LY28-1685-0
File No. S370-36

Program Product

MVS/Extended Architecture System Logic Library: EXCP Processor

MVS/System Product:

JES3 Version 2	5665-291
JES2 Version 2	5740-XC6

IBM

This publication supports MVS/System Product Version 2 Release 2.0, and contains information that was formerly presented in MVS/Extended Architecture System Logic Library Volume 7, LY28-1230-4, which applies to MVS/System Product Version 2 Release 1.7. See the Summary of Amendments for more information.

First Edition (June, 1987)

This edition applies to Version 2 Release 2.0 of MVS/System Product 5665-291 or 5740-XC6 and to all subsequent releases 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 Bibliography, GC20-0001, for the editions that are applicable and current.

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 readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Department D58, Building 921-2, PO Box 390, Poughkeepsie, N.Y. 12602. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

PREFACE

The MVS/Extended Architecture System Logic Library is intended for people who debug or modify the MVS control program. It describes the logic of most MVS control program functions that are performed after master scheduler initialization completes. For detailed information about the MVS control program prior to this point, refer to MVS/Extended Architecture System Initialization Logic. For general information about the MVS control program and the relationships among the components that make up the MVS control program, refer to the MVS/Extended Architecture Overview. To obtain the names of publications that describe some of the components not in the System Logic Library, refer to the section Corequisite Reading in the Master Preface in MVS/Extended Architecture System Logic Library: Master Table of Contents and Index.

HOW THE LIBRARY IS ORGANIZED

SET OF BOOKS

The System Logic Library consists of a set of books. Two of the books provide information that is relevant to the entire set of books:

1. The MVS/Extended Architecture System Logic Library: Master Table of Contents and Index contains the master preface, the master table of contents, and the master index for the other books in the set.

2. The MVS/Extended Architecture System Logic Library: Module Descriptions contains module descriptions for all of the modules in the components documented in the System Logic Library and an index.

Each of the other books (referred to as component books) in the set contains its own table of contents and index, and describes the logic of one of the components in the MVS control program.

ORGANIZATION OF THE COMPONENTS

Most component books contain information about one component in the MVS control program. However, some component books (such as System Logic Library: Initiator/Terminator) contain more than one component if the components are closely related, frequently referenced at the same time, and not so large that they require a book of their own.

A three or four character mnemonic is associated with each component book and is used in all diagram and page numbers in that book. For example, the mnemonic ASM is associated with the book MVS/Extended Architecture System Logic Library: Auxiliary Storage Management. All diagrams in this book are identified as Diagram ASM-n, and all pages as ASM-n, where n represents the specific diagram or page number. Whenever possible, the existing component acronym is used as the mnemonic for the component book. The Table of Book Titles in the Master Preface in MVS/Extended Architecture System Logic Library: Master Table of Contents and Index lists the book titles, the components included in each book (if a book contains more than one component), the mnemonics for the books, and the order number for each book.

HOW TO USE THE LIBRARY

To help you use this library efficiently, the following topics cover

- How to find information using book titles and the master index
- What types of information are provided for each component
- How to obtain further information about other books in the System Logic Library

FINDING INFORMATION USING THE BOOK TITLES

As you become familiar with the book titles, MVS component names and mnemonics, and the book contents, you will be able to use the System Logic Library as you would an encyclopedia and go directly to the book that you need. We recommend that you group the books in alphabetical order for easy reference, or, if you are familiar with MVS, that you to group the books by related functions.

The Table of Book Titles in the Master Preface in MVS/Extended Architecture System Logic Library: Master Table of Contents and Index contains a list of book titles and mnemonics. It provides a quick reference to all the books, and their corresponding components, in the System Logic Library.

FINDING INFORMATION USING THE MASTER INDEX

If you are not sure which book contains the information you are looking for, you can locate the book and the page on which the information appears by using the master index in System Logic Library: Master Table of Contents and Index. For the component books, the page number in an index entry consists of the mnemonic for the component and the page number; for System Logic Library: Module Descriptions, the page number consists of the mnemonic "MOD" and the page number.

For example:

ASM-12	refers to MVS/Extended Architecture System Logic Library: Auxiliary Storage Management, page ASM-12.
MOD-245	refers to MVS/Extended Architecture System Logic Library: Module Descriptions, page MOD-245.

INFORMATION PROVIDED FOR MOST COMPONENTS

The following information is provided for most of the components described in the System Logic Library.

1. An introduction that summarizes the component's function

2. Control block overview figures that show significant fields and the chaining structure of the component's control blocks

3. Process flow figures that show control flow between the component's object modules

4. Module information that describes the functional organization of a program. This information can be in the form of:

   - Method-of-Operation diagrams and extended descriptions.

   - Automatically-generated prose. The automated module information is generated from the module prologue and the code itself. It consists of three parts: module description, module operation summary, and diagnostic aids.

5. Module descriptions that describe the operation of the modules (the module descriptions are contained in System Logic Library: Module Descriptions)

Some component books also include diagnostic techniques information following the Introduction.

FURTHER INFORMATION

For more information about the System Logic Library, including the order numbers of the books in the System Logic Library, see the Master Preface in MVS/Extended Architecture System Logic Library: Master Table of Contents and Index.

CONTENTS

FIGURES

**Summary of Amendments
for LY28-1685-0
for MVS/System Product Version 2 Release 2.0**

This publication is new for MVS System Product Version 2 Release 2.0. It contains information that was reorganized from the EXCP Processor section in MVS/XA System Logic Library Volume 7, LY28-1230-4, which applies to MVS/XA System Product Version 2 Release 1.7.

This publication contains changes to support MVS/System Product Version 2 Release 2.0. The changes include:

- The CCW Translation Operation Tables module, IECVOTBL, was deleted. Each table that IECVOTBL contained is now in a separate module.
- Method of Operation information for the following new modules:

  IECVOPTB
  IECVOPTC
  IECVOPTD
  IECVOPTE
  IECVOPTG
  IECVOPTH
  IECVOPTI
  IECVOPTJ
  IECVOPTK
  IECVOPTL
  IECVOPTM
  IECVOPTN
  IECVOPTT
  IECVOPTU

- Minor technical and editorial changes throughout the publication.

EXCP PROCESSOR

INTRODUCTION

EXCP communicates information between access methods (including VTAM, JES2, and JES3) and the input/output supervisor (IOS). IOS is described in the IOS section of the System Logic Library. EXCP's role as a communication function includes these responsibilities:

- Communicating an access-method request for an I/O operation to IOS by (a) gathering information from the "access-method interface" (defined below), (b) consolidating the information into a single control block, and (c) passing the address of the control block to IOS.
- Communicating the status of an I/O operation to channel-end, abnormal-end, and PCI (program-controlled interrupt) appendages by (a) gaining control at the IOS exits and (b) moving IOS-collected information to access-method control blocks.
- Telling the access method what the final disposition of its I/O request is by posting its ECB (event control block).

As one of the callers of IOS, EXCP takes part in purging and restoring I/O requests. Its role is complementary to the I/O supervisor's: if IOS halts certain EXCP-initiated requests (all those initiated from a certain address space, for instance), EXCP deletes the control information it has kept for them; if IOS quiesces certain EXCP-initiated requests, EXCP saves a block of control information for each such request not yet sent to IOS, chains the blocks together, and gives IOS the address of the chain. When a restore operation is subsequently requested, IOS returns the address of the chain to EXCP, and EXCP resumes the processing of those requests.

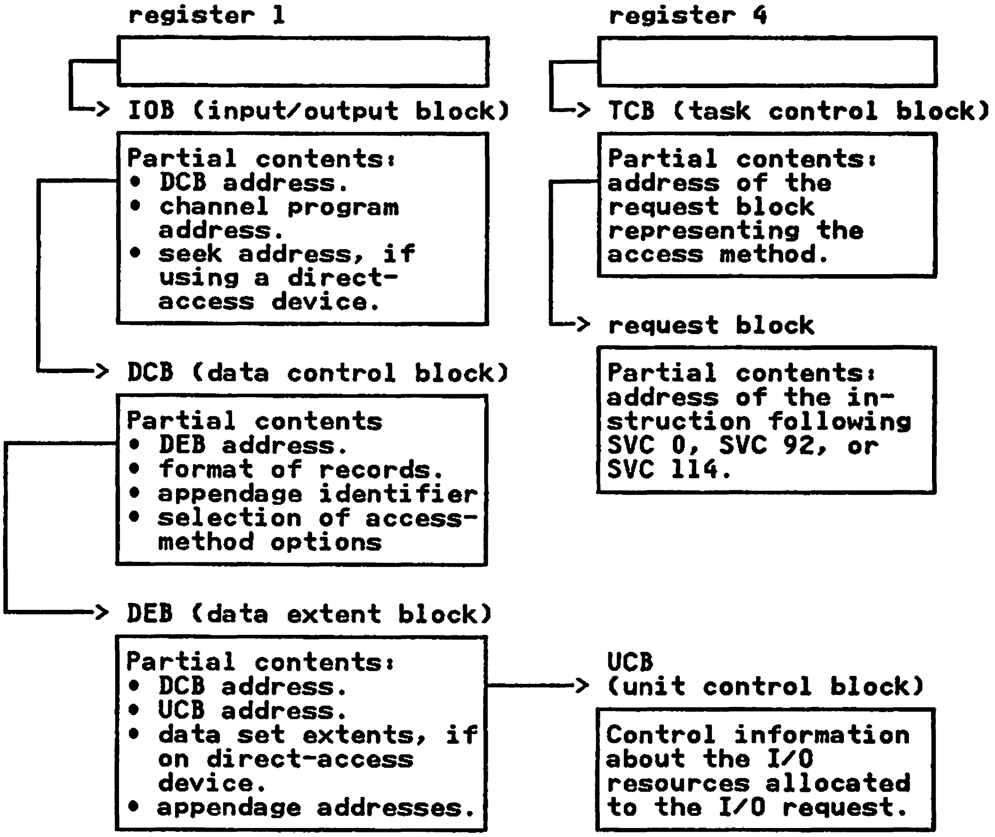
PROGRAMS THAT QUALIFY AS ACCESS METHODS

In the discussion of EXCP, the term "access method" means any program that builds channel programs and passes them to EXCP for execution. This definition includes some of the IBM access methods (such as SAM, BDAM, ISAM, BTAM, TCAM, VTAM, GAM, and PAM), JES2 and JES3, and any user program, utility program, or SVC routine that builds a channel program and gives it to EXCP for execution (even though building a channel program may not be its main purpose).

THE ACCESS-METHOD INTERFACE

To give control to EXCP, an access method issues an EXCP or EXCPVR macro instruction, which expands into an SVC 0 or SVC 114 instruction, respectively. The SVC interrupt handler then gives control to EXCP.

On acquiring control, EXCP finds:



register 1

IOB (input/output block)

Partial contents:
- DCB address.
- channel program address.
- seek address, if using a direct-access device.

DCB (data control block)

Partial contents
- DEB address.
- format of records.
- appendage identifier
- selection of access-method options

DEB (data extent block)

Partial contents:
- DCB address.
- UCB address.
- data set extents, if on direct-access device.
- appendage addresses.

register 4

TCB (task control block)

Partial contents: address of the request block representing the access method.

request block

Partial contents: address of the instruction following SVC 0, SVC 92, or SVC 114.

UCB (unit control block)

Control information about the I/O resources allocated to the I/O request.

The control blocks illustrated above constitute the access-method interface. They contain everything EXCP needs to build:

- An interface that IOS will use to start the I/O operation.
- An internal record, called an RQE (request queue element), that represents the access-method request for an I/O operation.

Communicating an I/O request to IOS

Preparing to go to IOS with an I/O request requires the following steps:

1. EXCP verifies the access method interface.

   Some of the errors EXCP checks for are conflicting DCB pointers, an invalid UCB, an invalid DEB, or an IOB, ECB, or DCB that is not in the protection key of the caller.

2. EXCP makes a record of the request and puts it in a related request queue (RRQ) if it is a related request. (The next topic describes related requests.)

   EXCP builds a request queue element (RQE), containing information such as the addresses of the TCB, UCB, IOB, and DEB, which are needed for later processing.

   If the IOB indicates that the I/O request is a related request, EXCP puts the RQE at the end of a related request queue (RRQ).

3. EXCP finds out if a VIO (virtual input/output) data set will be used and, if so, does not go to IOS with the request but to the VIO component instead.

   EXCP goes to the VIO component via the WIEXCP macro. The VIO component either simulates the transfer of data or uses another caller of IOS, the auxiliary storage manager, to read or write data. See VIO Logic for more information about VIO processing.

4. EXCP puts all the information IOS needs to process the request into an SRB (service request block) and IOSB (input/output supervisor block).

5. If necessary, EXCP calls the access method's PGFX (page fix) and EOE (end-of-extent) appendages.

   EXCP gives a PGFX appendage control if the access method either issued an EXCPVR macro or a virtual EXCP. Pages in the list returned by the PGFX appendage are fixed if EXCP was entered by an EXCPVR macro.

   For requests from a V=R address space, EXCP checks whether the DEB has been fixed. If not, EXCP does a TCB-associated pagefix, using th^ TCB address in the DEB. The PGFX appendage is not entered.

   EXCP enters the EOE appendage if a direct access device was allocated and the seek address in the IOB does not fall within the extent boundaries recorded in the DEB.

   EXCP also invokes the EOE appendage if, after IOS tries to start an I/O operation, the direct access ERP alters the seek address and wants the new seek address verified.

6. EXCP calls the access method's SIO (start I/O) appendage.

7. If the access method is not running in a V=R address space and did not issue an EXCPVR macro, EXCP calls:

   - IECVTCCW to copy the channel program in fixed storage and substitute real storage addresses for virtual ones

   - A system routine that fixes buffers

8. EXCP passes the I/O request to IOS.

   EXCP calls the IOS code that starts I/O operations. This call is made by issuing a STARTIO macro or by a direct branch from EXCP's DIE procedure. (IOS enters the DIE procedure of its caller after a solicited I/O event occurs.)

RELATED REQUESTS

Related requests are I/O requests with these characteristics:

- They are directed to the same data set and share the same DEB.

- They are processed by EXCP in the order received, but with some overlap; that is, request _n_ in a group of related requests need not be completely processed before some processing, short of channel-program execution, can be done on request _n+1_.

- If a related request returns from IOS with an I/O error, none of the related requests remaining to be sent can be successful. The subsequent requests depend on the success of the earlier request.

By examining the IOB, EXCP can tell if the access method has given it a related request and, if the access method has, what _type_ of related request it is - type denoting the amount of overlap permissible between a given related request, n, and n+1. Three types exist:

Type 1. The I/O operation for this type must complete, and the channel-end appendages must look at the status of the operation, before the next related request can be handled by the SIO appendage.

Type 2. The I/O operation for this type must complete, and the channel-end-appendage must look at the status of the operation, before the next related request can be sent to IOS. EXCP will have processed the next related request so that it is ready to send to IOS.

Type 3. The I/O operation for this type must complete before the next related request can be sent to IOS. The EXCP disabled interrupt exit (DIE) examines the subchannel status word (SCSW) for device-end or channel-end. For either condition, the DIE passes the next related request to IOS. (EXCP will have processed the next related request so it is ready to send to IOS.) If the SCSW for the I/O operation shows anything other than a device-end or channel-end indication, the next related request cannot be sent to IOS until the channel-end or abnormal-end appendage has executed.

ADDRESSING AND RESIDENCY MODE OF EXCP MODULES

The four modules comprising EXCP (IECVEXCP, IECVEXPR, IECVEXFR, and IECVTCCW) execute in 24-bit addressing mode and reside below 16 megabytes. This forces certain restrictions on users of the EXCP macro:

- Control blocks passed to EXCP must reside below the 16 megabytes.
- Appendages must execute in 24-bit addressing mode.
- The CCW translation operation tables must reside below 16 megabytes.

Also, EXCP can use only format-0 CCWs. (Format-0 CCWs use only 24-bit addresses; format-1 CCWs use 31-bit addresses.)

Virtual addresses above 16 megabytes are supported through virtual IDAWs. For each data transfer CCW to a location above 16 megabytes, a single virtual IDAW is required.

EXCP ABEND CODES

The following table lists abend codes with the EXCP module and symbolic names of the EXCP procedures that issue them. For the meanings of the abend codes, refer to Message Library: System Codes.

Code	Module	Procedure - Name
X'15C'	IECVEXCP	XCP000 - Validity check
X'172'	IECVEXCP	XCP000 - Validity check
X'200'	IECVEXFR	Functional recovery
X'300'	IECVEXCP	XCP000 - Validity check
X'400'	IECVEXCP	XCP000 - Validity check
X'500'	IECVEXCP	XCP000 - Validity check
X'700'	IECVEXCP	XCPTERM - Termination
	IECVEXFR	Functional recovery
X'800'	IECVEXCP	XCP090 - PGFX interface
	IECVEXCP	XCPTERM - Termination
	IECVEXCP	XCP115 - Translation interface
X'A00'	IECVEXCP	XCPTERM - Termination
	IECVEXFR	Functional recovery
X'B00'	IECVEXFR	Functional recovery
X'C22'	IECVEXCP	XCP036 - Building RQE
X'E00'	IECVEXCP	XCPTERM - Termination

EXCP DEBUGGING AREA (XDBA)

EXCP's functional recovery procedure, IECVEXFR, does not put diagnostic data in the SDUMP buffer. Instead, it gets storage for its own debugging area (the XDBA) and puts diagnostic data there. (Note that an XDBA is not provided for E00 abend codes.)

To locate the debugging area (XDBA) in a SYSABEND, SYSMDUMP, or SYSUDUMP dump, you must:

1. Get the address of the CVT from location X'4C' (PSA field FLCCVT2) in the dump.

2. Get the address of the TCB from the first word of the CVT (CVTTCBP).

3. Look X'C0' bytes into the TCB (TCBEXCPD) and get the address of the debugging area. If the address of the debugging area is zero then no debugging area is available.

The format and contents of the EXCP debugging area (XDBA) are as follows:

Hex Offset	Contents
0	XDBA identifier
10	XDBA chain pointer or zero
14	EXCP abend completion code
16	SDWA original abend code
18	SDWA PSW at time of error
20	Translation exception address
24	Reserved
30	SDWA registers at time of error
70	FRR parameter area identifier
78	EXCP FRR parameter area
90	RQE block identifier
94	RQE block size and 8-byte storage manager header

B9 A byte that shows where the error occurred. (This byte is in byte 37 (X'25') of the RQE.) The possible bit settings and their meanings are:
X'80': The error occurred while EXCP was preparing to send an I/O request to IOS.
X'40': The error occurred while EXCP was processing an I/O request that IOS was finished with.
X'21': The error occurred in a PCI appendage.
X'11': The error occurred in a channel end (CHE) appendage.
X'09': The error occurred in an abnormal end (ABE) appendage.
X'05': The error occurred in an end-of-extent (EOE) appendage.
X'03': The error occurred in a PGFX appendage.
X'01': The error occurred in a SIO appendage.

D4 Reserved.

EC Number of large blocks in the XDBA. The large blocks are moved into the remaining XDBA area starting at offset X'100' in the following sequence (if present): SRB/IOSB, ERP work area (EWA), translation control block (TCCW), indirect data address list (IDAL), list of fixed pages (FIX), beginning-end block (BEB), and channel program scan parameter list (CPS). Only valid large blocks are moved.

F0 Large block area identifier

100 Start of large blocks.

Note: For errors that occur in the PCI appendage during disabled interruption exit (DIE) processing, the IOS module IOSVIRBA provides a SYS1.LOGREC record and an SVC dump. The register contents and PSW at the time of the original error are contained in the SYS1.LOGREC record and the dump. EXCP uses the DIE exit when processing type 3 related requests, V=R requests, and EXCPVR requests.

CCW TRANSLATION OPERATION TABLE

The CCW translation operation tables communicate to IECVTCCW, the CCW tr▸nslator, information about how each CCW should be handled for a given device. IECVTCCW obtains the pointer to the appropriate CCW operation table from the device descriptor table (DDT) associated with the device.

A CCW translation operation table is 256 bytes in length, one byte per possible channel command. Normal processing is for IECVTCCW to treat a CCW as a data transfer command, translate the data address from a virtual address to a real address, and fix the data area.

For more information about the CCW translation operation tables, the device classes, and the specific devices with their corresponding CSECT names, see Channel Command Word (CCW) Translation Operation Tables Modules in this book.

MISCELLANEOUS HINTS

- During abend processing, the EXCP debugging areas are not freed. When you find the area pointed to by the TCB, scan that area for previously-obtained areas to help with EXCP analysis.

- IECVEXCP processing does all the interfacing to the EXCP appendages. Appendages are entered in SRB mode, physically enabled, and with the address of a save area in register 13.

- IECVEXCP maps the IOSB to the IOB before interfacing with an appendage. On return from the appendage, IECVEXCP re-maps the IOB to the IOSB.

- All the RQE blocks are maintained in an RQE pool. To determine the current RQE status, scan the RQE pool areas.

These areas can be located as shown in Figure EXCP-1. An RQE block is 64 (X'40') bytes in length; at offset X'3A' is a two-byte allocation indication. If the two bytes contain X'0075', the RQE is allocated and represents an active EXCP request. Offset X'38' contains the two-byte address space identifier associated with the request.

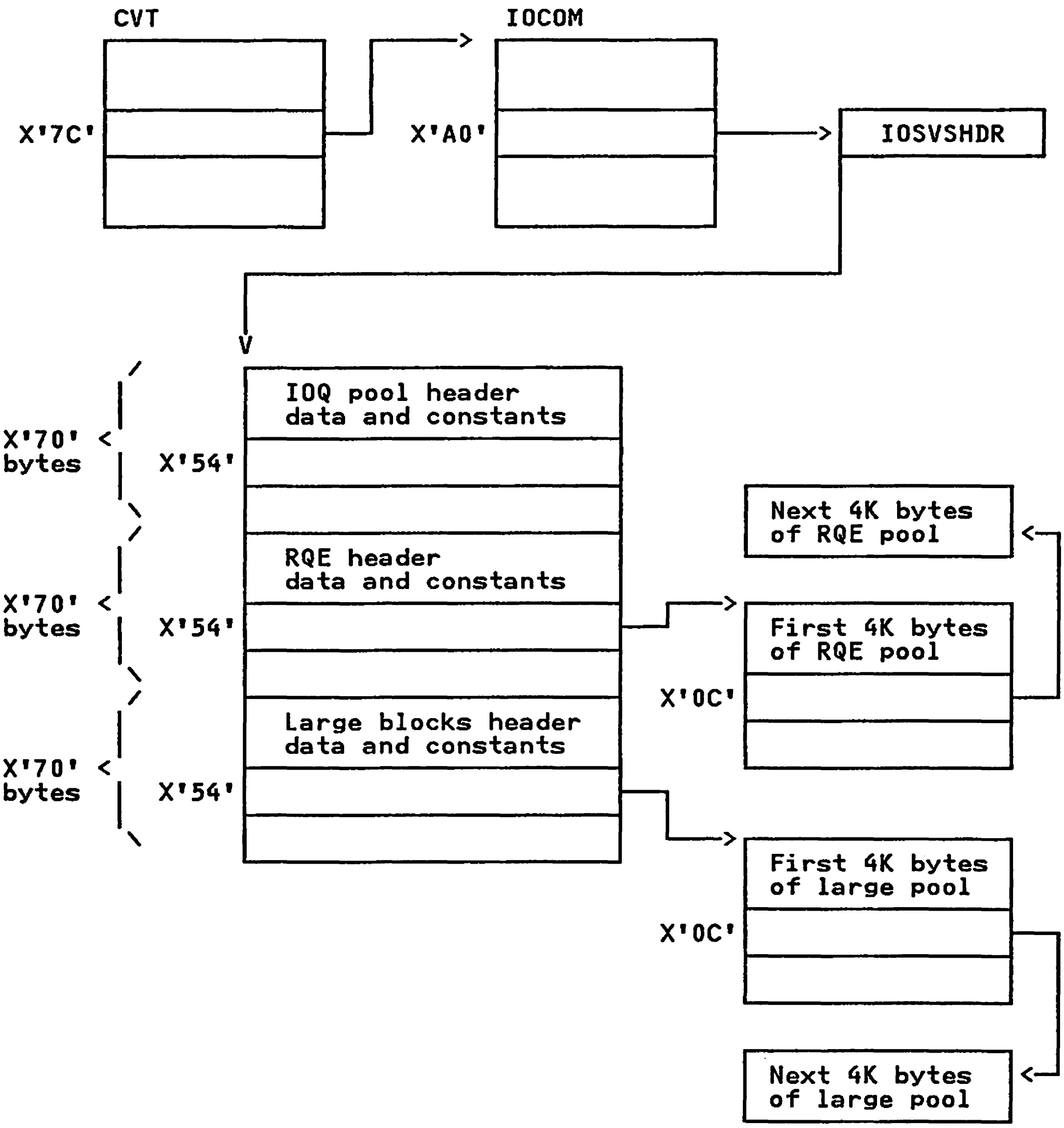
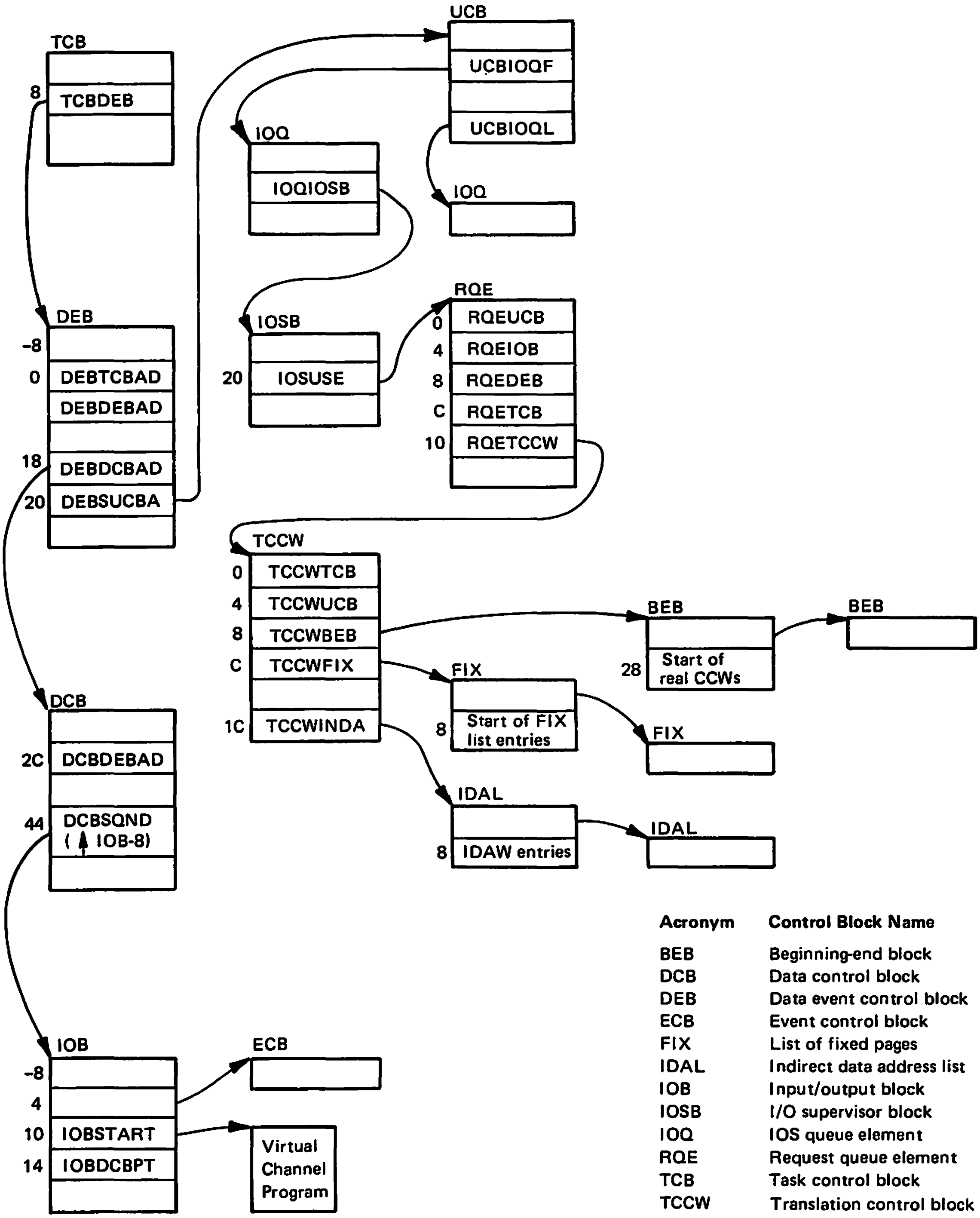

CVT
X'7C'
IOCOM
X'A0'
IOSVSHDR
X'70' bytes
X'54'
IOQ pool header data and constants
X'70' bytes
X'54'
RQE header data and constants
X'70' bytes
X'54'
Large blocks header data and constants
Next 4K bytes of RQE pool
First 4K bytes of RQE pool
X'0C'
First 4K bytes of large pool
X'0C'
Next 4K bytes of large pool

Figure 1. Locating RQE Pool Areas

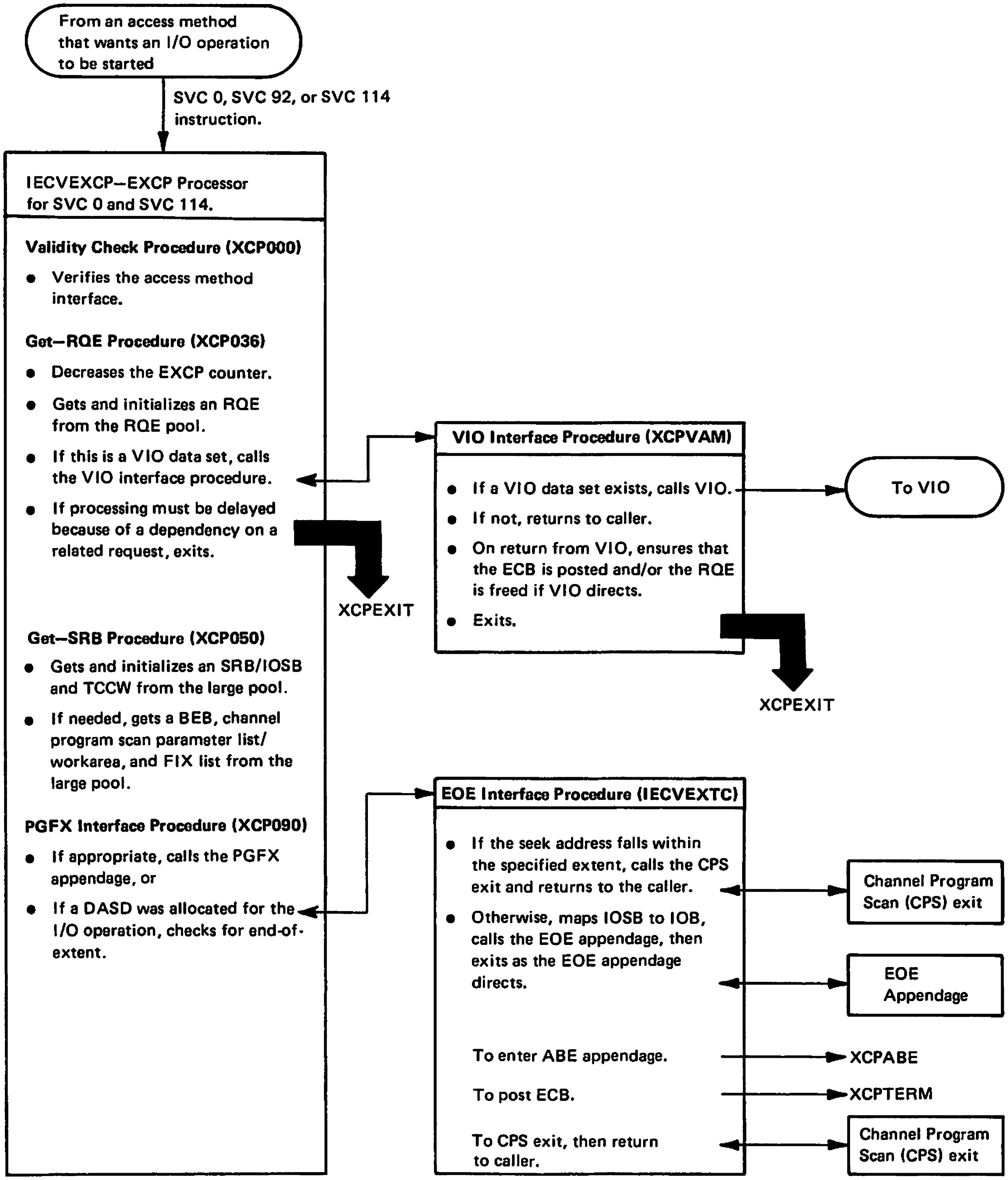
CONTROL BLOCK OVERVIEW



Acronym	Control Block Name
BEB	Beginning-end block
DCB	Data control block
DEB	Data event control block
ECB	Event control block
FIX	List of fixed pages
IDAL	Indirect data address list
IOB	Input/output block
IOSB	I/O supervisor block
IOQ	IOS queue element
RQE	Request queue element
TCB	Task control block
TCCW	Translation control block

PROCESS FLOW

The figures in this section show control flow within EXCP modules. In the figure for IECVEXCP, calls to procedures within IECVEXCP appear as external references. They can be distinguished from external references by the word "procedure" in the title and by the appearance of the label, for example, VIO Interface Procedure (XCPVAM).



From an access method that wants an I/O operation to be started

SVC 0, SVC 92, or SVC 114 instruction.

IECVEXCP—EXCP Processor for SVC 0 and SVC 114.

Validity Check Procedure (XCP000)

- Verifies the access method interface.

Get—RQE Procedure (XCP036)

- Decreases the EXCP counter.
- Gets and initializes an RQE from the RQE pool.
- If this is a VIO data set, calls the VIO interface procedure.
- If processing must be delayed because of a dependency on a related request, exits.

XCPEXIT

Get—SRB Procedure (XCP050)

- Gets and initializes an SRB/IOSB and TCCW from the large pool.
- If needed, gets a BEB, channel program scan parameter list/workarea, and FIX list from the large pool.

PGFX Interface Procedure (XCP090)

- If appropriate, calls the PGFX appendage, or
- If a DASD was allocated for the I/O operation, checks for end-of-extent.

VIO Interface Procedure (XCPVAM)

- If a VIO data set exists, calls VIO.
- If not, returns to caller.
- On return from VIO, ensures that the ECB is posted and/or the RQE is freed if VIO directs.
- Exits.

To VIO

XCPEXIT

EOE Interface Procedure (IECVEXTC)

- If the seek address falls within the specified extent, calls the CPS exit and returns to the caller.
- Otherwise, maps IOSB to IOB, calls the EOE appendage, then exits as the EOE appendage directs.

Channel Program Scan (CPS) exit

EOE Appendage

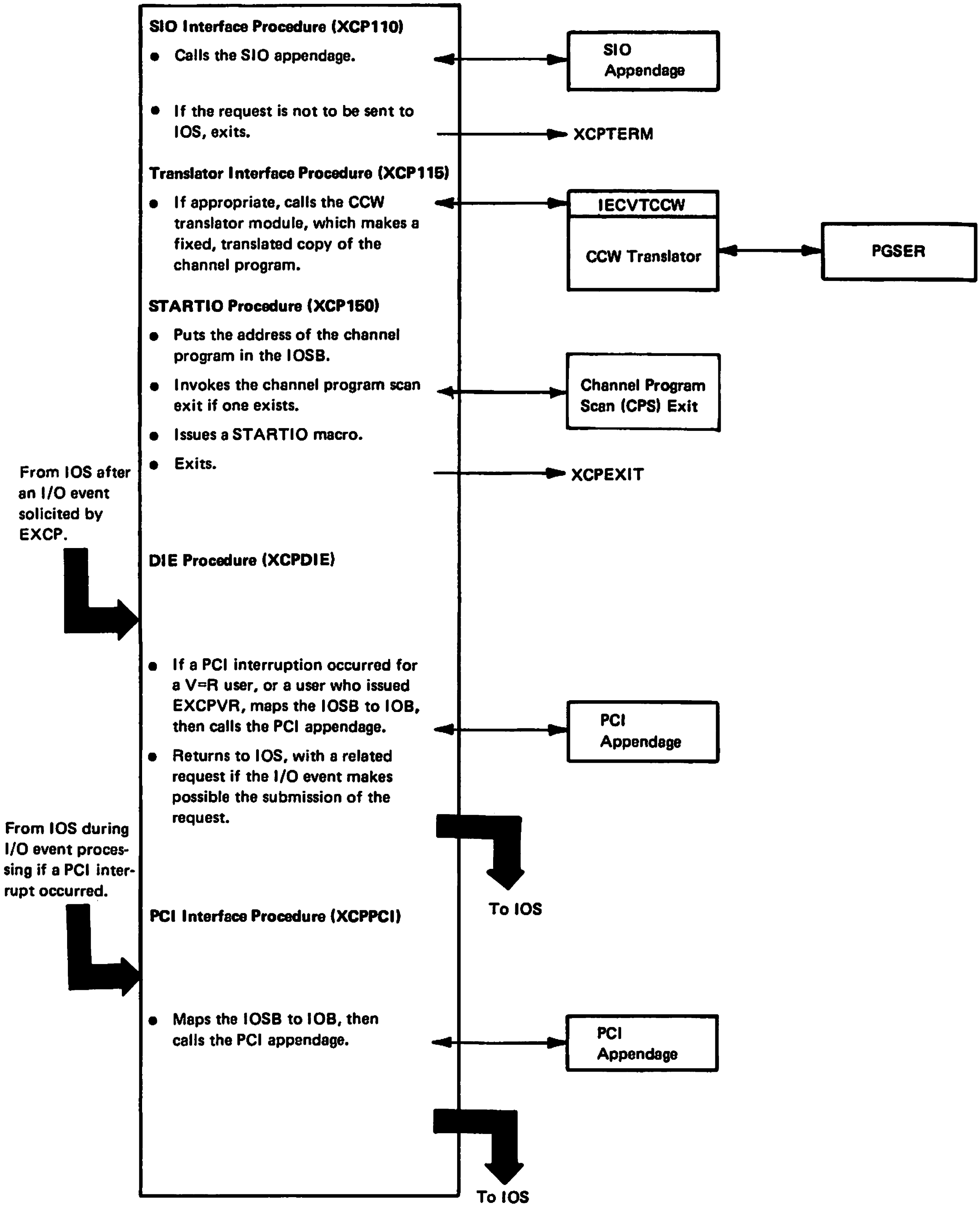
To enter ABE appendage. XCPABE

To post ECB. XCPTERM

To CPS exit, then return to caller.

Channel Program Scan (CPS) exit

Figure 2 (Part 1 of 4). IECVEXCP Process Flow



SIO Interface Procedure (XCP110)

- Calls the SIO appendage.
- If the request is not to be sent to IOS, exits.

Translator Interface Procedure (XCP115)

- If appropriate, calls the CCW translator module, which makes a fixed, translated copy of the channel program.

STARTIO Procedure (XCP150)

- Puts the address of the channel program in the IOSB.
- Invokes the channel program scan exit if one exists.
- Issues a STARTIO macro.
- Exits.

From IOS after an I/O event solicited by EXCP.

DIE Procedure (XCPDIE)

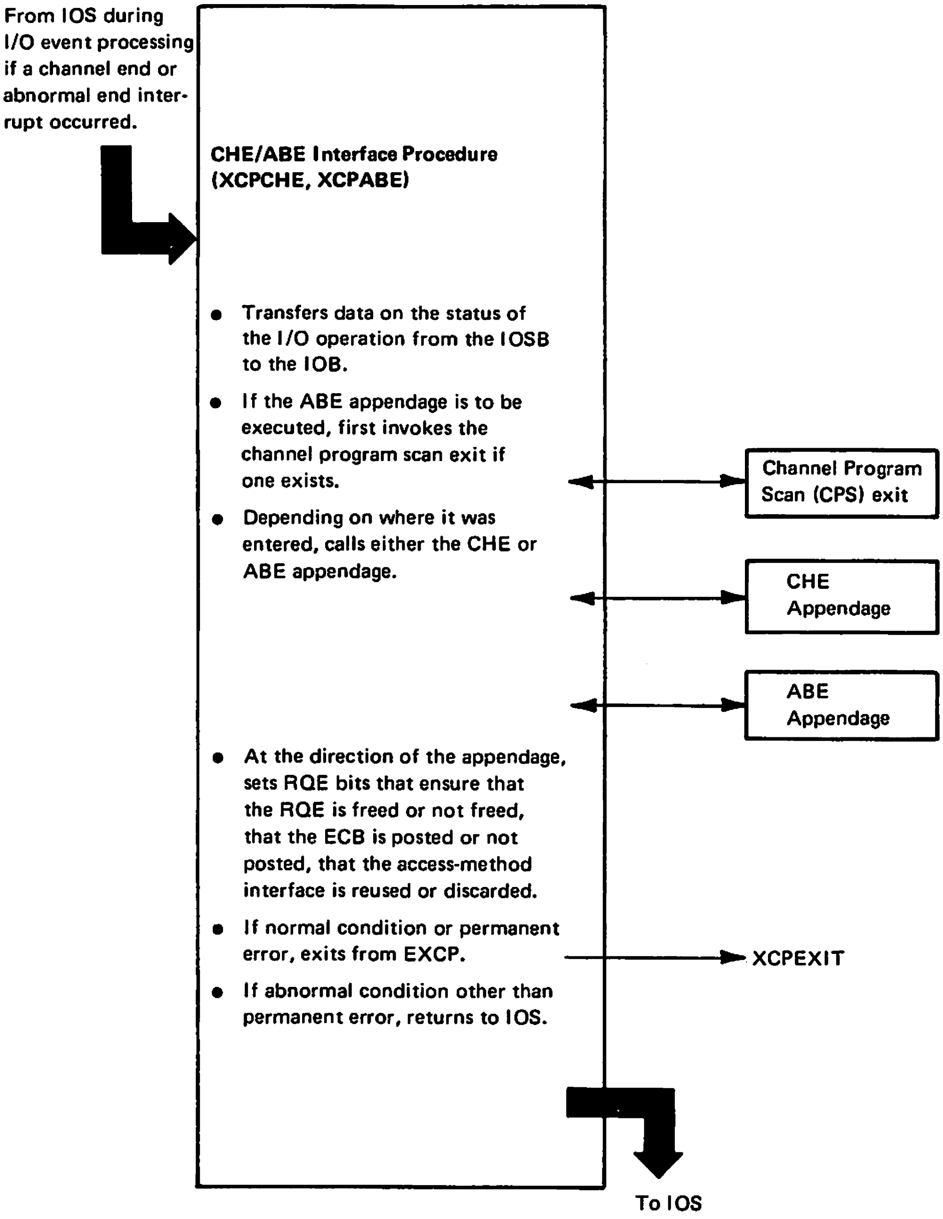
- If a PCI interruption occurred for a V=R user, or a user who issued EXCPVR, maps the IOSB to IOB, then calls the PCI appendage.
- Returns to IOS, with a related request if the I/O event makes possible the submission of the request.

From IOS during I/O event processing if a PCI interrupt occurred.

PCI Interface Procedure (XCPPCI)

- Maps the IOSB to IOB, then calls the PCI appendage.

Figure 2 (Part 2 of 4). IECVEXCP Process Flow



From IOS during I/O event processing if a channel end or abnormal end interrupt occurred.
CHE/ABE Interface Procedure (XCPCHE, XCPABE)
Channel Program Scan (CPS) exit
CHE Appendage
ABE Appendage
XCPEXIT
To IOS

- Transfers data on the status of the I/O operation from the IOSB to the IOB.
- If the ABE appendage is to be executed, first invokes the channel program scan exit if one exists.
- Depending on where it was entered, calls either the CHE or ABE appendage.
- At the direction of the appendage, sets RQE bits that ensure that the RQE is freed or not freed, that the ECB is posted or not posted, that the access-method interface is reused or discarded.
- If normal condition or permanent error, exits from EXCP.
- If abnormal condition other than permanent error, returns to IOS.

