be prepared for insert. All page and block processing is done with the exception of the actual insertion of the row.

ARIYD80

ARIYD80 is the trace point descriptor module for the Data Manipulation component. It contains all trace point descriptors for all trace points contained in modules of the Data Manipulation DBSS component. All trace point descriptors in this module have trace point numbers in the range 1500 through 1999.

ARIYD81

ARIYD81 provides Level 2 tracing for Data Manipulation and Lock component DBSI calls for DBSS entry and exit. ARIYD81 is the entry point to trace a Data Manipulation call for DBSS entry. It displays the BASE control information and, depending on the call, DOMAINS, KDOMAINS, and SARGS. ARIYD811 is a secondary entry point to trace a Data Manipulation call for DBSS exit. It displays the BASE control information and, depending on the call, DOMAINS. ARIYD812 and ARIYD813 are secondary entry points to trace Lock and Unlock for DBSS entry and exit. They display LBASE.

ARIYD87

Entry Point: ARIYD87
Function: ARIYD87 writes the scan table to the system log (if LOGMODE is yes, otherwise it does nothing).

Entry Point: ARIYD871
Function: ARIYD871 computes the length of the scan-area and returns it in the LENG parameter.

Entry Point: ARIYD872
Function: ARIYD872 copies scans from the log into the scan table (restores the scan table from the log).

ARIYE01

ARIYE01 is the trace DBSS opcode entry and exit module.

Entry Point: ARIYE01
Function: Use ARIYE01 when ARIYM00 is called (DBSS opcode call) and DBSS entry tracing is active for the agent structure.

Entry Point: ARIYE011
Function: Use ARIYE011 when ARIYM00 is returning (DBSS opcode call completion) and DBSS exit tracing is active for the agent structure.

For DBSS entry:
- If the entry trace level is 2 and there is a DBSS entry trace routine for the opcode, ARIYE01 is called to provide trace output.
- If the entry trace level is 1 or there is no DBSS entry trace routine for the opcode, this module executes a trace point that displays:
  - DBSS ENTRY:
  - L_OPCODE = 'opcode-name'

For DBSS exit:
- If the exit trace level is 2 and there is a DBSS exit trace routine for the opcode, ARIYE01 is called to provide trace output.
- If the exit trace level is 1 or there is no DBSS exit trace routine for the opcode, this module executes a trace point that displays:
  - DBSS EXIT:
  - L_OPCODE = 'opcode-name'
  - RETCODE = Return code value

ARIYE02

ARIYE02 accepts as input a character string message and 0 to n variables that are to be formatted and displayed to the local/remote operator or display device. ARIYE02 is invoked by the SHOW or YSHOW macro.

ARIYE04

ARIYE04 scans a 80 character terminal or card input line looking for non-blank character strings. For each string, up to eight characters are placed into a token array (first string in first array element, second string in second array element, etc.). The number of strings found is counted and returned. The token array is initialized to all blanks. Encountering a string longer than eight characters causes processing to terminate. It terminates with the first eight characters of the long string in the token array as the last token and an error return code. Parameter NTOKENS identifies the token array entry in error.

ARIYE05

ARIYE05 converts an input character string of length n into its hex representation (an output character string of length 2n).

ARIYE06

ARIYE06 converts a hex character string of length zero to eight (terminated by a blank character if less than 8) to a pointer(32) binary value.

ARIYE07

ARIYE07 converts a fixed(31) signed binary integer to an unpacked zoned decimal string. The first byte is blank if the number is positive. The first byte contains a minus sign if the number is negative. Leading zeros are not generated. The length of the string (including sign byte) is returned.

ARIYE08

ARIYE08 converts an unsigned, unpacked decimal character string to a fullword binary integer. The decimal string can be zero to eight charcters long. If less than eight characters long, the string must be terminated by a blank character.

ARIYE09

ARIYE09 converts a decimal character string to a binary halfword. The character string can optionally have a plus or a minus sign as a prefix (default is plus if omitted). It also has zero to six (seven if sign omitted) decimal digits terminated by a blank character.

ARIYE10

ARIYE10 converts a fixed(15) signed binary integer to an unpacked zoned decimal string. The first byte is blank if the number is positive. The first byte contains a minus sign if the number is negative. Leading zeros are not generated. The length of the string (including sign byte) is returned.

ARIYE11

ARIYE11 converts a hex character string of length zero to eight (terminated by a blank character if less than eight) to a fixed(16) binary value.

ARIYE13

ARIYE13 is the DBSS/DSC line display routine module. It displays a line of data (or a SQL/DS message).

Entry Point: ARIYE13
   Function: ARIYE13 displays a line of data that is not a SQL/DS message, for example, SQL/DS command output. A flag (DSCOMODE) in the DSCAREA table of SQL/DS directs the module to display the line locally or to send the line to the ISQL linkage for display on the ISQL terminal (this is for support of SQL/DS commands entered from ISQL).

Entry Point: ARIYE131
   Function: ARIYE131 is used exclusively by the DBSS/DSC message processor for displaying SQL/DS messages. As with the main entry point above, output lines are directed to either local output or to ISQL for ISQL terminal output. The logic is such, however, that during SQL/DS operator command processing, only messages originating from the operator command linkage and processing modules go to the ISQL linkage. All others go to local output.

SQL/DS has an initialization parameter DSPLYDEV that directs all output lines routed to this module to be displayed on either the system print file or the operator's console or to both. A routing flag in the DS2CVT indicates whether the line display is to be controlled by the DSPLYDEV parameter value. If the flag is on, the line and the routing code is passed to module entry ARISYSDH for display on the specified output device(s)/file(s). This routing function is active only during SQL/DS initialization and termination. Thus it does not conflict with ISQL routing since SQL/DS operator commands cannot be entered during initialization or termination. When the DSPLYDEV routing function is not active, local output is routed to the operator display module (ARIYM50).

Note: In VM, the DSPLYDEV parameter is ignored, causing this output to be displayed only on the virtual machine console.

ARIYE14

ARIYE14 is the DBSS trace services routine.

Entry Point: ARIYE14
   Function: ARIYE14 causes trace point data to be collected and written to the SQL/DS trace output file. This function is invoked by the TRACE macro with the LOCK option omitted.

Entry Point: ARIYE141
   Function: ARIYE141 causes trace point data to be collected and written to the SQL/DS trace output file. It also leaves the trace output file 'locked' (see below) so that the caller, itself, can create additional trace point data records and cause them to be written to the trace output file. This function is invoked by the TRACE macro with the LOCK option specified.

Entry Point: ARIYE142
   Function: ARIYE142 causes the trace output file to be 'unlocked'. This function is invoked via the TRACE macro with the UNLOCK option specified.

Entry Point: ARIYE143
   Function: ARIYE143 causes caller-created trace point data records to be written to the trace output file. See entry point ARIYE141 above. This function is invoked via a call with the address of the record as the parameter.

On normal entry, entry point ARIYE14 and ARIYE141, tests are made to determine if trace is:
   * Active at all
   * Active for the current user-ID
   * Active for the invoking subcomponent or function

If trace is not active, return with no output.

A test is made to determine if a snap dump was requested (via the TRACE command) for this trace point. If it was, the SQL/DS partition or virtual machine snap dump routine is called and the snap dump request is disabled (dump first time only).

ARIYE15

ARIYE15 is the DBSS trace point descriptor module for the DBSS ENTRY and DBSS EXIT trace points. It contains the trace point descriptors for modules ARIYM00 (DBSS module called by RDS), ARIYE01 (module called by ARIYM00 when DBSS ENTRY or DBSS EXIT tracing is active), and all modules called by module ARIYE01 (for level two tracing) to create opcode specific trace output for DBSS ENTRY and DBSS EXIT tracing.

All trace point descriptors in this module have trace point numbers in the range 0000 through 0299.

- DBSS ENTRY trace points are 0000 through 0149
- DBSS EXIT trace points are 0150 through 0299

ARIYE16

ARIYE16 converts a fixed(31) signed binary integer to a printable hexadecimal string. Leading zeros are not generated (thus a binary one produces a one byte string with the character one). If the binary input is zero, the output is a one byte string with the character zero.

ARIYE37

ARIYE37 processes the SHOW SYSTEM operator command. It displays the date and time. Then it invokes other show modules to obtain their show output as follows:

- SHOW DBEXTENT Module
- SHOW LOG Module
- SHOW ACTIVE Module
- SHOW LOCK Module

ARIYI00

ARIYI00 acquires an internal DBSPACE (type 5).

Entry Point: ARIYI001
Function: ARIYI001 is used by the Sort component.

Entry Point: ARIYI002
Function: ARIY002 releases an internal DBSPACE, that is, it empties it, closes it, unallocates it, releases the lock and invalidates scans in it.

Entry Point: ARIYI003
Function: ARIYI003 is used by the Sort component.

Entry Point: ARIYI004
Function: ARIYI004 is used by the Transaction (work) component.

ARIYI01 - VSE System-Dependent Module

ARIYI01 is considered as part of the SHOW module set, thus message IDs have not been added. The contents of the ACB & RPL are displayed on the system console. If any errors occur from the SHOWCB or TESTCB VSAM macros, then the SQL/DS termination routine (ARIYM02) is called.

ARIYI02

ARIYI02 displays the number of HEADER, DATA, and INDEX pages that are defined and are occupied for a given DBSPACE. It also displays the percent of free space of the occupied pages (SHOW DBSPACE operator command).