Figure 2 (Part 3 of 4). IECVEXCP Process Flow

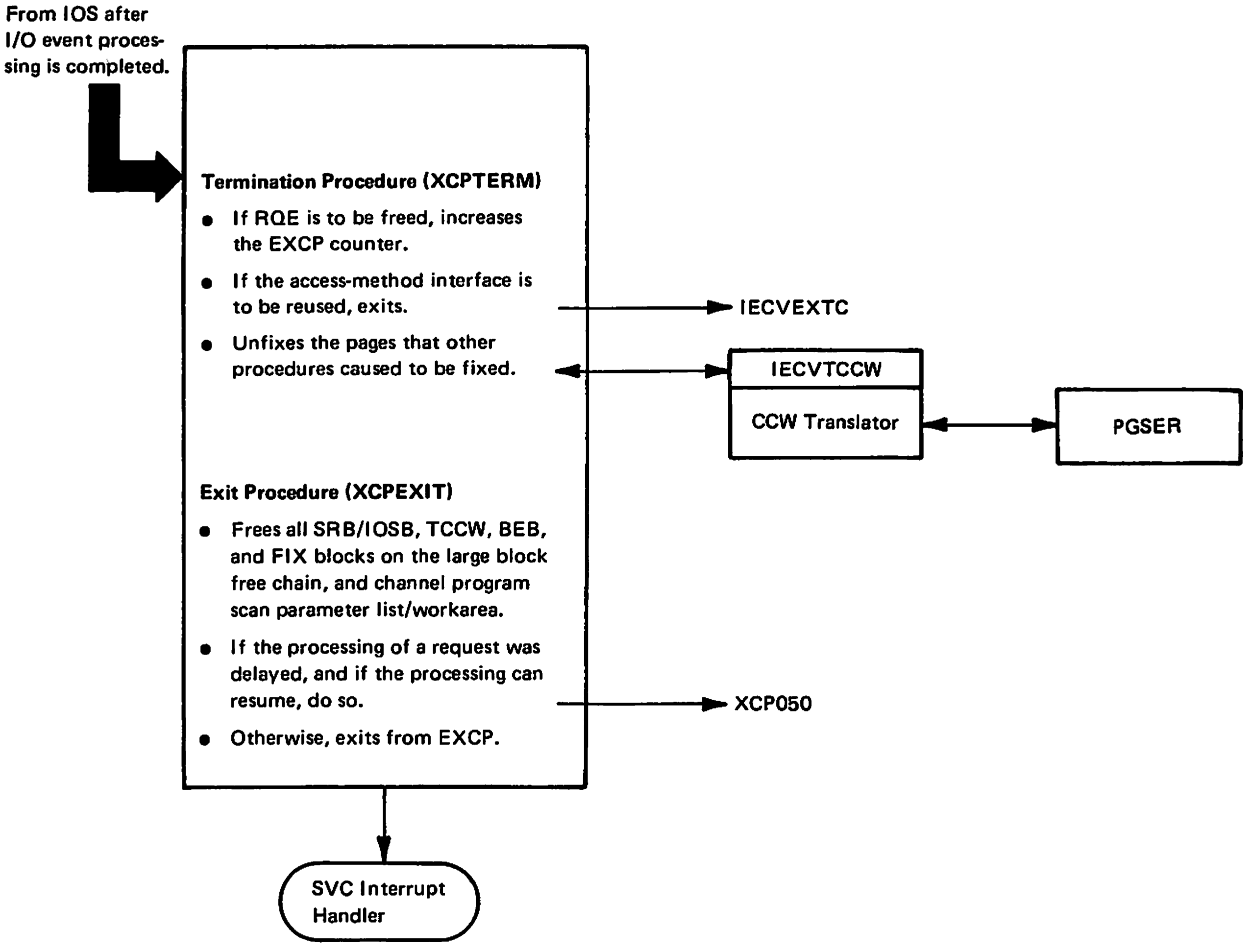
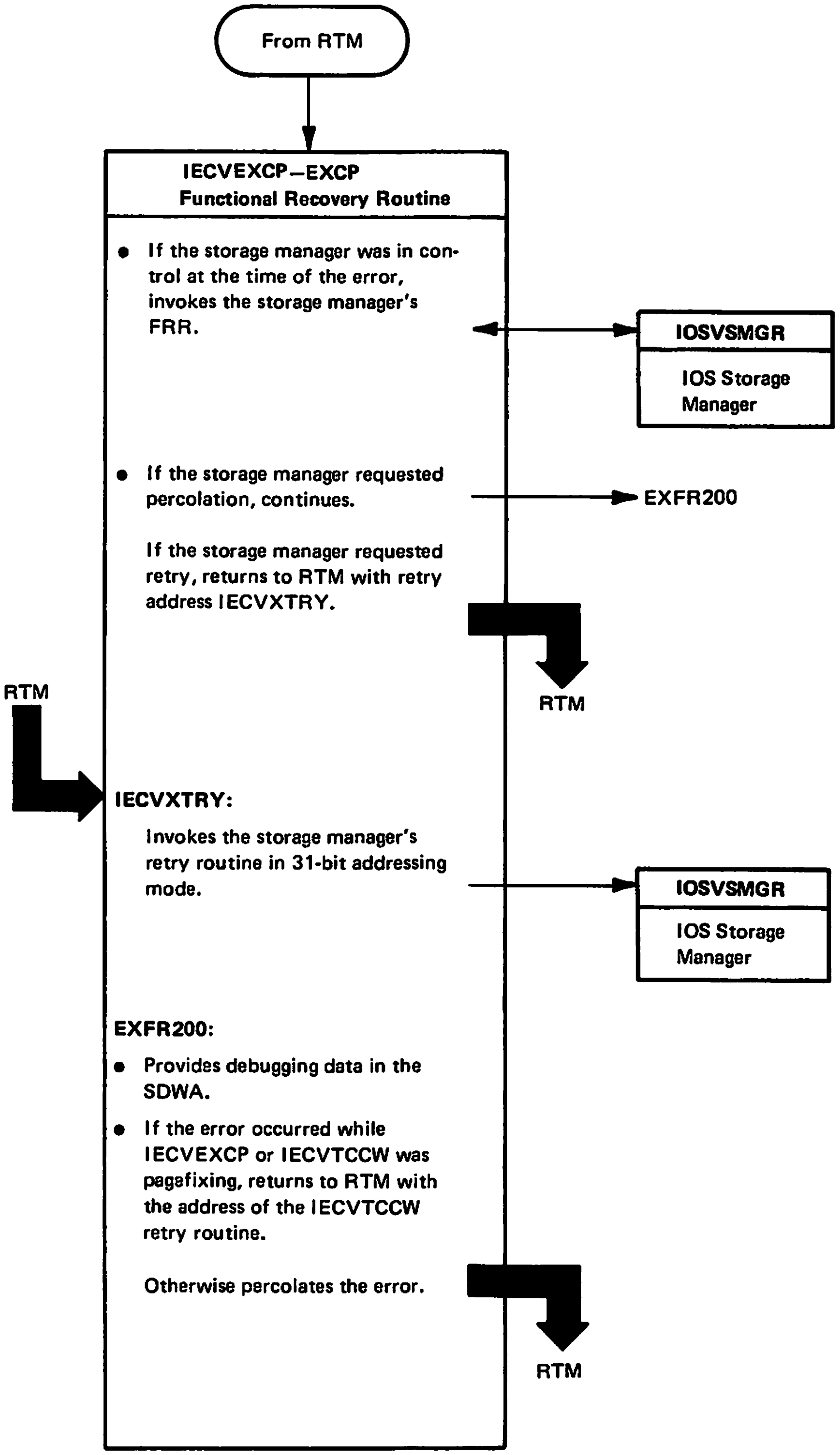

From IOS after I/O event processing is completed.
Termination Procedure (XCPTERM)
If RQE is to be freed, increases the EXCP counter.
If the access-method interface is to be reused, exits.
Unfixes the pages that other procedures caused to be fixed.
IECVEXTC
IECVTCCW
CCW Translator
PGSER
Exit Procedure (XCPEXIT)
Frees all SRB/IOSB, TCCW, BEB, and FIX blocks on the large block free chain, and channel program scan parameter list/workarea.
If the processing of a request was delayed, and if the processing can resume, do so.
Otherwise, exits from EXCP.
XCP050
SVC Interrupt Handler

Figure 2 (Part 4 of 4). IECVEXCP Process Flow



- If the storage manager was in control at the time of the error, invokes the storage manager's FRR.
- If the storage manager requested percolation, continues.

  If the storage manager requested retry, returns to RTM with retry address IECVXTRY.

IECVXTRY:

Invokes the storage manager's retry routine in 31-bit addressing mode.

EXFR200:

- Provides debugging data in the SDWA.
- If the error occurred while IECVEXCP or IECVTCCW was pagefixing, returns to RTM with the address of the IECVTCCW retry routine.

  Otherwise percolates the error.

Figure 3. IECVEXFR Process Flow

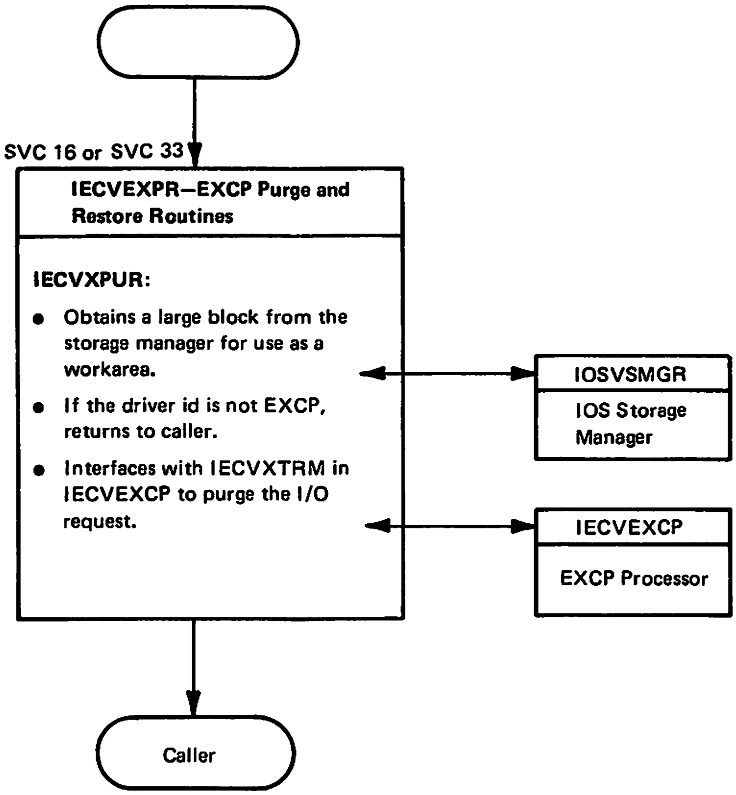

SVC 16 or SVC 33
IECVEXPR—EXCP Purge and Restore Routines
IECVXPUR:
● Obtains a large block from the storage manager for use as a workarea.
● If the driver id is not EXCP, returns to caller.
● Interfaces with IECVXTRM in IECVEXCP to purge the I/O request.
IOSVSMGR
IOS Storage Manager
IECVEXCP
EXCP Processor
Caller

Figure 4. IECVEXPR Process Flow

METHOD OF OPERATION

This section has detailed information for modules in this component. These modules are in alphabetic order. This detailed information is broken down into four different headings. The four headings and the topics they document are:

Module Description, which includes:

- Descriptive name
- Function (of the entire module)
- Entry point names, which includes:
  - Purpose (of the entry point)
  - Linkage
  - Callers
  - Input
  - Output
  - Exit normal
  - Exit error, if any
- External references, which includes:
  - Routines
  - Data areas, if any
  - Control blocks
- Tables
- Serialization

Note: Brief EXCP module descriptions are also included in MVS/Extended Architecture System Logic Library: Module Descriptions, which contains module descriptions for all the MVS/Extended Architecture components described in the System Logic Library.

Module Operation, which includes:

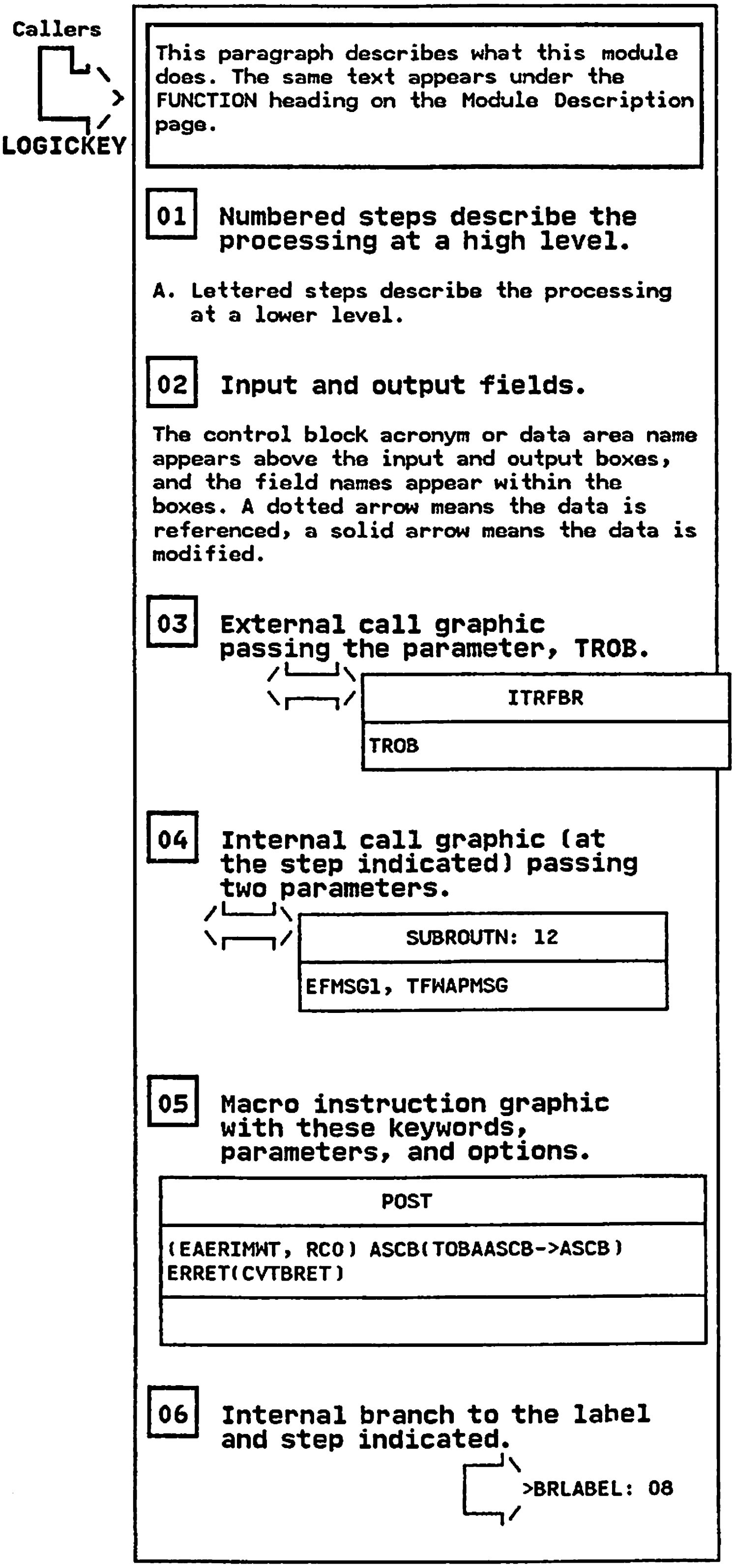
- Operation, which explains how the module performs its function.
- Recovery operation, which explains how the module performs any recovery.

Diagnostic aids, which provide information useful for debugging program problems; this includes:

- Entry point names
- Messages
- Abend codes
- Wait state codes
- Return codes for each entry point. Within each entry point, return codes might be further categorized by exit-normal and exit-error.
- Entry register contents for each entry point
- Exit register contents for each entry point

Logic Diagram, which illustrates the processing of the module, the input it uses, the output it produces, and the flow of control. Some modules do not have a logic diagram because the processing is sufficiently explained in the module description, the module operation, and the diagnostic aids sections. Figure 5 on page EXCP-22 illustrates the graphic symbols and format used in the logic diagrams.

LOGICKEY - Key to the Logic Diagrams STEP 01



This paragraph describes what this module does. The same text appears under the FUNCTION heading on the Module Description page.

01 Numbered steps describe the processing at a high level.

A. Lettered steps describe the processing at a lower level.

02 Input and output fields.

The control block acronym or data area name appears above the input and output boxes, and the field names appear within the boxes. A dotted arrow means the data is referenced, a solid arrow means the data is modified.

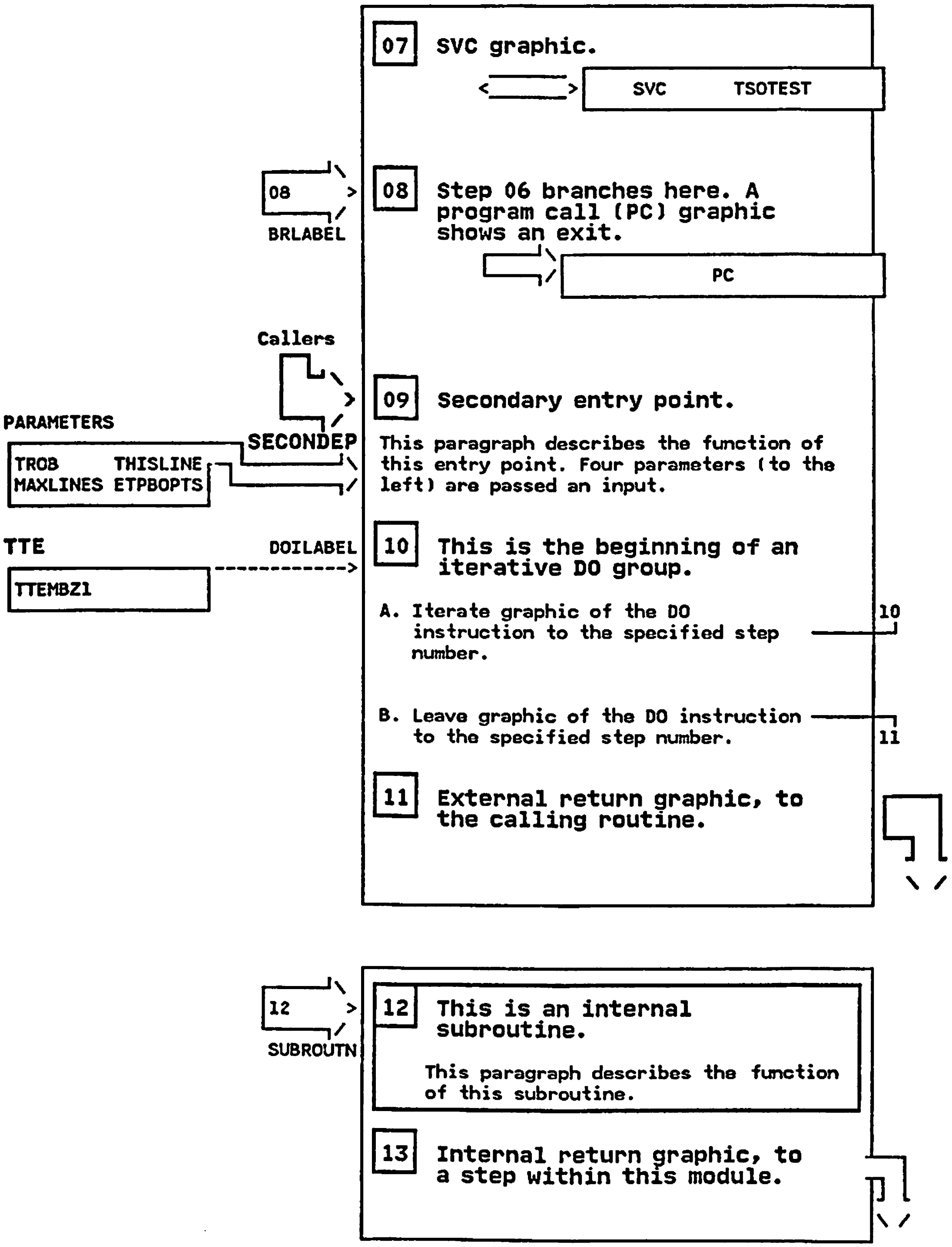
03 External call graphic passing the parameter, TROB.

04 Internal call graphic (at the step indicated) passing two parameters.

05 Macro instruction graphic with these keywords, parameters, and options.

06 Internal branch to the label and step indicated.

Figure 5 (Part 1 of 2). Key to the Logic Diagrams



SVC graphic.

Step 06 branches here. A program call (PC) graphic shows an exit.

Secondary entry point.

This paragraph describes the function of this entry point. Four parameters (to the left) are passed an input.

This is the beginning of an iterative DO group.

A. Iterate graphic of the DO instruction to the specified step number.

B. Leave graphic of the DO instruction to the specified step number.

External return graphic, to the calling routine.

This is an internal subroutine.

This paragraph describes the function of this subroutine.

Internal return graphic, to a step within this module.

Figure 5 (Part 2 of 2). Key to the Logic Diagrams

IECVEXCP - EXCP PROCESSOR FOR SVC 0(EXCP) AND SVC 114(EXCPVR)

IECVEXCP - MODULE DESCRIPTION

DESCRIPTIVE NAME: EXCP Processor for SVC 0 (EXCP)
and SVC114 (EXCPVR)

FUNCTION:
This module processes EXCP and EXCPVR I/O requests. As a driver of IOS, this module handles the initiation of a caller's request to IOS, handles the I/O interruption from IOS, and passes the results back to the caller through its appendages.

ENTRY POINT: IGC000

PURPOSE: To process EXCP (SVC 0) requests.

LINKAGE: SVC

CALLERS: Issuers of SVC 0

INPUT: IOB, TCB

OUTPUT: EXCP request readied for I/O initiation.

EXIT NORMAL: Return to SVC type 1 exit

EXIT ERROR: To RTM

ENTRY POINT: IGC114

PURPOSE: To process EXCPVR (SVC 114) requests.

LINKAGE: SVC

CALLERS: Issuers of SVC 114

INPUT: IOB, TCB

OUTPUT: EXCPVR request readied for I/O initiation.

EXIT NORMAL: Return to SVC type 1 exit

EXIT ERROR: To RTM

ENTRY POINT: IGC092

PURPOSE:
To process EXCP or EXCPVR requests for TSO restore (SVC 92).

LINKAGE: SVC

CALLERS: Issuers of SVC 92

INPUT: IOB, TCB

OUTPUT: EXCP or EXCPVR request readied for I/O initiation

EXIT NORMAL: Return to SVC type 1 exit

EXIT ERROR: To RTM

ENTRY POINT: XCPCHE - Normal-end Exit

PURPOSE:
To interface with the requestor's channel-end appendage.

LINKAGE: Branch and link

CALLERS:
IECVPST (IOS Post Status),
IECVEXCP front-end and back-end (termination)
processing routines

INPUT: IOSB

OUTPUT:
IOB updated to reflect the completed request.
ECB posted with one of the following completion codes:

7F - Normal completion
41 - Permanent error
42 - Extent violation
48 - Request purged

EXIT NORMAL: Return to caller

EXIT ERROR: To RTM

ENTRY POINT: XCPABE - Abnormal-end Exit

PURPOSE:
To interface with the EXCP requestor's
abnormal-end appendage.

LINKAGE: Branch and Link

CALLERS:
IECVPST (IOS Post Status),
IECVEXCP front-end and back-end (termination)
processing routines

INPUT: IOSB

OUTPUT:
IOB updated to reflect the completed request.
ECB posted with one of the following completion codes:

7F - Normal completion
41 - Permanent error
42 - Extent violation
48 - Request purged

EXIT NORMAL: Return to caller

EXIT ERROR: To RTM

ENTRY POINT: XCPDIE - Disabled Interrupt Exit (DIE)

PURPOSE:
To initiate a type 3 related request and
interface with a caller's program controlled
interrupt (PCI) appendage.

LINKAGE: Branch and Link

CALLERS:
IOS Disabled Interruption Routine that
interfaces with the driver's DIE exits.

INPUT: IOSB

OUTPUT: None

EXIT NORMAL: Return to caller

ENTRY POINT: XCPPCI - PCI Exit

PURPOSE: To interface with the caller's PCI appendage.

IECVEXCP - MODULE DESCRIPTION (Continued)

LINKAGE: Branch and Link

CALLERS: IECVPST (IOS Post Status)

INPUT: IOSB

OUTPUT: None

EXIT NORMAL: Return to caller

EXIT ERROR: To RTM

ENTRY POINT: IECVEXTC

PURPOSE: To perform extent check for DASD devices.

LINKAGE: BALR

CALLERS: IECVDERP (DASD error recovery procedure (ERP))

INPUT: IOSB

OUTPUT: None

EXIT NORMAL: Return to caller

EXIT ERROR: To RTM

ENTRY POINT: IECVX025

PURPOSE: To free the request queue element (RQE).

LINKAGE: BASR, BASSM

CALLERS:
SVC 3 exit routine,
IECVEXPR purge routine - Purge halt for
RB and AEQ purging.

INPUT: RQE block to be freed

OUTPUT: RQE block returned to the storage manager

EXIT NORMAL: Return to caller

EXIT ERROR: To RTM

ENTRY POINT: IECVXTRM

PURPOSE: To process a purge or FRR termination request.

LINKAGE: BALR

CALLERS:
IECVEXPR - EXCP purge routine,
IECVEXFR - FRR termination request

INPUT: RQE block

OUTPUT: RQE and large blocks returned to the storage manager.

EXIT NORMAL: Return to caller

EXIT ERROR: To RTM

EXIT ERROR: ABEND

IECVEXCP - MODULE DESCRIPTION (Continued)

EXTERNAL REFERENCES:

ROUTINES:

IARPSIV	- Perform page fix services
IEAOPT02	- Post with validity check
IEASMFEX	- Count the EXCP request and accumulate the device connect time (DCTI)
IECVQCNT	- Decrease quiesce count
IECVRCHN	- Add an IOB to the quiesce chain
IECVSMGR	- Obtain and return RQE and large blocks
IECVTCCW	- Translate a caller's virtual channel program
IECVEXFR	- Perform functional recovery processing
IFGDEBVR	- Perform DEB check

CONTROL BLOCKS:

ASCB	-- Address space control block
ASXB	-- Address space extension block
CVT	-- Communications vector table
DCB	-- Data control block
DEB	-- Data extent block
ECB	-- Event control block
FRRS	-- Functional recovery routine setrp
ICQE	-- Interrupt control queue block
IOB	-- I/O block
IOCOM	- I/O communication area
IOSB	-- I/O supervisor block
IPIB	-- IOS purge interface block
JSCB	-- Job step control block
PIRL	-- Purged I/O restore list
PSA	-- Prefixed save area
RD	-- Region descriptor
RRQ	-- Related request queue
RQE	-- Request queue element
SRB	-- Service request block
TCB	-- Task control block
TCCW	-- Translate CCW control block
UCB	-- Unit control block
WSAVT	-- Work save area vector table

IECVEXCP - MODULE OPERATION

This module processes EXCP and EXCPVR I/O requests (Also handles the SVC 92 request). It accepts the caller's IOB, DCB, DEB, and ECB and maps them into an IOS driver SRB/IOSB interface for initiation and I/O interrupt processing.

To perform this driver interface between the EXCP or EXCPVR callers and IOS, this module provides three functions: front-end processing (see label XCP000), normal-end and abnormal-end exit processing (see label XCP203A), and back-end (termination) processing (see label XCPTERM). Also, this module provides a disabled interrupt routine (DIE), PCI exit routine and an extent checking routine for the DASD ERP.

The portion that maps the caller's control block to the SRB/IOSB interface is called EXCP front-end processing, which includes the following:

- Validity checking the user's control blocks and issuing abends for inconsistencies.
- Issuing a C22 abend if the number of allowable outstanding EXCP/EXCPVR requests has been exceeded. The maximum per address space is 500.
- Obtaining and initializing an EXCP request queue element (RQE) control block as the EXCP anchor for the caller's request.
- If the UCB indicates the request is for a VIO data set (UCBURDEV), interfacing with VIO for the request.
- If the caller's IOB indicates that this is a related request, chaining the RQE block to the related request queue (RRQ) in the DEB and determining whether to process the request now or to wait for the completion of a previous request on the RRQ.
- Obtaining large blocks needed to process the caller's I/O request. These include blocks for the SRB/IOSB, TCCW, CPS (optional), BEB, and FIX (the last two are required for a virtual EXCP request).
- Initializing the IOSB and SRB.
- Creating a CPS block when the DDT indicates that the device supports channel program scan.
- Determining if the DEB block needs to be fixed for an EXCP V=R request that provides a PCI appendage.
- For EXCPVR and virtual EXCP requests, entering the caller's page-fix appendage.
- For EXCPVR requests, interfacing with the system paging services(PGSER) to fix the caller's fix list.
- For DASD devices, performing extent checking.
- Interfacing with the caller's SIO appendage.
- For virtual EXCP requests, interfacing with the EXCP module IECVTCCW to translate the caller's virtual channel program to a real channel program.
- If a device channel program scan (CPS) exit is provided, interfacing with it with the STARTIO indication.
- Issuing the STARTIO macro to pass the caller's request to IOS for execution.
- Returning to the caller via the exit for type 1 SVCs.

Upon completion of an I/O request, the IOS post status module (IECVPST) passes control back to this module for normal-end or abnormal-end exit processing. This module in turn interfaces with the requestor's channel-end or abnormal-end appendages. EXCP'S normal-end and abnormal-end exit processing consists of the following:

- Validity checking the caller's control blocks.
- Interfacing with the SMF routine to accumulate the device connect time and/or the EXCP count.
- Mapping the results of the I/O request from the IOSB to the caller's IOB.
- If a channel program scan (CPS) exit exists, interfacing with the CPS exit routine. If entry is to the normal-end appendage, the CPS function code is set to the normal function code. Otherwise, for the abnormal-end appendage call, enters the CPS routine with the I/O error function code.
- Interfacing with the caller's normal-end or abnormal-end appendage.
- Handling the possible return conditions from the caller's appendages, as follows:
  - a) Normal completion - two conditions can exist, depending on the setting of the IOB exception bit (IOBIOERR):
    - If the exception bit is off in the IOB, the I/O request has completed successfully and EXCP proceeds to its back-end processing to terminate the request.
    - If the exception bit is on in the IOB, the I/O operation did not complete successfully and the appendage requests the following error recovery procedure (ERP) processing:
      - Mapping bits and fields from the IOB to the IOSB.
      - If this is a related request and the IOBECBCC field indicates a permanent error, setting the DCB permanent bit in the caller's DCB.
      - If a channel program scan (CPS) exit exists, interfacing with the device channel program scan (CPS) exit with the STARTIO condition (the virtual and real starting addresses in the IOSB have been updated from the IOB).
      - If the IOS completion code is not a permanent error code, returning to IOS post status. This is done to allow the IOS post status routine to interface with ERP processing.
      - If the IOSB completion code indicates a permanent error, proceeding to EXCP back-end processing to terminate the request.
  - b) Do not post - The appendage indicates that the I/O request is complete but that the caller's ECB is not to be posted.
    EXCP proceeds to its back-end processing to terminate the request without posting the caller's ECB.
  - c) Retry request - The appendage indicates that the I/O request is to be retried (normally this means that the I/O request completed successfully and that the driver wants to start another I/O request). EXCP proceeds to its back-end processing to terminate the request without posting the caller's ECB and then returns to the EXCP front-end processing to perform the retry request.
  - d) Retry request from the top of the related request queue (RRQ) - The appendage indicates that the I/O request at the top of the RRQ is to be retried.
    EXCP proceeds to its back-end processing to terminate the request without posting the caller's ECB and then returns to the EXCP front-end processing to perform the retry

request.

EXCP back-end processing terminates the caller's request. Normally, EXCP back-end processing receives control from the EXCP normal/abnormal-end exit, as indicated above. The IOS post status module enters back-end processing when error recovery processing (ERP) indicates that the caller's request is in permanent error.
The IOS post status module also enters back-end processing when the IOS3 completion code indicates abnormal completion (X'45').
EXCP back-end processing includes:

. If the IOSB completion code indicates abnormal completion (X'45'), issuing a CALLRTM macro with type=ABTERM with a completion code of E00.
. For tape devices, increasing the DCB block count from the IOB block count.
. If the caller's appendage requested retry, performing the retry request.
. If page fixing was done for the request, performing the unfix processing.
. If the request is to be posted, interfacing with the system post routine to post the caller's ECB.
. Returning the control blocks (RQE and large blocks) obtained for the caller's request to the IOS storage manager.
. If this is a related request and EXCP has readied the caller's next I/O request, issuing the STARTIO macro to send the next request to IOS for initiation.
. Returning to IOS post status with return code 16 to indicate that back-end processing is complete, or returning to the EXCP front-end processing to perform the retry request.

IECVEXCP provides a DIE (disabled interrupt exit) for the following situations:

. When the caller specifies related request type 3 (see explanation below) in the IOB
. For EXCPVR or EXCP (V=R) requests when the caller has specified a PCI appendage.

The EXCP DIE routine is entered from IOS when IOS is performing I/O disabled interruption handling.

IECVEXCP provides a PCI (program controlled interrupt) exit to support the virtual EXCP request that provides a valid PCI appendage (not just a BR 14 instruction).
Upon completion of an I/O request, the IOS post status module (IECVPST) passes control to the EXCP PCI exit routine when the PCI bit is set in the subchannel status'and EXCP has set an address in the IOSPCI field of the IOSB. The EXCP PCI exit routine interfaces with the requestor's PCI appendage.

EXCP provides a special facility known as related request processing. Related request processing is indicated by the caller in his IOB block. Related requests are directed to the same data set and share the same DCB. Handling of related requests is as follows:

. EXCP chains related requests on the associated DEB related request queue (RRQ) in the order that they are received.
. They are processed by EXCP in the order received, with some overlap. The amount of overlap is dependent on the related request type.

If a related request is considered in permanent error (I/O request did not complete sucessfully), none of the remaining related requests are processed. They are purged and returned to the caller before the request with the permanent error is posted to the caller.

There are three types of related requests. These three types tell EXCP when to proceed with the next EXCP request upon completion of this EXCP request. The three types are:

1) Related request type 1:
   The I/O operation for request 'n' must complete and the callers channel-end or abnormal-end appendage must look at the status of the operation before EXCP starts processing request 'n+1' on the related request queue. For this type, EXCP back-end processing returns to EXCP front-end processing at the point where large blocks are obtained to begin handling the request.
2) Related request type 2:
   The I/O operation for request 'n' must complete and the caller's channel-end or abnormal-end appendage must look at the status of the operation before EXCP can issue the STARTIO macro to send request 'n+1' to IOS for initiation.
   The difference between type 1 and 2 is that, for type 2, EXCP has processed request 'n+1' up to the point of issuing the STARTIO macro to send the request to IOS for initiation. The STARTIO macro is issued in EXCP back-end processing.
   In fact, EXCP will process up to four requests to the point where they are ready for I/O initiation.
3) Related request type 3:
   The I/O operation for request 'n' must complete and the EXCP DIE routine must examine the IOSB subchannel status (SCSW) for normal completion (channel end bit set without any any error condition bits set). If the SCSW indicates normal status, the EXCP DIE requests that IOS issue the STARTIO macro for request 'n+1'. If the SCSW indicates other than normal completion, or if the request is being purged or being retried out of the ERP, the EXCP DIE does not initiate request 'n+1'. In this case, it will be handled like type 2.
   The difference between type 1 and 3 is that, for type 3, EXCP has processed request 'n+1' up to the point of issuing the STARTIO macro to send the request to IOS for initiation.
   In fact, EXCP will process up to four requests to the point where they are ready for I/O initiation.

IECVEXCP - DIAGNOSTIC AIDS

ENTRY POINT NAMES: IGC000
IGC114
IGC092
XCPCHE - Normal-end Exit
XCPABE - Abnormal-end Exit
XCPDIE - Disabled Interrupt Exit (DIE)
XCPPCI - PCI Exit
IECVEXTC
IECVX025
IECVXTRM

MESSAGES: None

ABEND CODES:

The following abends are generated in IECVEXCP (via SVC 13) and are processed by IECVEXFR (EXCP functional recovery routine).

```
 15C - Issuer of SVC 92 is not in supervisor state.
 172 - SVC 114 was issued and:
       - Protect key is not 0, or
       - Request was not issued in supervisor state, or
       - Authorization bit is not set in JSCB.
 300 - One of the following:
       - DEB not found on the DEB chain (validity check failure).
       - DEB is not an EXCP or ISAM DEB.
       - The IOBM index is larger than the DEBNMEXT index or
         both indexes are not zero.
 400 - DCB pointers in the IOB and DEB do not match.
 500 - One of the following:
       - DEB does not point to a valid UCB.
       - An ISAM IOB IOBM field specified extent 0.
 800 - One of the following:
       - Error attempting to fix pages for the request.
       - Error attempting to unfix pages for the request.
```

The following completion codes are set by IECVEXFR (EXCP functional recovery routine) as a result of a program check or indeterminate error during EXCP processing:

```
 200 - IOB, DCB, or ECB protect key is not
       the same as the user's key.
 700 - A program check occurred while in a supervisor service
       routine invoked by EXCP.
 A00 - A program check occurred in a user appendage.
 B00 - Indeterminate error.
```

The following abends are generated and processed in IECVEXCP via CALLRTM:

```
 C22 - Address space exceeded the maximum number of
       outstanding EXCPs.
 E00 - A program check occurred in IOS and no EXCP debug area
       is available.
```

WAIT STATE CODES: None

RETURN CODES:

ENTRY POINT IGC000: None

ENTRY POINT IGC114: None

ENTRY POINT IGC092: None

ENTRY POINT XCPCHE:

EXIT NORMAL:

Register 15 contains a decimal value:
0 - Normal completion with the IOSEX bit set in the IOSB
16 - Indicate to IOS post status that EXCP has performed its termination processing

ENTRY POINT XCPABE:

EXIT NORMAL:

Register 15 contains a decimal value:
0 - Normal completion with the IOSEX bit set in the IOSB
16 - Indicate to IOS post status that EXCP has performed its termination processing

ENTRY POINT XCPDIE:

EXIT NORMAL:

In register 15
0 - Normal return
4 - Initiate a new IOSB request (handle a type 3 related request)
8 - Ignore return

ENTRY POINT XCPPCI: None

ENTRY POINT IECVEXTC:

EXIT NORMAL:

Register 15 contains the following:
0 - Retry the request
4 - Post the requestor

ENTRY POINT IECVX025: None

ENTRY POINT IECVXTRM: None

REGISTER CONTENTS ON ENTRY:

ENTRY POINT IGC000:

Register 0	- Irrelevant
Register 1	- IOB address
Register 2	- Irrelevant
Register 3	- CVT address
Register 4	- TCB address
Register 5	- Current RB pointer
Register 6	- Entry address
Register 7	- ASCB address
Registers 8-12	- Irrelevant
Register 13	- Save area address
Register 14	- Return address
Register 15	- Irrelevant

ENTRY POINT IGC114:

Register 0	- Irrelevant
Register 1	- ICB address, with byte 0 set to X'F4'.
Register 2	- Irrelevant
Register 3	- CVT address

IECVEXCP - DIAGNOSTIC AIDS (Continued)

Register 4	- TCB address
Register 5	- Current RB pointer
Register 6	- Entry address
Register 7	- ASCB address
Registers 8-12	- Irrelevant
Register 13	- Save area address
Register 14	- Return address
Register 15	- Irrelevant

ENTRY POINT IGC092:

Register 0	- TCB address
Register 1	- IOB address, with byte 0 set to X'00' (EXCP) or to X'F4' (EXCPVR)
Register 2	- Irrelevant
Register 3	- CVT address
Register 4	- TCB address
Register 5	- Current RB pointer
Register 6	- Entry address
Register 7	- ASCB address
Registers 8-12	- Irrelevant
Register 13	- Save area address
Register 14	- Return address
Register 15	- Irrelevant

ENTRY POINT XCPCHE:

Register 0	- Irrelevant
Register 1	- IOSB address
Registers 2-5	- Irrelevant, but must not be destroyed
Registers 6-12	- Irrelevant, and are available to the exit if a save area is not available.
Register 13	- Address of the local lock save area
Register 14	- Return address
Register 15	- Entry point address

ENTRY POINT XCPABE:

Register 0	- Irrelevant
Register 1	- IOSB address
Registers 2-5	- Irrelevant, but must not be destroyed
Registers 6-12	- Irrelevant, and are available to the exit if a save area is not available.
Register 13	- Address of the local lock save area
Register 14	- Return address
Register 15	- Entry point address

ENTRY POINT XCPDIE:

Register 0-1	- Irrelevant
Register 2	- IOSB address
Registers 2-12	- Irrelevant
Register 13	- Address of a save area
Register 14	- Return address
Register 15	- Entry point address

ENTRY POINT XCPPCI:

Register 0	- Irrelevant
Register 1	- IOSB address
Register 2-5	- Irrelevant, but must not be destroyed
Registers 6-12	- Irrelevant
Register 13	- Address of the local lock save area when entered from IECVPST
	- Address of a save area in the TCCW block when entered from IECVEXCP DIE routine
Register 14	- Return address

IECVEXCP - DIAGNOSTIC AIDS (Continued)

```
  Register 15    - Entry point address
```

ENTRY POINT IECVEXTC:

```
  Register  0    - Irrelevant
  Register  1    - IOSB address
  Registers 2-12 - Irrelevant
  Register 13    - Irrelevant
  Register 14    - Return address
  Register 15    - Entry address
```

ENTRY POINT IECVX025:

```
  Register  1    - RQE address
  Registers 2-12 - Irrelevant
  Register 13    - Save area address
  Register 14    - Return address
  Register 15    - Entry address
```

ENTRY POINT IECVXTRM:

```
  Register  1    - RQE address
  Registers 2-12 - Irrelevant
  Register 13    - Save area address
  Register 14    - Return address
  Register 15    - Entry address
```

REGISTER CONTENTS ON EXIT:

ENTRY POINT IGC000:

EXIT NORMAL:

```
  Register  0    - Unpredictable
  Register  1    - ICB address
  Registers 2-15 - Unpredictable
```

ENTRY POINT IGC114:

EXIT NORMAL:

```
  Register  0    - Unpredictable
  Register  1    - ICB address
  Registers 2-15 - Unpredictable
```

ENTRY POINT IGC092:

EXIT NORMAL:

```
  Register  0    - Unpredictable
  Register  1    - IOB address
  Registers 2-15 - Unpredictable
```

ENTRY POINT XCPCHE:

EXIT NORMAL:

```
With return code in register 15 = 0
     Registers 0-13 - Same as on entry
     Register 14    - Return address
     Register 15    - Return code = 0

With return code in register 15 = 16
     Registers 0-3  - Unpredictable
     Register  4    - Same as on entry
     Register  5    - Same as on entry
     Registers 6-12 - Unpredictable
     Register 13    - Same as on entry
     Register 14    - Return address
     Register 15    - Return code = 16
```

ENTRY POINT XCPABE:

EXIT NORMAL:

```
  With return code in register 15 = 0
     Registers 0-13 - Same as on entry
     Register 14    - Return address
     Register 15    - Return code = 0

  With return code in register 15 = 16
     Registers 0-3  - Unpredictable
     Register  4    - Same as on entry
     Register  5    - Same as on entry
     Registers 6-12 - Unpredictable
     Register 13    - Same as on entry
     Register 14    - Return address
     Register 15    - Return code = 16
```

ENTRY POINT XCPDIE:

EXIT NORMAL:

```
  Registers 0-14 - Same as on entry
  Register 15    - Return code
```

ENTRY POINT XCPPCI:

EXIT NORMAL:

```
  Registers 0-15 - Same as on entry
```

ENTRY POINT IECVEXTC:

EXIT NORMAL:

```
  Registers 0-3  - Unpredictable
  Registers 4-14 - Same as on input
  Register 15    - Return code
```

ENTRY POINT IECVX025:

EXIT NORMAL:

```
  Register   0    - Same as on entry
  Register   1    - Unpredictable
  Registers  2-9  - Same as on entry
  Register  10    - Unpredictable
  Registers 11-13 - Same as on entry
  Registers 14-15 - Unpredictable
```

ENTRY POINT IECVXTRM:

EXIT NORMAL:

```
  Registers 0-12 - Unpredictable
  Register 13    - Same as on entry
  Register 14    - Same as on entry
  Register 15    - Unpredictable
```

IECVEXCP

This module processes EXCP and EXCPVR I/O requests. As a driver of IOS, this module handles the initiation of a caller's request to IOS, handles the I/O interruption from IOS, and passes the results back to the caller through its appendages.

Issuers of SVC 92

IGC092

01 This entry point handles the SVC 92 (5C) request.

The TCB address provided in register 0 is set in register 4, overlaying the environmental TCB address.