ARIYI03

ARIYI03 traces DBSPACE operation DBSI (ARIYM00) calls before entry and after exit and returns the opcode, DBSPACE number and return code.

ARIYI04

ARIYI04 links to supporting routines for the SHOW DBEXTENT, SHOW DBCONFIG, SHOW DBSPACE, and SHOW BUFFERS operator commands.

ARIYI05

ARIYI05 is called to free all temporary DBSPACEs acquired by the LUW. Then, ARIYI05 reads each log record of the LUW. Its process depends on the record type as follows:

- If the record represents an update to the data base ARIYL07 is called. Based on the type of update, ARIYL07 calls the appropriate DBSS module to undo the update to the data base.
- If the record is a Prepare-to-Commit type there is no process because the data base was not updated when the Prepare-to-Commit record was written
- If the record is a begin work or a save work, ARIYT26 is called to free all locks up to the save point, to reset the USERPROB flags in the TPMAP (if the undo process is complete), to reset the scans and to determine if the undo process should continue.
- If the record is a begin work type, ARIYT20 is called to reset the USERPROB flags in TPMAP, to free all locks of the LUW, to write the abort log record for the LUW, and to clear work information in YTABLE1 and the TPMAP entry.

After the undo process completes, ARIYT19 calls ARIYT22 to clear the USERPROB flags and get the reason for the undo. Then, ARIYT19 clears some fields in the TPMAP entry and YTABLE1, forces a checkpoint if the LUW was aborted because of log full, and returns to the caller with the reason that the undo took place.

ARIYI06

ARIYI06 communicates with ARIYI07 for reading and writing a specific block from or to the directory. ARIYI06 is used to access all system blocks and page map table blocks for DBSPACEs.

ARIYI07

ARIYI07 is the I/O routine for all data base activity to the directory, log(s), or DBEXTENTs.

ARIYI09

ARIYI09 displays information about the occupied page and directory block buffers (SHOW BUFFERS operator command).

ARIYI10

ARIYI10 empties DBSPACE by releasing all its pages (for DROP DBSPACE).

ARIYI12

ARIYI12 finds the first one bit in a string of bits pointed to by Register 1 whose length is in Register 2. A mask contained in Register 0 is applied to the first byte of the string so those bits are not included in the scan. Returned to the caller is the address of the byte containing a one bit, or zero if no bits are found, in Register 1; the index of the first one bit in the string base zero in Register 15; and a mask for clearing the one bit in the byte referenced by Register 1 in Register 0.

ARIYI13

ARIYI13 initializes YTABLE4 with device constants.

ARIYI14

Entry Point: ARIYI14
Function: ARIYI14 gets a page map table block from the directory. Then ARIYI14 puts the table block into storage.

Entry Point: ARIYI141
Function: ARIYI14 gets a page map table block from the directory. Then ARIYI14 puts the table into storage (used inside the I/O component) with a DBSPACE number specified.

Entry Point: ARIYI142
Function: ARIYI142 records the modification of a directory block that is already in the buffer at BPTR.

ARIYI15

ARIYI15 displays information pertaining to storage pools for which is backed by DBEXTENTs (for SHOW DBEXTENT and SHOW SYSTEM operator commands).

ARIYI17

ARIYI17 locates a free page or directory block buffer by scanning the buffer controls from the most recently used to the end of the buffer chain, testing the fix count for zero. When one is found it is made the most recently used, the address of the buffer is placed in Register 2, and the fix count is set to one.

ARIYI19

ARIYI19 gets a page into real storage for access.

ARIYI20

ARIYI20 allocates a page slot from the storage pool to which the DBSPACE belongs.

ARIYI21

ARIYI21 initializes the buffers and tables needed by the DBSS Storage (I/O) component.

ARIYI22 - VM System-Dependent Module

ARIYI22 allocates all the VMCBLOCKs (device and agents) for the DBEXTENTs, log(s) and directory. It also allocates the VMCBLOCKs (device and agents), and the VMQ for the vector table. The directory is opened in this routine. ARIYI22 calls ARIYI23 (VM) to open all the other data base files (DBEXTENTs, log(s)).

ARIYI22 - VSE System-Dependent Module

ARIYI22 generates the ACBs and RPLs for the DBEXTENTs, log(s) and directory. The directory is open in this routine.

ARIYI23 - VM System-Dependent Module

ARIYI23 opens all the DBEXTENTS and logs.

ARIYI23 - VSE System-Dependent Module

ARIYI23 generates the ACB (on first call - function code = A) and the RPL for the DBEXTENTs and the log(s). For subsequent calls (FUNCTION code = R) just the RPL is generated for the agent structure being created.

ARIYI24

ARIYI24 retrieves pointers to the last system checkpoint from the master record for the Log component.

ARIYI26

ARIYI26 makes DBSPACEs available for processing. This routine is called for the purpose of opening all synchronous DBSPACEs at initialization, or opening all synchronous DBSPACEs after a rollback in nolog mode, or opening temporary DBSPACE after it has been emptied.

ARIYI29

ARIYI29 is the logical page access module.

ARIYI31

ARIYI31 is a procedure to read/write storage management control blocks (from/to the directory file).

Entry Point: ARIYI311
   Function: ARIYI311 reads the data base control block.

Entry Point: ARIYI312
   Function: ARIYI312 reads and writes master record.

Entry Point: ARIYI313
   Function: ARIYI313 reads or writes the copy of YTABLE2 indicated by byte YT2BYTE.

Entry Point: ARIYI314
   Function: ARIYI314 reads or writes block alternation vector from block pointed to by BLKALT in the master record.

Entry Point: ARIYI315
   Function: ARIYI315 reads or writes a summary bit map from block indicated by MBYTE in the master record.

ARIYI36

ARIYI36 is involved during system checkpoint.

Entry Point: ARIYI36
   Function: ARIYI36 is for saving asynchronous DBSPACEs, which are not supported by the RDS.

Entry Point: ARIYI361
   Function: ARIYI361 is for closing asynchronous DBSPACEs, which are not supported by the RDS.

Entry Point: ARIYI362
   Function: ARIYI362 is invoked by empty DBSPACE processing, COMMIT to release temporary DBSPACEs.

Entry Point: ARIYI363
   Function: ARIYI363 is invoked by empty DBSPACE processing, ABORT WORK to release temporary DBSPACEs.

Entry Point: ARIYI364
   Function: ARIYI364 is invoked by empty DBSPACE processing, SORT to release temporary DBSPACEs.

Entry Point: ARIYI365
   Function: ARIYI365 is called by system checkpoint. Checkpoint occurs but, DBSPACEs are still open when complete.

Entry Point: ARIYI366
   Function: ARIYI366 is called by system checkpoint. DBSPACEs are closed after completion.

ARIYI41

ARIYI41 obtains the DBSPACE lock for the DBSPACE specified in the input parameter INSEGM if input parameter K is non-zero. If input parameter K is zero, then the DBSI latch is obtained by calling ARIYK08 (RSSENTER) and INSEGM is ignored.

ARIYI42

ARIYI42 fetches the attribute for the specified DBSPACE in input paramter SN and returns the byte value in input parameter B.

Entry Point: ARIYI421
   Function: ARIYI421 returns the number of pages in the DBSPACE obtained from the SEGTABLE in input parameter NPG.

Entry Point: ARIYI422
   Function: ARIYI422 returns the number of pages and the storage pool number form the SEGTABLE in input parameter pool.

Entry Point: ARIYI423

Function: ARIYI423 is a special entry point for the log component which returns the number of pages in the log dataset in the input parameter NPG.

ARIYI45

ARIYI45 unchains the specified buffer (input parameter PTRBUFC) from the HASH table (input parameter THASHP).

ARIYI46

ARIYI46 decrements the fix count in the buffer contents entry associated with the buffer pointed to by input parameter P.

ARIYI47

ARIYI47 is called during SQL/DS SQLEND/SHUTDOWN process or during abnormal termination, to close the directory, log(s), and all DBEXTENTs files.

ARIYI48

ARIYI48 is a special routine for the log component to communicate with the I/O component for the purpose of writing a page of log data to the log data set.

ARIYI49

ARIYI49 reads a log page specified by input parameter PAGENO into a buffer pointed to by input parameter BUFFADDR.

ARIYI50 - VSE System-Dependent Module

ARIYI50 is provided to VSAM to give control to after the EXCP for the I/O operation has been issued and before the wait for I/O completion. ARIYI50 tests to see if the I/O has completed and if not, puts the SQL/DS task (AGENT) in an I/O wait condition. When the I/O completes, control is given to this routine who returns control back to VSAM.

ARIYI51

ARIYI51 is called at initialization and after every checkpoint to see if any storage pool has less free pages than the cushion. The cushion = the total number of pages in the pool * YSOSLEV / 100.

ARIYI52

ARIYI52 is the trace point descriptor module for the Storage (I/O) component. It contains all trace point descriptors for all trace points contained in modules of the DBSS Storage (I/O) component. This module contains all static data declares required to describe and process trace points in the range 2000 through 2299.

ARIYK00

ARIYK00 releases access to the designated lock (gate). If the gate is held in medium duration, then the count is decremented and the gate is not released unless the new count is zero. Long locks are not released by this module.

ARIYK01

ARIYK01 (YLOCK) is the DBSI call to lock a DBSPACE, table, or a row.

ARIYK02

ARIYK02 releases a designated named lock owned by the caller.

ARIYK03

ARIYK03 determines if the named gate is held by anyone.

ARIYK04

ARIYK04 waits for access to X-LATCH.

ARIYK05

ARIYK05 releases access to X-LATCH and post all waiters for it.

ARIYK06

ARIYK06 waits for access to a S-LATCH mode in either 'S' for share or 'X' for exclusive.

ARIYK07

ARIYK07 gives up access to a S-LATCH and if a S-LATCH is now free, ARIYK07 posts all waiting requestors.

ARIYK08

ARIYK08 raises the DBSS wait flag and calls ARIYK18 which causes a wait, if the identified agent is stopped (that is, its RSSTOP (in TPMAP entry) bit is on). Corresponding post will be done by ARIYK11. ARIYK08 turns on INRSS (in TPMAP entry), the bit in the STATE byte of this LUW which indicates that this LUW is inside the DBSS.

ARIYK09

ARIYK09 marks the identified agent as not inside the DBSS and posts any agent which is waiting to stop the identified agent.

ARIYK10

ARIYK10 is called only by ARIYL08 the Checkpoint Routine. Its purpose is to cause all agents to leave the DBSS. Prior to this call to ARIYK10, ARIYL08 called ARIYK08 which turned on the INRSS bit in the state byte of ARIYL08.

ARIYK11

ARIYK11 turns off the stopped flag and resets the Stopper ID of all LUWs except the caller, so that they can continue executing the DBSS. Posts any agent whose LUW is in a DBSI wait.

ARIYK12

ARIYK12 looks for a deadlock cycle and chooses a victim in each one that is found. Cycles are detected by a recursive search of the lockwait graph.

ARIYK18

ARIYK18 waits to be posted and returns immediately if post waiting. ARIYK18 then sets LUW's state bit according to WAITTYPE and calls ARICDWT, Dispatcher Wait Routine. ARIYK18 clears WAITTYPE from the LUW's state bit.

ARIYK19

ARIYK19 is the Lock Manager Post routine.

ARIYK21

ARIYK21 resets a lock, clears LRBKEEP bit, sets LRB duration to long, and changes modes: (SIX | S | X) to S and IX | IS to IS; unless it is a lock for a temporary DBSPACE.

ARIYK22

ARIYK22 is called when the agent or SQL/DS has run out of lock request blocks (LRBs). An attempt is made to salvage and to consolidate LRBs owned by the caller by finding the DBSPACE containing the most locks. Then, the DBSPACE is acquired in a mode which covers all these locks and subsidiary locks are released. Then, the lock cache in YTABLE1 is edited and if the requested lock is not subsidiary to the DBSPACE, it requests the lock that caused all the trouble. The return code is positive and gives the granted mode or is negative and caller should back up.

ARIYK23

ARIYK23 requests that the named gate be kept beyond LUW commit and sets the keep bit on in the requestor's LRB for this lock.

ARIYK24

ARIYK24 sets the keep bit on all locks of a caller. ARIYK24 is used to keep all locks beyond LUW commit.

ARIYK25

ARIYK25 clears the keep bit on any locks and unlocks any unkept locks except the DB lock. ARIYK25 then sets duration of kept locks to long and downgrades mode of kept locks X | SIX | S to S and IX | IS to IS. ARIYK25 does not downgrade DBLOCK or temporary DBSPACE locks.

ARIYK26

ARIYK26 sets the keep bit on all locks of temporary DBSPACEs held by the user.

ARIYK27

ARIYK27 clears the keep bit on any locks. ARIYK27 is used to back out of a commit call and it undoes any ARIYK24 or ARIYK23 calls.

ARIYK28

ARIYK28 is the DBSI call to release a lock on a row.

ARIYK37

ARIYK37 displays lock status at the request of the SHOW LOCK command.

ARIYK38

ARIYK38 retrieves the DBSPACE header of INSEG and copies it into YTABLE1. ARIYK38 releases all acquired buffers.

ARIYK39

ARIYK39 loops through all child links and locks the DBSPACE where parent is a different DBSPACE.

ARIYK40

ARIYK40 obtains the DBSPACE attributes, lock, and header page for the input DBSPACE. If the header page is fetched, update the DBSPACE data in YTABLE1.

ARIYK41

ARIYK41 requests access to a named gate in the designated mode for the specified duration. Control specifies whether or not the requestor wants to wait for the request to be granted or to cancel the request if the request must wait.

ARIYK42

ARIYK42 is used for initialization for Lock component.

ARIYK43

ARIYK43 is the trace point descriptor module for the Lock component. It contains all trace point descriptors for all the trace points contained in modules in the Lock DBSS component. All trace point descriptors in this module have trace point numbers in the range 600 through 799.

DBSS/DSC LINK MAP MODULES

ARIYLNK

ARIYLNK is the DBSS/DSC link map module for both VSE and VM. ARIYLNK may be found in a dump, and shows the module name and corresponding link address of each DBSS/DSC module.

For VM only: ARIYLNK contains the first 200 DBSS/DSC module names and link addresses. The remaining entries are in ARIYLN1 (255 is the maximum allowable number of VCONS in one text module).

ARIYLN1 (VM Only)

ARIYLN1 is part 2 of the DBSS/DSC link map module, and contains entries after the first 200 that were found in ARIYLNK. ARIYLN1 may be found in a dump, as can ARIYLNK.

ARIYL00

ARIYL00 initializes the log. There are three types of initialization:

Parameter COLDFLAG = 'C' - Cold start the data base
= 'L' - Cold start the log only
= 'W' - Warmstart the log

ARIYL01

ARIYL01 starts a new log record of the specified length and type. (Called via YSTARTLG macro.)

ARIYL02

ARIYL02 writes data into the current log record. (Called via YWRITELG macro.)

ARIYL03

ARIYL03 terminates the writing of a log record with or without flushing the current page. (Called via YENDWRIT macro.)

ARIYL04

ARIYL04 begins the reading of a log record that begins at segment address SADDR. It returns the header of the record in LHBUFF and the (absolute) segment address of the previous record in PREDADDR. (If the record is of type BEGIN_LUW, PREDADDR will be meaningless.) (Called via YSTARTRD macro.)

ARIYL05

ARIYL05 reads the specified number of bytes from the current log record. (Called via YREADLOG macro.)

ARIYL06

ARIYL06 moves the log cursor delta bytes forward or backward. It also fixes the resulting log page in the page buffer. (Called via YMOVECUR macro.)

ARIYL07

ARIYL07 is the linkage between the work management and recovery subsystems and the various component data base update recovery modules.

ARIYL08

ARIYL08 performs a system checkpoint. This causes the DBSS-consistent state of all public and private DBSPACEs and the log to be saved, facilitating later recovery. If an archive is required, module ARIYL23 is called (reason = 'B').

Valid reasons are:
'I' - Log just initialized
'R' - System just recovered
'C' - Periodic or shutdown system checkpoint begin taken
'B' - Archive of the data base started
'A' - Archive of data base completed

ARIYL09

ARIYL09 is the SQL/DS system checkpoint routine. A test is made to determine if a checkpoint (or checkpoint and archive) is required. If a checkpoint is required, ARIYL08 is called. The test is then repeated, and now a checkpoint is not required. If a checkpoint is not required, ARIYK18 is called to cause a SQL/DS wait of type 'CHECKPOINT'. If it is a termination checkpoint or archive ARICTRM is called to end SQL/DS.

ARIYL10

ARIYL10 advances to the next log record during sequential READ SCAN. It returns the starting address in the log and header of this record.

ARIYL11

ARIYL11 fills the page, beyond the end of the log, with zeros. ARIYL11 is used by ARIYL13 to erase fractional log records at restart.

ARIYL12

ARIYL12 updates a fullword in a log record. The fullword does not have to be on a fullword boundary.

ARIYL13

ARIYL13 recovers SQL/DS from a system crash which did not involve media failure. It scans the log determining which logical units of work must be undone or redone. Then, those LUWs are undone or redone. Finally, agents which were prepared for commit or rollback when the crash occurred, are recreated.

ARIYL14

ARIYL14 changes the log in some way, for example, add a secondary log file, remove a secondary log file, or replace a log file with a new log file. The log file is always completely reformatted.

ARIYL15

ARIYL15 displays the status of the log (for SHOW LOG and SHOW SYSTEM commands).

ARIYL19

ARIYL19 checks that the specified log pages are the same for the primary and secondary log (if no secondary, then return). If the pages are not the same, then scan the entire log, bringing each pair of pages into agreement.

ARIYL21

ARIYL21 controls the restoring of the block disk (directory) from the archive tape(s).

ARIYL22

ARIYL22 controls the restoring of the data disks (DBEXTENTs) from the archive tape(s).

ARIYL23

ARIYL23 controls the archiving process. It is called by ARIYL08 whenever an archive checkpoint (B type) has been requested and taken. The archive writing takes place as a function of the checkpoint agent, that is, this program runs under the checkpoint agent. This is possible since all checkpointing will be inhibited during the writing of an archive tape, thus ensuring the consistency of the archived data base. ARIYL23 is called after other logical units of work have been allowed to run. That is, ARIYK09 and ARIYK11 have been called by ARIYL08.

ARIYL24

ARIYL24 causes ARIYL08 to be activated, requesting a 'B' checkpoint and in turn the activation of ARIYL23 (perform archive).

ARIYL25

ARIYL25 is the trace point descriptor module for the Log/Recovery component. It contains all trace point descriptors for all trace points contained in modules of the Log/Recovery DBSS component. All trace point descriptors in this module have trace point numbers in the range 300 through 599.