A. If the caller is not in supervisor state, issues ABEND 15C to abend the task.

>ABEND000: 72

B. If the IOB parameter register is positive, handles the request as an SVC 0 (EXCP).

C. If the IOB parameter register is negative, handles the request as an SVC 114 (EXCPVR).

RQE

RQE114

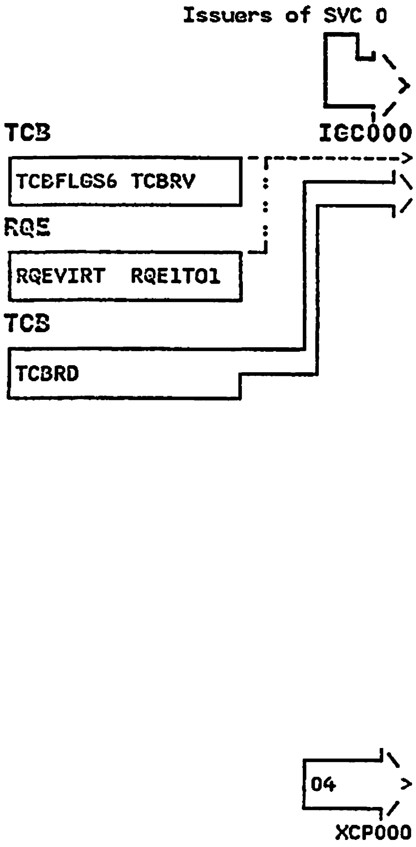
IGC114A

02 This entry point handles the SVC 114(72) EXCPVR request.

A. If the requestor is not authorized (not in system key 0-7, not in supervisor state, or not authorized in the JSCB), sets an indicator that further validity checks are required in the EXCP mainline for the EXCPVR request.

B. If the requestor is authorized, goes to IECVEXCP front-end processing.

>XCP000: 04



03 This entry point handles the SVC 0 (EXCP) request.

Determines if the EXCP request is a virtual EXCP or virtual=real (V=R) EXCP request.

If the bit in the TCB (TCBRV) is set indicating a V=R request, does the following V=R validity checks:
. If the first CCW of the channel program is not in the real region, treats the request as a virtual EXCP request.
. If the first CCW is in the real region, checks if the CVT authorizes V=R requests. If not, treats the request as a virtual EXCP request.
. Otherwise, determines if the V=R request is authorized to be a V=R user, in the following order: 1) Running in system key (0-7); 2) Running in supervisor state; or 3) JSCB indicates authorization. If the V=R request is not authorized, the V=R request is treated as a virtual EXCP request.

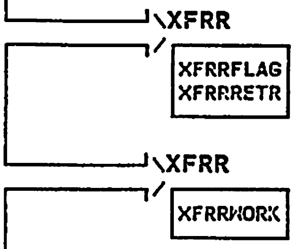
If the bit in the TCB (TCBRV) is off, the request is treated as a virtual EXCP request.

04 Performs IECVEXCP front-end processing.

Receives control from the EXCP or EXCPVR SVC entry point. WKREG6 indicates the type of entry and whether further EXCPVR validity checking is required.

A. Establishes module addressability, a functional recovery environment, and the IOB register.

B. Indicates that front-end processing is active and sets the type 1 SVC exit address in the FRR parameter area as the return address.



05 Validity checks the caller's control blocks.

In general, the validity check consists of accessing the caller's control block in the caller's key (key obtained from the RB old psw). If a protection check occurs, the functional recovery routine (IECVEXFR) will abend the requestor. If accessing the control block does not result in a protection check, then unique testing is done on the caller's control blocks and the appropriate abend is issued if an error is found.

The validity check is performed on the following blocks.

A. In caller's Key, the DCB pointer is loaded from the ICB and the DEB address is obtained from the DCB. The ECB address is obtained from the IOB and the ECB field is set to zero.

B. If the DCB pointers from the IOB and DEB blocks do not match, issues abend 400 to abend the task.

>ABEND000: 72

XFRR

XFRRWORK

XCP010 ---------->

C. If the caller is in problem program key, calls the DEB validity check routine to check if the DEB is on the DEB chain.

REGISTER APBSRG

RETURN REGISTER: LNKREG

05D >

XCP013

D. Other EXCP front-end routines that perform DEB validity checks enter at this point when the validity checks fail.

E. If the DEB was not found on the DEB chain, issues abend 300 to abend the task.

>ABEND000: 72

RQE

RQEKOBYP

XCP014 ---------->

F. Performs additional EXCPVR validity checking, as follows:

If the DEB (DEBESMVR) shows authorization, sets a bit in the FRR parameter area (RQEKOBYP) to indicate that this is a SAM request and that the IOB is in protected storage.

The DCBIOBAD field contains a pointer to a SAM control block (ICQ) which contains the ICB address. If the ICB address in the ICQ does not match the input ICB address, issues an abend 172 to abend the task.

If the DEB does not show authorization, issues an abend 172 to abend the task.

\XFRR

XFRRWORK

G. If the EXCPVR request fails the validity check, issues abend 172 to abend the task.

>ABEND000: 72

XFRR

XFRRWORK

DEB

DEBBASIC

XCP016

06 For problem program callers, performs additional validity checking, as follows.

A. In the key of the problem program caller, accesses the IOB and DCB and references the ECB. (For SAM requests, bypasses accessing the IOB.)

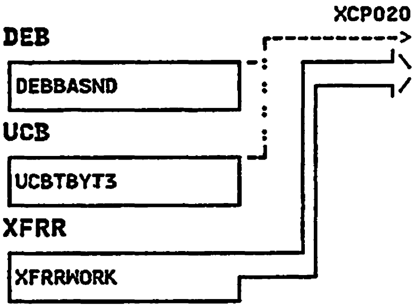
If a protection check occurs, the functional recovery routine (IECVEXFR) will issue abend 200 to abend the task.

B. If the DEB is not intended to be used by EXCP and the DEB type (DEBAMTYP) is not ISAM (DEBAMTYP), considers the DEB invalid and issues abend 300 to abend the task.

>XCP013: 05D

C. If this is a valid ISAM DEB and the IOBM field is zero, issues abend 500 to abend the task.

>ABEND000: 72

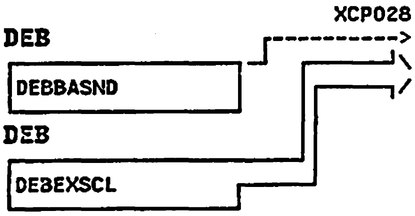


07 Performs IOBM and DEBNMEXT validity checking for direct access, graphic, and teleprocessing devices.

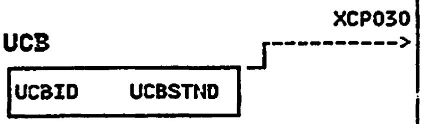
The IOBM field must be less than the DEB extent (DEBNMEXT) field. The IOBM is indexed from 0-n while the DEBNMEXT is indexed from 1-n (except that system-built DEBS may have an index of 0).

A. If the IOBM is larger than the DEBNMEXT or both indexes are not zero, issues abend 300 to abend the task.

>XCP013: 05D



08 For a valid IOBM extent, indexes into the DEB extent table to obtain the UCB address.



09 Checks if the UCB is valid (byte 3 not X'FF').

A. If the UCB is not valid, issues abend 500 to abend the task.

>ABEND000: 72

XCP035

10 Performs the following IOB initialization. (All of the caller's control blocks are valid at this point.)

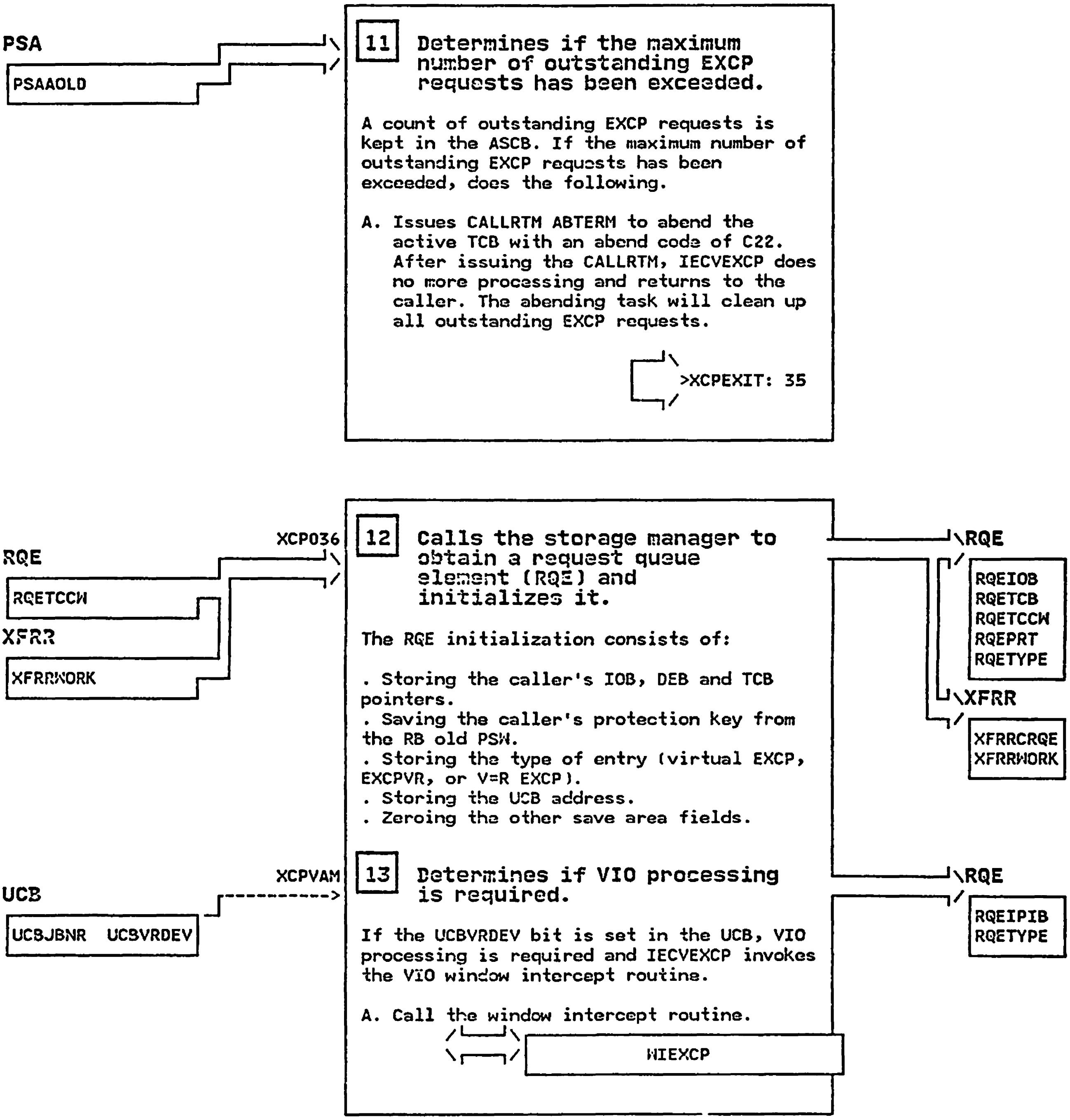
Zeroes the IOB error count.

Zeroes the CSW field.

Resets IOB flags.

Sets the ECB completion code to normal completion (X'7F').

IECVEXCP - EXCP Processor for SVC 0 (EXCP) and SVC114 (EXCPVR)   STEP 11



11 Determines if the maximum number of outstanding EXCP requests has been exceeded.

A count of outstanding EXCP requests is kept in the ASCB. If the maximum number of outstanding EXCP requests has been exceeded, does the following.

A. Issues CALLRTM ABTERM to abend the active TCB with an abend code of C22. After issuing the CALLRTM, IECVEXCP does no more processing and returns to the caller. The abending task will clean up all outstanding EXCP requests.

12 Calls the storage manager to obtain a request queue element (RQE) and initializes it.

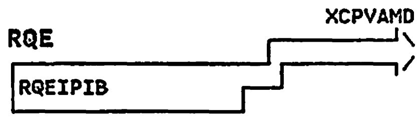
The RQE initialization consists of:

. Storing the caller's IOB, DEB and TCB pointers.
. Saving the caller's protection key from the RB old PSW.
. Storing the type of entry (virtual EXCP, EXCPVR, or V=R EXCP).
. Storing the UCB address.
. Zeroing the other save area fields.

13 Determines if VIO processing is required.

If the UCBVRDEV bit is set in the UCB, VIO processing is required and IECVEXCP invokes the VIO window intercept routine.

A. Call the window intercept routine.



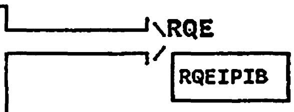
XCPVAMD

B. Upon return to IECVEXCP, the window intercept routine must at least restore the RQE pointer and the IECVEXCP base register (register 5).

The possible addresses returned to from the VAM window intercept routine are:
. Register 14+0 - Posts the caller's ECB and returns the RQE block to the storage manager.
. Register 14+4 - Does not post the caller's ECB and returns the RQE block to the storage manager.
. Register 14+8 - Does not post the caller's ECB and does not return the RQE block to the storage manager.

C. Goes to exit from IECVEXCP.

>XCPEXIT: 35

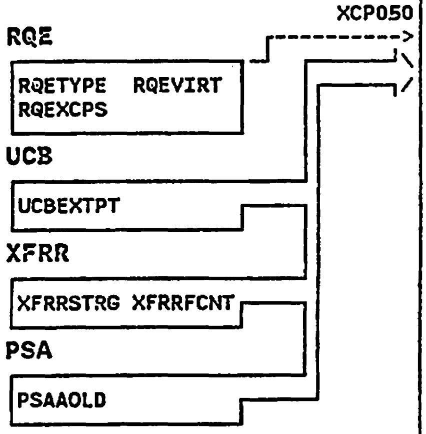


XCP038

14 If the caller's request is a related request, chains the RQE off the RRQ.

The related request handling routine (XCPRRQ00) determines whether to process the request or return to caller.

A. Goes to related request handling routine (XCPRRQ00).

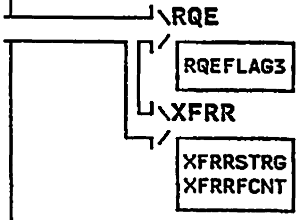


XCP050

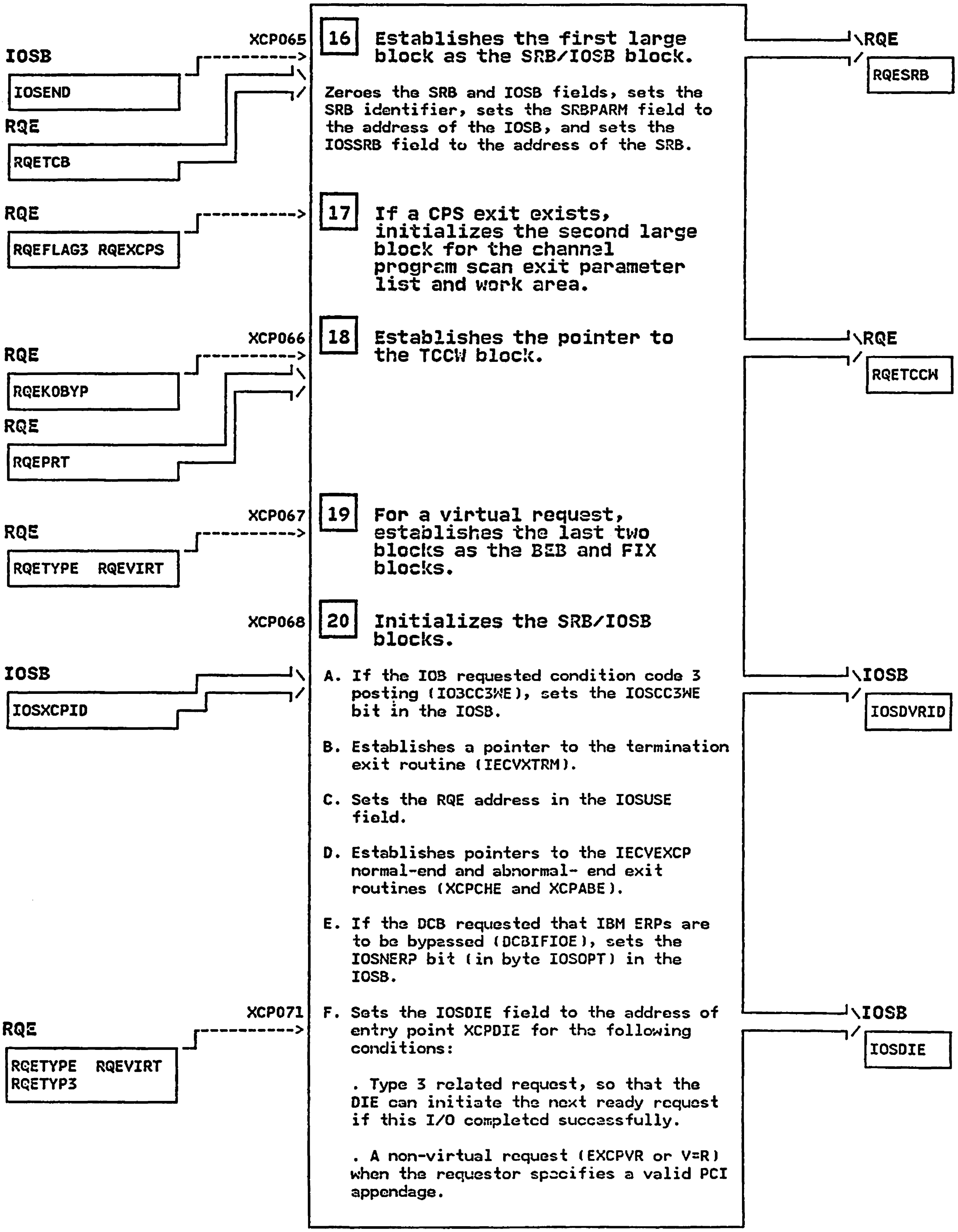
15 Obtains from the storage manager the number of large blocks required for the caller's request.

Two blocks are always obtained to be used as the SRB/IOSB and the TCCW blocks. For a virtual EXCP, two additional blocks are obtained for the BEB and FIX blocks. If a channel program scan (CPS) DDT exit exists, an additional block is obtained to contain the exit parameter list and exit work area (XCPS).

The storage manager returns the large blocks by chaining them together by the first word of each block. The chain field in the last block is zero.



IECVEXCP - EXCP Processor for SVC 0 (EXCP) and SVC114 (EXCPVR) STEP 16



16 Establishes the first large block as the SRB/IOSB block.

Zeroes the SRB and IOSB fields, sets the SRB identifier, sets the SRBPARM field to the address of the IOSB, and sets the IOSSRB field to the address of the SRB.

17 If a CPS exit exists, initializes the second large block for the channel program scan exit parameter list and work area.

18 Establishes the pointer to the TCCW block.

19 For a virtual request, establishes the last two blocks as the BEB and FIX blocks.

20 Initializes the SRB/IOSB blocks.

A. If the IOB requested condition code 3 posting (IOBCC3WE), sets the IOSCC3WE bit in the IOSB.

B. Establishes a pointer to the termination exit routine (IECVXTRM).

C. Sets the RQE address in the IOSUSE field.

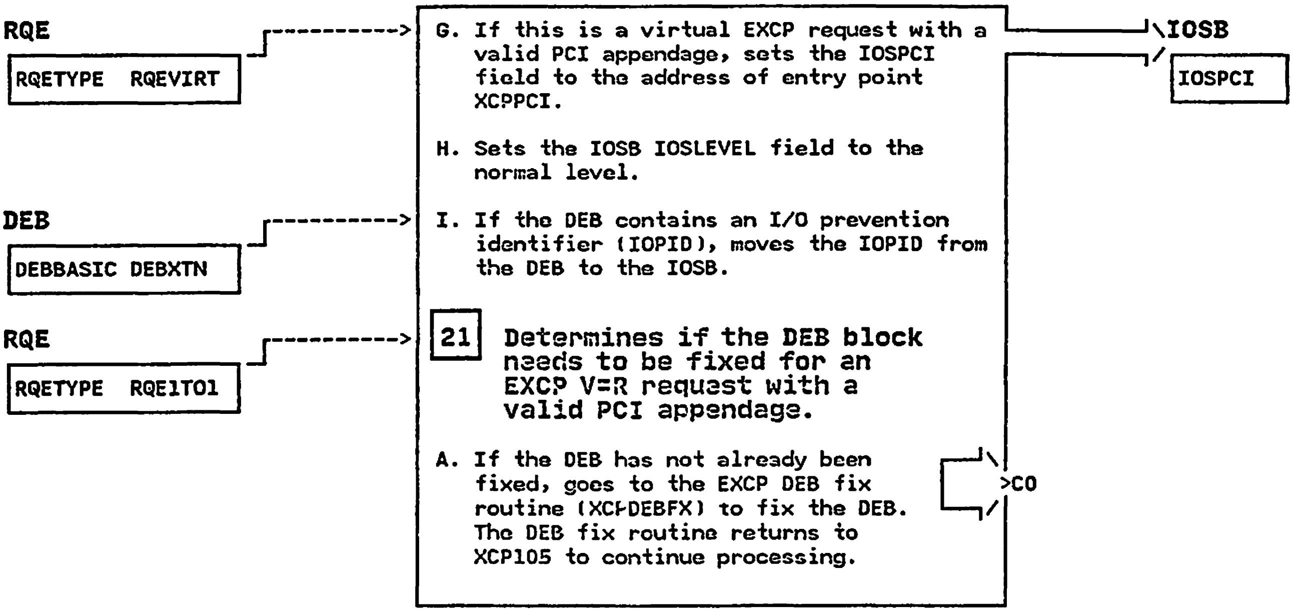
D. Establishes pointers to the IECVEXCP normal-end and abnormal- end exit routines (XCPCHE and XCPABE).

E. If the DCB requested that IBM ERPs are to be bypassed (DCBIFIOE), sets the IOSNERP bit (in byte IOSOPT) in the IOSB.

F. Sets the IOSDIE field to the address of entry point XCPDIE for the following conditions:

. Type 3 related request, so that the DIE can initiate the next ready request if this I/O completed successfully.

. A non-virtual request (EXCPVR or V=R) when the requestor specifies a valid PCI appendage.



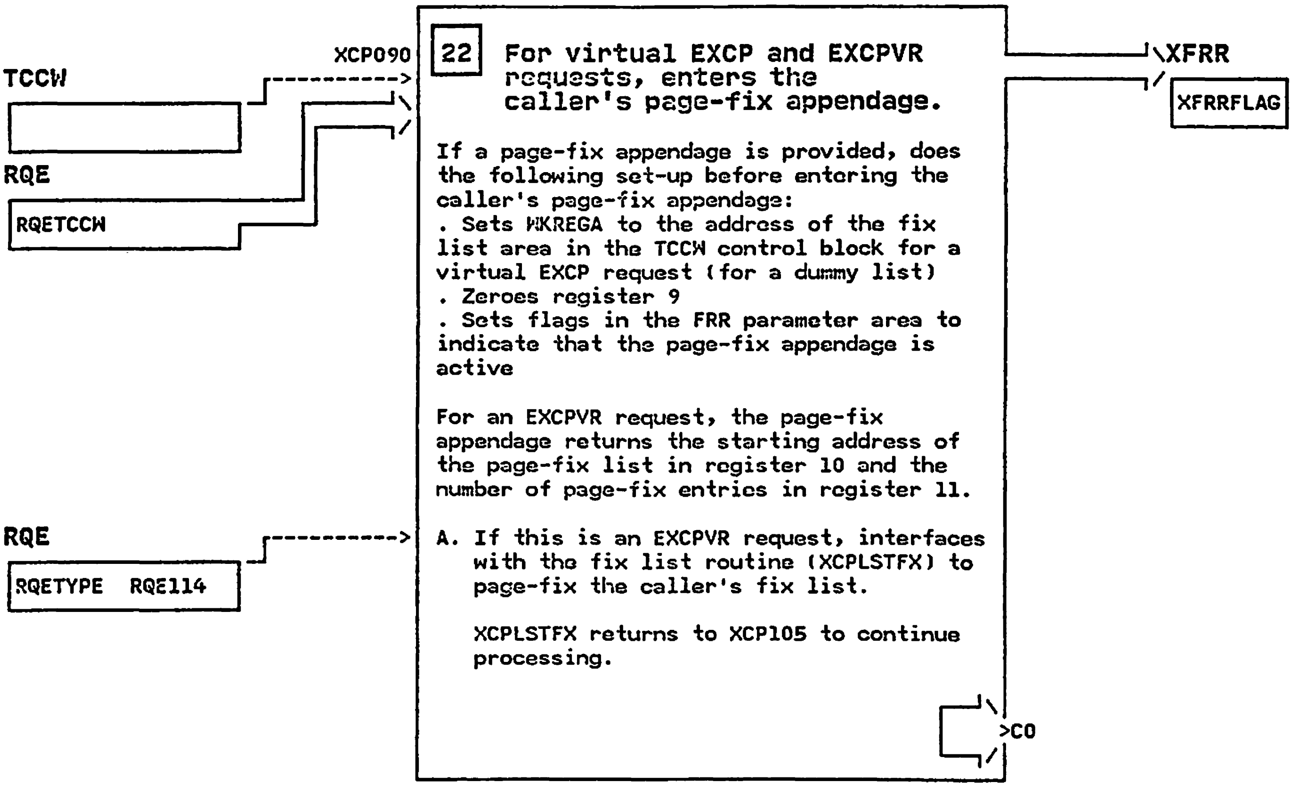
G. If this is a virtual EXCP request with a valid PCI appendage, sets the IOSPCI field to the address of entry point XCPPCI.

H. Sets the IOSB IOSLEVEL field to the normal level.

I. If the DEB contains an I/O prevention identifier (IOPID), moves the IOPID from the DEB to the IOSB.

21 Determines if the DEB block needs to be fixed for an EXCP V=R request with a valid PCI appendage.

A. If the DEB has not already been fixed, goes to the EXCP DEB fix routine (XCPDEBFX) to fix the DEB. The DEB fix routine returns to XCP105 to continue processing.



22 For virtual EXCP and EXCPVR requests, enters the caller's page-fix appendage.

If a page-fix appendage is provided, does the following set-up before entering the caller's page-fix appendage:
. Sets WKREGA to the address of the fix list area in the TCCW control block for a virtual EXCP request (for a dummy list)
. Zeroes register 9
. Sets flags in the FRR parameter area to indicate that the page-fix appendage is active

For an EXCPVR request, the page-fix appendage returns the starting address of the page-fix list in register 10 and the number of page-fix entries in register 11.

A. If this is an EXCPVR request, interfaces with the fix list routine (XCPLSTFX) to page-fix the caller's fix list.

XCPLSTFX returns to XCP105 to continue processing.

23 XCP105

UCB
UCBTBYT3

RQE
RQEFLAG

23 For DASD devices, handles the interface to the extent check routine.

A. Moves the IOB seek address (IOBSEEK) to the IOSB (IOSEEKA).

B. Calls the IECVEXCP extent check routine (XCPEXT) to see if the seek address is within the DEB low and high extents. If not, XCPEXT interfaces with the caller's end-of-extent appendage.

REGISTER WKREGB
RETURN REGISTER: LNKREG

C. If the extent is not within bounds (register 15 = 4), and the end-of-extent appendage has indicated to skip this operation, goes to IECVEXCP back-end processing at XCP508 to terminate the request.

>XCP508: 84

IOSB
IOSEEKA

XFRR
XFRRABE
RQE
RQETYPE

RRQ
RRQFIRST

D. If the extent is not within bounds (register 15 = 4) and this is not a related request, goes to interface with the caller's abnormal-end appendage at XCP202C.

>XCP202C: 50

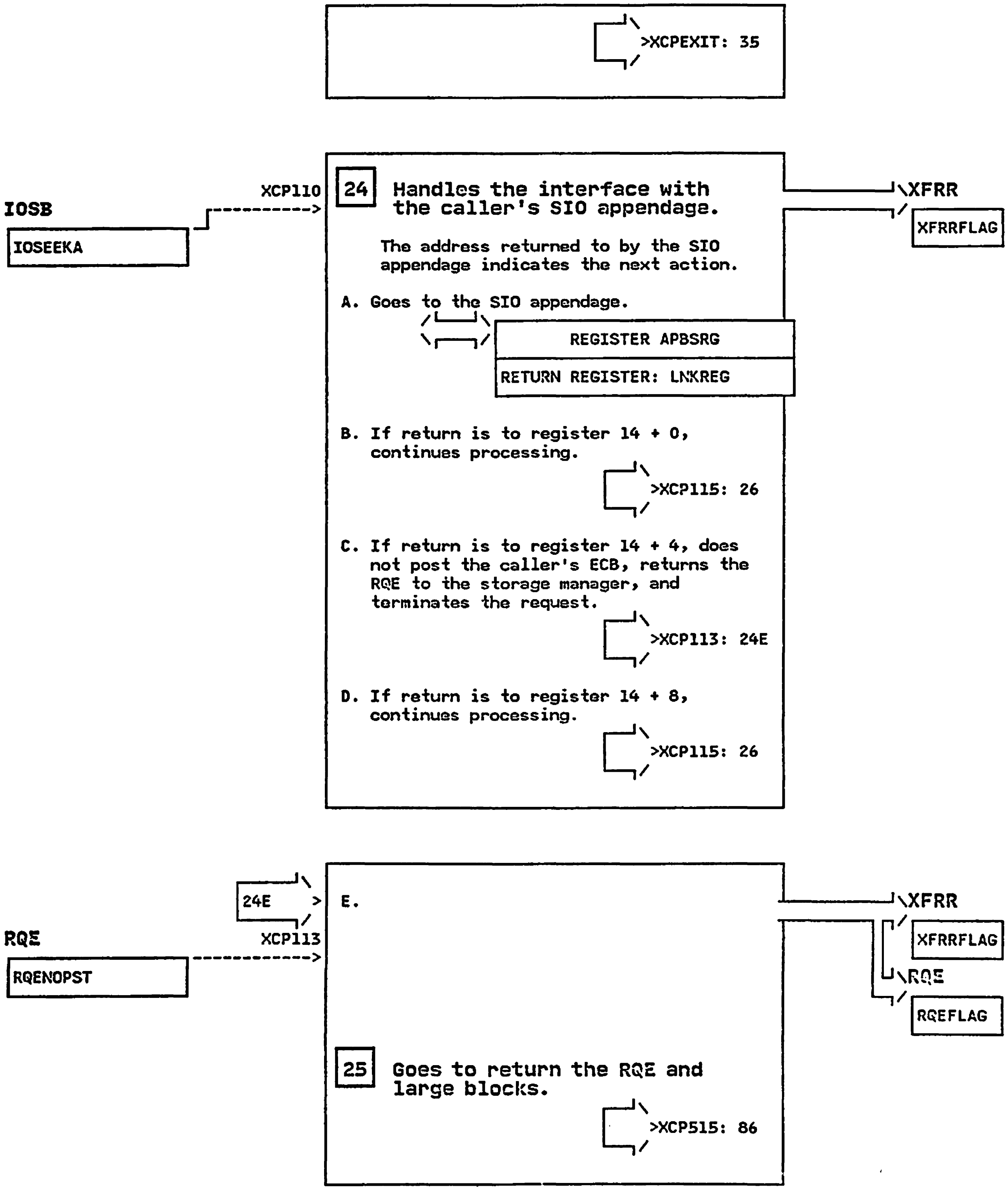
E. If the extent is not within bounds (register 15 = 4), and this is a related request, and the RQE is at top of the related request queue (RRQ), goes to interface with the caller's abnormal-end appendage at XCP203AA.

>XCP203AA: 52

F. If the extent is not within bounds (register 15 = 4), and this is a related request, and the RQE is not at the top of the related request queue (RRQ), sets a flag (RQEEOEE) in this RQE and exits from IECVEXCP processing.

XFRR
XFRRFLAG

RQE
RQETYPE



24 Handles the interface with the caller's SIO appendage.

The address returned to by the SIO appendage indicates the next action.

A. Goes to the SIO appendage.

B. If return is to register 14 + 0, continues processing.

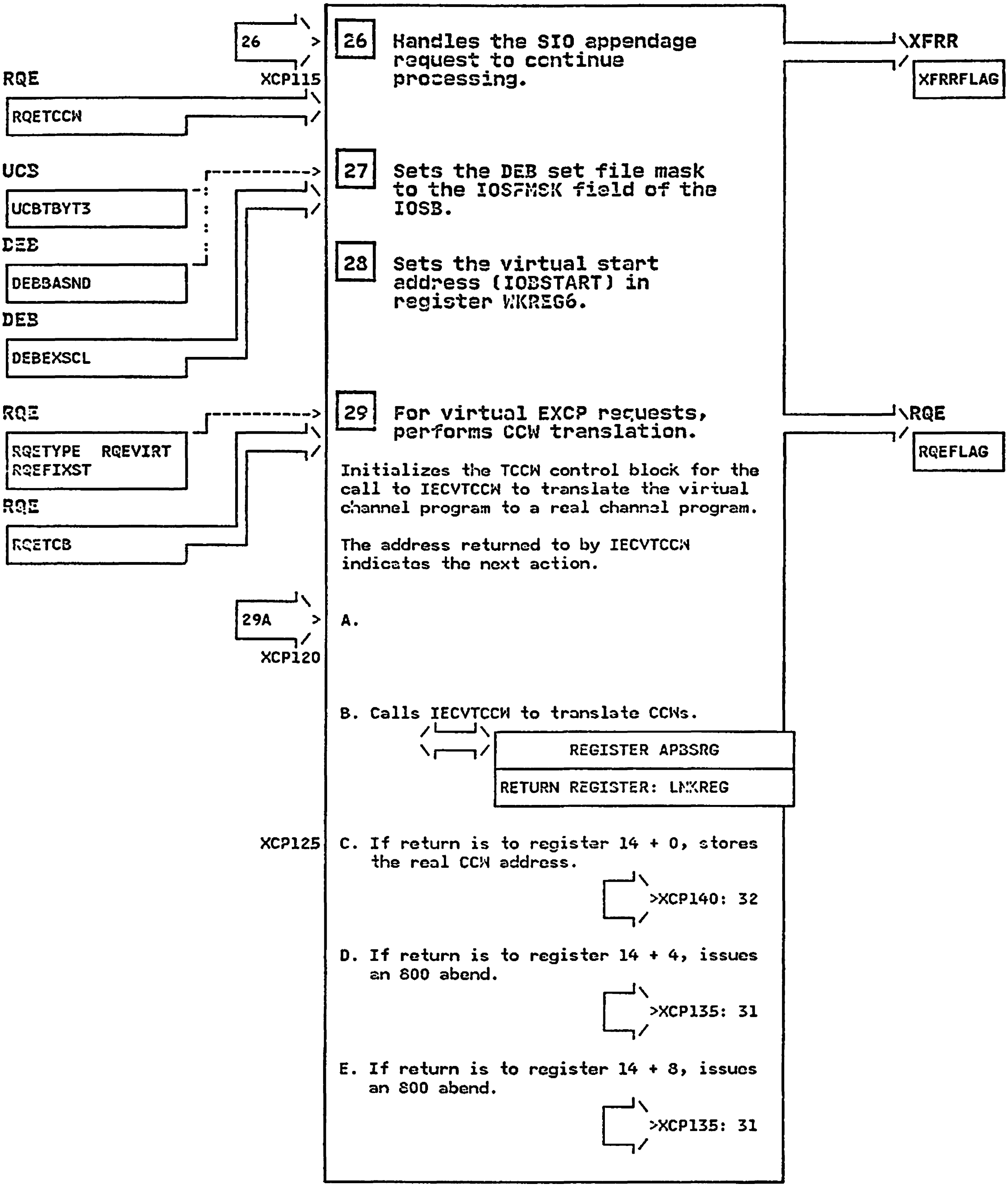
C. If return is to register 14 + 4, does not post the caller's ECB, returns the RQE to the storage manager, and terminates the request.

D. If return is to register 14 + 8, continues processing.

E.

25 Goes to return the RQE and large blocks.

IECVEXCP - EXCP Processor for SVC 0 (EXCP) and SVC114 (EXCPVR)   STEP 26



26 Handles the SIO appendage request to continue processing.

27 Sets the DEB set file mask to the IOSFMSK field of the IOSB.

28 Sets the virtual start address (IOBSTART) in register WKREG6.

29 For virtual EXCP requests, performs CCW translation.

Initializes the TCCW control block for the call to IECVTCCW to translate the virtual channel program to a real channel program.

The address returned to by IECVTCCW indicates the next action.

A.

B. Calls IECVTCCW to translate CCWs.

REGISTER APBSRG

RETURN REGISTER: LNKREG

C. If return is to register 14 + 0, stores the real CCW address.

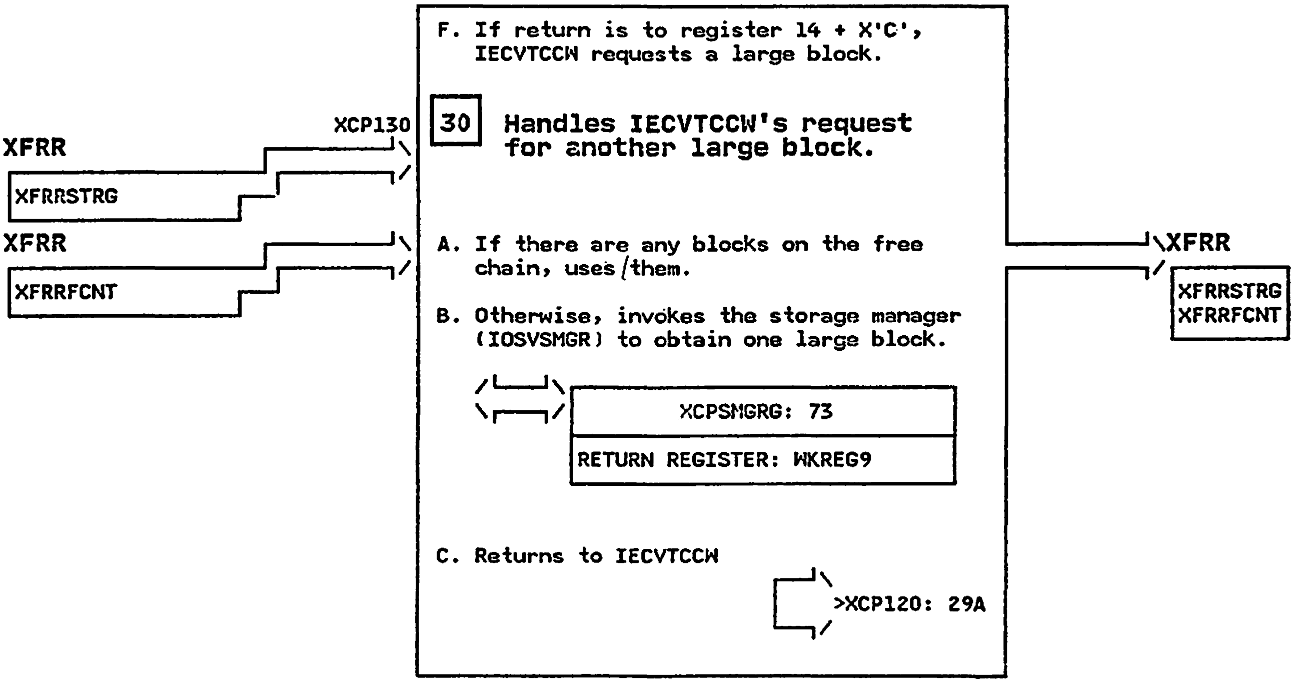
>XCP140: 32

D. If return is to register 14 + 4, issues an 800 abend.

>XCP135: 31

E. If return is to register 14 + 8, issues an 800 abend.

>XCP135: 31



F. If return is to register 14 + X'C', IECVTCCW requests a large block.

30 Handles IECVTCCW's request for another large block.

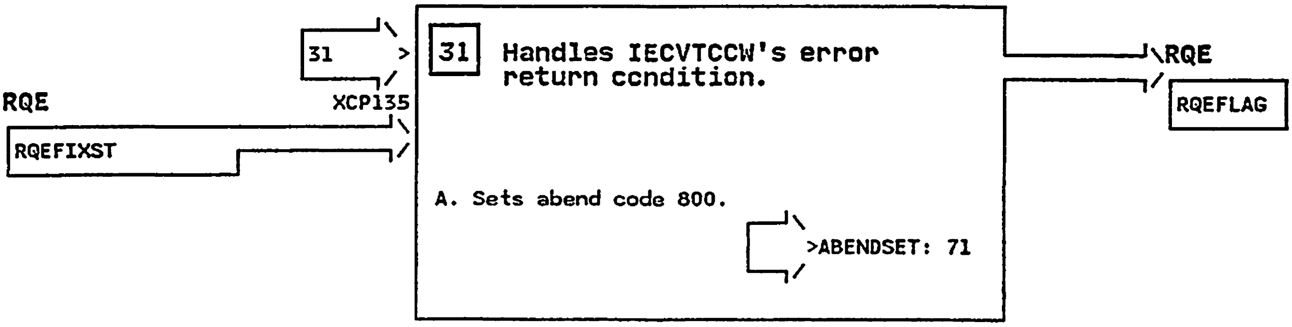
A. If there are any blocks on the free chain, uses/them.

B. Otherwise, invokes the storage manager (IOSVSMGR) to obtain one large block.

XCPSMGRG: 73
RETURN REGISTER: WKREG9

C. Returns to IECVTCCW

>XCP120: 29A



31 Handles IECVTCCW's error return condition.

A. Sets abend code 800.

>ABENDSET: 71

32

RQE
RQETYPE RQE1TO1

XCP140

32 With a successful IECVTCCW translation of the virtual channel program, updates the IOSB virtual and real start channel program addresses.

IOSB
IOSRST
IOSVST

RQE
RQEFLAG3 RQEXCPS

IOSB

TCCW

RQE
RQETCCW

XCP150

33 If the device provides a channel program scan exit (CPS), invokes the CPS exit with the STARTIO function.

A. Invokes the CPS exit.

REGISTER APBSRG

RETURN REGISTER: LNKREG

RQE
RQETYPE RQESTBL
RQESRBS

RRQ
RRQFIRST

XCP152

34 Determines if the request can be sent to IOS for initiation.

Sets a flag in the RQE (RQESTBL) to indicate that the request is startable.

If this is a type 2 or 3 related request and this RQE is not the first RQE on the RRQ, bypasses starting the request and exits from IECVEXCP processing (goes to XCPEXIT).

Otherwise, performs the following:
. Saves all the IECVEXCP registers in the local lock save area.
. Sets a flag in the RQE (RQESRBS) to indicate that the call to IOS is in progress for initiating the request.
. Issues the STARTIO macro passing the IOSB as the parameter.
. Upon return, restores all the IECVEXCP registers.
. Sets a flag in the RQE (RQEINIOS) to indicate that the request is in IOS.

RQE
RQEFLAG
RQEFLAG3

35

XCPEXIT

35 Exits from EXCP processing.

XFRR
XFRRCRQE XFRRFLAG
XFRRBKE XFRRWORK

RQE
RQETCB

A. If exit processing (XCPEXIT) was entered from back-end processing, determines, for type 2 and 3 related requests, whether further translation of requests is required.

- If the DCB is marked in permanent error, further translation is not required.

XFRR
XFRRCRQE
XFRRPRQE

B. Otherwise, goes to determine if there are RQEs to be processed.

>XCPEXITK: 42

XFRR

XFRRSTRG

XCPEXITB

36 Returns all free large blocks to the storage manager.

If the FRR large block chain pointer is not zero, calls the storage manager to return all large blocks chained on the large block free chain.

XCPSMGR: 73A

RETURN REGISTER: WKREG9

XFRR

XFRRFLAG XFRRBKE
XFRRWORK

XCPEXITC

37 If IOS post status called back-end processing, goes to XCPEXITD to free the local lock.

>XCPEXITD: 39

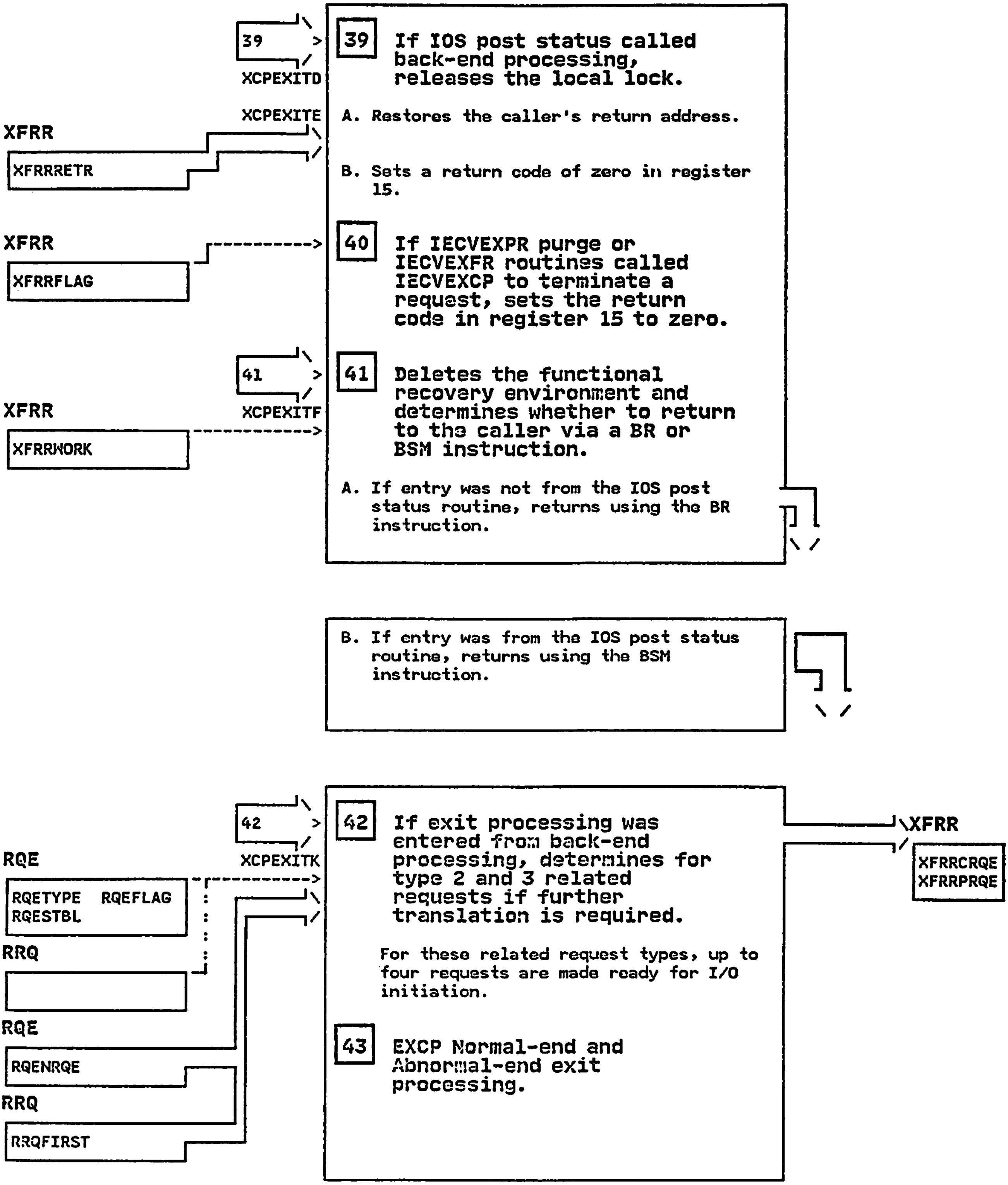
XFRR

XFRRRETR

38 If channel-end or abnormal-end exit processing called back-end processing, does the following:

. Restores post status registers
. Sets a return code of 16 in register 15
. Restores the post status return address
. Deletes the FRR
. Issues the BSM instruction to return to IOS post status

>XCPEXITF: 41



39 If IOS post status called back-end processing, releases the local lock.

A. Restores the caller's return address.

B. Sets a return code of zero in register 15.

40 If IECVEXPR purge or IECVEXFR routines called IECVEXCP to terminate a request, sets the return code in register 15 to zero.

41 Deletes the functional recovery environment and determines whether to return to the caller via a BR or BSM instruction.

A. If entry was not from the IOS post status routine, returns using the BR instruction.

B. If entry was from the IOS post status routine, returns using the BSM instruction.

42 If exit processing was entered from back-end processing, determines for type 2 and 3 related requests if further translation is required.

For these related request types, up to four requests are made ready for I/O initiation.

43 EXCP Normal-end and Abnormal-end exit processing.

XFRR

XFRRABE

RQE

RQEFLAG

IOSB

IOSFLA

44 If exit processing was entered from the IECVEXCP abnormal- end exit entry point (XCPABE) via IOS post status, does the following.

Saves post status registers in the local lock save area.

Indicates abnormal-end entry.

Establishes the RQE and DEB pointers from the IOSB.

If the RQE indicates that purge is active (RQEPURGE), bypasses the abnormal-end appendage and continues processing as if a return code of 0 was returned from the appendage.

A. Goes to the common IECVEXCP exit processing.

>XCP200: 46

45 If exit processing was entered from the IECVEXCP normal-end exit entry point (XCPCHE) via IOS post status, does the following.

Saves post status registers in the local lock save area.

Indicates channel-end entry.

Establishes the DEB pointer from the IOSB.

46 >
XCP200

RQE

RQEPRT

46 Performs common IECVEXCP exit processing.

Establishes a functional recovery routine.

Indicates a return to post status via a BSM instruction.

Establishes the RQE pointer from the IOSB.

Resets the request-in-IOS flag (RQEINIOS).

Establishes pointers to the IOB, DCB and UCB from the RQE.

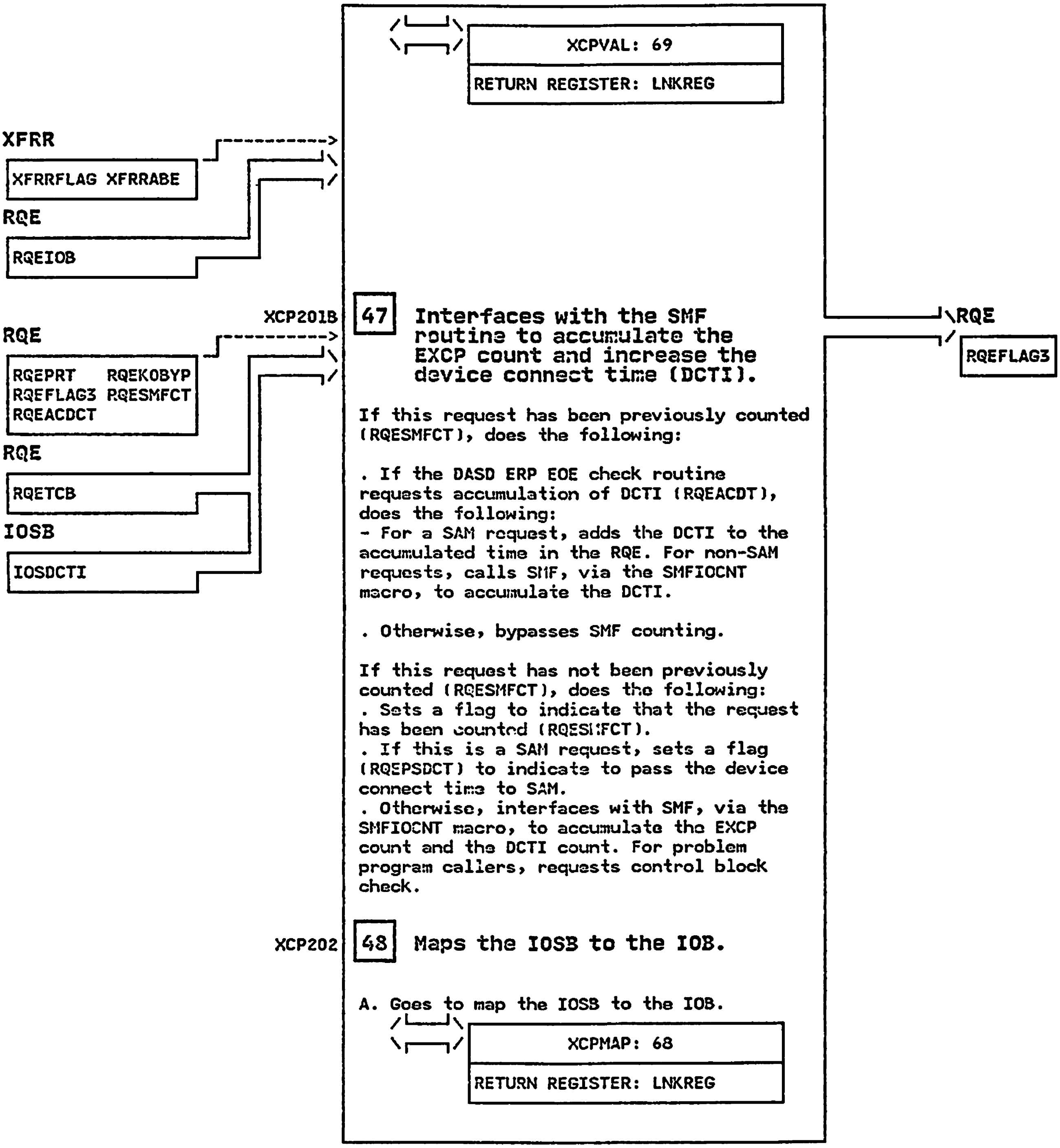
For problem program callers, validates the caller's blocks.

XFRR

XFRRCRQE
XFRRFLAG
XFRRWORK
XFRRRETR

RQE

RQEFLAG3



47 Interfaces with the SMF routine to accumulate the EXCP count and increase the device connect time (DCTI).

If this request has been previously counted (RQESMFCT), does the following:

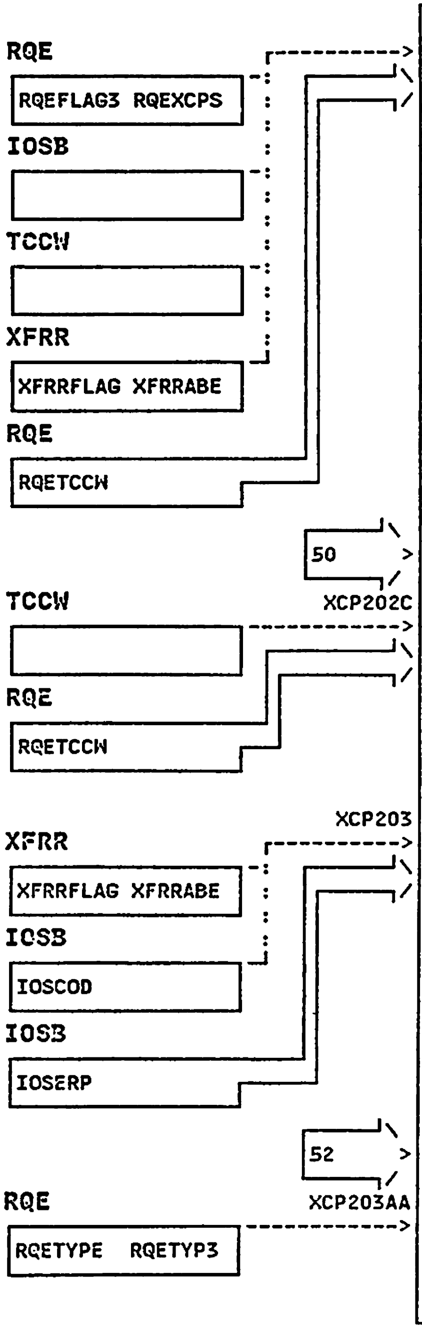
. If the DASD ERP EOE check routine requests accumulation of DCTI (RQEACDT), does the following:
- For a SAM request, adds the DCTI to the accumulated time in the RQE. For non-SAM requests, calls SMF, via the SMFIOCNT macro, to accumulate the DCTI.

. Otherwise, bypasses SMF counting.

If this request has not been previously counted (RQESMFCT), does the following:
. Sets a flag to indicate that the request has been counted (RQESMFCT).
. If this is a SAM request, sets a flag (RQEPSDCT) to indicate to pass the device connect time to SAM.
. Otherwise, interfaces with SMF, via the SMFIOCNT macro, to accumulate the EXCP count and the DCTI count. For problem program callers, requests control block check.

48 Maps the IOSB to the IOB.

A. Goes to map the IOSB to the IOB.



49 If the device provides a channel program scan exit (CPS), performs the following:

If the entry is to the normal-end appendage, invokes the CPS exit with the normal-end function.

Otherwise for the abnormal-end appendage, invokes the CPS exit with the I/O function.

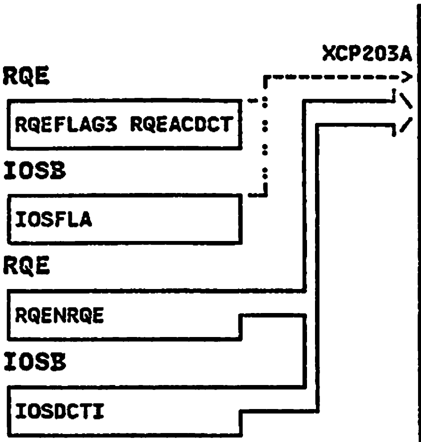
REGISTER APBSRG

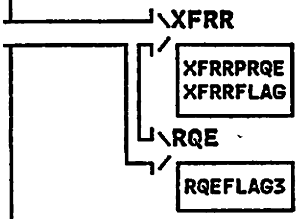
RETURN REGISTER: LNKREG

50 This is the entry point from the front-end end-of-extent appendage processing when the extent is not within bounds and this is not a related request.

51 If IECVEXCP was entered at the abnormal-end exit entry point (XCPABE), if the IOSB completion code indicates permanent error, and if this is a type 2 or type 3 related request, purges all related requests before entering the abnormal-end appendage.

52 This is the entry point from the front-end cnd-of-extent appendage processing when the extent is not within bounds and this is a related request.





53 **Prepares to enter the caller's channel-end or abnormal-end appendage.**

If the device connect time (DCTI) is to be passed to the SAM appendage, sets the DCTI saved in the RQE in register 9. Otherwise, sets register 9 to zero.

Issues a TM instruction on the IOSEX bit in order to set the condition code for the appendage.

A. Invokes the appendage.

REGISTER APBSRG

RETURN REGISTER: LNKREG

54 **Handles the return vectors from the appendage.**

The address the appendage returns to indicates the next action.

A. If return is to register 14 + 0, continues processing.

>XCP220: 59

B. If return is to register 14+4, does not post the caller's ECB; terminates the request.

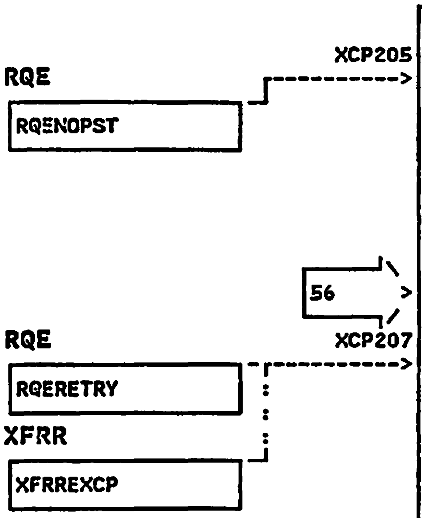
>XCP215: 58

C. If return is to register 14+8, retries the EXCP request. (Goes to the IECVEXCP termination routine to clean up, then goes to front-end processing to retry.)

>XCP207: 56

D. If return is to register 14 + X'C', does not post the caller's ECB and does not return the RQE. Terminates the request.

>XCP208: 57



55 **If return is to register 14+X'10', retries from the top of the RRQ.**

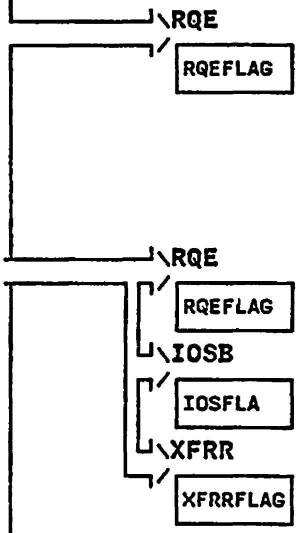
The setting of the RQENOPST flag and the RQERETRY flag indicates to the retry routine in the EXCP termination routine that the retry is to top of the RRQ.

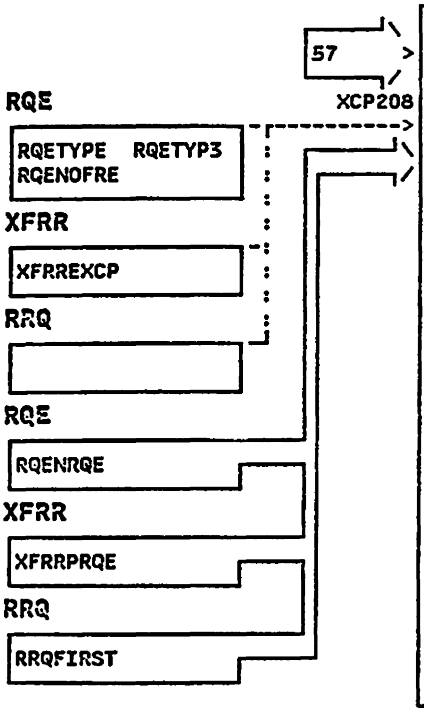
56 **If return is to register 14 + 8, re-issues (retries) the caller's request.**

The setting of the RQERETRY flag indicates to the retry routine in the EXCP termination routine that this request is to retried.

A. For returns to register 14+8 and register 14+X'10', goes to prepare to exit from the normal- end/abnormal-end exit processing.

>XCP250: 66

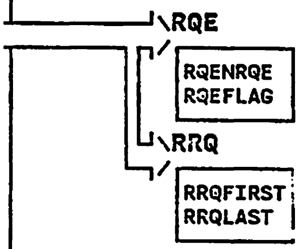




57 **Indicates that the RQE is not to be freed.**

. Sets the no-free-RQE flag (RQENOFRE) in the RQE.

. For a related request RQE, removes the RQE from the chain.



58 XCP215

RQE
RQENOPST

58 Indicates that the caller's ECB is not to be posted.

. Sets the no-post-ECB flag (RQENOPST) in the RQE.

. Resets the exception flag (IOSEX) in the IOSB.

A. For returns to register 14+4 and register 14+X'0C', goes to prepare to exit from normal-end/ abnormal-end exit processing.

>XCP250: 66

RQE
RQEFLAG
IOSB
IOSFLA
XFRR
XFRRFLAG

59 XCP220

XFRR
XFRRFLAG
XFRR
XFRRBKE

59 Continues normal processing.

XFRR
XFRRFLAG

60 If entry was not from the IOS post status routine, goes to prepare to enter the IECVEXCP termination routine.

>XCP251: 66A

61 Resets the IOSB exception and error flags.

IOSB
IOSFLA

62 If the IOB exception flag is off (implying successful completion), goes to prepare to enter IECVEXCP termination.

>XCP250: 66

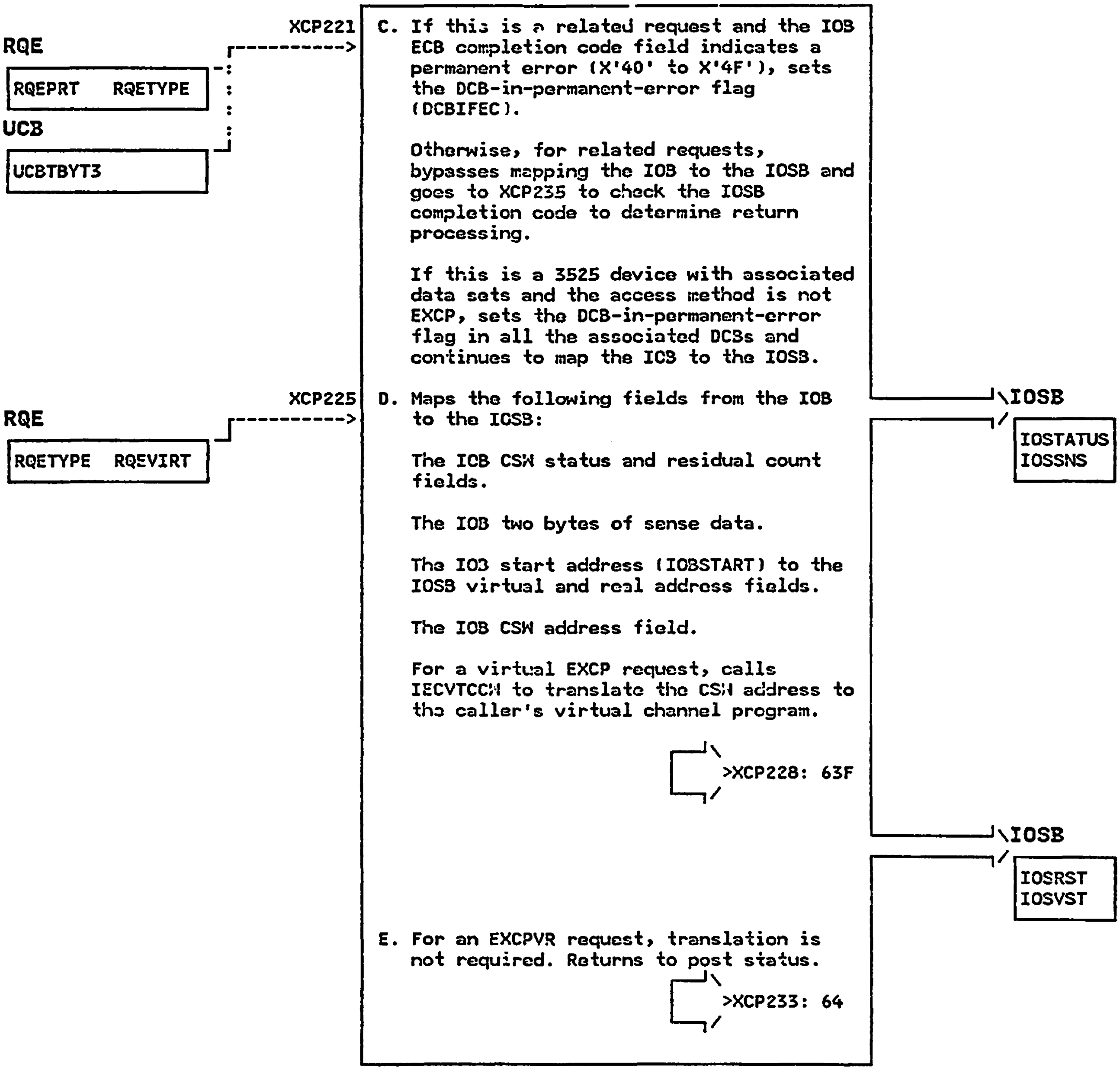
63 Otherwise, with the IOB exception bit set on, does the following:

A. Maps the IOB flags to the IOSB.

UCB
UCBTBYT3

B. For DASD, moves the IOB seek address to the IOSB.

IOSB
IOSEEKA



C. If this is a related request and the IOB ECB completion code field indicates a permanent error (X'40' to X'4F'), sets the DCB-in-permanent-error flag (DCBIFEC).

Otherwise, for related requests, bypasses mapping the IOB to the IOSB and goes to XCP235 to check the IOSB completion code to determine return processing.

If this is a 3525 device with associated data sets and the access method is not EXCP, sets the DCB-in-permanent-error flag in all the associated DCBs and continues to map the IOB to the IOSB.

D. Maps the following fields from the IOB to the IOSB:

The IOB CSW status and residual count fields.

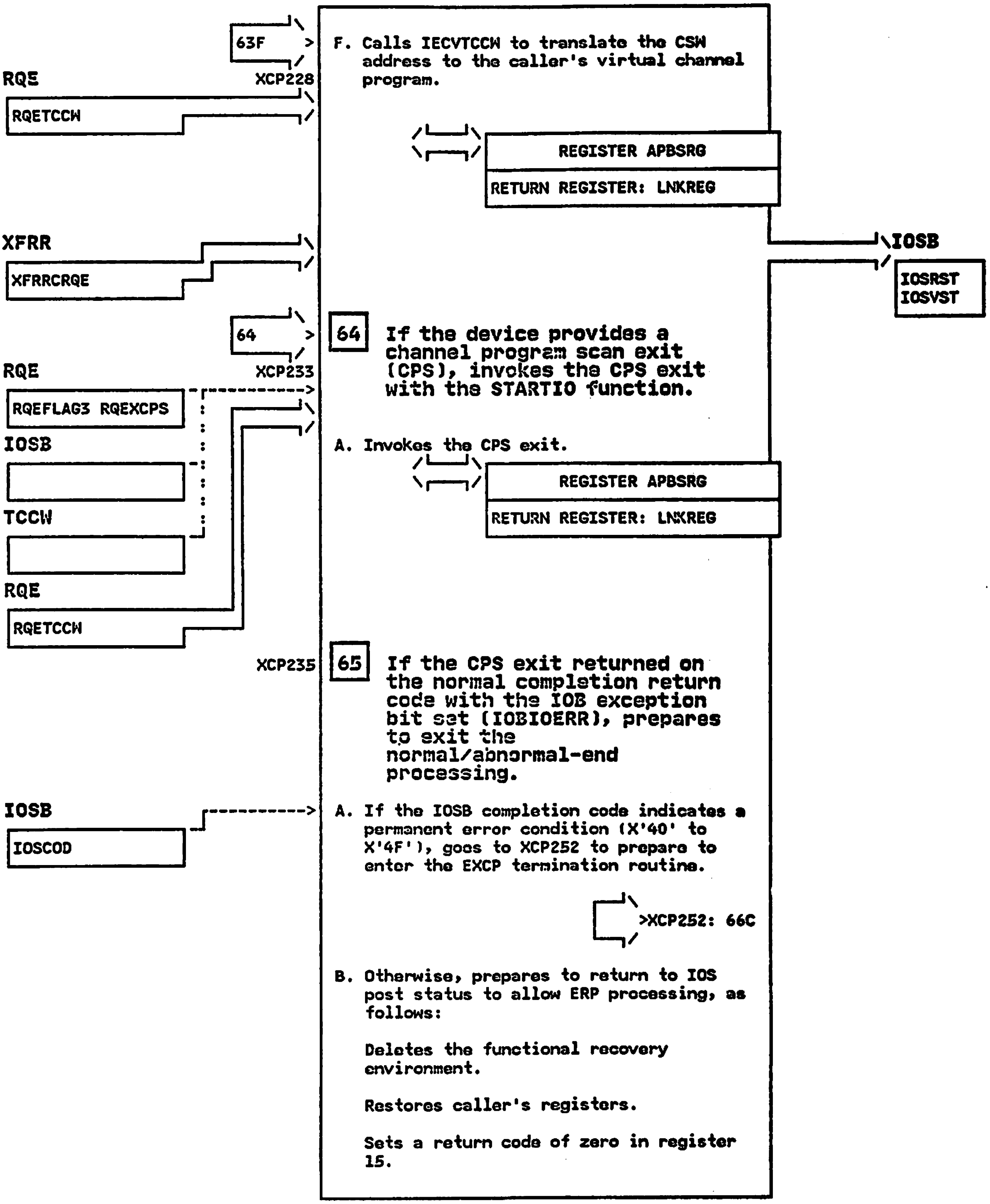
The IOB two bytes of sense data.

The IOB start address (IOBSTART) to the IOSB virtual and real address fields.

The IOB CSW address field.

For a virtual EXCP request, calls IECVTCCW to translate the CSW address to the caller's virtual channel program.

E. For an EXCPVR request, translation is not required. Returns to post status.



F. Calls IECVTCCW to translate the CSW address to the caller's virtual channel program.

64 If the device provides a channel program scan exit (CPS), invokes the CPS exit with the STARTIO function.

A. Invokes the CPS exit.

65 If the CPS exit returned on the normal completion return code with the IOB exception bit set (IOBIOERR), prepares to exit the normal/abnormal-end processing.

A. If the IOSB completion code indicates a permanent error condition (X'40' to X'4F'), goes to XCP252 to prepare to enter the EXCP termination routine.

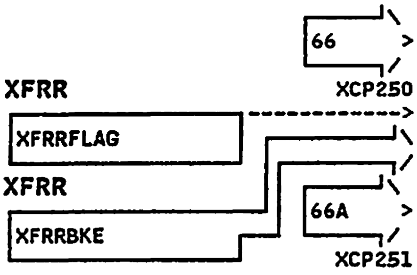
B. Otherwise, prepares to return to IOS post status to allow ERP processing, as follows:

Deletes the functional recovery environment.

Restores caller's registers.

Sets a return code of zero in register 15.

C. Returns to post status (IECVPST).



66 Prepares to exit from normal-end or abnormal-end exit processing, as follows.

A.

B. If entry was from IECVEXCP front-end or back-end processing, enters IECVEXCP back-end processing at XCP508 to terminate the request.

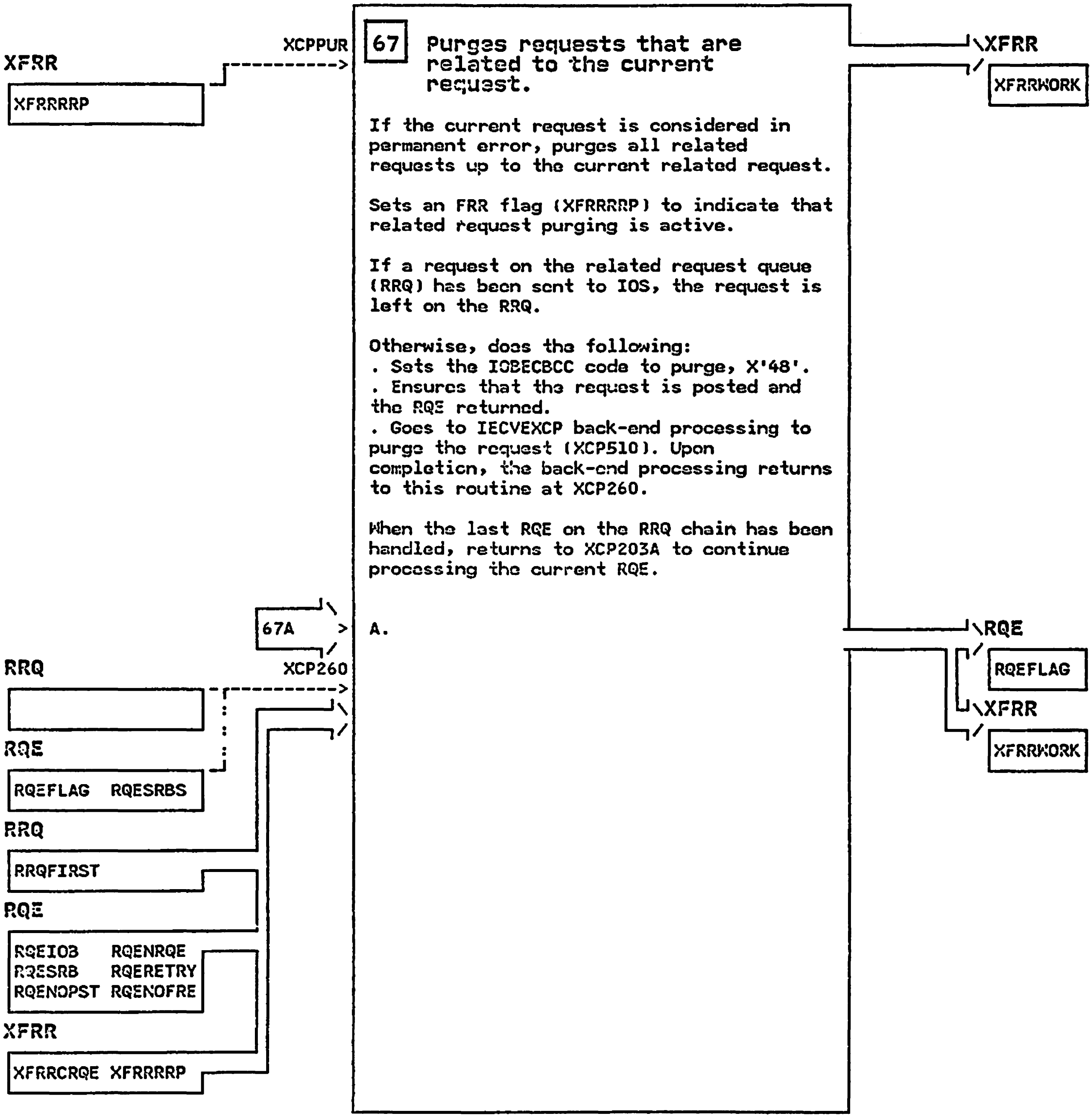
>XCP508: 84

66C
XCP252

C.

D. Otherwise, enters IECVEXCP back-end processing at XCPTERM to terminate the request.

>XCPTERM: 80B



67 Purges requests that are related to the current request.

If the current request is considered in permanent error, purges all related requests up to the current related request.

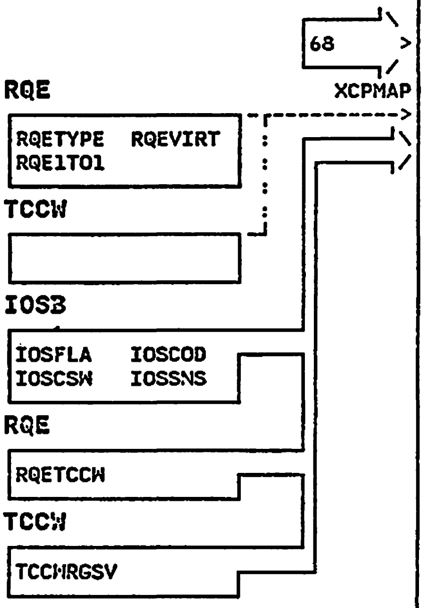
Sets an FRR flag (XFRRRRP) to indicate that related request purging is active.

If a request on the related request queue (RRQ) has been sent to IOS, the request is left on the RRQ.

Otherwise, does the following:
. Sets the IOBECBCC code to purge, X'48'.
. Ensures that the request is posted and the RQE returned.
. Goes to IECVEXCP back-end processing to purge the request (XCP510). Upon completion, the back-end processing returns to this routine at XCP260.

When the last RQE on the RRQ chain has been handled, returns to XCP203A to continue processing the current RQE.

A.

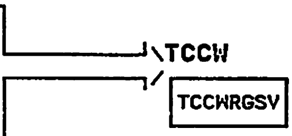


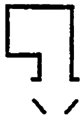
68 Maps the contents of fields in the IOSB to the IOB.

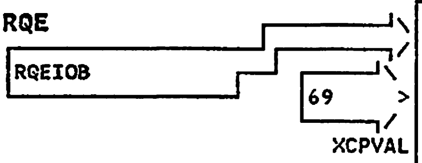
This mapping is done on completion of an I/O request. The I/O request results need to be mapped to the IOB before IECVEXCP enters any of the appendages or DIE routines.

Maps the following fields:
. The IOSFLA field to the IOBFLAG1
. The IOSCSW to the IOBCSW
. The IOSSNS (two bytes of sense data) to IOBSENSO
. The IOSCC (condition code) to IOBCC.
. If the IOSB completion code is X'74' (unknown to caller), leaves IOBECBCC set to X'7F'
. If the IOSB completion code is X'51' (new error code), sets IOBECBCC to X'41' (permanent error).
. Otherwise, the IOSB completion code (IOSCOD) is moved to IOBECBCC
. If the DIE (XCPDIE) routine was in control and the request was not an EXCP V=R request, translates the real CCW address to virtual and stores it in the IOB CSW address.
. If this is a virtual EXCP request, goes to IECVTCCW to translate the virtual address of the real channel program to the virtual address of the caller's virtual program.

A. Returns to caller.







69 Validates a problem program caller's control blocks.

If a protection check occurs while accessing the caller's control blocks, the functional recovery routine IECVEXFR will issue abend 200 to abend the task.

Issues a MODESET macro to get in the caller's key.

If the caller is not SAM, then accesses the beginning to the end of the normal IOB.

Accesses the DCB.

Accesses the ECB.

A. Returns to the caller.

70 > XCPPOST

RQE

RQEPRT ------------>

70 Interfaces with the system post routine.

If a protection check occurs while accessing the caller's ECB, the functional recovery routine IECVEXFR will issue abend 200 to abend the task.

A. Issues a MODESET macro to get in the caller's Key to access the ECB.

XCPPOST1 B. If the wait bit is on, then calls the post routine to post the ECB with the IOBECBCC code.

>XCPPOST5: 70D

C. Otherwise, issues a CS instruction in the caller's Key to set the IOBECBCC code to the ECB.

70D > XCPPOST5 D.

E. Invokes the post routine IEAOPT02.

REGISTER APBSRG
RETURN REGISTER: LNKREG

F. Returns to the caller.

71
ABENDSET

71 Sets register 1 to abend code 800 for page fix and unfix errors.

72
ABEND000

72 Issues the ABEND macro to abend the task.

The abend code to be issued has been set in register 1.

The functional recovery routine IECVEXFR receives control as a result of the SVC 13 and abends the task.

A. Issues the ABEND macro.

ABEND
(REG1), DUMP, , SYSTEM

73
XCPSMGRG

PSA

PSAAOLD

73 Interfaces with the IOS storage manager to get and free large blocks.

For IECVEXCP requests for large blocks, the storage manager does not establish a functional recovery routine. Instead, IECVEXCP establishes a storage manager FRR parameter area in the local lock area and places the address of this parameter area in register 4 (storage manager FRR parameter register).

If the storage manager is in control when RTM gets control as a result of an unexpected error, the functional recovery routine IECVEXFR must call the storage manager's functional recovery routine.

On a get request, register 6 contains the ASID of the request and register 11 contains the count of large blocks requested. On return from the storage manager, register 11 contains the pointer to the first large block.

On a free request, register 11 contains the pointer to the first block to be freed. The first word of each block contains a chain pointer to the next block to be freed. The last block chain pointer must be zero.

73A
XCPSMGR

A.

B. Invokes the storage manager (IOSVSMGR).

BASSM

LNKREG, APBSRG

C. Returns to the caller.

XFRR

XFRRFCNT

UCB

UCBTBYT3

RQE

RQEPRT

DEB

DEBBASIC

RQE

RQETCB

XCPRRQ00

74 Performs IECVEXCP front-end related request handling.

This routine is called from the IECVEXCP front-end processing to handle a caller's related request. If necessary, this routine obtains a related request queue element (RQE) and queues this RQE to the RRQ.

A. Determines if a related request queue (RRQ) element block is available.

If the DEBRRQ field is zero and this is not a 3525 device, obtains a related request queue (RRQ) and anchors it off the DEBRRQ field.

If the DEBRRQ field is zero and this is a 3525 device, does the following:
. If the 3525 is being handled by an EXCP access method, does not check for associated data sets; obtains a related request queue (RRQ) element.
. If there are no associated data sets, obtains a related request queue element.
. If there is an associated data set with an RRQ, the RRQ is used. The associated DCB and DEB are validity checked.
. If there is an associated data set but without an RRQ, obtains a related request queue element.

XCPRRQ10

B. Scans the 3525 DCBs for an available DEB to determine if an RRQ element is available.

For problem program callers, all new DC3s are validity checked. If a protection check occurs, the EXCP functional recovery routine will abend the task with abend code 300.

RQE

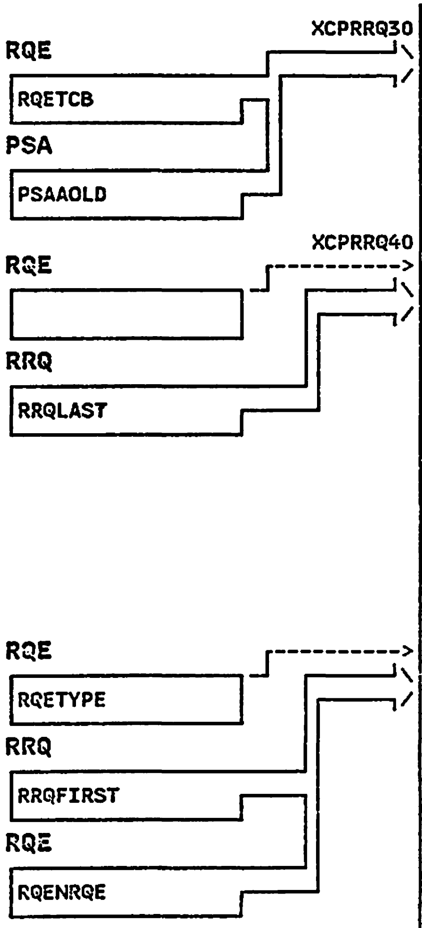
RQETYPE

IOSB

IOSFLA

DEB

DEBECBB



C. If an RRQ is not available, issues a GETMAIN to obtain an RRQ from subpool 254 and anchor it off the DEBRRQ.

D. Determines where in the chain to add the RQE block.

If the RRQ element is new, then the first and last entry contain this RQE block address.

If the RRQ element is null, the last entry address field points to the first entry address. For this case, the first and last point to this RQE address.

If RQE blocks exists on the RRQ, then this RQE block is stored at the end of the RRQ chain and the last entry address points to this RQE block.

E. Determines if related request processing is required.

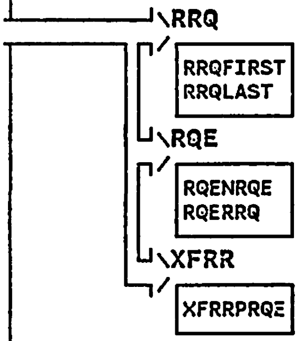
If the DCB indicates permanent error (DCBIFEC flag), then the IOBECBCC field is set to the purge condition and this routine passes control to the IECVEXCP abnormal-end exit processing (XCPABE) to interface with the caller's abnormal-end appendage.

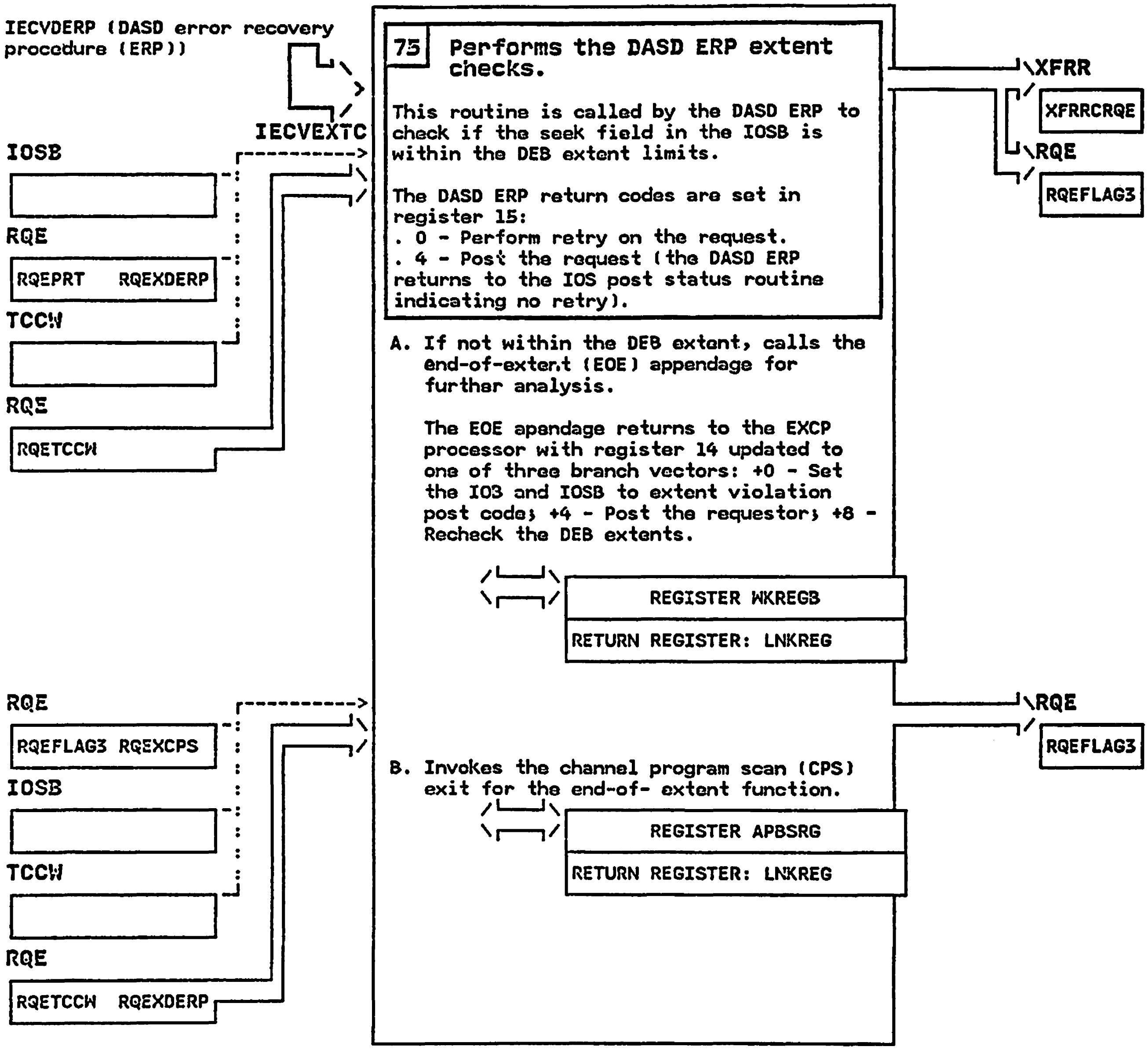
If this RQE is the first RQE on the RRQ or this RQE is for a type 1 related request, continues processing to prepare the request for I/O initiation.

For related request types 2 and 3, determines if the maximum number of requests has been readied. For type 2 and 3 requests, up to four requests can be made ready for I/O initiation.