ARIYM00

ARIYM00 is the DBSS entry point for executing DBSS operation codes. It determines what module entry point should be called to execute the operation code. Then ARIYM00 calls that entry point, passing the pointer to the operation parameter structure. On entry and on exit, tests are made for DBSS ENTRY and DBSS EXIT tracing. If that trace function is active, it makes a call to perform the trace function. Also, on entry and exit, a test is made to determine if the current DBSS logical unit of work has been flagged for asynchronous termination. If so, calls are made to DBSS routines to rollback the logical unit of work and to obtain the cause (for return code). Also the operation code execution is bypassed.

For each operation code call, a test is made to determine if a logical unit of work needs to be created for the operation (and possible subsequent operations). If so, a call is made to the DBSS BEGIN WORK function (implicit begin work).

ARIYM01

ARIYM01 is invoked by ARICIP2 (via the SQL/DS Dispatcher) to complete the SQL/DS initialization process.

In multiple user mode, ARIYM01 establishes operator communications (via STXIT OC) in the VSE environment, and issues the "Ready for Operator Communications" message in the VM environment. In both environments it also issues the "SQL/DS Initialization Complete" message.

In single user mode, only the initialization complete message is issued. ARIYM01 then calls the SQL/DS operator command processor entry module ARIYM10. ARIYM10 in turn calls the SQL/DS Dispatcher to wait on the operator agent. (In single user mode this is a 'permanent' wait condition for the operator agent.)

ARIYM02

ARIYM02 is the SQL/DS system error routine. ARIYM02 terminates SQL/DS with a call to ARICABE. It prints a character string that identifies the caller and the calling point (ARIxxxx nn). The caller is the module name (ARIxxxx). The calling point is a decimal value (nn). If the calling component is terminating due to a limit error problem (for example, a storage request cannot be satisfied due to insufficient storage), it sets the limit error indicator (DSCLIMER). This indicator causes ARIYM02 to set the "pass the limit error" return code to the SQL/DS termination routine (ARICTRM). A hardware error is treated in the same manner as a limit error. All other errors are treated as severe SQL/DS errors. This type of error condition causes a call to the SQL/DS abend handler routine (ARICABE). The error is treated as a condition similar to a program check condition. However, it sets the DSCTRACK flag (DSCTRSER) to indicate that the SQL/DS abend handler was called by this routine and to cause it to skip the displaying of information such as the abnormal termination code and the PSW. ARIYM02 initializes the SQL/DS abend savearea prior to calling the SQL/DS abend handler routine. It places the registers of the calling routine into the abend savearea and the PSW that points to the BALR instruction of the calling routine. The abend save area information is moved into a mapping of the CMS abend save area. ARIYM02 sets register 0 (abend code) to 0. For VSE, the pointer to the SQL/DS abend save area is set in register 1. For VM, the pointer to the mapping of the CMS abend save area is set in register 1. The SQL/DS abend handler is called, simulating the linkage similar to the operating system calling the SQL/DS abend handler, for both VSE and VM.

Note: As this is a terminal error condition and only messages and dump data is displayed, all calls to the SQL/DS dispatcher are suppressed. Therefore, the 'DS2WTOWT' bit is turned off to tell the operator communication routine not to call the SQL/DS Dispatcher Wait routine.

ARIYM03

ARIYM03 tests whether a given string is 'like' a given pattern. The string to be compared to the pattern, and its length, are supplied as parameters 1 and 2. The pattern, and its length, are supplied as parameters 3 and 4. The string is 'like' the pattern if the characters of the pattern can be placed in correspondence with the characters of the string. A '%' character in the pattern may correspond to an arbitrary sequence of 0 or more characters. A '_' character in the pattern may correspond to any single arbitrary character.

ARIYM04

ARIYM04 is invoked to:

1. Write a DBSS, DSC, or RDS message to the operator (VSE system operator or virtual machine terminal operator, or ISQL) and optionally read a reply (both VSE and VM) and/or write the message to the system print file (VSE only) or
2. Write a resource manager message to the VSE system operator or virtual machine terminal and optionally read a reply.

ARIYM04 invokes the message formatter for message retrieval and variable substitution. For multi-line messages, ARIYM04 loops displaying each line of the message.

ARIYM05

ARIYM05 is the message formatter module. It is invoked to retrieve a message (or line of a multi-line message) and to substitute any caller provided message variables into the message (or line). Using the caller provided message number or SQL code, the message index or SQL index table is searched to locate the address of the message module containing the message. It is the twin module of ARIMFMT and ARIIFOR (used in load modules, other than ISQL, which have stack storage). See ARIMFMT for more detailed description.

ARIYM10

ARIYM10 is the DBSS operator command linkage. In multiple-user mode SQL/DS:

ARIYM10 provides the linkage to the SQL/DS operator command processing routines from either the ISQL or system operator consoles. The system or ISQL operator is notified of the success or failure of the command processing.

In single-user mode SQL/DS:
ARIYM10 places the operator agent structure in permanent SQL/DS dispatcher wait state at the completion of SQL/DS initialization.

Entry Point: ARIYM10
Function: In multiple-user mode, ARIYM10 processes SQL/DS operator commands from the system operator console.

Entry Point: ARIYM101
Function: In multiple-user mode, ARIYM101 processes SQL/DS operator commands from ISQL terminal users.

ARIYM11

ARIYM11 provides the linkage between the DBSS operator command linkage (DOCI) module (ARIYM10) and the SQL/DS operator command processing modules. Also, the shownamе (operand one) of SHOW commands is validated.

ARIYM50

ARIYM50 writes a display line to the system operator console (VSE) or to the virtual machine terminal.

ARIYM51

ARIYM51 writes a message to the system operator console (VSE) or to the virtual machine terminal and reads an operator reply. The reply buffer is cleared to blanks before the I/O. The buffer is also translated to upper case after the I/O (folding lower case alphabetics to upper case).

ARIYM52

ARIYM52 obtains the date and time in printable format. The date and time are obtained from the operating system supervisor. The date is in either mm-dd-yy or dd-mm-yy format (VSE), or mm-dd-yy format only (VM). The time is in hh:mm:ss format (the VSE ZONE JCS adjusts time to local time as displaced from Greenwich Mean Time).

ARIYPAT

ARIYPAT provides a common patch area for DBSS and DSC modules residing in the SQL/DS partition.

ARIYSTP

ARIYSTP is the DBSS trace point descriptor module for the Sort component. It contains all trace point descriptors for all trace points contained in modules of the DBSS Sort component. The trace point numbers are in the range 2300 through 2353.

ARIYS00

ARIYS00 sorts a set of rows from a table or a sequential list using an opened scan to identify the rows.

Entry Point: ARIYS001
  Function: ARIYS001 sorts index pages using log information as opposed to sort base structure during backup.

Entry Point: ARIYS002
  Function: ARIYS002 is used to do an internal sort call for index creation.

ARIYS01

ARIYS01 checks whether the submitted search clause is satisfied by the row pointed to by Register 3.

ARIYS02

ARIYS02 inserts a row T in re-do or deletes a row T in un-do and backup using logged information. ARIYS02 reads the logged data and advances the log cursor for an amount equal to LOGL.

ARIYS03

ARIYS03 obtains, in TLTH, the length of the row to be inserted. It also checks submitted domain lengths, acceptability of NREQDOM, and total row length.

ARIYS04

ARIYS04 moves the user row (domains) to an output list page. It also updates the page header and completes the row with undefined fields, if NREQDOM < YDEGREE.

ARIYS05

ARIYS05 moves requested field values and lengths from a row in entity space to the user domains.

ARIYS08

ARIYS08 repeats a complete sort in re-do and empties a DBSPACE in un-do or backup using logged information. ARIYS08 reads the logged data and advances the log cursor for an amount equal to LOGL.

ARIYS09

ARIYS09 fills a sequential list with rows which are stored in an internal storage area.

ARIYS10

ARIYS10 fills an intermediate page with encoded rows which are stored in an internal storage area.

ARIYS11

ARIYS11 writes the header page for a DBSPACE containing a sequential list.

ARIYS12

ARIYS12 assembles a sort key in IKEY using fields from a stored row (list or table row) pointed at entry by TUPLEPTR.

Entry Point: ARIYS121
  Function: ARIYS121 assembles a sort key in IKEY using fields from a stored row (list) pointed at entry by TUPLEPTR.

ARIYS14

ARIYS14 encodes a row, according to the sort specifications, into an internal structure using fields from a stored row pointed at entry by TUPLEPTR.

Entry Point: ARIYS141
Function: ARIYS141 is used with an optimized index.

Entry Point: ARIYS142
Function: ARIYS142 is used with a list scan.

ARIYS17

ARIYS17 decodes the single domains stored in IKEYO and encodes them again in IKEY according to the sequence and order specification of SORTSPEC. The last field of the sort key is not encoded if variable.

Entry Point: ARIYS17
Function: ARIYS17 allows SORT the ability to use information in an index page without needing to go to the data page. All the fields needed by the sort are in the index key.

ARIYS18

ARIYS18 sorts the random ordered variable length rows using the vector TOSORT. The rows are stored contiguously in an internal DBSPACE page.

ARIYS19

ARIYS19 sorts the random ordered fixed length rows using the vector TOSORT. The rows are stored contiguously in an internal DBSPACE page.

ARIYS20

ARIYS20 merges a number of strings (NSTRINGS) stored in a temporary DBSPACE (INTSEGM).

ARIYS21

ARIYS21 decodes and reorders a row from its internal representation for the sort to its final format for the sequential list.