F. Processing is complete; the maximum number of RQEs, if any, have been readied.

>XCPEXIT: 35





75 Performs the DASD ERP extent checks.

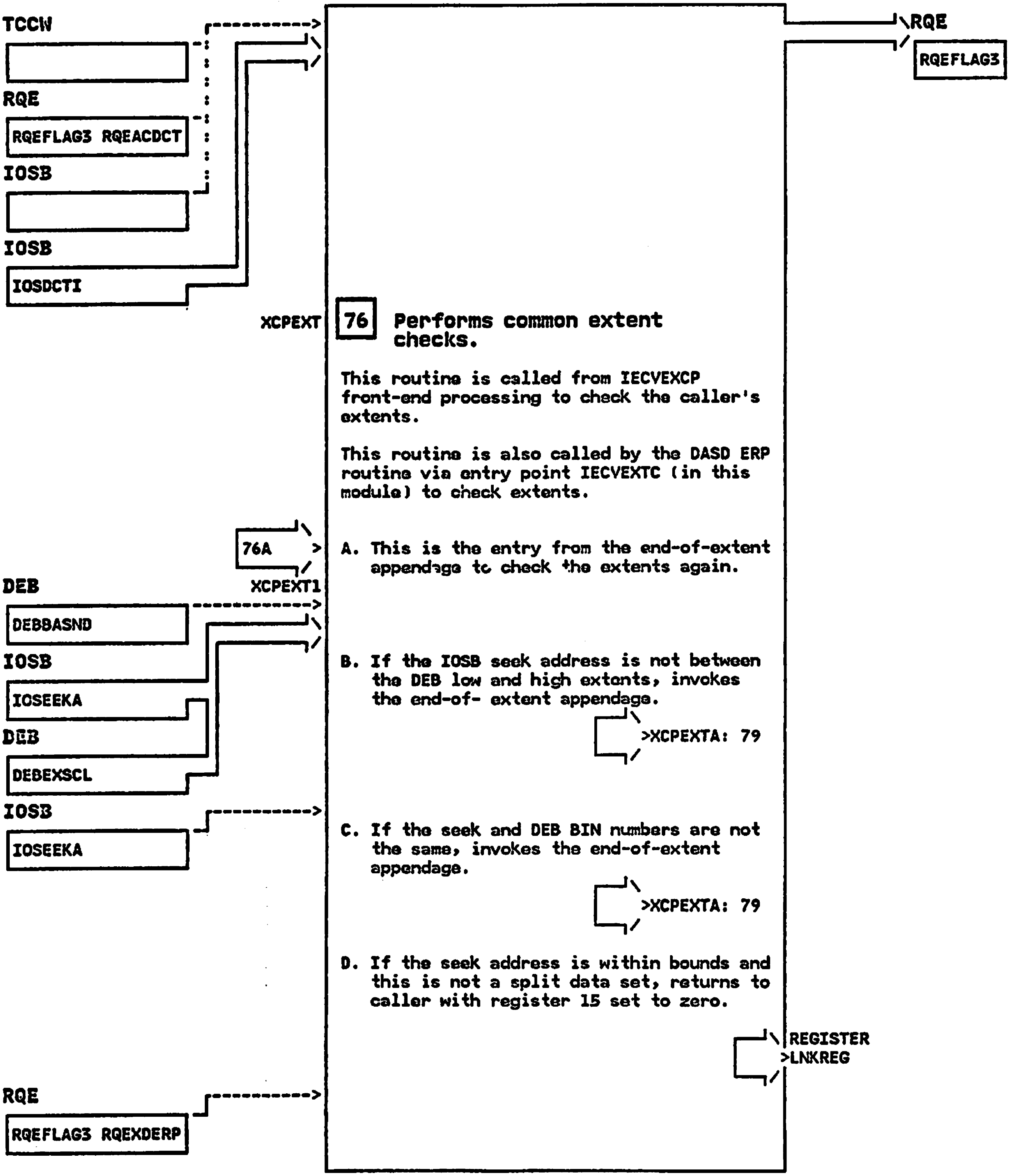
This routine is called by the DASD ERP to check if the seek field in the IOSB is within the DEB extent limits.

The DASD ERP return codes are set in register 15:
. 0 - Perform retry on the request.
. 4 - Post the request (the DASD ERP returns to the IOS post status routine indicating no retry).

A. If not within the DEB extent, calls the end-of-extent (EOE) appendage for further analysis.

The EOE apendage returns to the EXCP processor with register 14 updated to one of three branch vectors: +0 - Set the IOB and IOSB to extent violation post code; +4 - Post the requestor; +8 - Recheck the DEB extents.

B. Invokes the channel program scan (CPS) exit for the end-of- extent function.



76 Performs common extent checks.

This routine is called from IECVEXCP front-end processing to check the caller's extents.

This routine is also called by the DASD ERP routine via entry point IECVEXTC (in this module) to check extents.

A. This is the entry from the end-of-extent appendage to check the extents again.

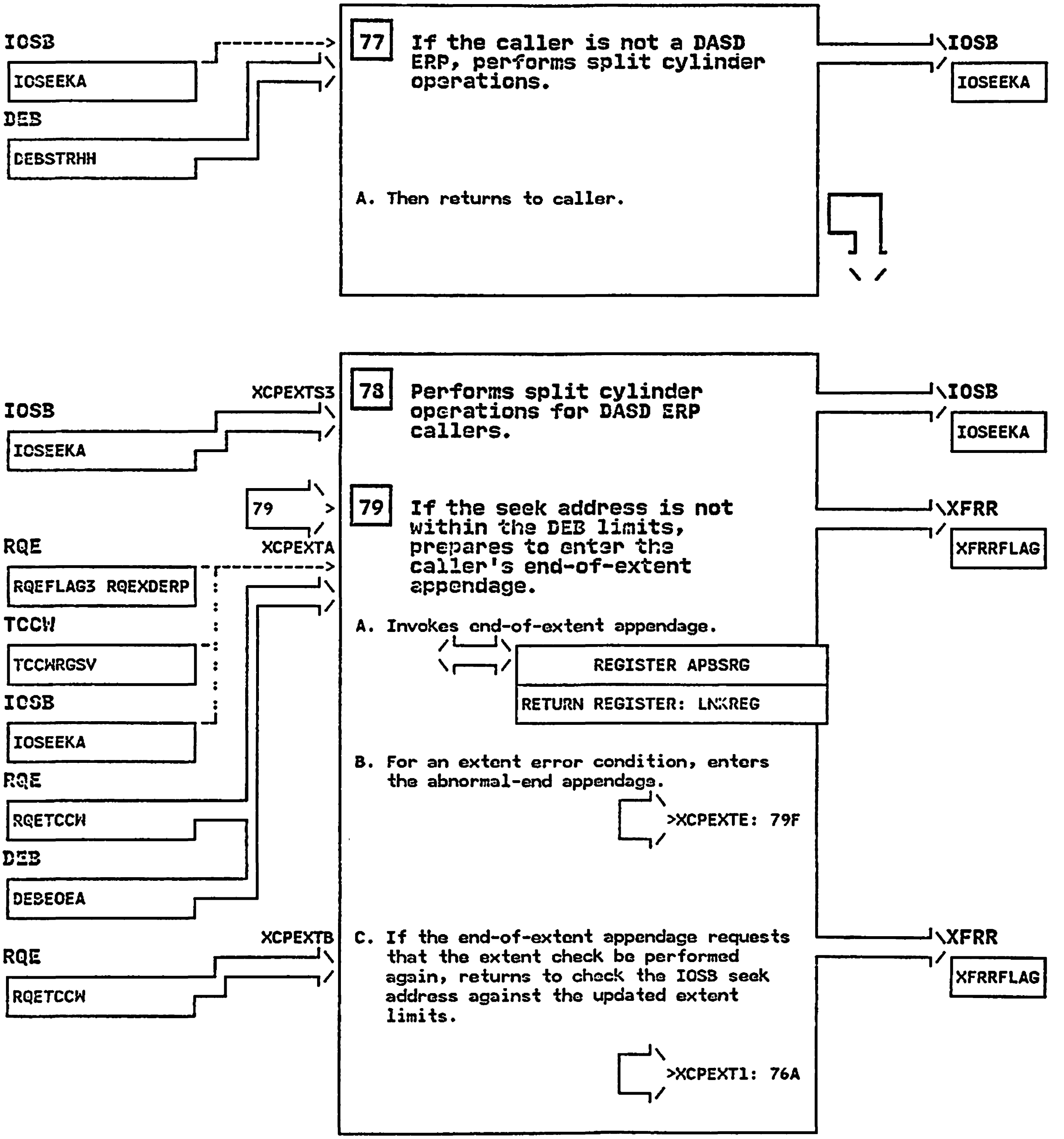
B. If the IOSB seek address is not between the DEB low and high extents, invokes the end-of- extent appendage.

>XCPEXTA: 79

C. If the seek and DEB BIN numbers are not the same, invokes the end-of-extent appendage.

>XCPEXTA: 79

D. If the seek address is within bounds and this is not a split data set, returns to caller with register 15 set to zero.



IOSB
IOSEEKA
DEB
DEBSTRHH

77 If the caller is not a DASD ERP, performs split cylinder operations.

A. Then returns to caller.

IOSB
IOSEEKA

XCPEXTS3
IOSB
IOSEEKA

78 Performs split cylinder operations for DASD ERP callers.

IOSB
IOSEEKA

79
XCPEXTA
RQE
RQEFLAG3 RQEXDERP
TCCW
TCCWRGSV
IOSB
IOSEEKA
RQE
RQETCCW
DEB
DEBEOEA

79 If the seek address is not within the DEB limits, prepares to enter the caller's end-of-extent appendage.

A. Invokes end-of-extent appendage.

REGISTER APBSRG
RETURN REGISTER: LNKREG

B. For an extent error condition, enters the abnormal-end appendage.

>XCPEXTE: 79F

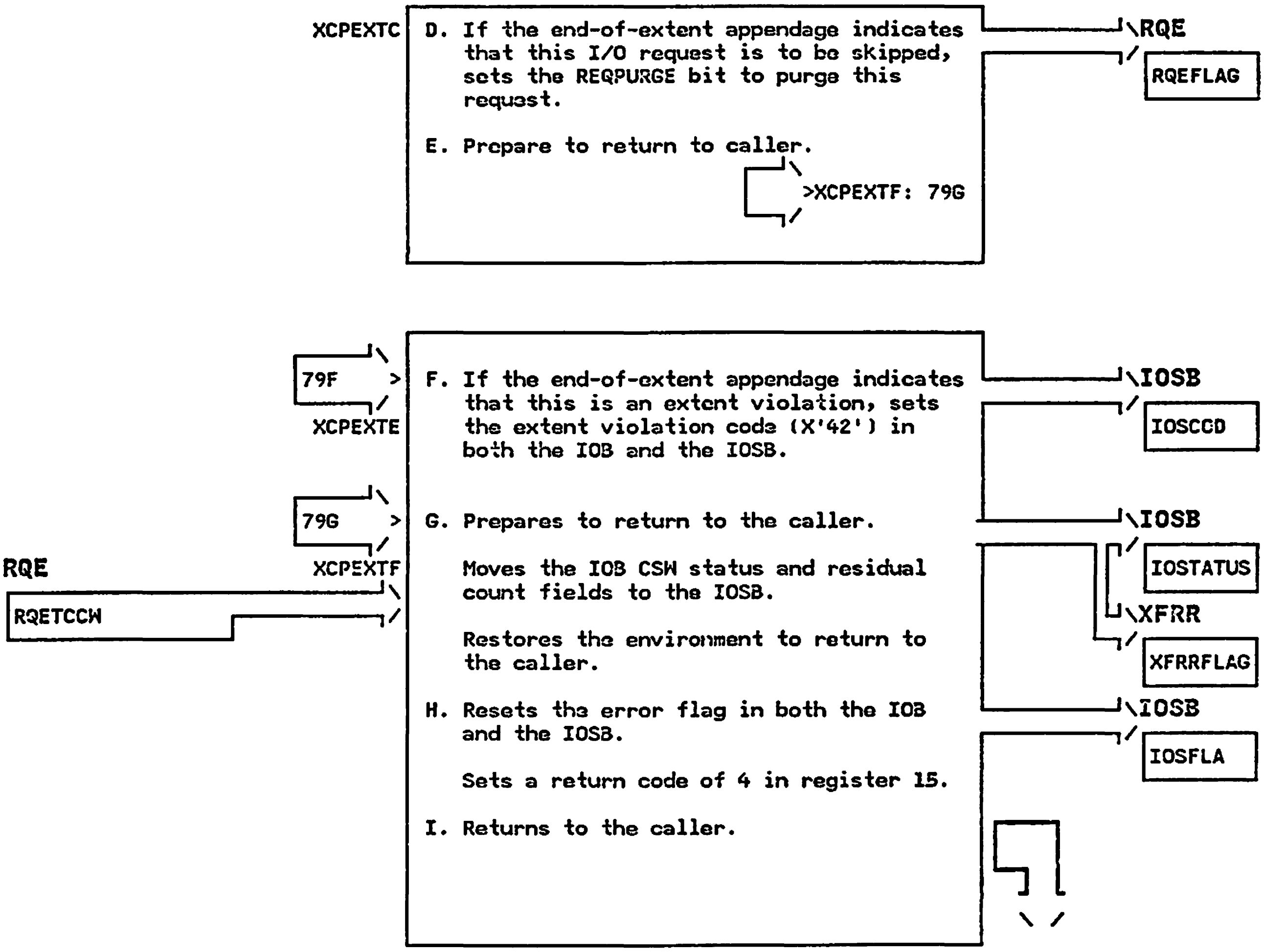
XFRR
XFRRFLAG

XCPEXTB
RQE
RQETCCW

C. If the end-of-extent appendage requests that the extent check be performed again, returns to check the IOSB seek address against the updated extent limits.

>XCPEXT1: 76A

XFRR
XFRRFLAG



D. If the end-of-extent appendage indicates that this I/O request is to be skipped, sets the REQPURGE bit to purge this request.

E. Prepare to return to caller.

F. If the end-of-extent appendage indicates that this is an extent violation, sets the extent violation code (X'42') in both the IOB and the IOSB.

G. Prepares to return to the caller.

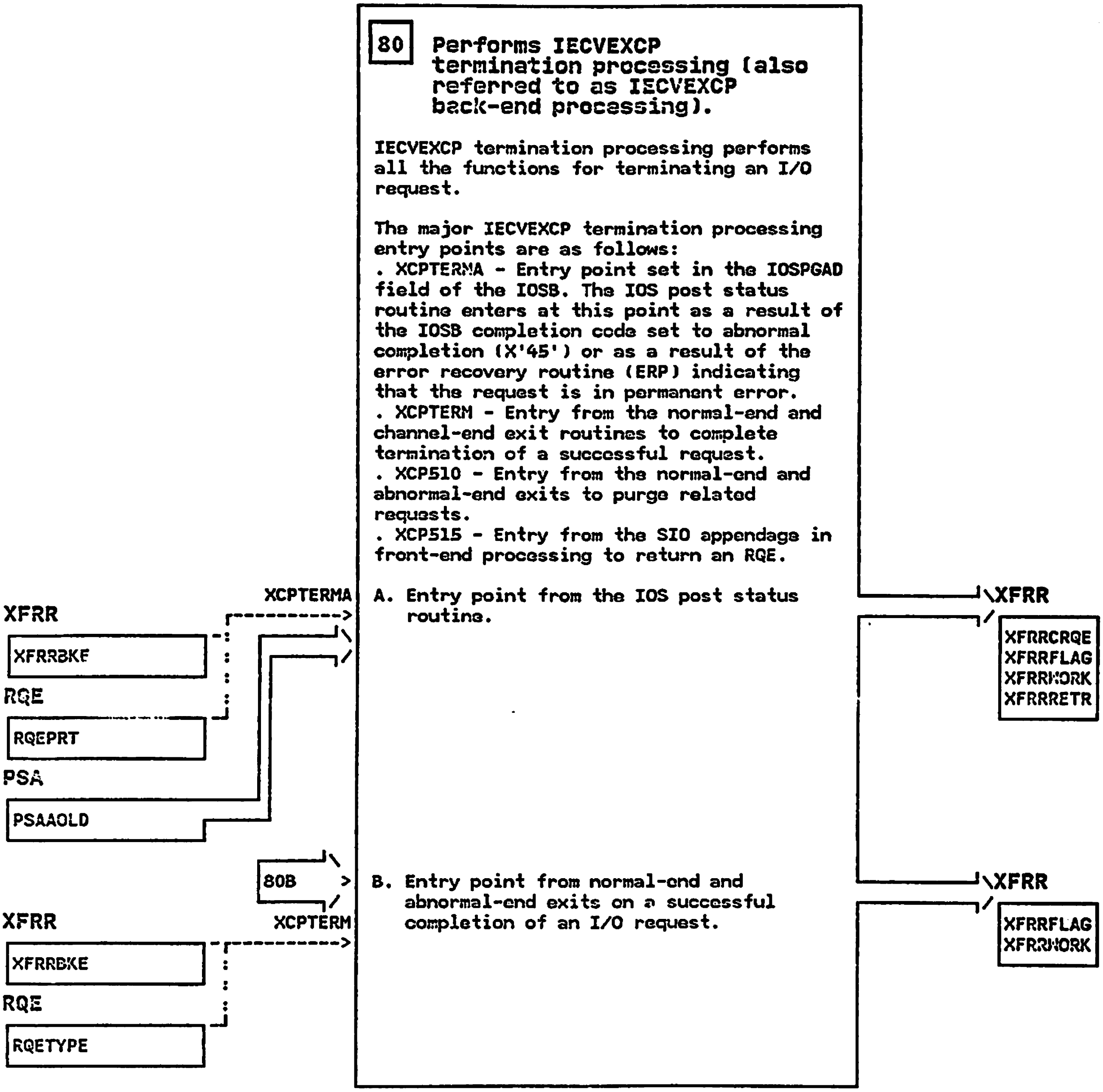
Moves the IOB CSW status and residual count fields to the IOSB.

Restores the environment to return to the caller.

H. Resets the error flag in both the IOB and the IOSB.

Sets a return code of 4 in register 15.

I. Returns to the caller.



80 Performs IECVEXCP termination processing (also referred to as IECVEXCP back-end processing).

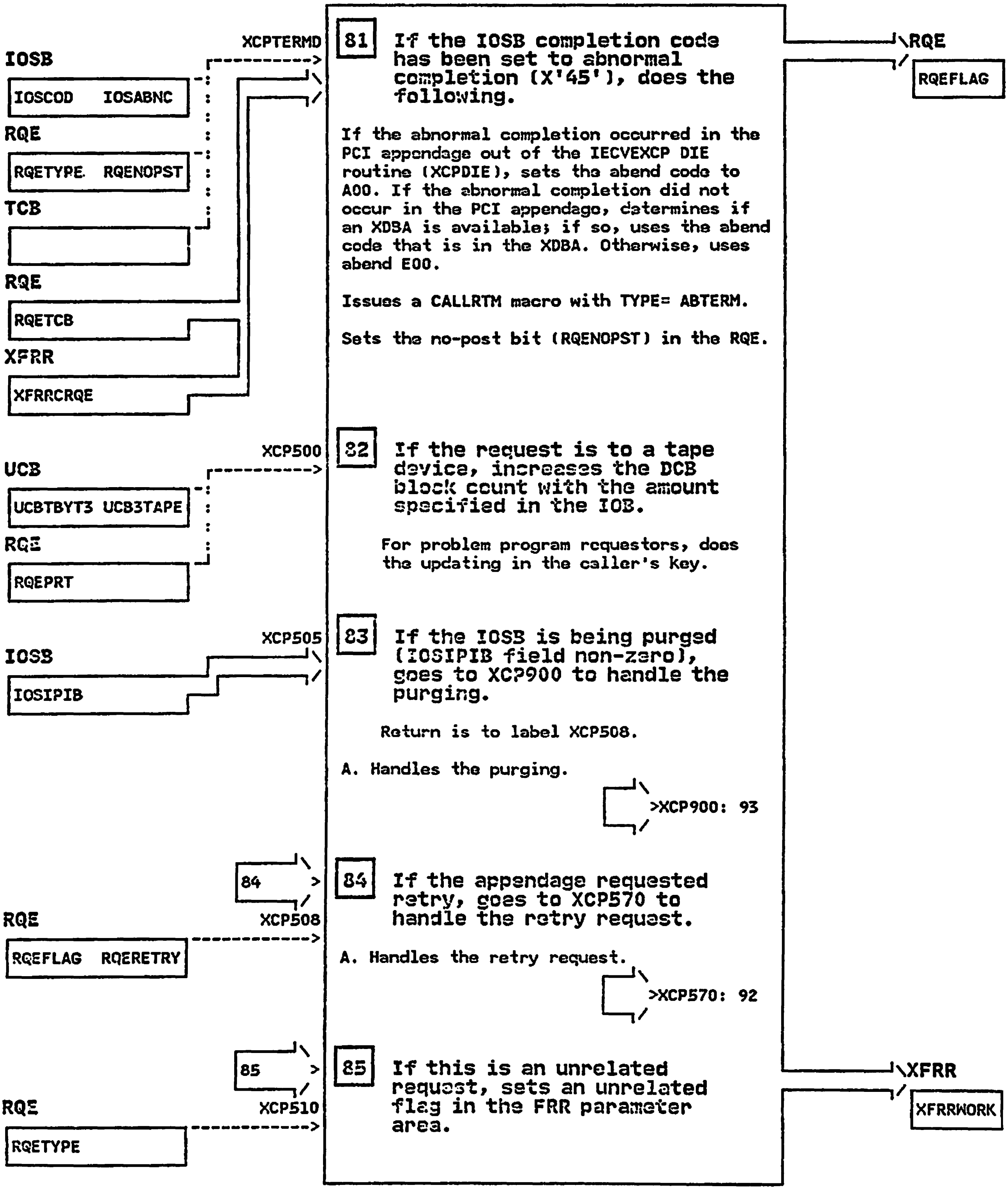
IECVEXCP termination processing performs all the functions for terminating an I/O request.

The major IECVEXCP termination processing entry points are as follows:
. XCPTERMA - Entry point set in the IOSPGAD field of the IOSB. The IOS post status routine enters at this point as a result of the IOSB completion code set to abnormal completion (X'45') or as a result of the error recovery routine (ERP) indicating that the request is in permanent error.
. XCPTERM - Entry from the normal-end and channel-end exit routines to complete termination of a successful request.
. XCP510 - Entry from the normal-end and abnormal-end exits to purge related requests.
. XCP515 - Entry from the SIO appendage in front-end processing to return an RQE.

A. Entry point from the IOS post status routine.

B. Entry point from normal-end and abnormal-end exits on a successful completion of an I/O request.

LY28-1685-0



81 If the IOSB completion code has been set to abnormal completion (X'45'), does the following.

If the abnormal completion occurred in the PCI appendage out of the IECVEXCP DIE routine (XCPDIE), sets the abend code to A00. If the abnormal completion did not occur in the PCI appendage, determines if an XDBA is available; if so, uses the abend code that is in the XDBA. Otherwise, uses abend E00.

Issues a CALLRTM macro with TYPE= ABTERM.

Sets the no-post bit (RQENOPST) in the RQE.

82 If the request is to a tape device, increases the DCB block count with the amount specified in the IOB.

For problem program requestors, does the updating in the caller's key.

83 If the IOSB is being purged (IOSIPIB field non-zero), goes to XCP900 to handle the purging.

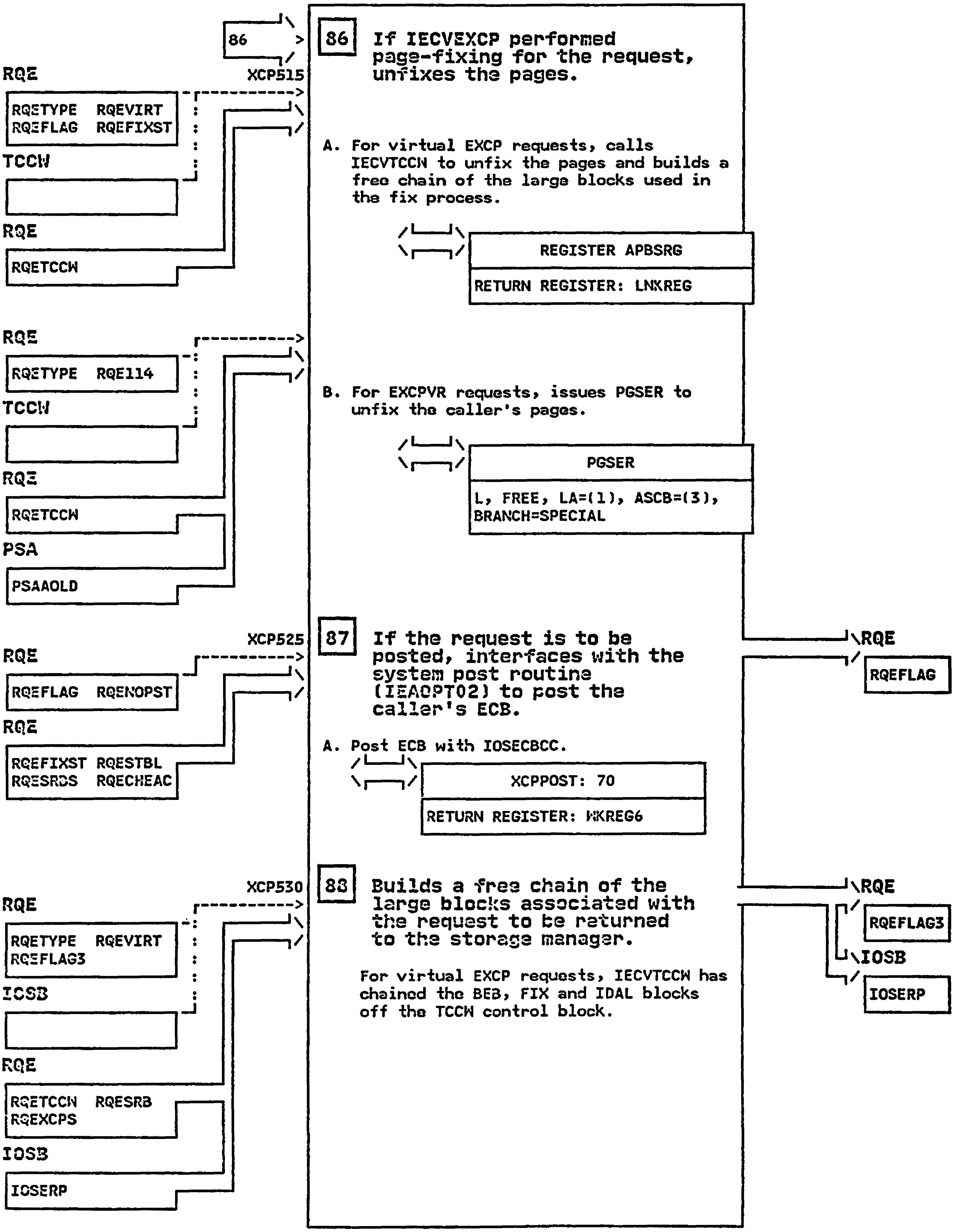
Return is to label XCP508.

A. Handles the purging.

84 If the appendage requested retry, goes to XCP570 to handle the retry request.

A. Handles the retry request.

85 If this is an unrelated request, sets an unrelated flag in the FRR parameter area.



86 If IECVEXCP performed page-fixing for the request, unfixes the pages.

A. For virtual EXCP requests, calls IECVTCCW to unfix the pages and builds a free chain of the large blocks used in the fix process.

REGISTER APBSRG
RETURN REGISTER: LNKREG

B. For EXCPVR requests, issues PGSER to unfix the caller's pages.

PGSER
L, FREE, LA=(1), ASCB=(3), BRANCH=SPECIAL

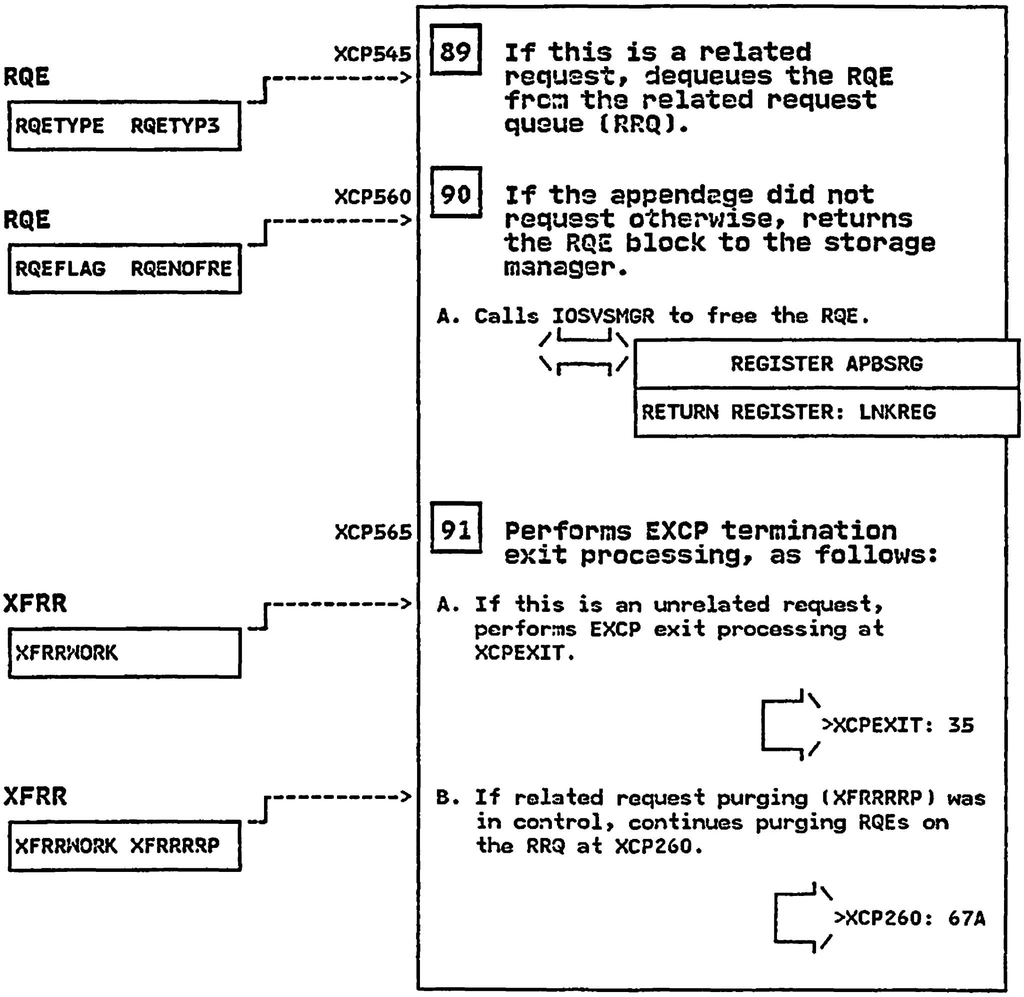
87 If the request is to be posted, interfaces with the system post routine (IEAOPT02) to post the caller's ECB.

A. Post ECB with IOSECBCC.

XCPPOST: 70
RETURN REGISTER: MKREG6

88 Builds a free chain of the large blocks associated with the request to be returned to the storage manager.

For virtual EXCP requests, IECVTCCW has chained the BEB, FIX and IDAL blocks off the TCCW control block.



89 If this is a related request, dequeues the RQE from the related request queue (RRQ).

90 If the appendage did not request otherwise, returns the RQE block to the storage manager.

A. Calls IOSVSMGR to free the RQE.

REGISTER APBSRG

RETURN REGISTER: LNKREG

91 Performs EXCP termination exit processing, as follows:

A. If this is an unrelated request, performs EXCP exit processing at XCPEXIT.

>XCPEXIT: 35

B. If related request purging (XFRRRRP) was in control, continues purging RQEs on the RRQ at XCP260.

>XCP260: 67A

C. For related requests, determines the processing to be done, as follows:

. If requestor's DCB indicates a permanent error, performs EXCP exit processing at XCPEXIT.

. If EXCP termination was entered from post status or from EXCP channel-end and normal-end exit routine (back-end bit is on - XFRRBKE), goes to XCPEXIT to perform exit processing.

. If the DEB RRQ field is zero (no RRQ) or there are no RQEs on the RRQ, goes to XCPEXIT to perform exit processing.

. Otherwise, establishes a pointer to the next RQE on the RRQ and does the following:

- If the new RQE indicates an end-of-extent error, goes to EXCP front-end processing at XCP105A to process the end-of-extent error.

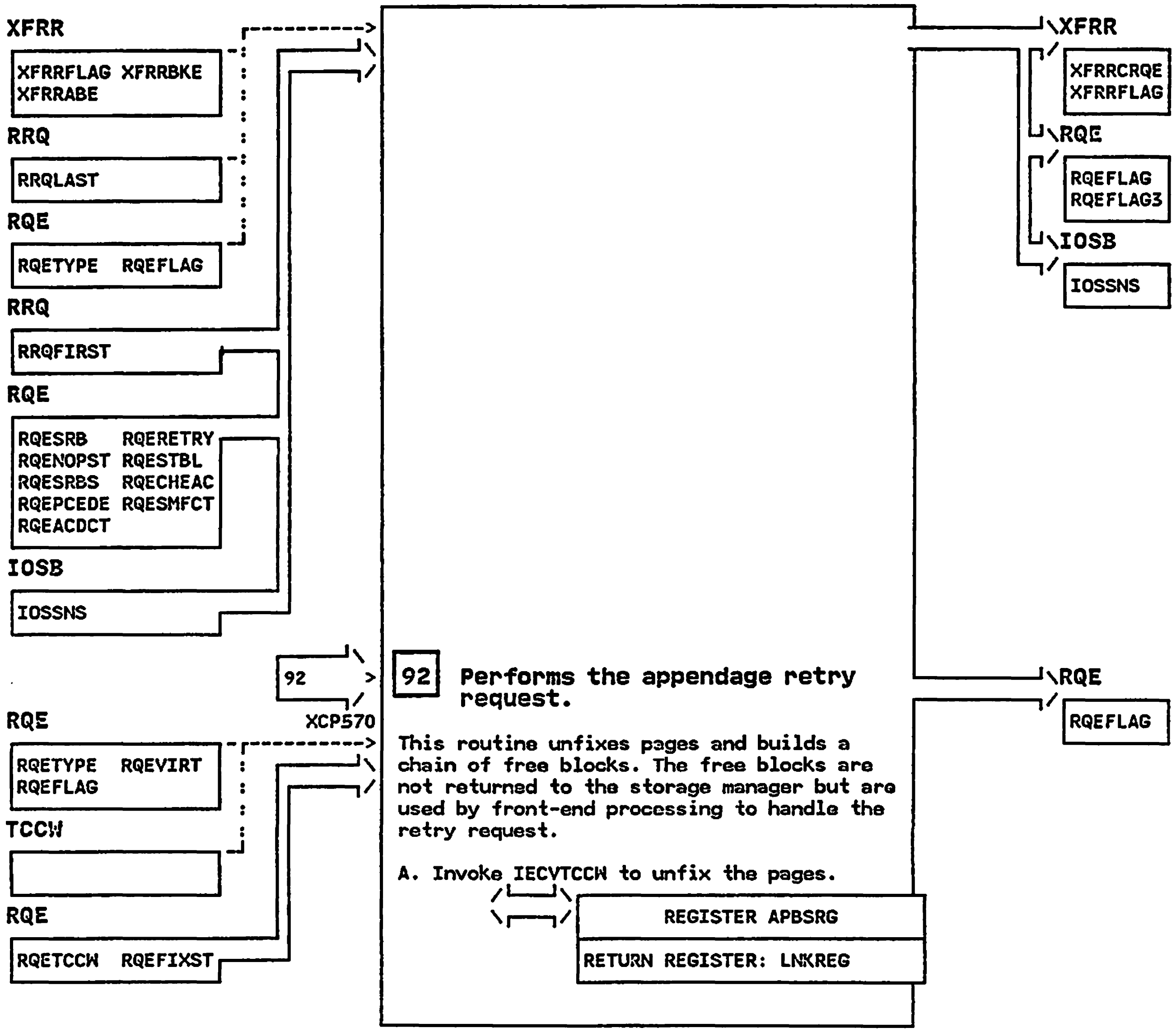
- If the new RQE indicates that the RQE has been sent to IOS, goes to XCPEXIT to perform exit processing.

- If the new RQE indicates that the RQE is startable, goes to EXCP front-end processing at XCP155 to issue the STARTIO macro to send the request to IOS.

- If the new RQE indicates a retry request, goes to EXCP front-end processing at XCP105 to prepare the caller's request for initiation.

- Otherwise, goes to EXCP front-end processing at XCP050 to obtain large blocks and prepare the caller's request for initiation.

>XCPEXIT: 35

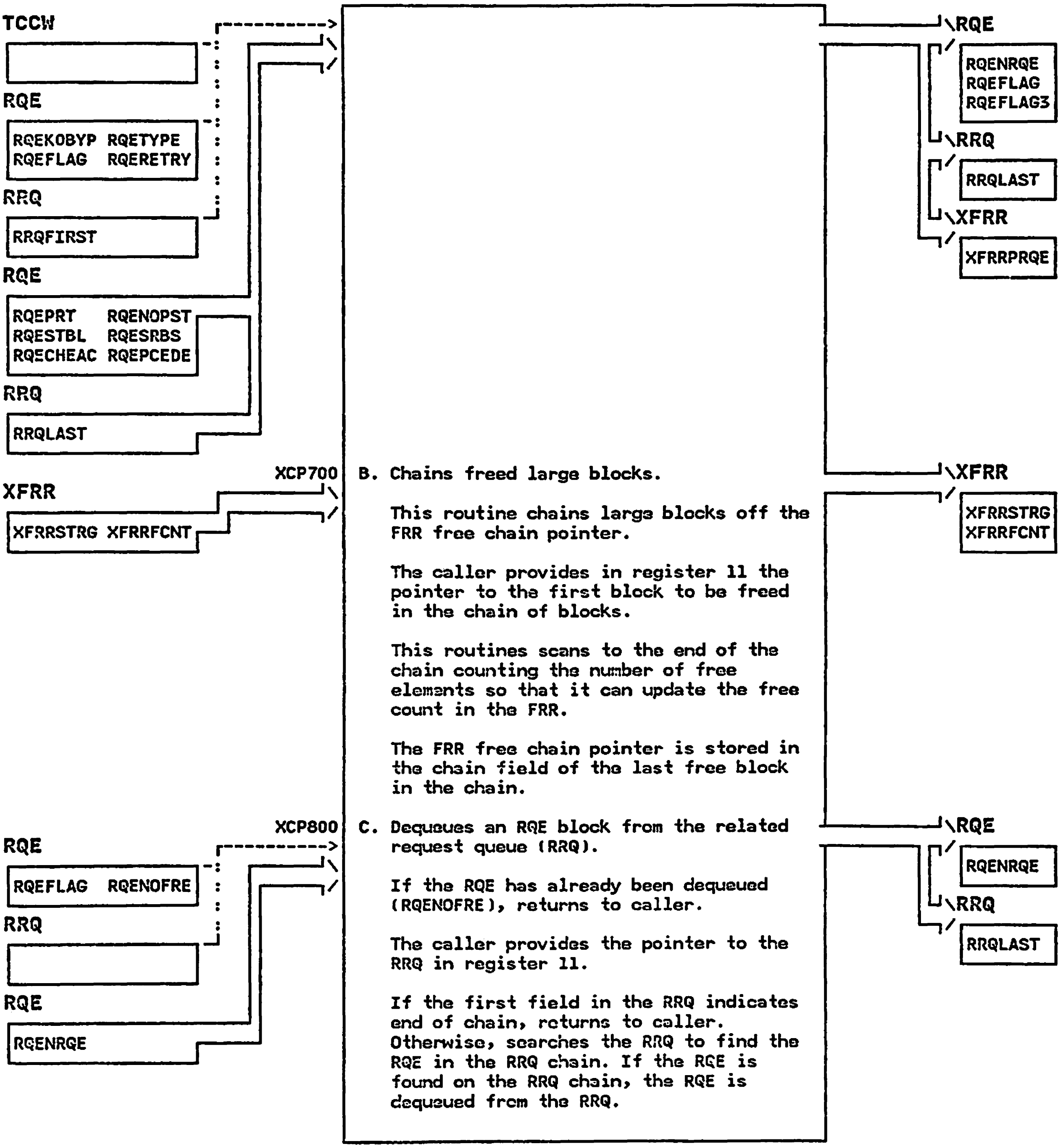


92 Performs the appendage retry request.

This routine unfixes pages and builds a chain of free blocks. The free blocks are not returned to the storage manager but are used by front-end processing to handle the retry request.

A. Invoke IECVTCCW to unfix the pages.

IECVEXCP - EXCP Processor for SVC 0 (EXCP) and SVC114 (EXCPVR) STEP 92B



B. Chains freed large blocks.

This routine chains large blocks off the FRR free chain pointer.

The caller provides in register 11 the pointer to the first block to be freed in the chain of blocks.

This routines scans to the end of the chain counting the number of free elements so that it can update the free count in the FRR.

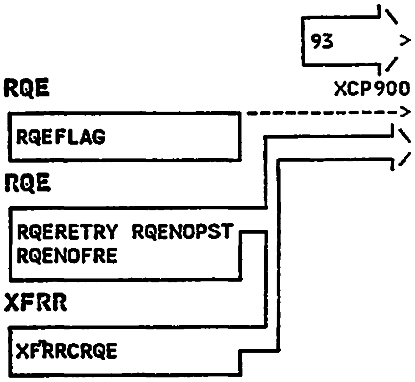
The FRR free chain pointer is stored in the chain field of the last free block in the chain.

C. Dequeues an RQE block from the related request queue (RRQ).

If the RQE has already been dequeued (RQENOFRE), returns to caller.

The caller provides the pointer to the RRQ in register 11.

If the first field in the RRQ indicates end of chain, returns to caller. Otherwise, searches the RRQ to find the RQE in the RRQ chain. If the RQE is found on the RRQ chain, the RQE is dequeued from the RRQ.



93 Determines how to handle the request that is being purged (IOSIPIB field points to an IPIB).

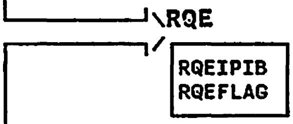
Sets the RQEIPIB field to the IPIB address.

If the RQE is to be freed (RQENOFRE) and retry is not requested, returns to IECVEXCP termination processing (XCP510) to continue termination processing. When the termination routine frees the RQE, the IPIB purge wait count is decreased.

If the RQE is to be freed (RQENOFRE) and retry is requested, calls the IECVRCHN routine in IECVEXPR to chain the caller's request on the EXCP PIRL data fields (IOB restore chain and the EPCB). The IPIB address is passed in register 0. If the caller's request is to be chained at the top of the IOB restore chain, the high-order bit of register 0 is set. Otherwise, the IOB associated with the caller's request is chained at the end of the IOB restore chain. Upon return from IECVRCHN, resets the retry (RQERETRY) and no-free (RQENOFRE) bits and sets the no-post (RQENOPST) bit and returns to IECVEXCP exit processing at XCP515.

If the appendage requests that the RQE not be freed (RQENOFRE) and the IPIB indicates an RB purge, returns to IECVEXCP termination processing (XCP508) to continue termination processing. When the requestor calls IECVEXCP at IECVX025 to free the RQE, the IPIB purge wait count will be decreased.

If the appendage requests that the RQE not be freed (RQENOFRE) and the IPIB does not indicate RB purging, calls IECVQCNT to decrease the purge wait count and zeroes the IOSIPIB field, and returns to IECVEXCP termination processing (XCP508) to continue termination processing.



IECVEXCP - EXCP Processor for SVC 0 (EXCP) and SVC114 (EXCPVR)   STEP 94

IOS Disabled Interruption Routine that interfaces with the driver's DIE exits.

XCPDIE

IOSB

IOSFLA IOSTSA
IOSTSB IOSIPIB

RQE

RQEPRT RQETYPE
RQEVIRT RQE1TO1
RQETYP3 RQEFLAG
RQESRBS RQEFLAG3

IOSB

IOSTATUS

RQE

RQETCCW RQENRQE
RQESRB RQESTBL
RQEPCEDE

94 Handles the IECVEXCP DIE processing.

The IECVEXCP DIE routine is entered from IOS post status after an I/O interrupt has occurred for an EXCP or EXCPVR request.

If the request was an EXCP V=R or EXCPVR and the PCI bit was set in the IOSB status field, does the following:

. If the PCI bit was set with ending status (channel and device end bits set in the device status), does not call the PCI appendage; goes to check for a type 3 related request.

. Otherwise, prepares to enter the PCI appendage:

- Loads pointers to the IOB, DCB, DEB and UCB.
- For problem program callers, validates the caller's IOB, DCB and ECB blocks.
- Maps the IOSB to the IOB.
- Calls the PCI appendage.
- Updates the IOSB real and virtual CCW start addresses from the IOB.
- Continues to check for type 3 related requests.

For type 3 related requests, when channel end status is indicated in the IOSB device status field, does the following:

. If the ERP is in control (IOSERR bit set on) or there is non-normal status, does not start the next ready I/O request.

Returns to the caller.

BSM

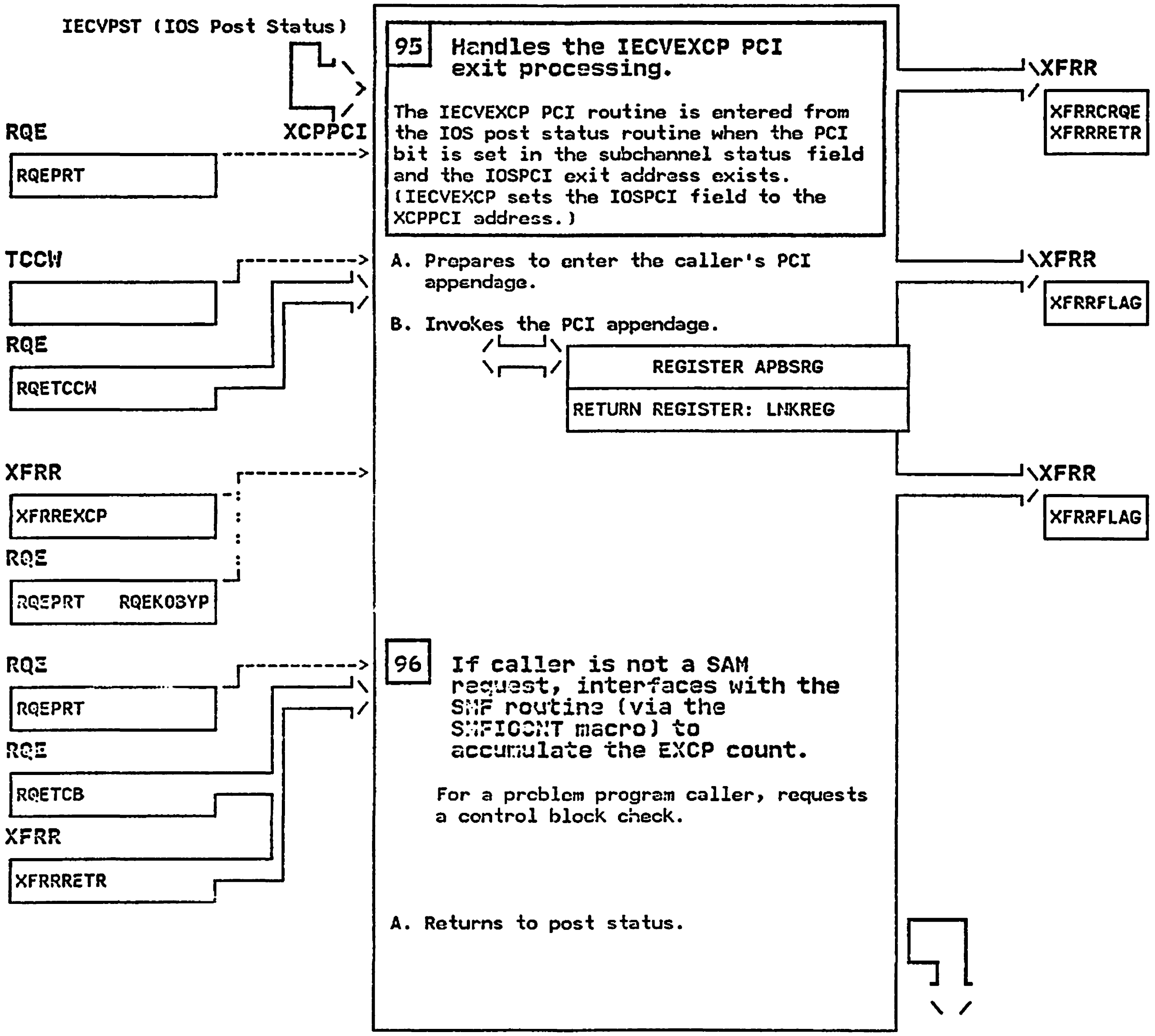
0, LNKREG

RQE

RQETYPE
RQEFLAG
RQEFLAG3

IOSB

IOSRST
IOSVST



IECVPST (IOS Post Status)

RQE: RQEPRT → XCPPCI

TCCW

RQE: RQETCCW

XFRR: XFRREXCP

RQE: RQEPRT RQEKOBYP

RQE: RQEPRT

RQE: RQETCB

XFRR: XFRRRETR

95 Handles the IECVEXCP PCI exit processing.

The IECVEXCP PCI routine is entered from the IOS post status routine when the PCI bit is set in the subchannel status field and the IOSPCI exit address exists. (IECVEXCP sets the IOSPCI field to the XCPPCI address.)

A. Prepares to enter the caller's PCI appendage.

B. Invokes the PCI appendage.

REGISTER APBSRG
RETURN REGISTER: LNKREG

96 If caller is not a SAM request, interfaces with the SMF routine (via the SMFIOCNT macro) to accumulate the EXCP count.

For a problem program caller, requests a control block check.

A. Returns to post status.

XFRR: XFRRCRQE XFRRRETR

XFRR: XFRRFLAG

XFRR: XFRRFLAG

IECVEXPR - EXCP purge routine,
IECVEXFR - FRR termination
request

IECVXTRM

RQE

RQEPRT

XFRR

XFRRCRQE
XFRRPRQE
XFRRSTRG
XFRRFLAG
XFRRRETR

97 Handles the termination request from IECVEXPR and IECVEXFR.

This routine is entered from IECVEXPR to terminate a purge request and from IECVEXFR (functional recovery routine) to terminate the request in error.

Sets up the environment to enter the IECVEXCP termination routine, as follows:
. No functional recovery environment is established. Builds a pseudo EXCP FRR parameter area in the area provided by the caller.
. Sets the EXCP FRR parameter processing byte to X'3E' to indicate IECVEXPR or IECVEXFR entry.
. Establishes pointers to the IOB, DCB, DEB and UCB.

A. If the caller is in system key, bypasses the validity checks and goes to the IECVEXCP termination routine.

>XCP510: 85

B. Otherwise, validity checks the caller's control blocks.

XCPVAL: 69
RETURN REGISTER: LNKREG

C. Goes to return the RQE and large blocks.

>XCP510: 85

SVC 3 exit routine, IECVEXPR
purge routine - Purge halt for
RB and AEQ purging.

IECVX025

RQE

RQEPRT RQEKOBYP
RQEFLAG3 RQESMFCT

RQE

RQEPRT

RQE

RQETCB RQEIPIB

98 Returns the RQE block for SVC 3 callers and the IECVEXPR purge routine.

This routine is entered from the SVC 3 exit routine and IECVEXPR purge routine to return an RQE block to the storage manager.

The local lock is held by the caller on entry.

A. If caller is not a SAM request, interfaces with the SMF routine (via the SMFIOCNT macro) to accumulate the EXCP count.

For a problem program caller, requests control block check.

B. If the request is undergoing purge quiesce, interfaces with the IOS quiesce count routine (IECVQCNT) in the IOS module IGC0001F to decrease the purge wait count.

RQE

RQENRQE

PSA

PSAAOLD

C. Increases the count of outstanding EXCP requests kept in the ASCB.

D. Returns the RQE block to the storage manager.

Sets the RQE block to be freed in register 1.

BASSM
LNKREG, APBSRG

E. Returns to the caller.

XCPFREE

RQE

RQEPRT	RQEKOBYP
RQEFLAG3	RQESMFCT

99 IECVEXCP common RQE free routine.

This routine is entered from the IECVEXCP VIO routine and the IECVEXCP termination routine to return an RQE block to the storage manager.

RQE

RQEPRT

XFRR

XFRRCRQE

RQE

RQETCB	RQEIPIB

A. If caller is not a SAM request, interfaces with the SMF routine (via the SMFIOCNT macro) to accumulate the EXCP count.

For a problem program caller, requests control block check.

XFRR

XFRRCRQE

RQE

RQEIOB

B. If the request is undergoing purge quiesce, interfaces with the IOS quiesce count routine (IECVQCNT) in the IOS module IGC0001F to decrease the purge wait count.

RQE

RQEIOB RQENRQE

PSA

PSAAOLD

C. Increases the count of outstanding EXCP requests kept in the ASCB.

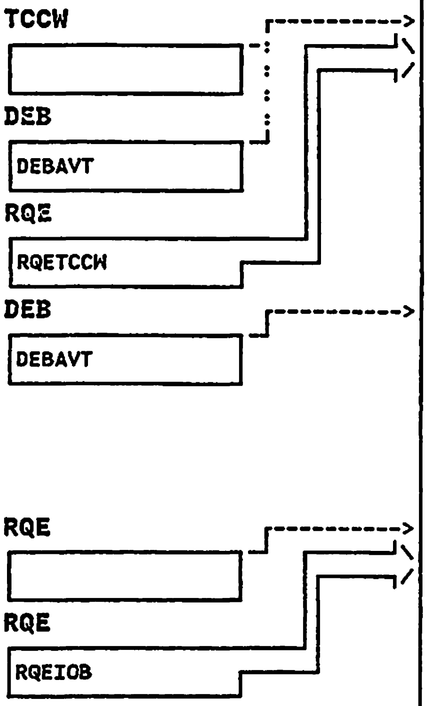
D. Returns the RQE block to the storage manager.

Sets the RQE block to be freed in register 1.

BASSM
LNKREG, APBSRG

E. Returns to the caller.

XCPDEBFX **100** Performs DEB fixing.



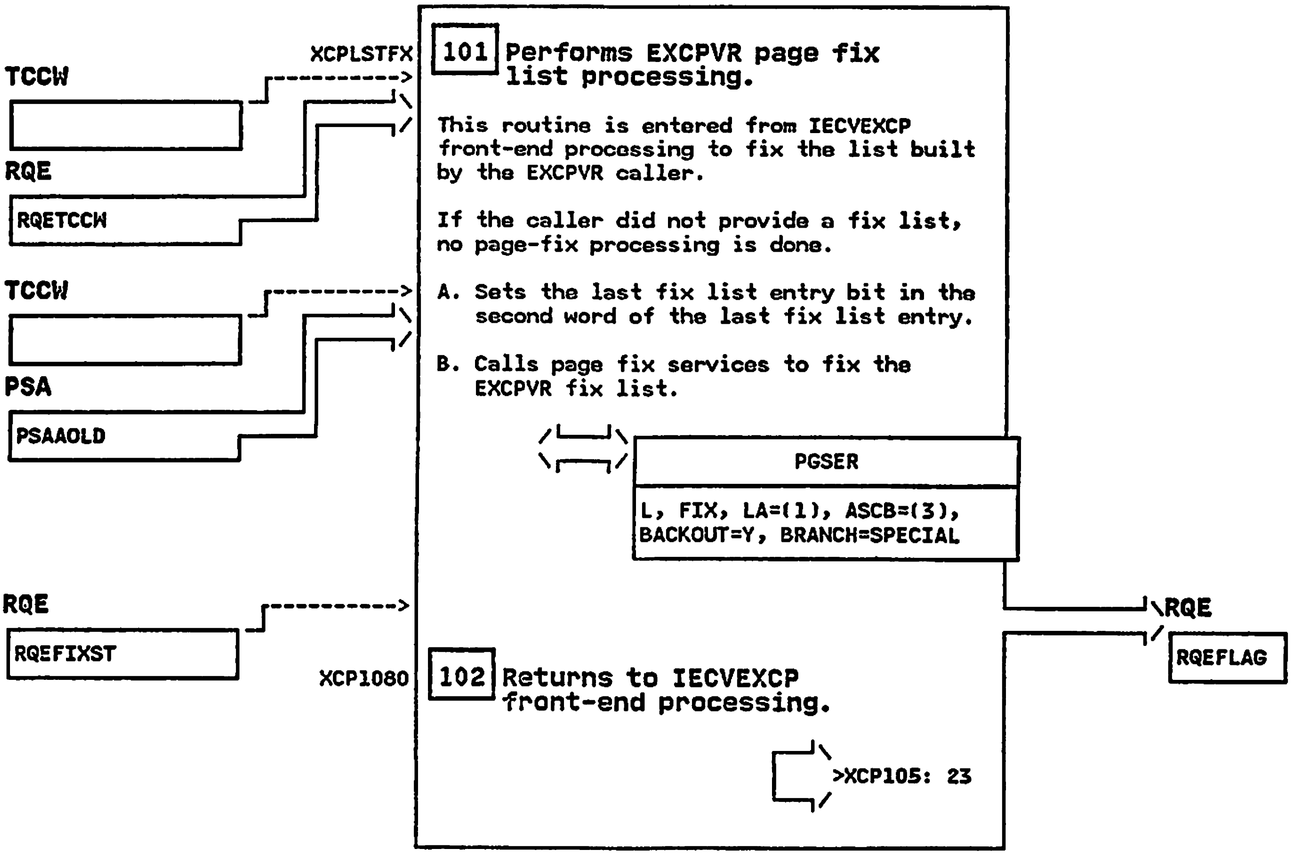
A. Builds a fix entry for the DEB storage area.

B. Interfaces with IARPSIV to fix the DEB storage area.

REGISTER APBSRG
RETURN REGISTER: LNKREG

C. If the return code from IARPSIV is non-zero, issues an 800 abend to abend the task.

>ABENDSET: 71



101 Performs EXCPVR page fix list processing.

This routine is entered from IECVEXCP front-end processing to fix the list built by the EXCPVR caller.

If the caller did not provide a fix list, no page-fix processing is done.

A. Sets the last fix list entry bit in the second word of the last fix list entry.

B. Calls page fix services to fix the EXCPVR fix list.

PGSER

L, FIX, LA=(1), ASCB=(3),
BACKOUT=Y, BRANCH=SPECIAL

102 Returns to IECVEXCP front-end processing.

IECVEXFR - EXCP FUNCTIONAL RECOVERY ROUTINE (FRR)

IECVEXFR - MODULE DESCRIPTION

DESCRIPTIVE NAME: IECVEXCP Functional Recovery Routine (FRR)

FUNCTION:

To recover from an unexpected error that caused exit to RTM. Also performs abend processing as a result of EXCP issuing an SVC 13 (D).

ENTRY POINT: IECVEXFR

PURPOSE: See Function.

LINKAGE: RTM linkage to FRR

CALLERS: RTM

INPUT: SDWA address, 200-byte work area address

OUTPUT:

- Serviceability data in the SDWA
- ABEND code set in the SDWA
- Control blocks copied to the XDBA debugging area.
- Indicator set to inform the Post Status FRR to schedule termination if an error occurred in an appendage.

EXIT NORMAL: Return to caller

EXTERNAL REFERENCES:

ROUTINES:

IECVSMFR	- The storage manager functional recovery routine
IECVTCFR	- CCW translation functional recovery routine.
IECVXTRM	- IECVEXCP termination routine
IECVX025	- IECVEXCP free RQE routine

DATA AREAS: XDBA - Debugging area

CONTROL BLOCKS:

ASCB	- Address space control block
ASXB	- Address space extension block
CVT	- Communication vector table
DCB	- Device control block
DEB	- Data extent block
ECB	- Event control block
FRRS	- Functional recovery routine setrp
IOB	- I/O block
IOCOM	- I/O communication area
IOSB	- I/O supervisor block
JSCB	- Job step control block
PSA	- Prefixed save area
QVPL	- Queue verifier parameter list
RB	- Request block
RQE	- Request queue element
RRQ	- Related request queue
SDWA	- System diagnostic work area
SRB	- Service request block
TCB	- Task control block
TCCW	- Translate CCW block
VRA	- Variable recording area
XDBA	- EXCP debugging area

SERIALIZATION: Local lock held

IECVEXFR - MODULE OPERATION

The EXCP functional recovery routine (FRR) receives control from the recovery termination manager (RTM) when unexpected errors occur in IECVEXCP processing or when IECVEXCP issues an SVC 13 abend.
This functional recovery routine will analyze the error condition, provide the appropriate recovery, and request retry or percolation.

Module processing consists of the following:

1. If the storage manager was active with an RQE or large block request (either get or free request), prepares to enter the storage manager's functional recovery routine for handling.
   Does the following:
   A. Creates a storage manager FRR parameter area and puts its address in the SDWAPARM field.
   B. Ensures that register 0 (address of a 200-byte work area) and register 1 (SDWA pointer) are set correctly.
   C. Issues the BASSM instruction to enter the storage manager's functional recovery routine.
   D. The storage manager's FRR analyzes the error, fills in the SDWA serviceability data, determines whether to retry or percolate, and returns to this routine.
   E. Upon return from the storage manager's FRR, checks the SDWA to determine if the storage manager requested retry or percolation.
   F. If the storage manager's FRR requested retry, sets the retry address to IECVXRTY in IECVEXFR. (IECVEXCP runs in 24-bit mode and the storage manager runs in 31-bit mode. RTM gives control to IECVXRTY, which issues a BSM instruction to enter the storage manager retry routine in 31-bit mode.) The storage manager will perform the retry and continue processing in its mainline.
   G. If the storage manager's FRR requested percolation, this functional recovery routine continues with its processing.

2. If the storage manager was not active or the storage manager indicated percolation, does the following:
   A. Provides the SDWA serviceabilty data.
   B. Provides the following debugging data in the variable recording area (VRA):
      . The original abend code
      . The EXCP FRR parameter area
      . If an RQE is available, the RQE block
      . If a TCCW is available, the first 48 bytes of the TCCW.
   C. If the PCI appendage was active, sets the SDWA abend code to A00 and proceeds to step 3 to continue.
   D. If an abend (028, 171 or 18A) or a program check occurred while page fix services were active (in IECVEXCP or IECVTCCW), does the following:
      . If IECVTCCW was active at the time of the error, issues a SETRP macro to indicate retry without recording. The retry address is the IECVTCCW retry routine.
      . If IECVEXCP was active with an EXCPVR page fix, sets the SDWA completion code to 800 and proceeds to step 3.
   E. If the SDWA indicates percolation, sets the SDWA completion code to A00 and proceeds to step 3.
   F. If the SDWA completion code indicates protection check (0C4) and the protection check occurred in one of the IECVEXCP validity check routines, sets the SDWA completion code to 200 and proceeds

to step 3.

G. For all other errors, sets the SDWA completion code to B00 and continues with step 3.

3. Issues the SETRP macro to update the SDWA completion code, requests a dump and indicates recording of the error in SYS1.LOGREC.

4. If the functional routine was entered enabled, obtains and builds the XDBA and chains it off the abending TCB.

5. If the RQE indicates a VIO request, zeroes the VIO work area fields in the RQE.

6. Performs the following cleanup functions:
   A. If IECVEXCP back-end (termination) was active, does the following:
      - Sets the no-post flag in the RQE.
      - If entry to back-end processing was from the normal-end or abnormal-end exits, continues with step 7. (This routine will percolate to the IOS post status functional recovery routine, which will schedule the IECVEXCP termination routine.)
      - If entry to back-end processing was from the IOS post status routine or from IECVEXCP front-end processing, calls the IECVEXCP purge termination routine (IECVXTRM) to return the RQE and large blocks to the IOS storage manager.
   B. If IECVEXCP front-end processing was active, does the following:
      - If the request has been sent to IOS, continues with step 7.
      - Otherwise, sets the no-post flag in the RQE and calls the IECVEXCP purge routine (IECVXTRM) to return any RQE and large blocks to the storage manager.
   C. Otherwise, IECVEXCP normal-end or abnormal-end exit interface processing was active. Does the following:
      - Sets the IOSB completion code to the abnormal completion code (X'45').
      - Continues with step 7.

7. If IECVEXCP back-end processing was active, frees the local lock.

8. Returns to RTM to percolate the error.

RECOVERY OPERATION: Same as module operation

IECVEXFR - DIAGNOSTIC AIDS

ENTRY POINT NAME: IECVEXFR

MESSAGES: None

ABEND CODES:

The following abends are generated in this functional recovery routine:

```
  200 - IOB, DCB, or ECB protect key is not the same as the
        user key.
  700 - A program check occurred while in a supervisor service
        routine invoked by EXCP.
  A00 - A program check occurred in a user appendage.
  B00 - Indeterminate error.
```

The following abends are detected and generated in IECVEXCP (via SVC 13) and are processed in this functional recovery routine (IECVEXFR):

```
  15C -  Issuer of  SVC 92 not in supervisor state.
  172 -  SVC 114 caller not authorized.
  300 -  DEB validity check failures:
         - DEB not an EXCP or ISAM DEB.
         - The IOBM index is larger than the DEBNMEXT index
           or both indexes are zero.
  400 -  DCB pointers failed validity check.
  500 -  DEB does not provide a valid UCB, or
         an ISAM IOB IOBM field specified extent 0.
  800 -  Error in attempting to fix/unfix pages.
  A00 -  A program check occurred in the PCI appendage
         when called out of the EXCP DIE routine.
```

WAIT STATE CODES: None

RETURN CODES:

EXIT NORMAL:

```
  In SDWARCDE:
  0- Continue with termination (percolate
     the error).
  4- Retry to the IECVTCCW retry routine.
     The TCCW block address is stored in the
     SDWA retry register 11 save area.
     Any return code contained in register 15
     at the time of the error is in field
     SDWASR06.
```

REGISTER CONTENTS ON ENTRY:

```
Register  0    - Address of a 200-byte work area
Register  1    - Address of the SDWA.
Register  2-13- Irrelevant
Register 14    - Return address
Register 15    - Entry point of IECVEXFR
```

REGISTER CONTENTS ON EXIT:

EXIT NORMAL:

Same as on entry.

IECVEXFR - IECVEXCP Functional Recovery Routine (FRR) STEP 01

RTM

IECVEXFR

To recover from an unexpected error that caused exit to RTM. Also performs abend processing as a result of EXCP issuing an SVC 13 (D).

01 Determines if the storage manager (IOSVSMGR) was in control at the time of the error.

02 If the storage manager was active with either an RQE or large block request (get or free request), returns to the IOSVSMGR functional recovery routine (FRR) to handle the error.

The IOSVSMGR FRR routine will determine whether to retry or percolate the request and then return.

Sets up the registers to simulate entry from RTM:
. Register 0 - Address of the RTM 200-byte workarea. Note: The 200-byte work area is used to contain the queue verification parameter list (QVPL), the FRR parameter area passed to the storage manager FRR, and an EXCP FRR save area.
. Register 1 - Address of the SDWA. SDWAPARM contains the address of the FRR parameter area formatted for storage manager usage.

SFRR
SFRRSLEN

A. Builds the storage manager FRR parameter area in the 200-byte work area.

For RQE block requests, builds an FRR parameter area.

For large block requests, moves the FRR parameter area that is in the local lock save area (last six words) to the FRR parameter area in the 200-byte work area.

EXFR106 B. Issues BASSM to enter the storage manager's FRR routine in 31-bit mode.

BASSM
REG1→, REG15

SDWA
SDWARCDE

03 Upon return from the storage manager, retries or percolates the error as requested by the storage manager.

SDWA

A. If the storage manager FRR routine requested percolation, continues with EXCP FRR processing.

>EXFR200: 04B

B. If the storage manager FRR routine requested retry, does the following:

Stores the IECVEXCP FRR retry address (IECVXRTY) in the SDWA.

Moves the FRR parameter list from the 200-byte work area to the last six words of the local lock work area (the 200-byte work area is released by RTM before entering the retry routine). Only the word containing the ASID and flags are moved (bytes 56-59 of the local lock save area). The rest of this FRR parameter area must not be changed by this routine or by the storage manager retry routine.

C. Returns to RTM with the EXCP FRR retry routine address.

SDWA

SDWARTYA

IECVXRTY

04 Interfaces with the storage manager's retry routine.

This routine receives control to switch addressing mode from 24-bit to 31-bit and to issue a BSM to the storage manager's FRR retry routine.

A. Goes to the storage manager's FRR retry routine.

BSM
0, REG15

04B > EXFR200

B.

05 Performs EXCP mainline recovery processing.

SDWA: SDWAXPAD SDWASRVP

A. Moves the load module name, module name and the FRR routine name into the SDWA.

06 Provides debugging data in the SDWA variable recording area (VRA).

SDWA: SDWACMPC

A. Saves the original abend code in the SDWA VRA area and provides an area for storing the adjusted abend code.

B. Saves the EXCP FRR parameter area.

RQE: RQEBL

C. If an RQE is available, moves the RQE block into the VRA.

D. If a TCCW block is available, moves the first 48 bytes of the TCCW control block to the VRA.

EXFR202

07 If the RQE pointer is valid, determines if the PCI appendage was active.

A. If the error occurred in the PCI appendage called out of the EXCP DIE routine, sets the registers in the SDWA register save area (SDWAGRSV) to zero.

B. Goes to issue the abend.

>EXFR290: 11A

TCCW

EXFR205

08 Determines if the abend occurred while IECVEXCP or IECVTCCW was active with a page fix request.

Checks if system abend code 028, 171 or 18A was issued by the page fix services, or if a program check occurred while page fix services were active.

TCCW

A. If IECVTCCW was active at the time of the error, saves the original abend code in the SDWA VRA area and sets the retry address to the IECVTCCW FRR retry routine.

Saves any return code that was in register 15 at the time of the error in field SDWASR06 and establishes the TCCW block pointer in field SDWASR11.

Issues the SETRP macro to establish the IECVTCCW retry address and to indicate no recording.

B. Returns to RTM to perform the IECVTCCW retry.

EXFR212

C. If IECVEXCP was active with an EXCPVR page fix, sets the abend code to 800.

D. Goes to issue the SETRP macro.

>EXFR290: 11A

EXFR215

09 If this FRR received control as the result of percolation (SDWAPERC is on), leaves the abend code set to 700. Otherwise, sets the abend code to A00.

SDWA

SDWACMPC

10 Determines if entry was a result of an IECVEXCP validity check.

Issues the 200 abend if the SDWA completion code indicates a protection check (0C4 completion code) and the error occurred in an IECVEXCP validity check routine.

EXFR230 **11** If the error condition was not identified by any of the previous checks, set the abend code to B00.

11A > EXFR290 A. Issues the SETRP macro to update the SDWA completion code, requesting a dump and indicating a recording in LOGREC.

EXFR300 B. Saves the issuing abend code in the VRA area next to the original abend code.

EXFR310 **12** Determines if XDBA debugging data is to be provided to the abending TCB.

A. If the FRR routine was entered enabled, obtains and builds an XDBA block.

EXFRXDBA: 18
RETURN REGISTER: REG15

EXFR315 **13** For a VIO request, zeroes the VIO work area fields in the RQE control block.

EXFR320 **14** Determines what clean-up functions to perform.

A. If IECVEXCP back-end processing was active, sets the no-post flag in the RQE (RQENOPST) to bypass posting the ECB.

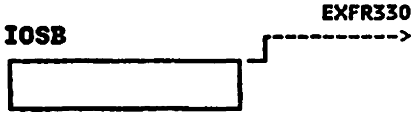
If back-end processing was entered from the channel-end or abnormal-end appendage, continues processing at EXFR350.

If back-end processing was entered from the IOS post status routine or from front-end processing, returns all storage blocks associated with the EXCP request.

EXFR325 B. If IECVEXCP front-end processing was active, does the following:

If the EXCP request is in IOS (RQEINIOS), continues processing at EXFR350.

Otherwise, sets the no-post flag (RQENOPST) in the RQE to indicate that the ECB is not to be posted and continues processing at EXFR350.



EXFR330

C. If channel-end or abnormal-end appendage interface processing was active, does the following:

Saves the IOSB completion code (IOSCOD) in the SDWA VRA area.

Sets the IOS completion code to IOSABNC (X'45') to indicate that a processing error occurred.

EXFR350

15 If the local lock was obtained, frees it.

EXFR360

16 Determines if the error is to be recorded on SYS1.LOGREC.

If IECVEXFR was entered enabled and established an abend code of 700 or less, issues a SETRP macro to request a dump without recording on LOGREC.

17 Returns to RTM.

18

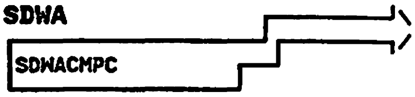
EXFRXDBA

18 Obtains storage and initializes the XDBA area.

A. Issues a GETMAIN to obtain storage from subpool 230 for the XDBA data and initializes it to zeroes.

B. Sets the XDBA pointer in field TCBEXCPD of the TCB.

Sets any previous TCB XDBA pointer in field XDBACHAN of the XDBA.



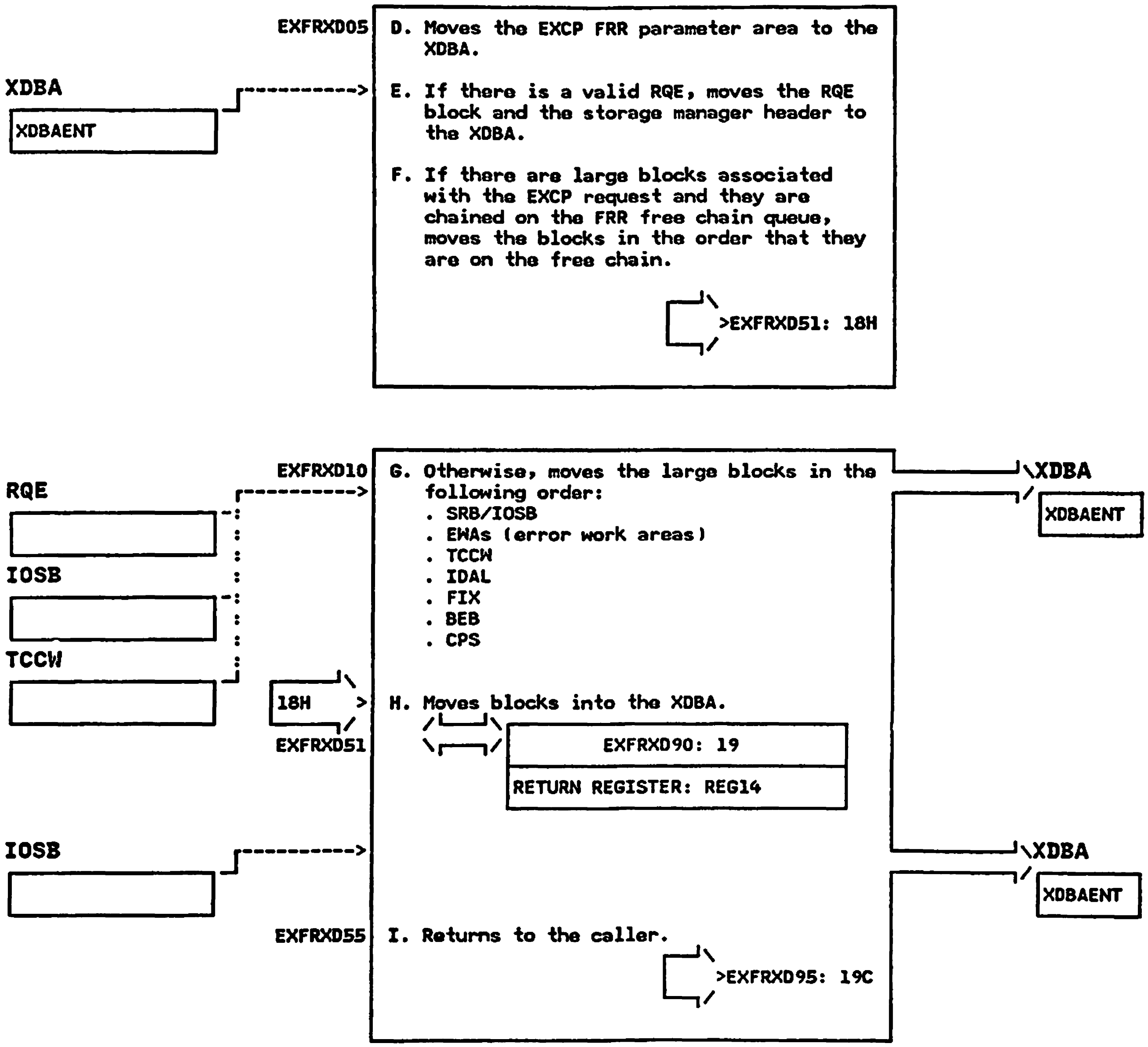
C. If an SDWA is available, moves data from the SDWA to the XDBA.

Moves the original abend code from the SDWA.

Sets the SDWA system completion code (SDWACMPC) fields in the XDBA. This represents the adjusted abend code.

Moves the translation exception address from the SDWA (SDWATRAN) to the XDBA.

If the registers and PSW are valid, moves these fields from the SDWA to the XDBA.



EXFRXD05 D. Moves the EXCP FRR parameter area to the XDBA.

E. If there is a valid RQE, moves the RQE block and the storage manager header to the XDBA.

F. If there are large blocks associated with the EXCP request and they are chained on the FRR free chain queue, moves the blocks in the order that they are on the free chain.

>EXFRXD51: 18H

EXFRXD10 G. Otherwise, moves the large blocks in the following order:

- SRB/IOSB
- EWAs (error work areas)
- TCCW
- IDAL
- FIX
- BEB
- CPS

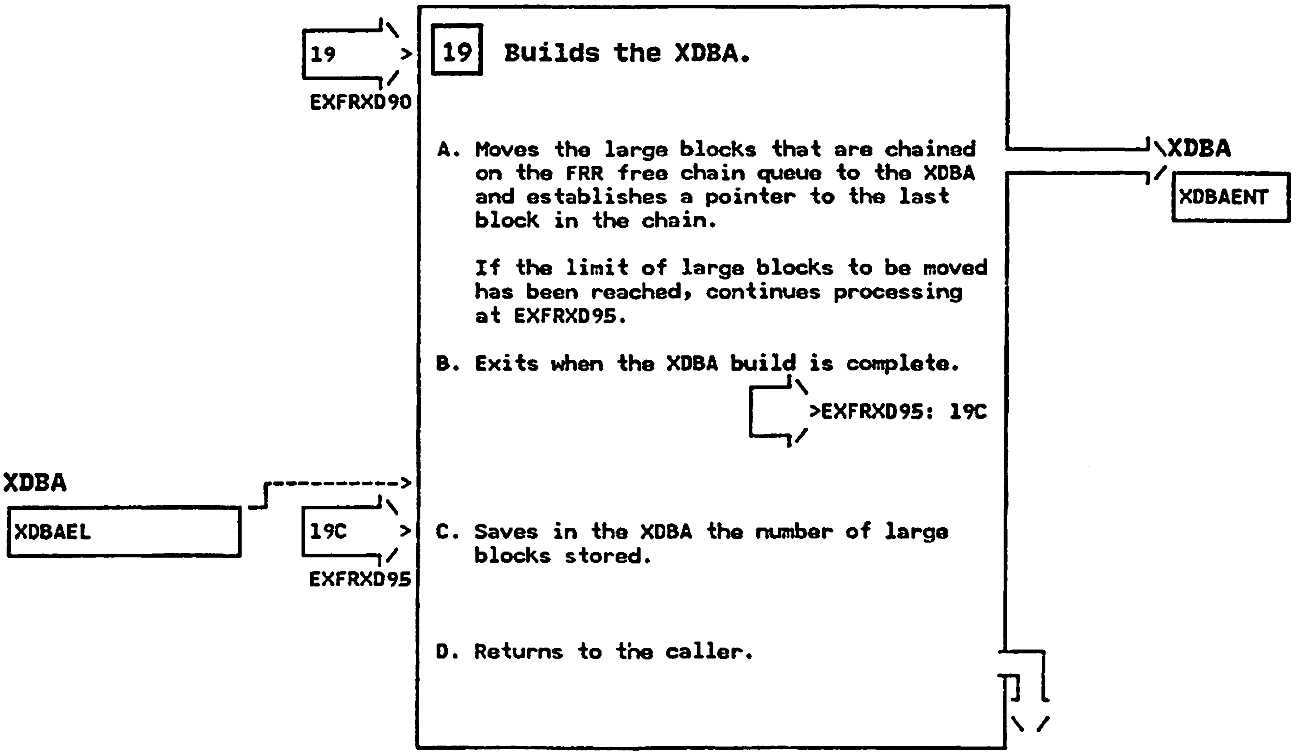
18H > EXFRXD51 H. Moves blocks into the XDBA.

EXFRXD90: 19

RETURN REGISTER: REG14

EXFRXD55 I. Returns to the caller.

>EXFRXD95: 19C



19 Builds the XDBA.

A. Moves the large blocks that are chained on the FRR free chain queue to the XDBA and establishes a pointer to the last block in the chain.

If the limit of large blocks to be moved has been reached, continues processing at EXFRXD95.

B. Exits when the XDBA build is complete.

C. Saves in the XDBA the number of large blocks stored.

D. Returns to the caller.

IECVEXPR - PROCESSOR'S PURGE AND RESTORE ROUTINES

IECVEXPR - MODULE DESCRIPTION

DESCRIPTIVE NAME: EXCP Processor's Purge and Restore Routines

FUNCTION:
IECVEXPR controls and manages the EXCP processor purge and restore functions. Four entry points are provided for callers to interface with the EXCP processor for purge and restore functions.

ENTRY POINT: IECVXPUR+0 - Purge SVC 16

PURPOSE:
This entry point accepts from the IOS purge function (IGC0001F) an IOS purge interface block (IPIB) which describes the purge function(s) that are to be performed. The IPIB indicates one of two options, halt or quiesce.

LINKAGE: Branch and link to entry point IECVXPUR

CALLERS: IOS purge module IGC0001F

INPUT: Address of the purge IPIB block

OUTPUT:
For halt, all associated requests are purged and, if indicated in the IPIB block, the callers of all purged requests are posted with a purge completion code. For quiesce, a list of IOBs to be restored with the associated EPCB blocks are chained off the IOS PIRL block.

EXIT NORMAL: Return to caller

ENTRY POINT: IECVXPUR+4 - I/O Halt (SVC 33 for BTAM)

PURPOSE:
This entry point receives control from the IOS SVC 33 function (IGC0003C) to terminate an active channel program. This function is provided for BTAM for terminating a teleprocessing program. This routine changes a real channel program CCW operation code to a NOP command code, so that the channel program will go to completion.

LINKAGE: Branch and link to entry point IECVXPUR+4

CALLERS: IOS I/O Halt SVC module IGC0003C

INPUT:
Offset to the CCW command that is to be changed to a NOP operation code.

OUTPUT:
The corresponding real channel CCW changed to a NOP operation code.

EXIT NORMAL: Return to caller

ENTRY POINT: IECVXRES

PURPOSE:
This entry point receives control from the IOS SVC 17 function (IGC0001G) to restore EXCP I/O requests that are chained on the IOB restore chain. Register 1 points to the EXCP entry in the IOS PIRL block.

LINKAGE: Branch and link to entry point IECVXRES

CALLERS: IOS I/O Restore SVC module IGC0001G

INPUT:
The two words provided by the EXCP processor in the IOS PIRL block on a purge quiesce request.

OUTPUT:
All the IOBs on the IOB restore chain have been re-issued.

EXIT NORMAL: Return to caller

ENTRY POINT: IECVRCHN

PURPOSE: This entry point builds an EXCP restore IOB chain.

LINKAGE: Branch and link to entry point IECVRCHN

CALLERS:
From entry point IECVXPUR in this module or from IECVEXCP

INPUT:
When provided, register 0 contains the IPIB block and register 1 the address of the RQE block.

OUTPUT:
The IOB has been put on the IOB restore chain and information about the IOB is put in the EPCB block. If an IOS PIRL block is not provided (no IPIB block or the DEBPIRL pointer is zero), a PIRL is obtained and a pointer is set in the DEB.

EXIT NORMAL: Return to caller

ENTRY POINT: IECVEXCL

PURPOSE:
This entry point dequeues IOBs from the IOB restore chain associated with the DCB to be closed.

LINKAGE: Branch and link to entry point IECVEXCL

CALLERS: RTM module IEAVTAS3

INPUT:
Register 1 contains a pointer to a two-word parameter list which contains the address of the PIRL and the address of the DCB being closed.

OUTPUT:
All the IOBs associated with the DCB being closed are dequeued from the IOB restore chain.

EXIT NORMAL: Return to caller

EXTERNAL REFERENCES:

ROUTINES:
IEAOPT02 - Post with validity check
IECVSMLF - Storage manager to free a large block
IECVSMLG - Storage manager to get a large block
IECVXTRM - EXCP termination routine
IECVX025 - Free RQE routine

CONTROL BLOCKS:
ASCB - Address space control block
ASXB - Address space extension block

IECVEXPR - MODULE DESCRIPTION (Continued)

CVT	- Communications vector table
DCB	- Data control block
DEB	- Data extent block
ECB	- Event control block
EPCB	- EXCP purge control block
IOB	- Input/output block
IOCOM	- IOS communication area
IOSB	- I/O supervisor block
IPIB	- IOS purge interface block
JSCB	- Job step control block
PIRL	- Purge I/O restore list
PSA	- Prefixed save area
RB	- Request block
RQE	- Request queue element
RRQ	- Related request queue
SRB	- Service request block
TCB	- Task control block
TCCW	- Translate CCW control block

SERIALIZATION: Local lock held

IECVEXPR - MODULE OPERATION

This module controls and manages the EXCP processor purging and restore functions.

This module consists of four separate entry points to perform the following functions:

1) IECVXPUR - This entry point provides a table for two SVC functions.

- +0 - SVC 16(10), Purge SVC.
  This SVC performs the halt or quiesce function on EXCP (SVC 0) and EXCPVR (SVC 114) operations.
- +4 - SVC 33(21), I/O Halt SVC for BTAM.
  This SVC performs a request from BTAM to terminate its active channel program.

The halt and quiesce functions are handled as follows:

a. Halt option, DEB purge -- Frees all requests associated with the DEB. These include ali requests passed by IGC0001F from the IOS queues, all related requests (for the 3525, only the requests associated with this DEB are freed), and all requests on the asynchronous exit queues (AEQs). If RB purge was also specified, the TCB RB chain is searched for IRBs with RQEs associated with the DEB. If requested in the IPIB, posts the caller.

b. Halt option, TCB purge -- Performs the same function as a DEB purge for each DEB on the TCB chain.

c. Halt option, address space purge -- No processing is done by IECVXPUR. In the process of its address space processing, IOS will issue a purge request to the storage manager to free all RQE and large blocks associated with the purged address space.

d. Quiesce option, DEB purge -- Builds an IOB restore chain containing all the IOBs that have not been sent to IOS. In the process of building the IOB restore chain, builds an EXCP purge control block (EPCB) in protected storage which contains, for each IOB, its corresponding TCB address, the protect key of the originator of the request, and type of IOB (EXCP or EXCPVR).
The pointer to the first IOB on the restore chain and the pointer to the first EPCB are stored in the two words for the EXCP driver in the IOS PIRL block. For all requests that have been sent to IOS, the IPIB quiesce count is increased by 1. The EXCP module IECVEXCP decreases the IPIB count when the I/O interruption is received for processing. Only when the count has gone to zero does IOS know that all outstanding I/O requests have completed.

e. Quiesce option, TCB purge -- Performs the same processing as DEB purge for each DEB on the TCB chain.

f. Quiesce option, address space purge -- Performs the same function as TCB purge for each TCB in the address space.

2) IECVXRES - SVC 17(11), RESTORE SVC.
Restores previously quiesced EXCP or EXCPVR requests by issuing SVCs 0, 92, or 114.

3) IECVRCHN - Builds a chain of EXCP or EXCPVR requests to be restored. This routine is called

IECVEXPR - MODULE OPERATION (Continued)

```
              by this module to build a restore chain for
              a caller's purge quiesce request.
 4) IECVEXCL - Dequeues IOBs on the restore chain for DCBs
              to be closed during RTM processing.
```

IECVEXPR - DIAGNOSTIC AIDS

ENTRY POINT NAMES: IECVXPUR+0 - Purge SVC 16
IECVXPUR+4 - I/O Halt (SVC 33 for BTAM)
IECVXRES
IECVRCHN
IECVEXCL

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES:

ENTRY POINT IECVXPUR+0: None

ENTRY POINT IECVXPUR+4:

EXIT NORMAL:

Register 15 contains one of the following decimal values:
0 - Valid request. The specified real CCW command code was changed to a NOP.
16 - Invalid request. The EXCP I/O request was not a virtual EXCP request (SVC 0).
20 - Invalid request. IECVEXCP is already in the process of translating the virtual EXCP request to a real channel program.
24 - Invalid request. The corresponding real channel program CCW associated with the virtual channel CCW address passed in register 0 could not be found.

ENTRY POINT IECVXRES:

EXIT NORMAL:

Register 15 contains:
0 - Restore processing complete.