ARIYS22

ARIYS22 writes the header to a filled output page, unfixes this page and gets a new output page if the current output string creation continues. It calls the unfix of the output directory block buffer only at string end and the transition to a new directory block. The second acquired temporary DBSPACE (if any) is released.

ARIYS23

ARIYS23 copies a total string of an intermediate input DBSPACE into a sequence of pages of an intermediate output DBSPACE fetching page by page. The page header description is changed and the unfixing of pages and directory blocks is handled by this module.

ARIYS24

ARIYS24 achieves the input loop of the sort by means of a scan, fills the qualified rows into a internal DBSPACE page, sorts that page indirectly and stores the sorted rows into pages (strings) of an intermediate DBSPACE.

Entry Point: ARIYS24
Function: ARIYS24 is used as an entry for link or index scan (index and data pages touched).

Entry Point: ARIYS241
Function: ARIYS241 is used as an entry for the optimized index case (only index pages touched).

Entry Point: ARIYS242
Function: ARIYS242 is used as an entry for a table scan.

Entry Point: ARIYS243
Function: ARIYS243 is used as an entry for a list scan.

ARIYS25

ARIYS25 traces a SORT call on DBSS entry. It displays the base control information and (depending on the call) SARGS, KDOMAINS, SORTLIST and SORTSPEC.

Entry Point: ARIYS251
Function: ARIYS251 traces a sort call on DBSS exit. It displays the base control information.

DBSS WORK

ARIYT00

ARIYT00 is called by DSC at SQL/DS initialization time. It initializes the DBSS subcomponents: work, lock, counter, storage and log. Also, ARIYT00 allocates a TPMAP entry for the initialization (operator) agent by calling ARIYT05. If trace was specified, ARIYT00 initializes the trace subcomponent of DSC by calling ARICTRI. If the data base is to be restored from an archive, the module ARIYL22 is called. If DBEXTENTs are to be added to the data base, ARISDSK is called. If DBSPACEs are to be added, ARISEGA is called.

ARIYT01

ARIYT01 is the router for the SQL/DS SHOW operator commands. It routes as follows: for SHOW LOG, call ARIYL15; for SHOW SYSTEM, call ARIYE37; for SHOW ACTIVE, call ARIYT12; for SHOW LOCK, call ARIYK37; and for SHOW other, display message ARI067I.

ARIYT02

ARIYT02 is called by ARIYT00, and sets the entire TPMAP to binary 0.

ARIYT05

ARIYT05 is called when an agent is built. It assigns a TPMAP entry for the agents. Additionally it sets: the index of the entry (WHOAMI) in YTABLE1, the pointer to the entry (WHOAMIP) in YTABLE1, and the PROCID of the entry is set to point to DSCAREA.

ARIYT06

ARIYT06 is called by RDS when an application specifies RELEASE as part of EXEC SQL COMMIT WORK or EXEC SQL ROLLBACK WORK. It is also called by DSC after an agent is reset because of termination. ARIYT06 is a no-op if the agent is still in LUW. Otherwise, ARIYT06 resets the TPMAP entry of the agent and releases overflow working storage by calling ARICWFR.

ARIYT08

ARIYT08 provides level 2 trace output for DBSS Entry and DBSS Exit tracing of work (LUW) component DBSS operation codes (and the process operator command DBSS operation code).

ARIYT09

ARIYT09 begins a new logical unit of work for an agent. This consists of: assigning a new LUW identifier from NEXTID of the YRSSCVT, initializing new LUW data into the TPMAP entry and YTABLE1, and writing the Begin Work log record.

ARIYT10

ARIYT10 establishes a save point for an LUW. (In general, rollback work undoes all updates for a LUW up to the begin work log record. When a save point is established, after some updates, rollback work could be requested to undo work up to the save point.) The save work linkage is not used. There is no SQL external to establish save points.

ARIYT12

ARIYT12 processes the SHOW ACTIVE command. See the SQL/DS Operations, listed in Preface, for a description of the command.

ARIYT14

ARIYT14 is the commit work process module.

ARIYT15

ARIYT15 is called when DSC or DBSS asynchronously detects that a rollback or commit should occur. If ARIYT15 detects that the agent is still in DBSS (RSSFLAG = 'Y') the call is a no-op. Otherwise:

- The EDSF (Extended Dynamic Statements Facility) termination support is invoked to allow proper termination of an access module (AUX) build or update.
  ARIYT22
  - ARIYT22 is called to determine if a rollback should occur. The return code from ARIYT22 indicates whether a rollback or commit should be done.

- ARIXERD1 (secondary entry point in ARIXERD) is called with an indicator as to whether a rollback or commit should be performed for the AUX build or update. The return code from ARIXERD1 indicates the following:
  0 - termination of the AUX was successful (or no AUX to terminate).
  4 - termination of the AUX failed, but the rollback was done.
  8 - termination of the AUX failed, and a rollback should be done. In this case, ARIYT23 is invoked to ensure the rollback flags are set properly.
- Set RSSFLAG = 'Y' at the beginning of the process and to 'N' at the end of the process.
- If USERPROB has SYSBACK = Set (1) or USERBACK = Set (1) achieve rollback by calling ARIYT19. Set SYSBACK or USERBACK to KO (2) so that ARIYT22 will be able to return the reason for the rollback. Return
- If TPMFORS is on and if the LUW is in work, achieve commit by calling ARIYT14. Reset TPMFORS and return.

ARIYT16

ARIYT16 is called when a LUW needs to be aborted because of an operator request or the log is full. The process is as follows:

- If the unit is already being rolled back the call is a no-op. (This prevents recursion of the message ARI230I.) Otherwise.....
- Schedule the rollback or commit by calling ARIYT23.
- If the agent is connected (linked) to an application and there is no request to break the connection, set DCERESET = '1'B and TPMREMDC = '1'B. DCERESET tells DSC that the agent reset should only include LUW abort and storage reset. TPMREMDC tells the work component to not "remember" (not clear out) the DBSSCODE which is the reason for the abort. This is saved so that the next SQL call from the application could return the appropriate SQL code.

  If the request includes a request to break the connection, set DCERESET = '0'B and DCEFORCE = '1'B. DCEFORCE tells the DSC component to disconnect and reconnect the communications link for an indoubt LUW.
- Set DCELKTRM = '1'B to tell the SQL/DS dispatcher to reset the agent after it detects that the scheduled abort has completed.
- If the agent is waiting on communications or a lock, the wait is cleared.
- Display message ARI230I.

ARIYT17

ARIYT17 processes DBSI requests to ROLLBACK WORK for a LUW. It uses ARIYT23 to establish the reason for the rollback (user request) and to establish the save point. ARIYT17 uses ARIYT19 to undo the updates for the LUW.

ARIYT18

ARIYT18 verifies the validity of the input parameters and schedules the rollback or commit of a logical unit of work by calling the abort routine, ARIYT16. ARIYT18 passes the following data to ARIYT16:

- TYPE which indicates whether or not the link should be disconnected from the application after rollback.
- Binary agent identifier
- Reason for the abort = 'OPERATOR REQUEST'
- The abort direction (COMMIT or ROLLBACK).

ARIYT19

ARIYT19 undoes the updates to the data base of a LUW. It is called primarily by ARIYT17 as a result of an application's ROLLBACK WORK request. Additionally, it is called by various SQL/DS modules when an LUW must be rolled back because of operator request or because of system problems.

ARIYT20

ARIYT20 resets the USERPROB in the TPMAP entry of the LUW, frees all locks, uses the log component to allocate and write an abort type log record, and clears work component information in the TPMAP entry and YTABLE1. ARIYT20 is called as part of the un-do work process.

ARIYT21

ARIYT21 is called as part of the un-do work process when a save point or begin work record is reached. Its primary function is to determine if the un-do process needs to continue. It does this by comparing the save point number in the log record with RESTORTO value in TPMAP. In general, the RESTORTO value is 0 indicating that the undo process should continue until the begin work log record. The RESTORTO value will remain 0 unless ARIYT10 was invoked.

ARIYT22

ARIYT22 is called to determine if a rollback should occur, and (if a rollback just completed) to give the reason for the rollback. ARIYT22 processes the UNDOFLAGS of the TPMAP entry as follows:

- Set SIPROB to 0. This is because all conditions of SIPROB are detected and cleared as part of the DBSI call.
- Process the asynchronous data of USERPROB as follows:
  - If flags are CLEAR (0), return with return code = 0.
  - If any one flag is SET (1), return with return code = -1. This tells the caller to start or continue a rollback process.
  - If any one flag is KO (2), set flag to CLEAR (0) and return the reason for the rollback.

ARIYT23

ARIYT23 sets the flags so that rollback or commit will occur at the appropriate time. The process is a no-op if the agent is not in LUW. For the COMMIT request the process is:

- If LUW is already being rolled back, the call is a no-op.
- If the agent belongs to a subsystem (SQL/DS online support) process the request as a ROLLBACK.
- If the agent is currently processing a request for the application, process the request as a rollback.
- Otherwise, set TPMFORS = '1'B, and return. Setting the flag will cause the next call to ARIYT15 to cause the LUW to be COMMITed.

ARIYT26

ARIYT26 gets recovery information, as defined by the control block RMRE, for the SQL/DS online support.

ARIYT27

ARIYT27 has the function of logging enough LUW information so that, in the event of a system or subsystem failure, the agent may be rebuilt at recovery time. The agent is rebuilt in a state that is prepared for a commit or rollback request which could be entered with the FORCE operator command.

ARIYT28

ARIYT28 is called via the DBSI after RDS establishes a new userid for an application. It saves the new userid in YTPMAP and DSCAREA. Additionally, if the userid has schedule authority, the TPMAP entry for the agent is marked as "belonging to a subsystem". This prevents the DISABLE option of the FORCE operator command.

ARIYT29

ARIYT29 is called by the warm start recovery process (ARIYL13) after all active LUWs that are not Prepared-to-Commit have been redone (COMMIT) or undone (ROLLBACK) by SQL/DS recovery. LUWs that are Prepared-to-Commit are coordinated by a subsystem such as CICS/DOS/VS. That means that SQL/DS may not determine their ending status (COMMIT or ROLLBACK). To SQL/DS, that status is said to be "indoubt."

For each LUW that SQL/DS is indoubt about, an indoubt agent structure is built by ARIYT29. The agent structure is built from information on the log and reflects the LUW in much the same way as a general agent structure did prior to SQL/DS termination. The indoubt agent structure represents the LUW until the coordinating subsystem or the SQL/DS operator requests a COMMIT WORK or ROLLBACK.

ARIYXTP

ARIYXTP is the SQL/DS trace point descriptor module for the Index component. It contains the trace point descriptors for all trace points in modules of the SQL/DS Index component. The trace point numbers are in the range 2600 through 2660.

ARIYX00

ARIYX00 allocates space for a new SCB in the Scan Table, and chains it off the old SCB specified in caller parameter SCBPT.

ARIYX01

ARIYX01 deletes a key/TID pair from the index.

ARIYX02

ARIYX02 drops an index by freeing its DBSPACE pages.

ARIYX03

ARIYX03 updates the index SCBs which are affected by the deletion of the key, TID pair in IKEY and ITID.

ARIYX04

ARIYX04 fills the index pages described by the Index Control Record (ICR).

ARIYX05

ARIYX05 is called only when doing 'next row' processing inside the DBSS via an index scan.

ARIYX06

ARIYX06 decodes a string from the format that the Sort component prefers to the format that the Index component prefers.

ARIYX07

ARIYX07 finds an unallocated index page, allocates it, gets it in the buffer, returns its page number in NPAGE, and its page and directory block pointers in Register 2 and BLKPTR.

ARIYX08

ARIYX08 fills in some information into the header of an index page.

ARIYX09

After an index being created is filled, ARIYX09 reflects the last key in the index leaf page onto the next level of an index non-leaf page. To accomplish this, ARIYX09 inserts the appropriate key/TID pair(s) into the index non-leaf page indicated in the input parameter level.

ARIYX10

ARIYX10 cleans up the non-leaf pages of the index just created by ARIYX02 and ARIYX09.

ARIYX11

ARIYX11 finds an unallocated index page, allocates it, locks it exclusive short, gets it in the buffer, and returns its page number in NPAGE and the buffer pointers in Register 2 and BLKPTR.

ARIYX12

ARIYX12 finds the next key/TID pair in the index. The current key/TID pair is in IKEY0 and ITID, version number is in VNUM in YTABLE1, and physical information (page, offset) is also in YTABLE1. If it is not already in the buffer, the page under the cursor is brought in. If ARIYX12 advances to a new key value, the variable NEWKEY is set to one.

Entry Point: ARIYX121
  Function: ARIYX121 finds the next pair in the index. It assumes the caller has put the page in the buffer and validated the physical information.

ARIYX14

ARIYX14 inserts data into an index leaf page, at GOODPTR.

ARIYX15

ARIYX15 inserts a (key, son) pair into an index non-leaf page.

ARIYX16

ARIYX16 updates the index SCBs which are affected by the insertion of the key/TID pair in IKEY and ITID.

ARIYX17

ARIYX17 maps the key supplied in LKEY into a gate name, and inserts it into YTABLE1, so that key value locking can be used.

ARIYX18

ARIYX18 inserts a key into an index.

ARIYX19

ARIYX19 allows more room in the SCB by allocating a larger SCB. ARIYX19 repositions the scan under the SCBPTR so as to be longer.

ARIYX20

ARIYX20 returns the next key/TID pair in sequence.

ARIYX21

ARIYX21 returns a key/TID pair that is either the first pair or is related to the input key.

ARIYX22

ARIYX22 updates the Scan Control Block (SCB) table so that SCBs which have been adjusted by a preceding insert or delete (via ARIYX03 and ARIYX16) are properly updated. ARIYX22 frees the new SCB which is chained off the old one, and is used only for backup.

ARIYX23

ARIYX23 creates a new Scan Control Block (SCB) or updates an old one. If the callname in YTABLE1 is 'OPEN' then ARIYX23 allocates a new SCB and sets it to a value established by a previous call to ARIYX21 (with SAVECODE = 'S'). If the callname is 'NEXT' then ARIYX23 updates an existing SCB (specified in ISCANID) from a temporary cursor which was established by a previous call to ARIYX20.

ARIYX24

ARIYX24 searches a leaf page of an index if the keys are fixed in length.

ARIYX25

ARIYX25 searches a nonleaf page of an index if the keys are fixed in length.

ARIYX26

ARIYX26 searches a leaf page of an index if the keys are variable in length.

ARIYX27

ARIYX27 searches a nonleaf page of an index if the keys are variable in length.

ARIYX28

ARIYX28 updates the SCB table so that any SCBs which have been adjusted by a preceding insert or delete (via ARIYX03 and ARIYX16) are properly updated. ARIYX28 replaces the old SCB with the new SCB chained off the old one, thus completing a normal DBSI call.

ARIYX29

ARIYX29 takes care of inserting a key/TID pair into an index in the special case that the current page (whose ID is PAGEID, pointed at by Register 2, BLKPTR) is too full. ARIYX29 obtains a free page from the DBSPACE and moves some of the key/TID pairs from the current page onto the new page. The parent page must be modified: one key must be inserted. This of course may cause another split. To avoid un-doing modifications when backup is signalled, ARIYX29 places the necessary exclusive short locks on the index before modifying any page. GOODPTR points at the place where the submitted key/TID pair is to be inserted. No pages are fixed when ARIYX29 returns.

ARIYX30

ARIYX30 finds the next larger key value to the one found under PAIRPTR in the page in the buffer pointed to by Register 2.

DBSS UPDATE STATISTICS/COUNTER COMMAND

ARIYZTP

ARIYZTP is the DBSS trace point descriptor module for the DBSS Update Statistics component. It contains the trace point descriptors for all trace points contained in modules of the Update Statistics DBSS component. It also contains trace point descriptors for Service Temporary trace point for all DBSS/DSC modules. The trace point numbers for DBSS Update Statistics are in the range 2900 through 2942. The trace point numbers for Service Temporary trace point are in the range of 9900 through 9999.

ARIYZ00

ARIYZ00 sets up the Update Statistics routines.

ARIYZ01

ARIYZ01 checks that BEGIN WORK was issued, checks the addressing boundaries of the user LDBASE structure, checks that attention flag was not set, and initializes a number of YTABLE1 variables.

ARIYZ02

ARIYZ02 creates TREESPEC and LINKINFO structures.

ARIYZ03

ARIYZ03 checks that the specified table (RID) is valid; that is, it has been defined in DBSS's catalog, the ordering path to it is valid, and it is the child relation of the given access path. It sets values of TEMPCPRE, TEMPPPRE, PARNTSEG, PARNTREL and TEMPAREA.

ARIYZ04

ARIYZ04 obtains the DBSPACE attributes and header pages for an input DBSPACE INSEGM and updates the DBSPACE data in YTABLE1. It is designed for multi-DBSPACE processing in the LOAD/UNLOAD facilities.

ARIYZ05

ARIYZ05 creates the SEGMDESC structure.

ARIYZ06

ARIYZ06 updates the internal statistical tables maintained by the LOAD/UNLOAD facilities.

Entry Point: ARIYZ061
   Function: ARIYZ061 initializes internal statistics related tables.

Entry Point: ARIYZ062
   Function: ARIYZ062 Updates statistics from a row in the DBSPACE scan of data pages (MODEFLAG = 'U') during 'STARTUS' call.

Entry Point: ARIYZ063
   Function: ARIYZ063 completes internal statistics tables.

ARIYZ10

ARIYZ10 unfixes all directory block and page buffers and moves the submitted return code INCODE to LDRCODE in user LDBASE structure if it is not ILLEGALP. It also ABENDs SQL/DS if there is an I/O error.

ARIYZ11

ARIYZ11 does a DBSPACE scan for every DBSPACE in which there are tables for which statistics are to be gathered, setting up ORPHTID.

ARIYZ15

ARIYZ15 is the DBSI call module to complete an UPDATE STATISTICS. ARIYZ15 finishes statistics collection processed by ARIYZ06, collects index statistics if requested, returns statistics to RDS, releases internal storage used by UPDATE STATISTICS, and turns off the Update statistics mode bit in YTABLE1.

ARIYZ16

ARIYZ16 scans the pages of an index, collects statistics, and stores them in TREESPEC.

ARIYZ19

ARIYZ19 displays DBSS counters for the COUNTER operator command.

ARIYZ20

ARIYZ20 resets DBSS counter(s) to zero for the RESET operator command.

ARIYZ23

ARIYZ23 initializes SQL/DS system counters (see the COUNTER operator command) and resets them to zero.

ARIYZ25

ARIYZ25 displays the TREESPEC for update statistics trace.

ARIYZ26

ARIYZ26 displays the SEGMDESC for update statistics trace.

ARIYZ27

ARIYZ27 displays statistics gathered after exit from an UPDATE STATISTICS call.