ENTRY POINT IECVRCHN: None

ENTRY POINT IECVEXCL: None

REGISTER CONTENTS ON ENTRY:

ENTRY POINT IECVXPUR+0:

Régister 0 -	Irrelevant
Register 1 -	Address of the IPIB block
Registers 2-12 -	Irrelevant
Register 13 -	Address of an 18-word savearea
Register 14 -	Return address to purge (IGC0001F)
Register 15 -	Entry point address

ENTRY POINT IECVXPUR+4:

Register 0 -	Offset to the caller's virtual channel program CCW that is to be changed to the NOP command code.
Register 1 -	Irrelevant
Register 2 -	Address of the EXCP processor IOSB
Registers 3- 5 -	Irrelevant

IECVEXPR - DIAGNOSTIC AIDS (Continued)

Register 6 - Caller's base register (must be maintained)
Registers 7-12 - Irrelevant
Register 13 - Address of an 18 word savearea.
Register 14 - Return address to I/O halt (IGC0003C)
Register 15 - Entry point address

ENTRY POINT IECVXRES:

Register 0 - Irrelevant
Register 1 - Pointer to a two-word area containing pointers to the IOB restore chain and EXCP's EPCB block which contains the TCB/IOB data. These two words were provided by EXCP in the IOS PIRL block on a purge-quiesce request.
Registers 2-5 - Irrelevant
Register 6 - Caller's base register (must be maintained)
Registers 7-12 - Irrelevant
Register 13 - Address of an 18-word savearea
Register 14 - Return address to I/O restore module IGC0001G
Register 15 - Entry point address

ENTRY POINT IECVRCHN:

Register 0 - Address of an IPIB block or zero.
When the register contains an IPIB address and is positive, queues the IOB at the end of the IOB restore chain.
When the register is negative, queues the IOB at the beginning of the IOB restore chain.
Register 1 - Address of an RQE block which contains the IOB to be put on the restore chain
Registers 2-12 - Irrelevant
Register 13 - Address of an 18-word savearea
Register 14 - Caller's return address
Register 15 - Entry point address

ENTRY POINT IECVEXCL:

Register 0 - Irrelevant
Register 1 - Pointer to a two-word area:
. Word 1 - Pointer to a fullword which contains the address of the PIRL
. Word 2 - Pointer to a fullword which contains the address of the DCB being closed
Registers 2-12 - Irrelevant
Register 13 - Address of an 18-word savearea
Register 14 - Caller's return address
Register 15 - Entry point address

REGISTER CONTENTS ON EXIT:

ENTRY POINT IECVXPUR+0:

EXIT NORMAL:

Registers 0-15 - Same as on entry.

ENTRY POINT IECVXPUR+4:

IECVEXPR - DIAGNOSTIC AIDS (Continued)

EXIT NORMAL:

Registers 0-14 - Same as on entry
Register 15 - Return code

ENTRY POINT IECVXRES:

EXIT NORMAL:

Registers 0-14 - Same as on entry
Register 15 - Return code

ENTRY POINT IECVRCHN:

EXIT NORMAL:

Registers 0-15 - Same as on entry

ENTRY POINT IECVEXCL:

EXIT NORMAL:

Registers 0-15 - Same as on entry

IECVEXPR - EXCP Processor's Purge and Restore Routines

IECVEXPR

IECVEXPR controls and manages the EXCP processor purge and restore functions. Four entry points are provided for callers to interface with the EXCP processor for purge and restore functions.

IOS I/O Halt SVC module
IGC0003C

IECVXPUR

01 IECVXPUR entry points for SVC 16(10) and SVC33(21).

SVC16

02 Saves the caller's registers and establishes a purge base register.

PUR000

03 Obtains the local lock and uses the local lock save area as a routine save area pointer.

IOCOM

04 Obtains a large block from the IOS storage manager for use as a work area.

When requesting the block, uses ASID zero to prevent the IOS purge function from freeing the large block when it requests the storage manager to free all large blocks in the address space associated with the address space termination function.

On this entry to the storage manager, the large block is zeroed by the storage manager.

BASSM
REG14, REG15

IPIB

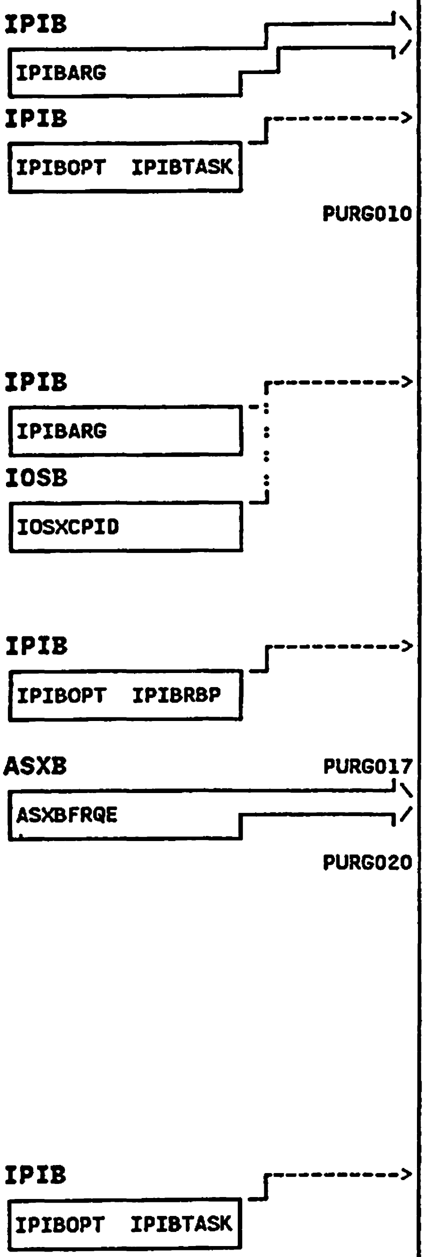
IPIBOPT IPIBMEM
IPIBHALT

IPIB

IPIBOPT IPIBRBP

05 For an address space quiesce, sets the search argument and search field to zeroes.

Register 0 is the search argument and WRKREGB is the search field.



06 For RB quiesce, sets the search argument and search field appropriately.

A. Sets register 0 (search argument) to the IPIB address.

B. If this is a TCB purge, sets the search field (WRKREGB) to the TCB address provided in the RQE (RQETCB).

C. Otherwise, this is a DEB purge. Does the following:

Sets the search field (WRKREGB) to the TCB address provided in the RQE (RQETCB).

D. Checks if the IPIBARG is the EXCP driver ID. For compatability, an IPIBARG of zero is also recognized as an EXCP driver request.

E. If neither, bypasses purge and prepares to return to the caller.

>PURG900: 12

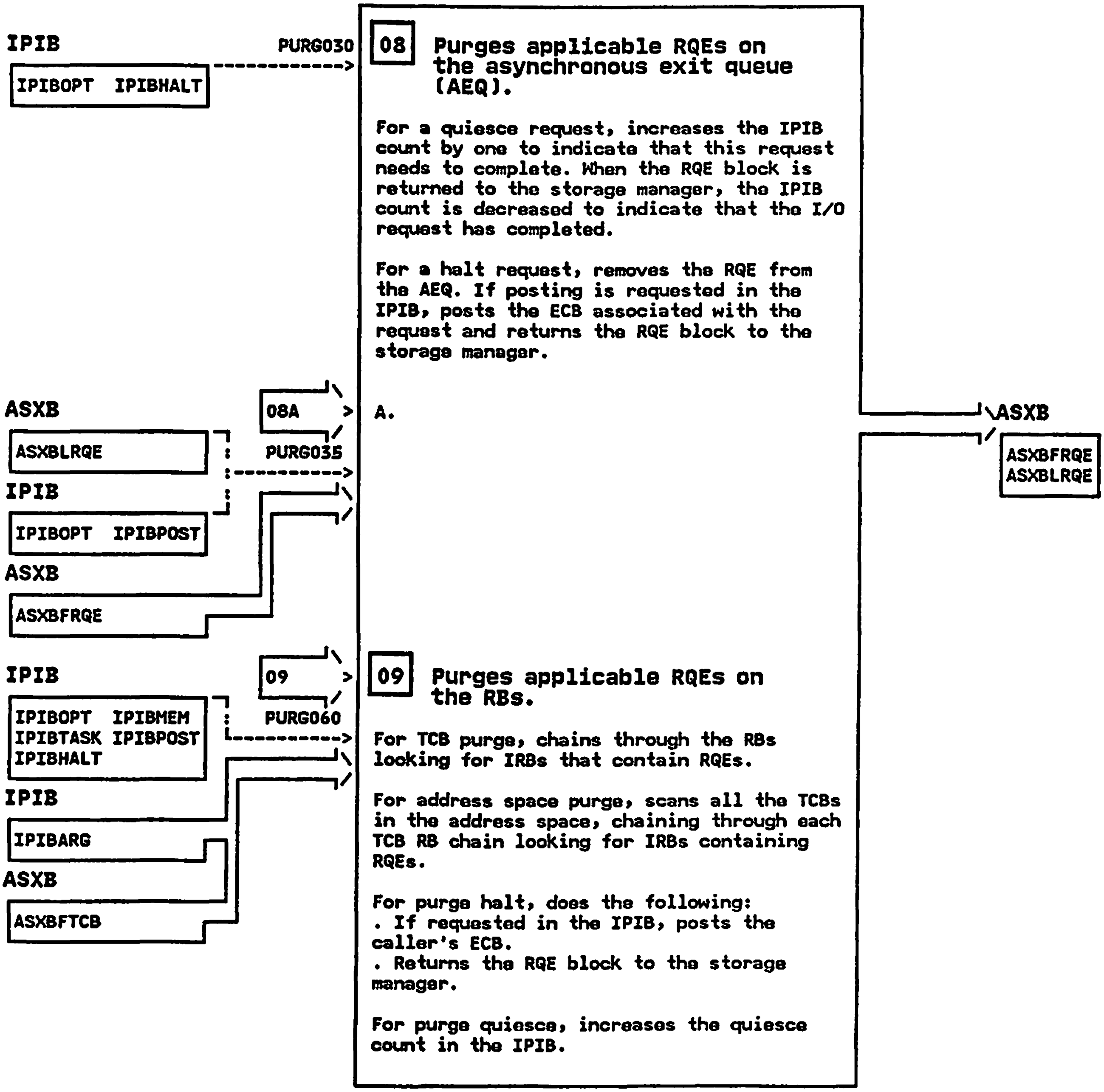
07 Checks if there are RQEs on the asynchronous exit queue (AEQ) to be processed.

A. If the ASCBFRQE field is zero, there are no more RQEs to process.

>PURG060: 09

B. Otherwise, continue searching the AEQ.

>PURG035: 08A



08 Purges applicable RQEs on the asynchronous exit queue (AEQ).

For a quiesce request, increases the IPIB count by one to indicate that this request needs to complete. When the RQE block is returned to the storage manager, the IPIB count is decreased to indicate that the I/O request has completed.

For a halt request, removes the RQE from the AEQ. If posting is requested in the IPIB, posts the ECB associated with the request and returns the RQE block to the storage manager.

A.

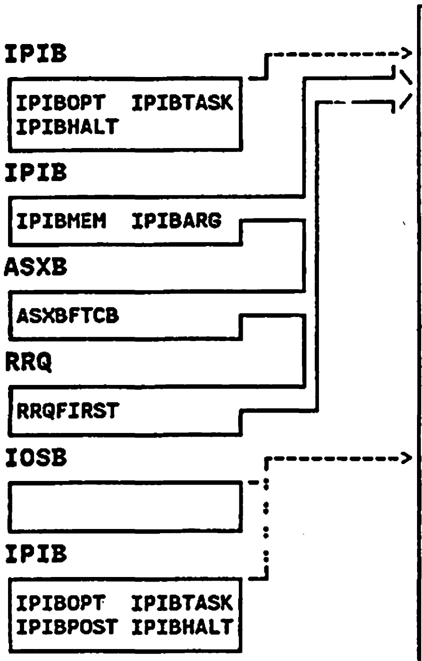
09 Purges applicable RQEs on the RBs.

For TCB purge, chains through the RBs looking for IRBs that contain RQEs.

For address space purge, scans all the TCBs in the address space, chaining through each TCB RB chain looking for IRBs containing RQEs.

For purge halt, does the following:
. If requested in the IPIB, posts the caller's ECB.
. Returns the RQE block to the storage manager.

For purge quiesce, increases the quiesce count in the IPIB.

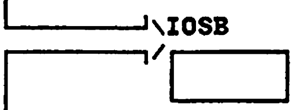


10 For DEB requests, purges applicable RQEs on the related request queue (RRQ).

For purge halt, removes all requests from the RRQ.

For purge quiesce, places all the IOBs on a restore chain.

If the SVC purge caller has identified requests that should not be purged, these requests are ignored in this routine.



A. Handles a purge quiesce for a related RQE that has been sent to IOS.

If EXCP had issued the STARTIO macro to initiate the related request, checks if IOS has initiated the I/O request, has set the IPIB field, and has accordingly increased the IPIB count field.

If IOS has set the IPIB field in the IOSB (IOSIPIB), there is no need to increase the count, but IECVXPUR needs to set the IPIB pointer in the RQEIPIB field of the RQE.

PURGCNT B. Increases the purge quiesce count in the IPIB.

The IPIB count represents the number of outstanding I/O requests that have to complete.

The IPIB address is set in field RQEIPIB of the RQE to associate it with the purge quiesce.

When the I/O request completes, the EXCP processor decreases the IPIB count just before returning the RQE block to the storage manager.

IPIB
IPIBOPT IPIBHALT

PURG170

11 For purge halt, purges all SRB/IOSBs on the IPIBSRB chain.

For each IOSB, does the following:
. Sets the IOSIPIB pointer value in the RQEIPIB field.
. Sets the associated ECB completion code to purge (X'48').
. If the post option was not specified in the IPIB, sets the RQENOPST flag to indicate no posting.
. Interfaces with IECVEXCP to purge the I/O request and return the RQE and large blocks to the storage manager.

PURG175 A. If all SRB/IOSBs have been examined, returns to the caller.

>PURG900: 12

IPIB

IPIBOPT	IPIBPOST

PURG185 B. Interfaces with EXCP.

PURGINTF: 15
RETURN REGISTER: REG14

IOCOM

12 >

PURG900

12 When purge processing is complete, does the following.

A. Returns the large block used as a work area to the IOS storage manager (IOSVSMGR).

BASSM
REG14, REG15

B. Issues the SETLOCK macro to release the local lock.

C. Restores the caller's registers and returns.

BSM
0, REG14

PURGPOST **13** Sets the ECB with a purge completion code (X'48') and interfaces with the system post routine (IEAOPT02).

If the caller is in a problem program key, the call to the system post routine requests a validity check of the ECB.

A. Returns to the caller.

PURGFREE

14 Interfaces with the IECVEXCP routine IECVX025 to return the RQE to the IOS storage manager (IOSVSMGR).

REGISTER REG15

RETURN REGISTER: REG14

A. Returns to the caller.

15 PURGINTF

15 Interfaces with the IECVEXCP purge routine IECVXTRM to purge the I/O request and to return the RQE and large blocks.

REGISTER REG15

RETURN REGISTER: REG14

A. Return to the caller.

PURGIOBV

16 Sets the purge completion code in the ECB.

For problem program callers, changes to the caller's key before setting the purge completion code in the ECB.

A. Returns to the caller.

From entry point IECVXPUR in this module or from IECVEXCP

IECVRCHN

17 **This entry point builds a restore chain for the following requests:**

A purge quiesce request from this module.

A related request purge (RRQ) from mainline IECVEXCP.

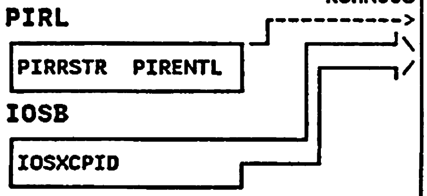
A request to add a request to the restore chain from the IECVEXCP abnormal-end exit routine.

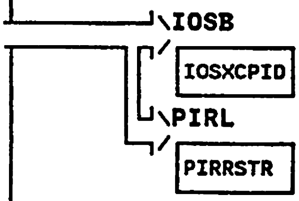
18 **Determines if an IOS PIRL block exists.**

A. If an IPIB was provided, establishes a pointer to the PIRL from the IPIBPIRL field of the IPIB.

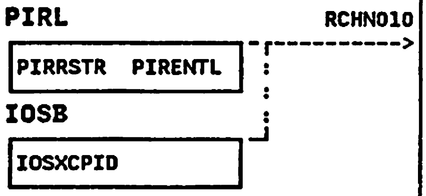
RCHN005 B. Otherwise, if provided in the DEB, establishes the PIRL pointer from the DEB.

RCHN008 C. If there is no PIRL, obtains storage from subpool 254 and initializes it as a PIRL.





RCHN010 19 **Establishes a pointer to the EXCP entry in the PIRL.**



20 **Establishes a pointer to the EXCP EPCB block in the PIRL EXCP entry.**

A. If the EXCP entry does contain an EPCB, goes to search for an available entry.

>RCHN015: 22

B. If the EXCP entry does not contain an EPCB, obtains an EPCB and initializes it.

RCHNGTMN: 30
RETURN REGISTER: WKREGC

PIRL

PIRRSTR

21 RCHN012

21 Establishes a pointer to the IOB chain field in the EXCP entry.

Sets register 0 to all ones (FF-FF).

Go to RCHN050 to place the IOB on the IOB restore chain and in the EPCB.

>RCHN050: 26

22 RCHN015

22 Searches the EPCB for an available entry for storing the IOB.

If there are no entries in this EPCB, checks to see if there is another EPCB chained to this one that has an available entry.

EPCB

EPCBNENT

22A RCHN020

A. If there is an available entry in this EPCB, continues.

>RCHN030: 24

EPCB

EPCBCHN

B. If there are no available entries in this EPCB and there is not another EPCB chained to this one, obtains an EPCB.

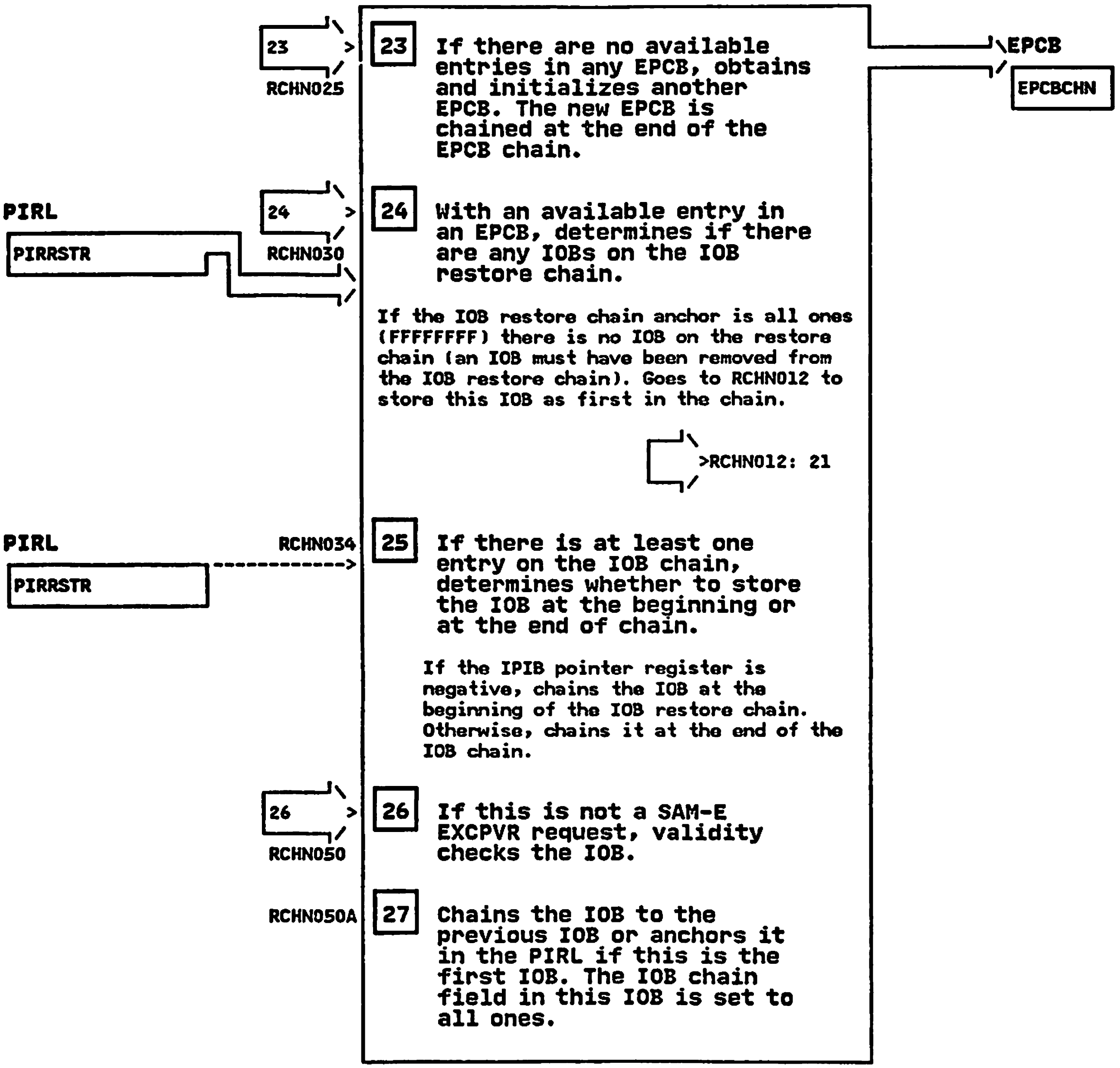
>RCHN025: 23

EPCB

EPCBCHN

C. If there is another EPCB chained to this one, checks to see if it has any available entries.

>RCHN020: 22A



23 If there are no available entries in any EPCB, obtains and initializes another EPCB. The new EPCB is chained at the end of the EPCB chain.

24 With an available entry in an EPCB, determines if there are any IOBs on the IOB restore chain.

If the IOB restore chain anchor is all ones (FFFFFFFF) there is no IOB on the restore chain (an IOB must have been removed from the IOB restore chain). Goes to RCHN012 to store this IOB as first in the chain.

25 If there is at least one entry on the IOB chain, determines whether to store the IOB at the beginning or at the end of chain.

If the IPIB pointer register is negative, chains the IOB at the beginning of the IOB restore chain. Otherwise, chains it at the end of the IOB chain.

26 If this is not a SAM-E EXCPVR request, validity checks the IOB.

27 Chains the IOB to the previous IOB or anchors it in the PIRL if this is the first IOB. The IOB chain field in this IOB is set to all ones.

IPIB
IPIBOPT

PIRL
PIRRSTR

28 Fills in an EPCB entry.

Each EPCB entry is 8 bytes in length and is filled in as follows:

Byte 0 - Contains the request key of the caller, moved from the RQEPRT field of the RQE. Bits 0-3 contain the caller's key and bits 4-7 are flags defined by EXCP.

Byte 1-3 - Contains the IOB address.

Byte 4 - Contains X'F4' if this is an EXCPVR request, otherwise zeroes.

Byte 5-7 - If an IPIB was provided and requests that the IOB be restored to the originating TCB, stores the restoring TCB address. Otherwise, with this field zero, restores the IOB under the restoring TCB.

With an available IPIB, the EPCB entry is stored in field IPIBIO of the IPIB.

EPCB
EPCBIOB
EPCBPKEY
EPCBTCB

EPCB
EPCBNENT

RCHN060

29 Performs clean-up and returns to caller.

Increases the count of EPCB entries by 1 and stores the address of the next available EPCB entry in field EPCBNENT.

Restores the caller's registers and returns to the caller via register 14.

EPCB
EPCBNENT

EPCB
EPCBIOB EPCBBL

30
RCHNGTMN

30 Obtains and initializes an EPCB block.

Obtains an EPCB from subpool 230 associated with the job step TCB.

Sets to zero the EPCB block and initializes the following fields:

. EPCBRTCB contains the address of the TCB used in the GETMAIN (used when freeing the EPCB).

. EPCBENT contains a pointer to the first EPCB.

Sets the number of available entries to the maximum.

EPCB
EPCBNENT

A. Returns to the main routine.

IOSB

SVC33

31 Interfaces with SVC33 for BTAM to terminate its active channel program.

Changes an active real channel program CCW operation code to a NOP operation code, so that the channel program will go to completion.

A. Returns to SVC 33 processing.

IOS I/O Restore SVC module
IGC0001G

IECVXRES

32 Restore I/O request (IOBs) subroutine.

This entry point receives control from the IOS SVC 17 function (IGC0001G) for the EXCP processor to restore I/O requests that are chained on the IOB restore chain.

33 Saves the caller's registers, establishes addressability, and establishes a pointer to the first IOB on the restore chain.

34 If the pointer to the first IOB on the restore chain is zero, there are no IOBs to be restored. Goes to free associated EXCP EPCB blocks.

>REST903: 37

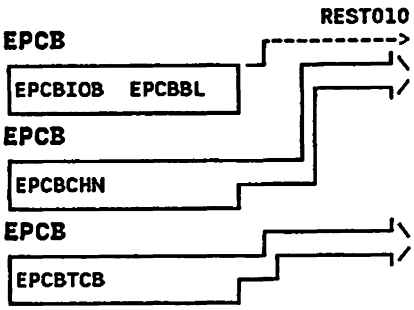
35
REST000

35 Otherwise, removes the IOB from the IOB restore chain and prepares to restore it.

Saves the next IOB on the IOB restore chain in the EXCP PIRL area.

IECVEXPR - EXCP Processor's Purge and Restore Routines

STEP 35A



REST010

A. Searches the EPCB block for a matching IOB address.

If a matching IOB is not found on the EPCB chain, ignores the IOB on the IOB restore chain and continues processing at REST080 for the next IOB on the restore chain.

B. If there is a matching EPCB IOB entry, performs the following:

Sets register 0 to the restoring TCB or zeros.

Issues a MODESET macro to get into the user's key.

If this is a virtual EXCP request, issues SVC 0 if there was no associated TCB indicated in the EPCB entry.

If this is an EXCPVR request, issues SVC 114 if there was no associated TCB indicated in the EPCB entry. Otherwise, issues SVC 92.

REST060

C. After issuing the appropriate SVC, issues MODESET to return to Key 0.

D. Determines if the ECB has been waited on.

If the ECB associated with the IOB is zero, the ECB has not been waited on. Goes to handle the next IOB.

Otherwise, obtains the local lock, sets the ECB to zero, then releases the local lock.

E. Gets the next IOB to be restored.

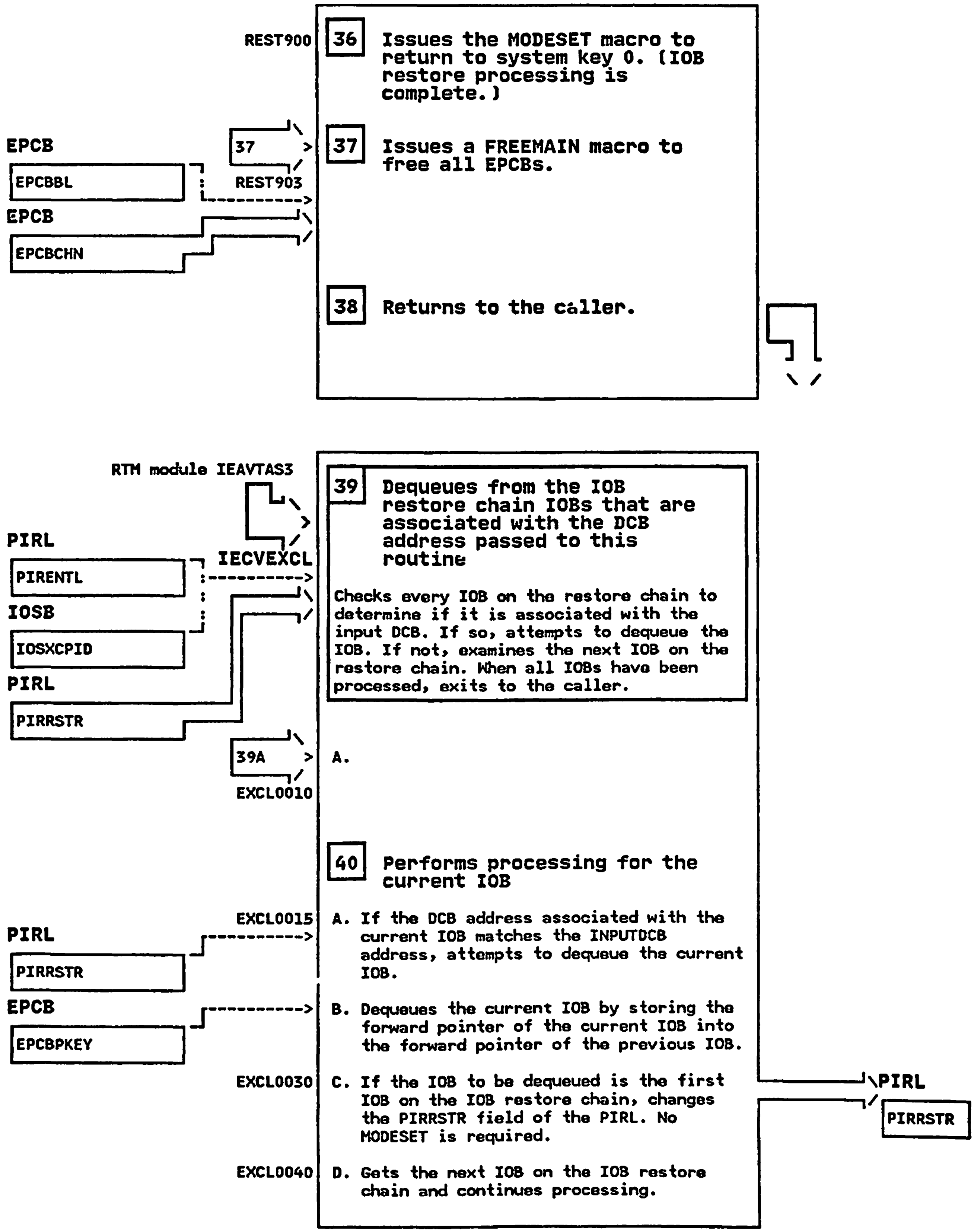
>REST080: 35F

35F >
REST080

F. Prepares to handle the next IOB on the restore chain.

>REST000: 35

IECVEXPR - EXCP Processor's Purge and Restore Routines STEP 36



36 Issues the MODESET macro to return to system key 0. (IOB restore processing is complete.)

37 Issues a FREEMAIN macro to free all EPCBs.

38 Returns to the caller.

39 Dequeues from the IOB restore chain IOBs that are associated with the DCB address passed to this routine

Checks every IOB on the restore chain to determine if it is associated with the input DCB. If so, attempts to dequeue the IOB. If not, examines the next IOB on the restore chain. When all IOBs have been processed, exits to the caller.

A.

40 Performs processing for the current IOB

A. If the DCB address associated with the current IOB matches the INPUTDCB address, attempts to dequeue the current IOB.

B. Dequeues the current IOB by storing the forward pointer of the current IOB into the forward pointer of the previous IOB.

C. If the IOB to be dequeued is the first IOB on the IOB restore chain, changes the PIRRSTR field of the PIRL. No MODESET is required.

D. Gets the next IOB on the IOB restore chain and continues processing.

>EXCL0010: 39A

41 Returns to the caller.

BSM
0, REG14

EXCL0060

42 Loops through the EPCB chain until an IOB is found that matches the IOB address in HOLDIOBR or until all EPCBs have been examined.

EPCB

EPCBIOB

42A

EXCL0070

A.

EXCL0080

EPCB

EPCBIOB

B. Loops through all the entries in the current EPCB block and, if any entry contains an IOB address that matches the passed IOB address, returns to the caller of this subroutine.

REGISTER
>REG1

EXCL0090

EPCB

EPCBCHN

C. If the passed IOB address is not equal to the IOB address of any of the entries in the current EPCB, updates the current EPCB pointer to point to the next EPCB on the chain.

D. Loops through the next EPCB on the chain.

>EXCL0070: 42A

CHANNEL COMMAND WORD (CCW) TRANSLATION OPERATION TABLE MODULES

Each of the following modules contains a CCW translation operation table:

- IECVOPTB - 3704/3705 communication device
- IECVOPTC - Teleprocessing class devices
- IECVOPTD - DASD class devices
- IECVOPTE - 3211 printer device
- IECVOPTG - Graphic class devices
- IECVOPTH - 3890 MICR device
- IECVOPTI - 3886 OCR device
- IECVOPTJ - 3895 printer
- IECVOPTK - 1287/1288 optical reader
- IECVOPTL - 3851 MSS controller
- IECVOPTM - 3540 diskette
- IECVOPTN - 3838 VPSS
- IECVOPTT - Tape class devices
- IECVOPTU - Unit record class devices

Channel command word (CCW) translation operation tables indicate how a device command code is translated. These tables include device classes (TAPE, TP, DASD, GRAPHICS, and unit record) and devices that have unique command codes that deviate from the standard five device class operation tables.

The UCB for a device points to the device dependent table (DDT) for that device. The DDT entry for that device contains the address of the CCW translation operation table for that device. The CCW translator adds the value of the command byte (in the channel command word) to the address of the CCW translation operation table to reach the correct entry.

Each translation operation table has 256 entries, one for each channel command word; each entry is one byte long.

The following bits are defined in each byte of a CCW translation operation table:

X'80' - The CCW provides status modifier (SM) support.
X'40' - The CCW is a non-data transfer type command. (No data areas to be fixed)
X'20' - IECVTCCW uses this byte, which is always ser, to indicate a NO-OP TIC command. IECVTCCW copies this byte into byte 5 of each CCW to be translated and sets a bit in byte 5 to indicate a NO-OP TIC command.
X'10' - Reserved
X'08' - Reserved
X'04' - Reserved
X'02' - Reserved
X'01' - Reserved

IECVTCCW - CCW TRANSLATOR

DESCRIPTIVE NAME: CCW Translator

FUNCTION:

This module performs four options involved with the translation of a caller's virtual channel program to a real channel program.
These options include:

- Translating the caller's virtual channel program to a real channel program and fixing the program's data areas
- Unfixing the program's data areas
- Translating a virtual real channel address to its corresponding virtual channel program address
- Translating a virtual channel program address to its real channel program address

Another option allows IECVTCCW to request from the caller an additional large block when more storage is required to translate the caller's virtual channel program to a real channel program.

ENTRY POINT: IECVTCCW

PURPOSE:

Performs the translate function specified in the TCCW control block. The TCCWOPTN byte is used with a branch table to select the routine that will handle the caller's option. (While these routines are not defined as entry points, they are documented as such.)

The branch vector table is as follows:

TCCWOPTN	Routine	Function
X'00'	TCCWI100	CCW translation
X'04'	TCCWR000	Address re-translation
X'08'	TCCWU100	Unfix caller's data areas
X'0C'	TCCWG000	Inform IECVTCCW that an additional block was obtained
X'10'	TCCWX000	Single address translation

LINKAGE: BALR

CALLERS:

IECVEXCP and others who require virtual channel program translation.

INPUT:

TCCW control block:
TCCWOPTN byte - Function to be performed
TCCWBEB - Address of a BEB block
TCCWFIX - Address of FIX list block
(See the appropriate entry point in this module for specific input requirements.)

Register 0 - For TCCW option code X'0C', the address of the requested large block.

OUTPUT: None

EXIT NORMAL: Return to caller.

EXIT ERROR:

To RTM (IECVTCCW does not establish a functional recovery routine)

ENTRY POINT: TCCWI100

IECVTCCW - MODULE DESCRIPTION (Continued)

PURPOSE:
To translate a caller's virtual channel program to a real channel program and to fix all the channel program's data areas (TCCWOPTN=X'00').

LINKAGE: BALR

CALLERS: IECVEXCP and others who require this function

INPUT:

```
TCCW control block:
TCCWOPTN byte - Option code X'00'
TCCWFVC  - Address of the virtual channel program
TCCWBEB  - Address of a BEB block
TCCWFIX  - Address of FIX list block
TCCWUCB  - Address of the UCB
TCCWMODB - The TCCWLBLK bit is on when the caller
           provided 248-byte blocks for translation.
           The rest of the byte must be zero.
```

OUTPUT:
The virtual channel program has been translated, and data areas fixed in storage. Control blocks initialized or created include the BEB, FIX list, and the IDAL.

EXIT NORMAL: Return to caller.

EXIT ERROR:
To RTM (IECVTCCW does not establish a functional recovery routine).

ENTRY POINT: TCCWR000

PURPOSE:
To re-translate a virtual address in the real channel program to its corresponding virtual address in the caller's virtual channel program (TCCWOPTN=X'04'). Typical use is to translate the virtual channel status word (CSW) address.

LINKAGE: BALR

CALLERS: IECVEXCP and others who require this function

INPUT:
TCCW control block with the same fields as for entry point TCCWI100.
TCCWOPTN - Option code X'04'.

OUTPUT: The virtual address of the virtual channel program.

EXIT NORMAL: Return to caller.

EXIT ERROR:
To RTM (IECVTCCW does not establish a functional recovery routine).

ENTRY POINT: TCCWU100

PURPOSE:
To unfix the data storage associated with the virtual channel program and to provide a free chain of all the blocks used in the translation of the virtual channel program (TCCWOPTN=X'08').

LINKAGE: BALR

CALLERS: IECVEXCP and others who require this function

INPUT:
TCCW control block with the same fields as for entry point TCCWI100
TCCWOPTN byte - Option code X'08'.

OUTPUT:
The virtual channel program data areas have been unfixed and a free chain of the blocks has been built.

EXIT NORMAL: Return to caller.

EXIT ERROR:
To RTM (IECVTCCW does not establish a functional recovery routine).

ENTRY POINT: TCCWG000

PURPOSE:
To inform IECVTCCW that the caller has provided another large block of storage so that translation can continue (TCCWOPTN=X'0C'). (During its processing, IECVTCCW might discover that it needs more storage. In this case, IECVTCCW returns to the caller with a return code indicating that the caller should obtain more storage and pass it back to IECVTCCW.)

LINKAGE: BALR

CALLERS:
IECVEXCP and others who require virtual channel program translation.

INPUT:
TCCW control block with the same fields as for entry point TCCWI100
TCCWOPTN byte - Option code X'0C'.

Register 1 - Address of a large block. If the flag TCCWLBLK is set in the TCCW control block, the caller must provide a 248-byte block.

OUTPUT: None

EXIT NORMAL:
Not applicable. Processing continues with the entry point that discovered the need for more storage.

ENTRY POINT: TCCWX000

PURPOSE:
To translate a virtual channel program address in the caller's virtual channel program to its corressponding virtual address in the real channel program (TCCWOPTN=X'10').

LINKAGE: BALR

CALLERS: IECVEXCP and others who require this function

INPUT:
TCCW control block with the same fields as for entry point TCCWI100

IECVTCCW - MODULE DESCRIPTION (Continued)

TCCWOPTN byte - Option code X'10'.

Register 0 - The virtual address of the virtual channel program to be translated.

OUTPUT:
The virtual channel program address corresponding to a virtual address in the caller's virtual channel program.

EXIT NORMAL: Return to caller.

EXIT ERROR:
To RTM (IECVTCCW does not establish a functional recovery routine).

ENTRY POINT: IECVTCFR

PURPOSE:
This IECVTCCW retry routine validity checks errors that occur during the translation of the caller's virtual channel program. IECVTCCW sets a flag in the TCCW control block (TCCWPGCK) before performing the validity check or issuing the call to the PGSER services for fixing and unfixing pages. The EXCP processor functional recovery routine, upon finding this bit set, will set this entry point as its retry routine.

LINKAGE: BALR

CALLERS: RTM

INPUT: TCCW control block

OUTPUT: None

EXIT NORMAL:
Returns to mainline IECVTCCW processing to continue processing or to terminate processing the caller's request

EXTERNAL REFERENCES:

ROUTINES: None

CONTROL BLOCKS:

ASCB	- Address space control block
ASXB	- Address space extension block
BEB	- Beginning-end block
CVT	- Communications vector table
DDT	- Device descriptor table
FIX	- Page fix list
IDAL	- Indirect address list
PSA	- Prefixed save area
PVT	- Paging vector table
RB	- Request block
TCB	- Task control block
TCCW	- TCCW control block
UCB	- Unit control block
WSAVT	- Work save area vector table

TABLES:

Translation operation tables. The specific table address is contained in the device descriptor table (DDT). See IECVTOBL's module operation for more information about the operation tables.

IECVTCCW - MODULE OPERATION

This module performs five options involved with the translation of a caller's virtual channel program to a real channel program.
The operations performed by IECVTCCW depend on the option code specified by the caller.in the TCCW option field (TCCWOPTN):

TCCWOPTN (decimal) Function

0 - Translates a virtual channel program to a real channel program and fixes the data area storage. This module will support a 31-bit virtual storage interface through virtual IDAWs.
This module will also support the fixing of virtual I/O buffers above 16 megabytes real.

4 - Retranslates a real channel program address to its corresponding virtual channel program address.

8 - Unfixes the data area storage and creates a free chain of the blocks used in the translation.

12 - After IECVTCCW informs the caller that another large block of storage is required for translating the channel program (via return code), the caller specifies this option to indicate that a large block has been provided; IECVTCCW processing continues.

16 - Translates a single virtual channel program address to its corresponding virtual address in the real channel program.

IECVTCCW - DIAGNOSTIC AIDS

ENTRY POINT NAMES: IECVTCCW
TCCWI100
TCCWR000
TCCWU100
TCCWG000
TCCWX000
IECVTCFR

MESSAGES: None

ABEND CODES: None

WAIT STATE CODES: None

RETURN CODES:

ENTRY POINT IECVTCCW:

EXIT NORMAL:

Register 15 contains one of the following decimal values:
0 - Translation option completed successfully.
4 - Translation option completed unsuccessfully (See Return Codes under individual entry points for the meaning of this return code.)
12 - Caller should obtain a block of storage and pass it back to this module. The block must be at least 160 bytes. Up to 248 bytes is allowed by this module.

ENTRY POINT TCCWI100:

EXIT NORMAL:

Register 15 contains one of the following decimal values:
0 - Translation option completed successfully.
4 - Translation option completed unsuccessfully as a result of a translation error or validity check error. The TCCWOPTN byte has been set to one of the following values:
X'80' - Page fix error
X'E0' - Validity check error
12 - Caller should obtain a block of storage and pass it back to this module. The block must be at least 160 bytes. Up to 248 bytes is allowed by this module.

ENTRY POINT TCCWR000:

EXIT NORMAL:

Register 15 contains one of the following decimal values:
0 - Translation option completed successfully.
4 - Translation option completed unsuccessfully.
The virtual address of the real channel program was not found in the BEB or the requestor is not in a system key.
The TCCWOPTN byte has been set to X'E0' if the requestor is not in a system key.

ENTRY POINT TCCWU100:

EXIT NORMAL:

Register 15 contains one of the following decimal values:
4 - Translation option completed unsuccessfully.
The TCCWOPTN byte has been set as follows:
X'E0' - The requestor was not in a system key or a validity check error occurred.
8 - Translation option completed successfully.
Register 1 contains the address of the first block on the free block chain.

ENTRY POINT TCCWG000: None

ENTRY POINT TCCWX000:

EXIT NORMAL:

Register 15 contains one of the following decimal values:
4 - Translation option completed unsuccessfully.
The requestor was not in system key.
The TCCWOPTN byte has been set as follows:
X'90' - Translation unsuccessful.
X'E0' - Caller not in a system key.
8 - Translation option completed successfully.
Register 0 contains the virtual address in the real channel program corressponding to the virtual address in the caller's virtual channel program.

ENTRY POINT IECVTCFR: None

REGISTER CONTENTS ON ENTRY:

ENTRY POINT IECVTCCW:

Register 0 - Address of the large block provided by the caller (TCCWOPTN=X'0C'); otherwise, irrelevant
Register 1 - TCCW control block address
Registers 2-13 - Irrelevant
Register 14 - Return address
Register 15 - Entry point address

ENTRY POINT TCCWI100:

Register 0 - Irrelevant
Register 1 - TCCW control block address
Registers 2-13 - Irrelevant
Register 14 - Return address
Register 15 - Entry point address

ENTRY POINT TCCWR000:

Register 0 - A virtual address within the real channel program
Register 1 - TCCW control block address
Registrs 2-13 - Irrelevant
Register 14 - Return address
Register 15 - Entry point address

ENTRY POINT TCCWU100:

Register 0 - Irrelevant
Register 1 - TCCW control block address
Registers 2-13 - Irrelevant
Register 14 - Return address
Register 15 - Entry point address

IECVTCCW - DIAGNOSTIC AIDS (Continued)

ENTRY POINT TCCWG000:

Register 0 - Address of the large block
Register 1 - TCCW control block address
Register 2-13 - Irrelevant
Register 14 - Return address
Register 15 - Entry point address

ENTRY POINT TCCWX000:

Register 0 - The virtual address of the virtual channel program to be translated.
Register 1 - TCCW control block address
Registers 2-13 - Irrelevant
Register 14 - Return address
Register 15 - Entry point address

ENTRY POINT IECVTCFR:

Registers 0-5 - Irrelevant
Register 6 - Caller's return code as set in register 15 on entry to the functional recovery routine
Register 7-9 - Irrelevant
Register 10 - On an ABEND 18A with a return code 4, the address of the first invalid page
Register 11 - Address of the TCCW control block
Register 12-14 - Irrelevant
Register 15 - Address of the IECVTCCW retry routine

REGISTER CONTENTS ON EXIT:

ENTRY POINT IECVTCCW:

EXIT NORMAL:

Register 0 - Real address of the first real CCW
Registers 1-14 - Restored to contents on entry
Register 15 - Return code

ENTRY POINT TCCWI100:

EXIT NORMAL:

Register 0 - Real address of first real CCW
Registers 1-14 - Restored to contents on entry
Register 15 - Return code

ENTRY POINT TCCWR000:

EXIT NORMAL:

Register 0 - A virtual address within the virtual channel program.
Registers 1-14 - Restored to contents on entry
Register 15 - Return code

ENTRY POINT TCCWU100:

EXIT NORMAL:

Register 0 - Unpredictable
Register 1 - Address of the first block on the free block chain
Registers 2-14 - Restored to contents on entry
Register 15 - Return code

IECVTCCW - DIAGNOSTIC AIDS (Continued)

ENTRY POINT TCCWG000: Irrelevant

ENTRY POINT TCCWX000:

EXIT NORMAL:

Register 0 - A virtual address in the real channel program corressponding to a virtual address in the caller's virtual channel program
Register 1 - TCCW control block address
Registers 2-14 - Restored to contents on entry
Register 15 - Return code

ENTRY POINT IECVTCFR: Irrelevant

IECVTCCW - CCW Translator

STEP 01

IECVEXCP and others who require virtual channel program translation.