SECTION 4: DIRECTORY

This section is divided into two parts.

The first part, Link Edit Book Directory, contains a list of the SQL/DS components by Link Edit Book names, the Link Book Names under which they can be found, and the phase names (VSE) or load module names (VM) in these Link Books.

The second part, the Module Directory, contains a list of the modules, the phase(s) (VSE) or load module(s) (VM) in which the module is included, and page number(s) on which the module is discussed.

LINK EDIT BOOK DIRECTORY - VSE

Link Edit Book for Component	Link Book	Phase Name(s)
DBSS / DSC	ARISLKDD	ARISQLDS ARICMOD
VSE Dependent Functions	ARISLKED	ARISD00D ARISD01D ARISD10D ARISD11D ARISD12D ARISD13D ARISD14D ARISD15D ARISD16D ARISD20D ARISD21D ARISD22D ARISD23D ARISD24D ARISD25D ARISD26D ARISD30D ARISD31D ARISD32D ARISD33D
RDS	ARISLKRD	ARIXRDS
GENCAT	ARISLKGD	ARIGCAT
Assembler PREP	ARISLKAD	ARIPRPA
COBOL PREP	ARISLKCD	ARIPRPC
PL/I PREP	ARISLKPD	ARIPRPP
FORTRAN PREP	ARISLKTD	ARIPRPF
DBS Utility	ARISLKUD	ARIDBS
ADDSEG (Add DBSPACE)	ARISLKSD	ARISEGB
Batch Resource Manager	ARISLKBD	ARIRBARM
Online Resource Manager	ARISLKZD	ARIOOLRM
Online Resource Manager Control (ENABLE/DISABLE CICS Resource Manager)	ARISLKYD	ARIRCONT
Trace Formatter	ARISLKFD	ARIMTRA
ISQL Mainline Transaction (without Extract) ISQL Mainline Transaction (with Extract)	ARISLKID ARISLKKD AXTILKED	ARIISQL ARIIEXT
ITRM Terminal Transaction	ARISLKJD	ARIITRM
Extract Utility Program	AXTILKUD	AXTUGET
Extract Utility Monitor	AXTILKMD	AXTUMON
Extract Utility Control Block	None	AXTUCBA
SQL/DS Copyright	None	$$$COSQL

LINK EDIT BOOK DIRECTORY - VM

Link Edit Book for Component	Link Book	Load Module Name(s)
DBSS	ARISLKDC	ARISQLDS
DSC/RM (DS2MODE)	ARISLKMC	ARICMOD
RDS	ARISLKRC	ARIXRDS
GENCAT	ARISLKGC	ARIGCAT
Assembler PREP	ARISLKAC	ARIPRPA
COBOL PREP	ARISLKCC	ARIPRPC
FORTRAN PREP	ARISLKTC	ARIPRPF
PL/I PREP	ARISLKPC	ARIPRPP
DBS Utility	ARISLKUC	ARIDBS
ADDSEG (Add DBSPACE)	ARISLKSC	ARISEGB
VM Resource Manager	ARISLKVC	ARIRVMRM
Trace Formatter	ARISLKFC	ARIMTRA
ISQL	ARISLKIC	ARIISQL

MODULE DIRECTORY

In the following list, you will see some of the module names or entry point names followed by "(VSE)" or "(VM)". This indicates that the particular name applies only to VSE/Advanced Function or to VM/SP, respectively.

Module	In Phase(s)(VSE)/ Load Mod(s)(VM)	Page Reference
ARIXC36	ARIXRDS	86, 522
ARIXC37	ARIXRDS	81, 522
ARIXC38	ARIXRDS	85, 81, 522
ARIXC39	ARIXRDS	92, 522
ARIXC40	ARIXRDS	85, 522
ARIXC41	ARIXRDS	85, 522
ARIXC42	ARIXRDS	85, 522
ARIXC43	ARIXRDS	85, 522
ARIXC45	ARIXRDS	86, 522
ARIXC47	ARIXRDS	92, 522
ARIXC48	ARIXRDS	92, 522
ARIXC49	ARIXRDS	92, 522
ARIXC51	ARIXRDS	85, 523
ARIXC53	ARIXRDS	523
ARIXC54	ARIXRDS	93, 523
ARIXDMY	ARIXRDS	
ARIXEAB	ARIXRDS	15, 524
ARIXEAD	ARIXRDS	524
ARIXEBR	ARIXRDS	15, 524
ARIXECK	ARIXRDS	15, 524
ARIXECL	ARIXRDS	15, 15
ARIXECW	ARIXRDS	15, 524
ARIXEDB	ARIXRDS	15, 524
ARIXEDC	ARIXRDS	15, 524
ARIXEDP	ARIXRDS	15, 525
ARIXEDR	ARIXRDS	18, 525
ARIXEDS	ARIXRDS	15, 525
ARIXEER	ARIXRDS	93, 525
ARIXEFB	ARIXRDS	526
ARIXELK	ARIXRDS	14, 526
ARIXELX	ARIXRDS	15, 526
ARIXEOC	ARIXRDS	526
ARIXEPH	ARIXRDS	526
ARIXEPP	ARIXRDS	28, 526
ARIXERD	ARIXRDS	15, 527
ARIXERO	ARIXRDS	14, 527
ARIXERP	ARIXRDS	15, 527
ARIXESP	ARIXRDS	14, 527
ARIXEST	ARIXRDS	14, 527
ARIXESX	ARIXRDS	28, 527
ARIXETR	ARIXRDS	192, 528
ARIXEUH	ARIXRDS	528
ARIXFA0	ARIXRDS	529
ARIXFTB	ARIXRDS	529
ARIXF01	ARIXRDS	529
ARIXF02	ARIXRDS	529
ARIXF03	ARIXRDS	529
ARIXF04	ARIXRDS	529
ARIXF05	ARIXRDS	529
ARIXF06	ARIXRDS	529
ARIXF07	ARIXRDS	529
ARIXF08	ARIXRDS	529
ARIXF09	ARIXRDS	529
ARIXF11	ARIXRDS	529
ARIXF12	ARIXRDS	529
ARIXF13	ARIXRDS	529
ARIXF14	ARIXRDS	529
ARIXF15	ARIXRDS	529
ARIXF16	ARIXRDS	529
ARIXF17	ARIXRDS	529
ARIXF18	ARIXRDS	529
ARIXF19	ARIXRDS	529
ARIXF20	ARIXRDS	529
ARIXF21	ARIXRDS	530
ARIXF22	ARIXRDS	530
ARIXF23	ARIXRDS	530
ARIXF24	ARIXRDS	530
ARIXF25	ARIXRDS	530
ARIXF26	ARIXRDS	530
ARIXF27	ARIXRDS	530
ARIXF28	ARIXRDS	530
ARIXF29	ARIXRDS	530
ARIXF30	ARIXRDS	530
ARIXF31	ARIXRDS	530
ARIXF32	ARIXRDS	530
ARIXF33	ARIXRDS	530
ARIXF34	ARIXRDS	530
ARIXF35	ARIXRDS	530
ARIXF36	ARIXRDS	530
ARIXF37	ARIXRDS	530
ARIXF38	ARIXRDS	530
ARIXF39	ARIXRDS	530
ARIXF40	ARIXRDS	530
ARIXF41	ARIXRDS	530
ARIXF42	ARIXRDS	530
ARIXF43	ARIXRDS	530
ARIXF44	ARIXRDS	530
ARIXF46	ARIXRDS	530
ARIXF47	ARIXRDS	530
ARIXF48	ARIXRDS	531
ARIXF49	ARIXRDS	531
ARIXF50	ARIXRDS	531
ARIXF51	ARIXRDS	531
ARIXF52	ARIXRDS	531
ARIXF53	ARIXRDS	531
ARIXF55	ARIXRDS	531
ARIXF56	ARIXRDS	531
ARIXF57	ARIXRDS	531
ARIXF58	ARIXRDS	531
ARIXF59	ARIXRDS	531
ARIXF60	ARIXRDS	531
ARIXF61	ARIXRDS	531

Module	In Phase(s)(VSE)/ Load Mod(s)(VM)	Page Reference
ARIXOML	ARIXRDS	547
ARIXOMS	ARIXRDS	547
ARIXONV	ARIXRDS	547
ARIXONW	ARIXRDS	547
ARIXOOP	ARIXRDS	54 - 77, 547
ARIXOOS	ARIXRDS	56, 59, 547
ARIXOQ1	ARIXRDS	63, 63, 547
ARIXOQ2	ARIXRDS	65, 56, 547
ARIXORM	ARIXRDS	547
ARIXOSC	ARIXRDS	547
ARIXOSG	ARIXRDS	547
ARIXOSL	ARIXRDS	75, 56, 547
ARIXOSO	ARIXRDS	547
ARIXOSR	ARIXRDS	547
ARIXOSS	ARIXRDS	548
ARIXOST	ARIXRDS	548
ARIXOSU	ARIXRDS	548
ARIXOS1	ARIXRDS	548
ARIXOS2	ARIXRDS	548
ARIXOS3	ARIXRDS	548
ARIXOS4	ARIXRDS	548
ARIXOTF	ARIXRDS	548
ARIXOTS	ARIXRDS	61, 548
ARIXOUS	ARIXRDS	548
ARIXOVC	ARIXRDS	59, 65, 548
ARIXOVD	ARIXRDS	548
ARIXOV1	ARIXRDS	548
ARIXOW1	ARIXRDS	56, 548
ARIXOW2	ARIXRDS	61, 548
ARIXO1S	ARIXRDS	548
ARIXPAT	ARIXRDS	549
ARIXPA0	ARIXRDS	33, 549
ARIXPA1	ARIXRDS	31, 549
ARIXPA2	ARIXRDS	31, 549
ARIXPA3	ARIXRDS	31, 549
ARIXPA4	ARIXRDS	33, 549
ARIXPTT	ARIMTRA	
ARIXSCF	ARIXRDS	15, 550
ARIXSCH	ARIXRDS	93, 550
ARIXSCN	ARIXRDS	55, 550
ARIXSCP	ARIXRDS	15, 550
ARIXSDB	ARIXRDS	550
ARIXSDT(D	ARIXRDS	550
ARIXSLN	ARIXRDS	550
ARIXSRT	ARIXRDS	550
ARIXSTM(D	ARIXRDS	550
ARIXSUT	ARIXRDS	550
ARIXTR0	ARIXRDS	551
ARIXTR1	ARIXRDS	551
ARIXTR2	ARIXRDS	551
ARIXTR3	ARIXRDS	551
ARIXTR4	ARIXRDS	551
ARIXTR5	ARIXRDS	551
ARIXTR6	ARIXRDS	551
ARIYC00	ARISQLDS	279, 552
ARIYC01	ARISQLDS	278, 280, 552
ARIYC02	ARISQLDS	278, 282, 552
ARIYC03	ARISQLDS	278, 552
ARIYC031 (EP)		552
ARIYC05	ARISQLDS	552
ARIYC06	ARISQLDS	278, 279, 552
ARIYC07	ARISQLDS	278 - 288, 552
ARIYC08	ARISQLDS	552
ARIYC09	ARISQLDS	552
ARIYC10	ARISQLDS	286, 552
ARIYC11	ARISQLDS	278, 288, 552
ARIYC13	ARISQLDS	278, 553
ARIYC14	ARISQLDS	278, 553
ARIYC15	ARISQLDS	278, 553
ARIYC18	ARISQLDS	294, 553
ARIYC19	ARISQLDS	295, 553
ARIYC20	ARISQLDS	296, 553
ARIYC21	ARISQLDS	295, 553
ARIYC22	ARISQLDS	294, 553
ARIYC23	ARISQLDS	293, 298, 553
ARIYC24	ARISQLDS	293, 300, 553
ARIYC25	ARISQLDS	293, 302, 553
ARIYC26	ARISQLDS	293, 304, 553
ARIYC27	ARISQLDS	553
ARIYDMY	ARISQLDS	
ARIYD00	ARISQLDS	307, 306, 554
ARIYD01	ARISQLDS	308, 554
ARIYD02	ARISQLDS	310, 554
ARIYD03	ARISQLDS	313, 554
ARIYD04	ARISQLDS	315, 554
ARIYD05	ARISQLDS	317, 554
ARIYD06	ARISQLDS	321, 554
ARIYD08	ARISQLDS	326, 554
ARIYD09	ARISQLDS	332, 554
ARIYD10	ARISQLDS	334, 554
ARIYD11	ARISQLDS	279, 554
ARIYD13	ARISQLDS	554
ARIYD14	ARISQLDS	555
ARIYD15	ARISQLDS	555
ARIYD16	ARISQLDS	555
ARIYD17	ARISQLDS	555
ARIYD18	ARISQLDS	555
ARIYD19	ARISQLDS	555
ARIYD20	ARISQLDS	555
ARIYD23	ARISQLDS	555
ARIYD26	ARISQLDS	279, 555
ARIYD27	ARISQLDS	556

Module	In Phase(s)(VSE)/ Load Mod(s)(VM)	Page Reference
ARIYS14	ARISQLDS	575
ARIYS17	ARISQLDS	576
ARIYS18	ARISQLDS	576
ARIYS19	ARISQLDS	576
ARIYS20	ARISQLDS	576
ARIYS21	ARISQLDS	576
ARIYS22	ARISQLDS	576
ARIYS23	ARISQLDS	576
ARIYS24	ARISQLDS	351, 352, 353, 576
ARIYS25	ARISQLDS	576
ARIYT00	ARISQLDS	272, 577
ARIYT01	ARISQLDS	577
ARIYT02	ARISQLDS	577
ARIYT05	ARISQLDS	387, 577
ARIYT06	ARISQLDS	397, 577
ARIYT08	ARISQLDS	577
ARIYT09	ARISQLDS	388, 577
ARIYT10	ARISQLDS	392, 577
ARIYT12	ARISQLDS	577
ARIYT14	ARISQLDS	390, 577
ARIYT15	ARISQLDS	577
ARIYT16	ARISQLDS	578
ARIYT17	ARISQLDS	393, 578
ARIYT18	ARISQLDS	578
ARIYT19	ARISQLDS	395, 578
ARIYT20	ARISQLDS	397, 578
ARIYT21	ARISQLDS	397, 578
ARIYT22	ARISQLDS	579
ARIYT23	ARISQLDS	579
ARIYT26	ARISQLDS	579
ARIYT27	ARISQLDS	394, 579
ARIYT28	ARISQLDS	579
ARIYT29	ARISQLDS	398,579
ARIYXTP	ARISQLDS	580
ARIYX00	ARISQLDS	580
ARIYX01	ARISQLDS	366, 580
ARIYX02	ARISQLDS	363, 580
ARIYX03	ARISQLDS	580
ARIYX04	ARISQLDS	358, 580
ARIYX05	ARISQLDS	580
ARIYX06	ARISQLDS	580
ARIYX07	ARISQLDS	580
ARIYX08	ARISQLDS	580
ARIYX09	ARISQLDS	580
ARIYX10	ARISQLDS	580
ARIYX11	ARISQLDS	580
ARIYX12	ARISQLDS	580
ARIYX14	ARISQLDS	581
ARIYX15	ARISQLDS	581
ARIYX16	ARISQLDS	581
ARIYX17	ARISQLDS	581
ARIYX18	ARISQLDS	374, 581
ARIYX19	ARISQLDS	581
ARIYX20	ARISQLDS	380, 581
ARIYX21	ARISQLDS	369, 581
ARIYX22	ARISQLDS	581
ARIYX23	ARISQLDS	581
ARIYX24	ARISQLDS	581
ARIYX25	ARISQLDS	581
ARIYX26	ARISQLDS	581
ARIYX27	ARISQLDS	581
ARIYX28	ARISQLDS	581
ARIYX29	ARISQLDS	582
ARIYX30	ARISQLDS	582
ARIYZTP	ARISQLDS	583
ARIYZ00	ARISQLDS	385, 583
ARIYZ01	ARISQLDS	583
ARIYZ02	ARISQLDS	583
ARIYZ03	ARISQLDS	583
ARIYZ04	ARISQLDS	583
ARIYZ05	ARISQLDS	583
ARIYZ06	ARISQLDS	583
ARIYZ10	ARISQLDS	583
ARIYZ11	ARISQLDS	583
ARIYZ15	ARISQLDS	583
ARIYZ16	ARISQLDS	584
ARIYZ19	ARISQLDS	584
ARIYZ20	ARISQLDS	584
ARIYZ23	ARISQLDS	584
ARIYZ25	ARISQLDS	584
ARIYZ26	ARISQLDS	584
ARIYZ27	ARISQLDS	584

INDEX

D

M

N

O

P

R

S

T

U

V

W

SQL/Data System Logic, Volume 1 (File No. S370/4300-50) Printed in U.S.A. LY24-5216-2

SQL/Data System Logic
Volume 1
LY24-5216-2

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. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate. Comments may be written in your own language; English is not required.

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:

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

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 or you may mail directly to the address in the Edition Notice on the back of the title page.)

Reader's Comment Form

Cut or Fold Along Line

Fold and Tape | Please Do Not Staple | Fold and Tape

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®

SQL/Data System Logic, Volume 1 (File No. S370/4300-50) Printed in U.S.A. LY24-5216-2

SQL/Data System Logic
Volume 1
LY24-5216-2

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. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate. Comments may be written in your own language; English is not required.

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:

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

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 or you may mail directly to the address in the Edition Notice on the back of the title page.)

Reader's Comment Form

Cut or Fold Along Line

Fold and Tape | Please Do Not Staple | Fold and Tape

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®

SQL/Data System Logic, Volume 1 (File No. S370/4300-50) Printed in U.S.A. LY24-5216-2

SQL/Data System Logic
Volume 1
LY24-5216-2

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. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate. Comments may be written in your own language; English is not required.

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:

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

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 or you may mail directly to the address in the Edition Notice on the back of the title page.)

LY24-5216-2

Reader's Comment Form

Cut or Fold Along Line

Fold and Tape | Please Do Not Staple | Fold and Tape

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®

SQL/Data System Logic, Volume 1 (File No. S370/4300-50) Printed in U.S.A. LY24-5216-2

SQL/Data System Logic
Volume 1
LY24-5216-2

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. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate. Comments may be written in your own language; English is not required.

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.*

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

	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 or you may mail directly to the address in the Edition Notice on the back of the title page.)

LY24-5216-2

Reader's Comment Form

Cut or Fold Along Line

Fold and Tape | Please Do Not Staple | Fold and Tape

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®

SQL/Data System Logic, Volume 1 (File No. S370/4300-50) Printed in U.S.A. LY24-5216-2

LY24-5216-2

SQL/Data System Logic, Volume 1 (File No. S370/4300-50) Printed in U.S.A. LY24-5216-2