IECVTCCW

This module performs four options involved with the translation of a caller's virtual channel program to a real channel program. These options include: o Translating the caller's virtual channel program to a real channel program and fixing the program's data areas o Unfixing the program's data areas o Translating a virtual real channel address to its corresponding virtual channel program address o Translating a virtual channel program address to its real channel program address

Another option allows IECVTCCW to request from the caller an additional large block when more storage is required to translate the caller's virtual channel program to a real channel program.

IECVTCCW

01 Performs module initialization.

The caller provides a pointer to the TCCW control block in register 1. The TCCW control block option byte indicates the function to be performed.

A. Saves the caller's registers in the TCCW control block.

TCCW
TCCWRGSV

B. Establishes a pointer to the TCCW control block.

TCCW
TCCWOPTN

C. Determines if the caller is in a system key.

Issues an IPK instruction and, if the caller is not in a system key, returns to the caller with a return code of 4.

TCCWI000

D. Determines which function is to be performed by branching on the option byte.

E. For option code X'00', translates a CCW.

>TCCWI100: 03

F. For option code X'04', retranslates an address.

>TCCWR000: 20

G. For option code X'08', unfixes the caller's data areas.

>TCCWU100: 22

H. For option code X'0C', continues processing with additional storage.

>TCCWG000: 02

I. For option code X'10', translates a single address.

>TCCWX000: 24

IECVEXCP and others who require virtual channel program translation.

TCCWG000

02 Handles the second entry from the caller when IECVTCCW requested another large block to be used as a BEB, FIX or IDAL block (option code X'0C').

The address of the block provided by the caller is in register 0.

A. Zeros the first four words of the obtained block.

B. Restores registers that were saved when IECVTCCW returned to the caller for the additional block.

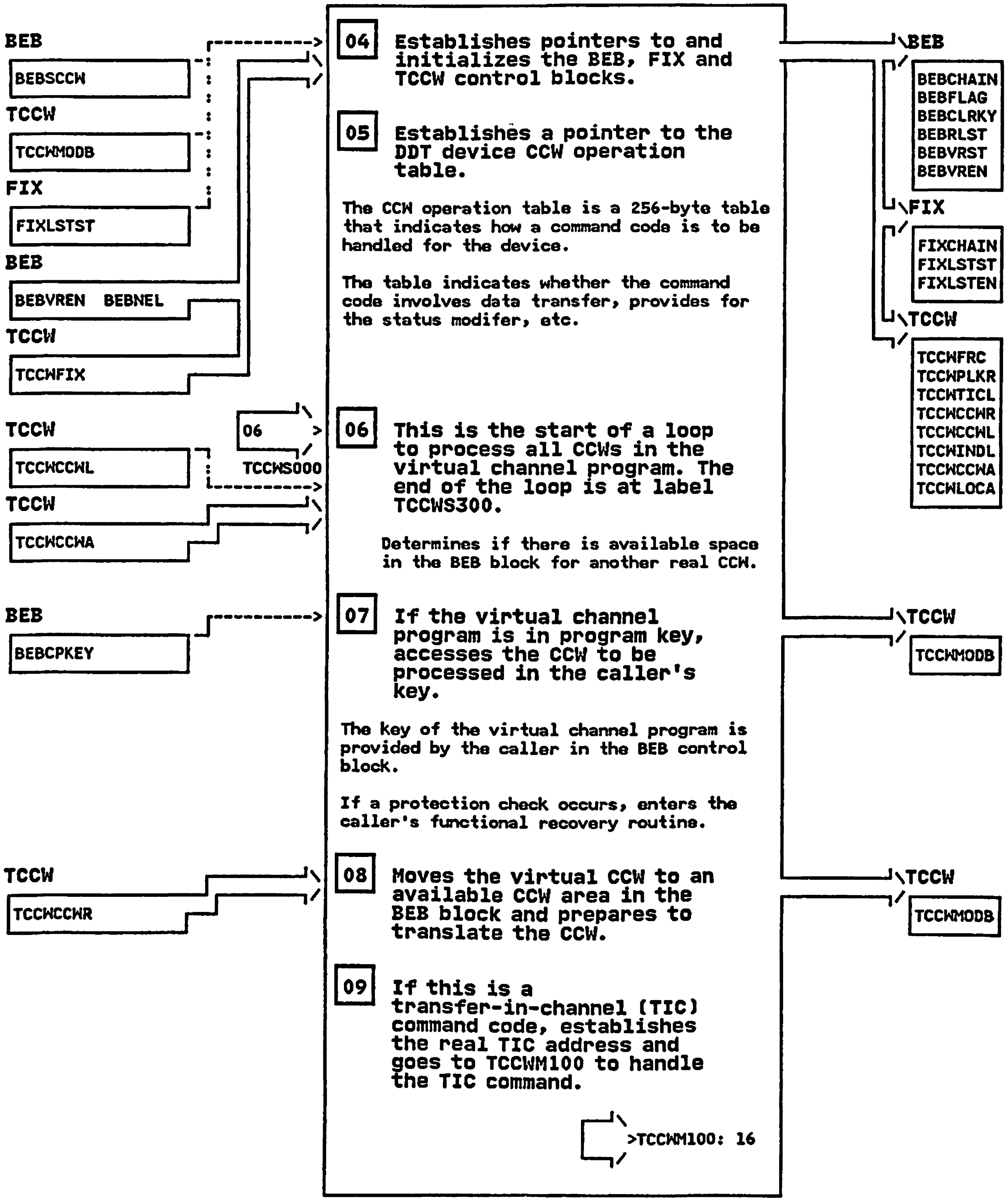
C. Returns to the routine that requested the additional block.

IECVEXCP and others who require this function

TCCWI100

03 Handles the caller's request to translate a virtual channel program to a real channel program (option code X'00').

This routine translates the CCW data addresses to real addresses and creates a real CCW string from the virtual string. It determines for each CCW whether the data pages need to be fixed and whether an indirect address word list (IDAL) is necessary.



BEB
BEBSCCW
TCCW
TCCWMODB
FIX
FIXLSTST
BEB
BEBVREN BEBNEL
TCCW
TCCWFIX

04 Establishes pointers to and initializes the BEB, FIX and TCCW control blocks.

BEB
BEBCHAIN
BEBFLAG
BEBCLRKY
BEBRLST
BEBVRST
BEBVREN
FIX
FIXCHAIN
FIXLSTST
FIXLSTEN
TCCW
TCCWFRC
TCCWPLKR
TCCWTICL
TCCWCCWR
TCCWCCWL
TCCWINDL
TCCWCCWA
TCCWLOCA

05 Establishes a pointer to the DDT device CCW operation table.

The CCW operation table is a 256-byte table that indicates how a command code is to be handled for the device.

The table indicates whether the command code involves data transfer, provides for the status modifer, etc.

TCCW
TCCWCCWL
TCCW
TCCWCCWA

06
TCCWS000

06 This is the start of a loop to process all CCWs in the virtual channel program. The end of the loop is at label TCCWS300.

Determines if there is available space in the BEB block for another real CCW.

BEB
BEBCPKEY

07 If the virtual channel program is in program key, accesses the CCW to be processed in the caller's key.

The key of the virtual channel program is provided by the caller in the BEB control block.

If a protection check occurs, enters the caller's functional recovery routine.

TCCW
TCCWMODB

TCCW
TCCWCCWR

08 Moves the virtual CCW to an available CCW area in the BEB block and prepares to translate the CCW.

TCCW
TCCWMODB

09 If this is a transfer-in-channel (TIC) command code, establishes the real TIC address and goes to TCCWM100 to handle the TIC command.

>TCCWM100: 16

TCCW

TCCWOPBT

TCCWS020

10 Obtains the command code byte from the CCW operation table and stores it in byte 5 of the real CCW.

If the previous CCW indicated data chaining, stores the previous command code saved in field TCCWOPBT as the command code for this CCW.

11 If this is a non-data transfer type command code, zeroes the data address field and ensures that the IDA bit is off.

A. Performs setup for the next virtual CCW. (Processing is complete for this CCW.)

>TCCWS260: 14

TCCWS080

12 For a non-write data transfer type command code with the skip bit set in the CCW, ensures that the IDA bit is set off.

A. Performs setup for the next virtual CCW. (Processing is complete for this CCW.)

>TCCWS260: 14

TCCWS100

13 For a data transfer CCW, establishes a pointer to the data address.

A. If the IDA bit is set in the virtual CCW, loads the address contained in the IDAW.

BEB

BEBCPKEY

B. If the channel program is in a user's key, enters the caller's key to load the data address.

TCCW

TCCWMODB

TCCWS120

C. Determines the virtual boundaries of the data area.

If this is a read backward CCW, ensures that the stop address is lower than the start address.

TCCW

TCCWLOCA

TCCWS160

D. Determines if this data area range is contained in the previous data area argument as saved in fields TCCWLOCA and TCCWHICA of the TCCW control block.

If this data area is not contained within the previous fixed area, goes to TCCWM000 to determine if this data area is contained within a previous fix list entry. If so, returns to continue CCW processing. If not contained within an existing fix list, builds a fix list to fix the data area. The page fix routine returns here (to TCCWS200) to continue CCW processing.

>TCCWM000: 15

13E

TCCWS200

E.

F. Determines if an IDAL entry is required.

If this data area crosses a page boundary, enters TCCWM400 to build IDAWs. Upon completion of building IDALs, processing is complete for this CCW and the IDAL routine goes to TCCWS260 to perform setup for the next CCW.

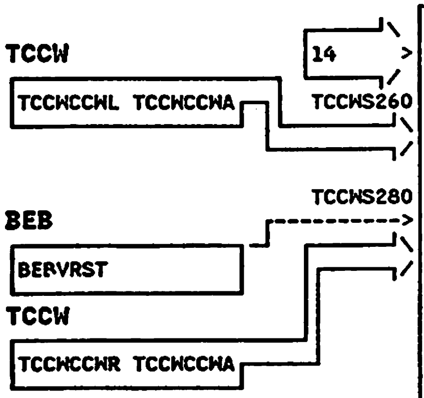
>TCCWM400: 19

G. If the IDA bit is set in the CCW, enters 31-bit mode to issue an LRA instruction on the data address. Otherwise, issues an LRA instruction in 24-bit mode.

If the real address is above the line, goes to the IDAL build routine at TCCWM402 to build IDAWS for the data above the 16-megabyte line.

Otherwise, stores the 24-bit real data address in the real CCW data address field.

>TCCWM402: 19B



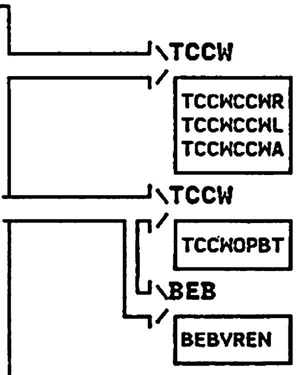
14 Prepares to translate the next virtual CCW. (Processing is complete for this CCW.)

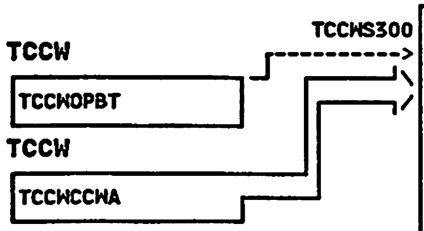
A. If this CCW does not indicate command or data chaining, determines if CCW translation is complete:

If a previous CCW exists in this BEB segment and provides for status modifier support, does the following:
. If the pevious CCW was a TIC CCW, goes to TCCWM200 to check for any unresolved TICs.
. Otherwise, treats the CCW as a chained CCW.

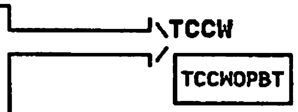
If a previous CCW does not exist in the BEB segment, goes to TCCWM200 to check for any unresolved TICs. (CCW translation is complete.)

>TCCWM200: 17



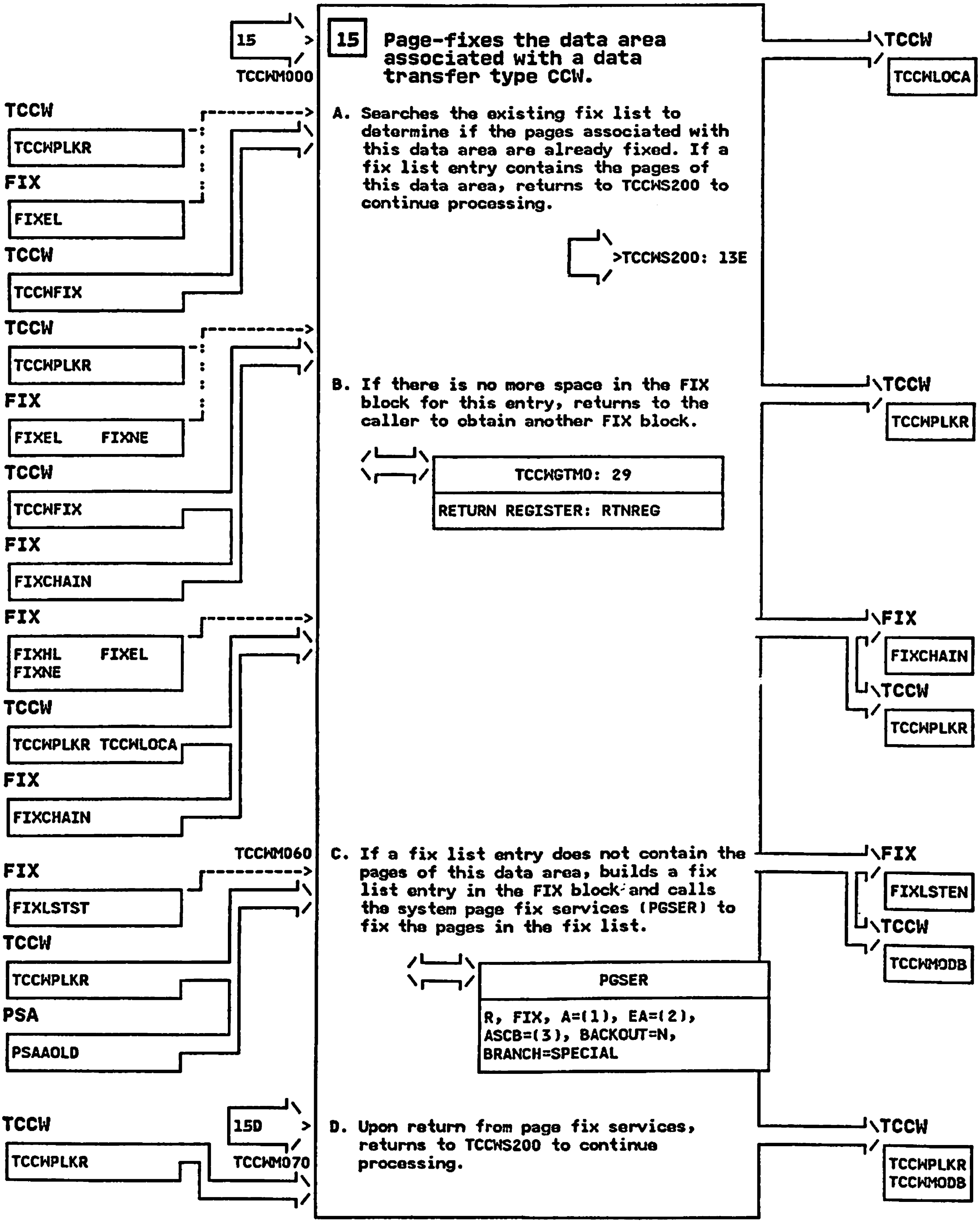


B. Prepares to handle the next virtual CCW.



C. Returns to translate the next virtual CCW.

>TCCWS000: 06



15 Page-fixes the data area associated with a data transfer type CCW.

A. Searches the existing fix list to determine if the pages associated with this data area are already fixed. If a fix list entry contains the pages of this data area, returns to TCCWS200 to continue processing.

>TCCWS200: 13E

B. If there is no more space in the FIX block for this entry, returns to the caller to obtain another FIX block.

TCCWGTMO: 29

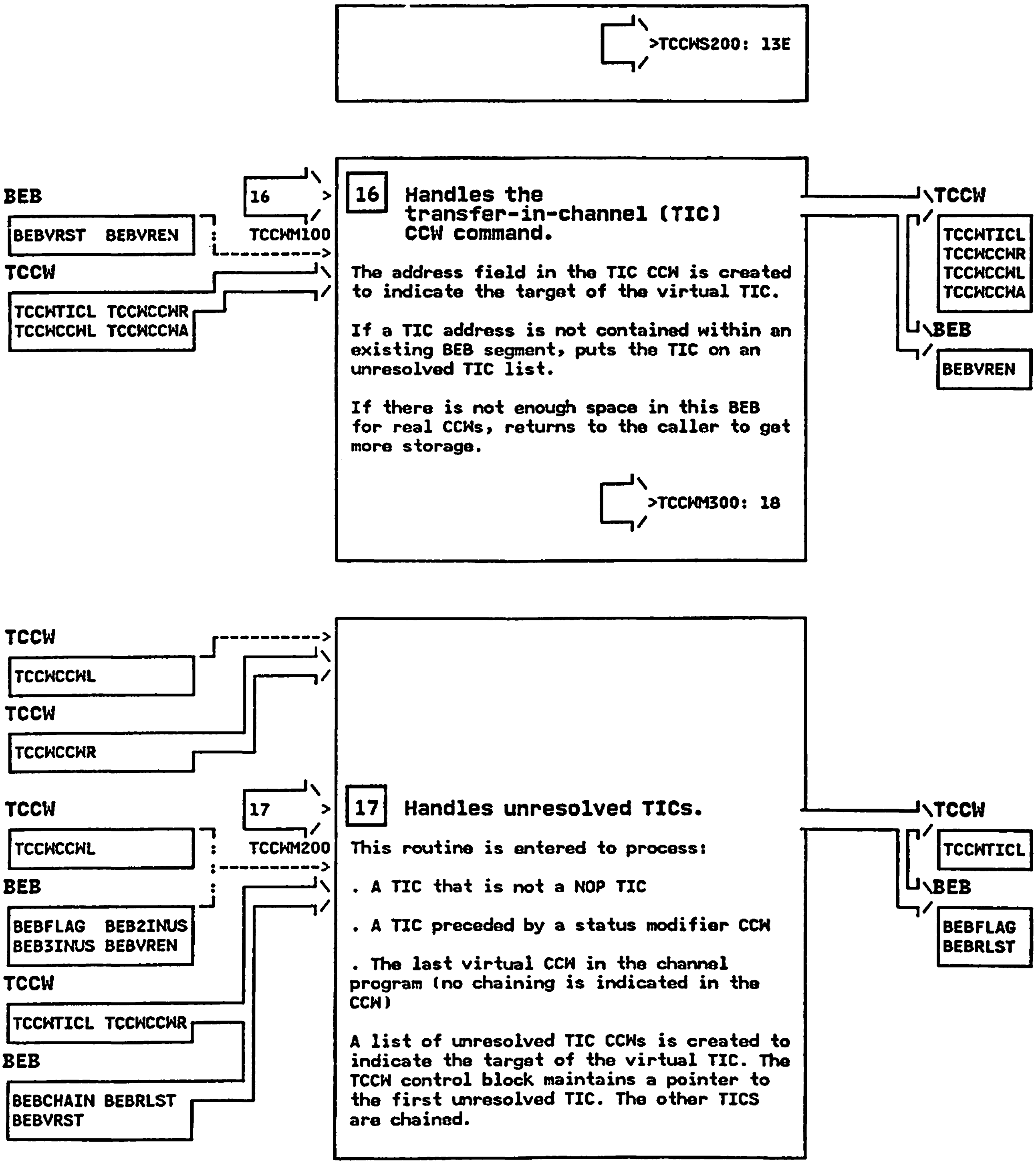
RETURN REGISTER: RTNREG

C. If a fix list entry does not contain the pages of this data area, builds a fix list entry in the FIX block and calls the system page fix services (PGSER) to fix the pages in the fix list.

PGSER

R, FIX, A=(1), EA=(2), ASCB=(3), BACKOUT=N, BRANCH=SPECIAL

D. Upon return from page fix services, returns to TCCWS200 to continue processing.



>TCCWS200: 13E

BEB

BEBVRST BEBVREN

TCCW

TCCWTICL TCCWCCWR
TCCWCCWL TCCWCCWA

16 >
TCCWM100

16 Handles the transfer-in-channel (TIC) CCW command.

The address field in the TIC CCW is created to indicate the target of the virtual TIC.

If a TIC address is not contained within an existing BEB segment, puts the TIC on an unresolved TIC list.

If there is not enough space in this BEB for real CCWs, returns to the caller to get more storage.

>TCCWM300: 18

TCCW

TCCWTICL
TCCWCCWR
TCCWCCWL
TCCWCCWA

BEB

BEBVREN

TCCW

TCCWCCWL

TCCW

TCCWCCWR

TCCW

TCCWCCWL

BEB

BEBFLAG BEB2INUS
BEB3INUS BEBVREN

TCCW

TCCWTICL TCCWCCWR

BEB

BEBCHAIN BEBRLST
BEBVRST

17 >
TCCWM200

17 Handles unresolved TICs.

This routine is entered to process:

. A TIC that is not a NOP TIC

. A TIC preceded by a status modifier CCW

. The last virtual CCW in the channel program (no chaining is indicated in the CCW)

A list of unresolved TIC CCWs is created to indicate the target of the virtual TIC. The TCCW control block maintains a pointer to the first unresolved TIC. The other TICS are chained.

TCCW

TCCWTICL

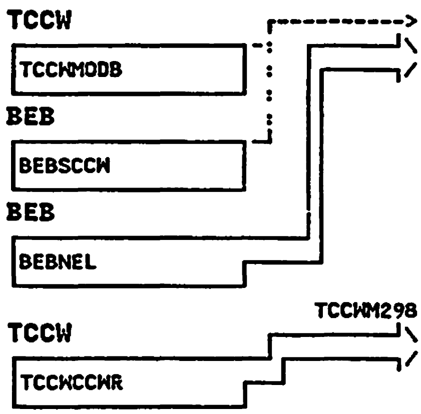
BEB

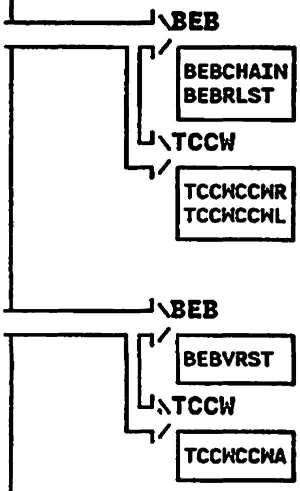
BEBFLAG
BEBRLST

TCCWM280 A. If there is insufficient space in the existing BEB to resolve the TIC, returns to the caller to obtain another large block to be used as a BEB block.

On return from the caller, register 15 contains the pointer to the large block. The large block is initialized as a BEB block, and processing continues.

TCCWGTM2: 29A
RETURN REGISTER: RTNREG

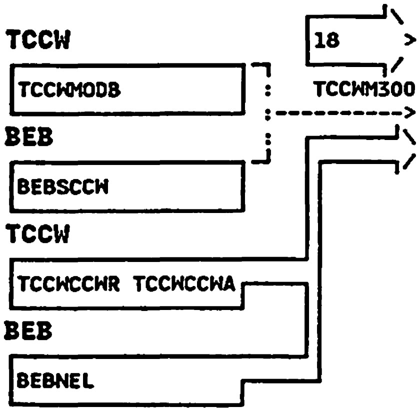




B. Establishes the virtual start address in the new BEB segment and sets the TIC real address.

C. Returns to process the next virtual CCW. (TIC processing is complete.)

>TCCWS000: 06

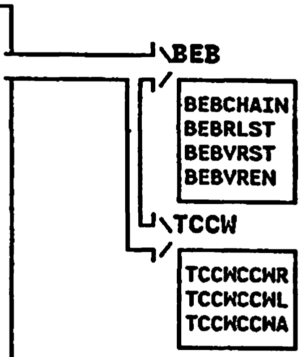


18 Returns to the caller to obtain another large block to be used as a BEB block.

This routine is called when there is not enough space in this BEB for real CCWs.

On return from the caller, register 15 contains the pointer to the large block. The large block is initialized as a BEB block, and processing continues.

Inserts a TIC command at the end of the current BEB segment of the previous BEB to point to the first segment of the new BEB. Checks to ensure that the TIC is not to another TIC or that a TIC CCW is not split from a status modifier CCW.



A. When the new BEB block is initialized, returns to continue virtual CCW processing.

>TCCWS000: 06

19

TCCWM400

TCCW

TCCWMODB TCCWPC10

19 Builds indirect address words (IDAWs) in an IDAL block.

A. Builds an indirect address word list for each CCW whose data area crosses one or more page boundaries or whose data area is fixed above the 16-megabyte line.

The IDAWs consist of the translated CCW address plus the address of each subsequent page referenced by the data area. The address of the first IDAW in the list replaces the data address in the CCW and the indirect data address (IDA) flag in the CCW flag byte is set on.

If an abend condition occurred during fixing, an additional invalid IDAW is built. If this IDAW is acted on by the channel, a channel program check (CPC) occurs.

TCCW

TCCWINDL

19B

TCCWM402

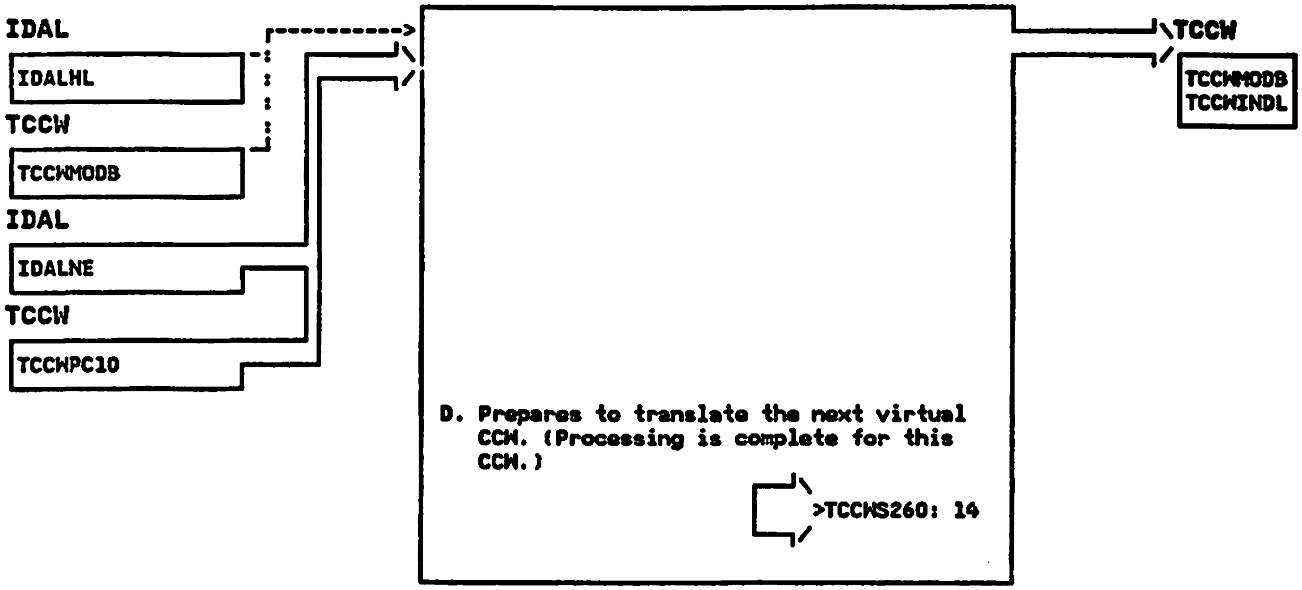
B.

C. If there are not enough IDAW slots available, returns to the caller to obtain another large block to be used as an IDAL block.

On return from the caller, register 15 contains the pointer to the large block. The large block is initialized as an IDAL block and processing continues.

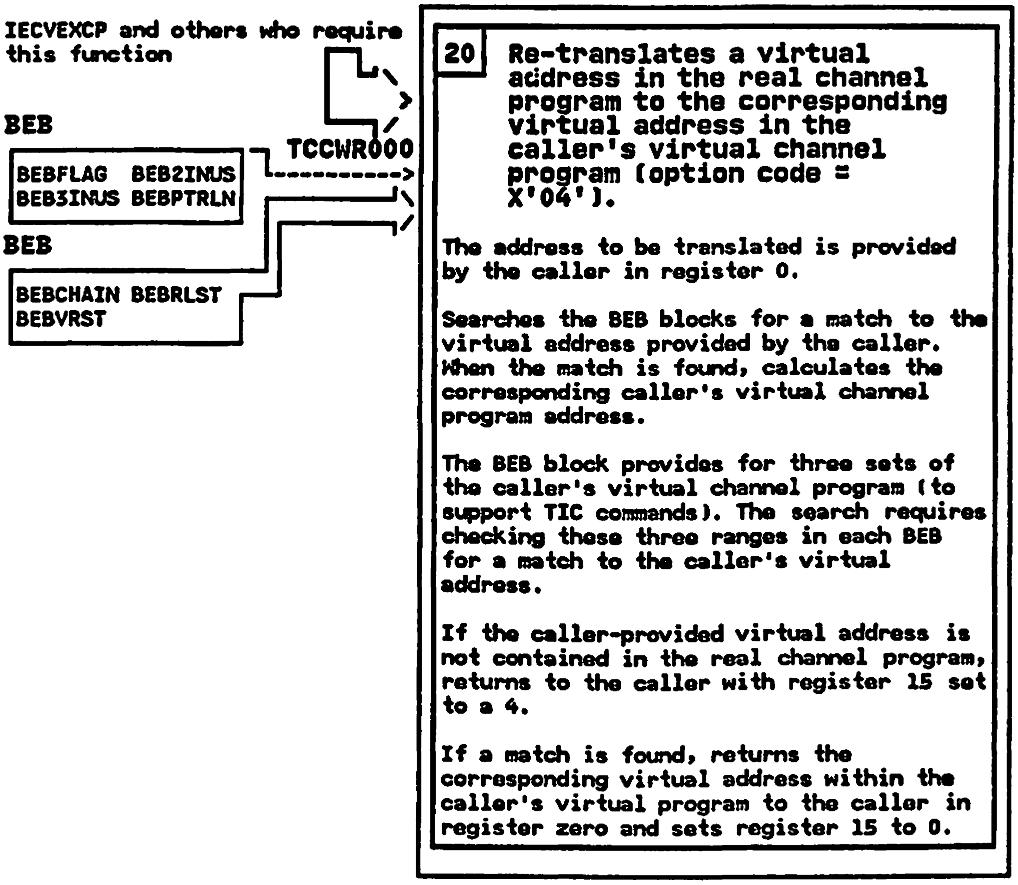
TCCWGTM0: 29

RETURN REGISTER: RTNREG



D. Prepares to translate the next virtual CCW. (Processing is complete for this CCW.)

>TCCWS260: 14



20 Re-translates a virtual address in the real channel program to the corresponding virtual address in the caller's virtual channel program (option code = X'04').

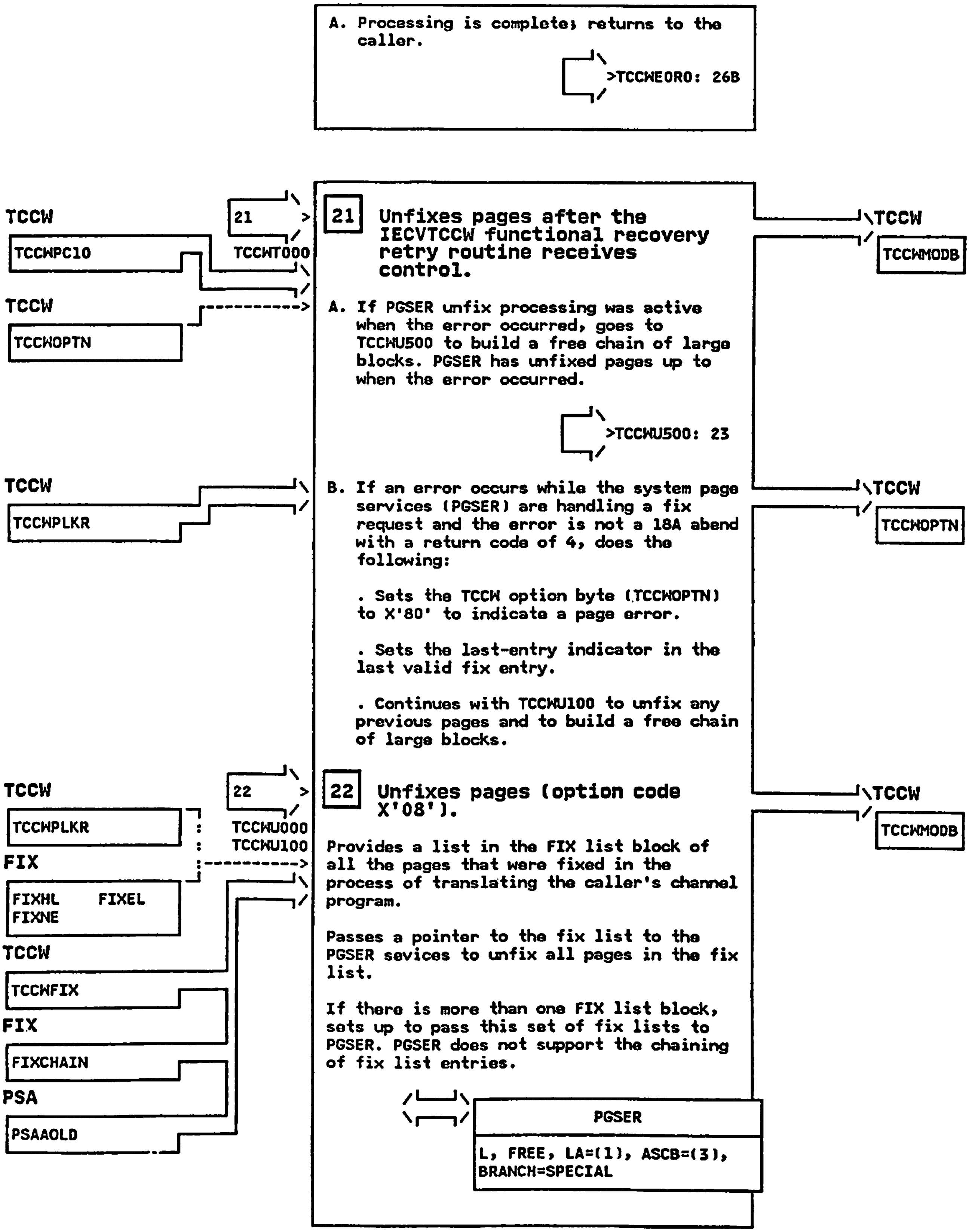
The address to be translated is provided by the caller in register 0.

Searches the BEB blocks for a match to the virtual address provided by the caller. When the match is found, calculates the corresponding caller's virtual channel program address.

The BEB block provides for three sets of the caller's virtual channel program (to support TIC commands). The search requires checking these three ranges in each BEB for a match to the caller's virtual address.

If the caller-provided virtual address is not contained in the real channel program, returns to the caller with register 15 set to a 4.

If a match is found, returns the corresponding virtual address within the caller's virtual program to the caller in register zero and sets register 15 to 0.



A. Processing is complete; returns to the caller.

>TCCWEORO: 26B

21 Unfixes pages after the IECVTCCW functional recovery retry routine receives control.

A. If PGSER unfix processing was active when the error occurred, goes to TCCWU500 to build a free chain of large blocks. PGSER has unfixed pages up to when the error occurred.

>TCCWU500: 23

B. If an error occurs while the system page services (PGSER) are handling a fix request and the error is not a 18A abend with a return code of 4, does the following:

. Sets the TCCW option byte (TCCWOPTN) to X'80' to indicate a page error.

. Sets the last-entry indicator in the last valid fix entry.

. Continues with TCCWU100 to unfix any previous pages and to build a free chain of large blocks.

22 Unfixes pages (option code X'08').

Provides a list in the FIX list block of all the pages that were fixed in the process of translating the caller's channel program.

Passes a pointer to the fix list to the PGSER sevices to unfix all pages in the fix list.

If there is more than one FIX list block, sets up to pass this set of fix lists to PGSER. PGSER does not support the chaining of fix list entries.

PGSER

L, FREE, LA=(1), ASCB=(3), BRANCH=SPECIAL

A. When all fix lists have been processed, goes to build a free chain of blocks.

>TCCWU500: 23

\TCCW

TCCWMODB

TCCW

TCCWOPTN TCCWMODB

TCCW

TCCWFIX

23 >

TCCWU500

23 Builds a free chain of large blocks.

This routine builds a free chain of all IDAL, FIX and BEB blocks (in this order) and chains them off the TCCW control block chain word.

\TCCW

TCCWMODB

IECVEXCP and others who require this function

BEB

BEBFLAG BEB2INUS
BEB3INUS BEBVREN
BEBPTRLN

BEB

BEBCHAIN BEBRLST
BEBVRST

TCCWX000

24 Translates a single address (option code X'10').

Translates a virtual address in the virtual channel program to its corresponding virtual address in the real channel program.

The caller has provided the virtual address in register 0.

If the translation is successful, sets the translated address in register 0 and sets a return code of 0 in register 15.

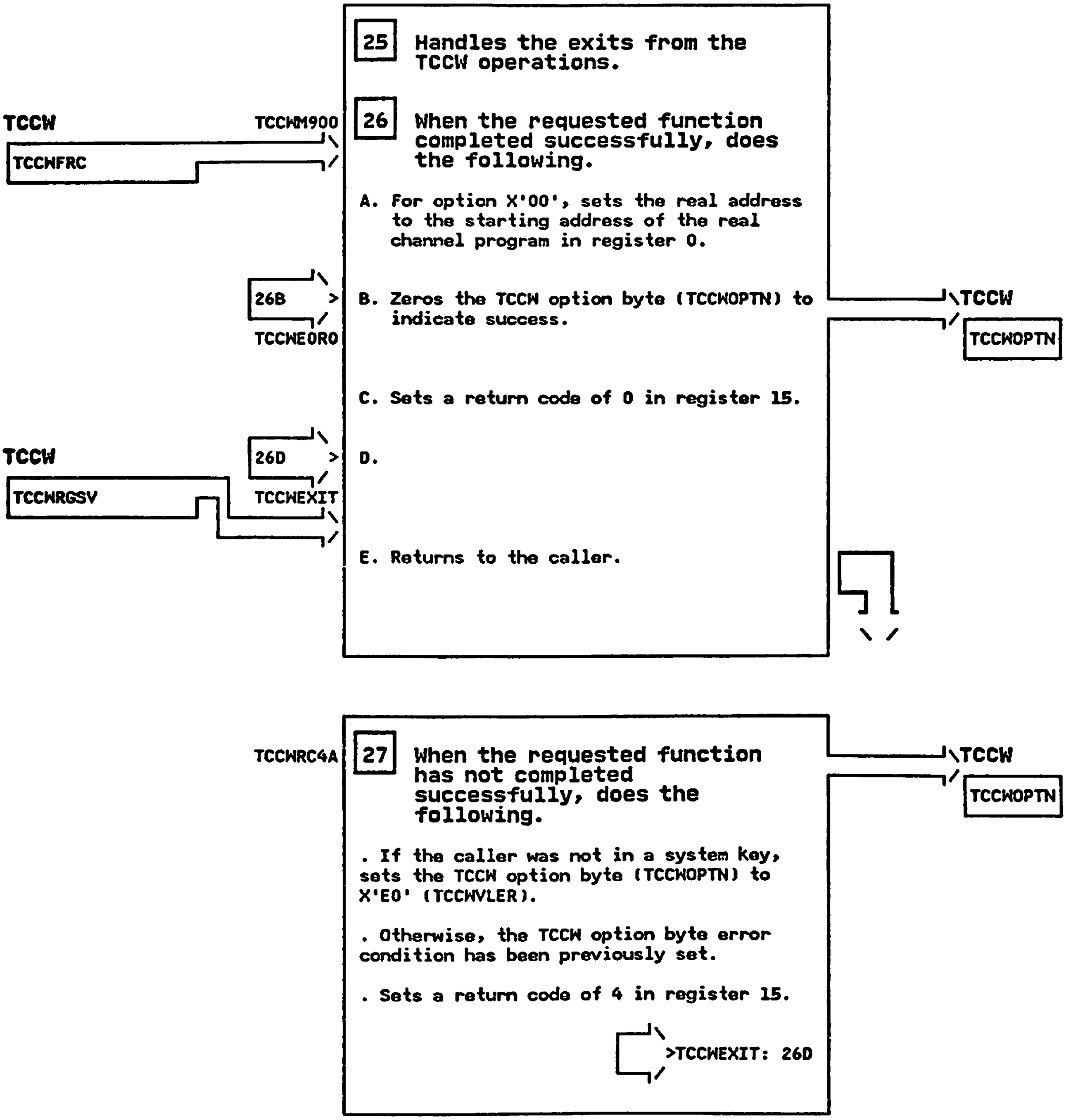
If the translation is unsuccessful, sets a return code of 4 in register 15.

A. Returns to the caller (processing is complete).

>TCCWEOR0: 26B

\TCCW

TCCWOPTN



25 Handles the exits from the TCCW operations.

26 When the requested function completed successfully, does the following.

A. For option X'00', sets the real address to the starting address of the real channel program in register 0.

B. Zeros the TCCW option byte (TCCWOPTN) to indicate success.

C. Sets a return code of 0 in register 15.

D.

E. Returns to the caller.

27 When the requested function has not completed successfully, does the following.

. If the caller was not in a system key, sets the TCCW option byte (TCCWOPTN) to X'E0' (TCCWVLER).

. Otherwise, the TCCW option byte error condition has been previously set.

. Sets a return code of 4 in register 15.

>TCCWEXIT: 26D

TCCWRC8

28 When function code X'08' completed successfully, sets a return code of 8.

>TCCWEXIT: 26D

29 >
TCCWGTM0

29 Returns to the caller of IECVTCCW to request another large block for translating the caller's virtual channel program. Does the following.
. Saves registers in the TCCW block.
. Saves the routine return address.
. Sets a return code of 12 in register 15.

29A >
TCCWGTM2

A.

TCCW
TCCWOPTN

B. Returns to the caller of IECVTCCW.

>TCCWEXIT: 26D

RTM
>
IECVTCFR

30 IECVTCCW Retry Recovery Routine.

This retry routine receives control from the EXCP processor functional recovery routine (IECVEXFR) when IECVTCCW is active and has set the TCCWPGCK bit in the TCCW control block for the following.

. ABEND 028
. ABEND 171
. ABEND 18A
. The SDWA indicates a program check.

The EXCP processor will specify, in the SDWA, this routine as its retry routine, and return to RTM requesting retry.

Actions taken are as follows:

TCCW
TCCWMODB

TCCW
TCCWPC10

A. If the error occurred while performimg a validity check (the TCCWVLCK bit is set in the TCCW control block), sets the TCCW option byte to the validity check error indication (X'E0') and returns to the TCCW mainline at TCCWU000 to unfix pages and return to the caller with a return code of 4.

>TCCWU000: 22

TCCW
TCCWOPTN
TCCWMODB

B. If the abend was not a 18A with a return code of 4, returns to the TCCW mainline at TCCWT000 to determine if page fixing or unfixing was active.

>TCCWT000: 21

TCCW
TCCWPLKR

C. If the abend was a 18A with a return code of 4 and if the first page of the fix request was invalid, returns to the TCCW mainline at TCCWT000 to set the TCCW option byte to indicate a fix error (X'80') and to prepare to unfix any previous page fix requests.

>TCCWT000: 21

D. If the abend was a 18A with a return code of 4 and if the caller was in system key zero, returns to the TCCW mainline at TCCWT000 to set the TCCW option byte to indicate a fix error (X'80') and to prepare to unfix any previous page fix requests.

>TCCWT000: 21

TCCW
TCCWPC10

E. For other errors, does the following:
. Indicates in the TCCW control block that an additional invalid IDAW (RCCWPC10) is to be provided after the last valid IDAW.
. Adjusts the fix list entry to the last valid page.
. Returns to TCCW mainline at TCCWM070 to continue the virtual channel program translation.

>TCCWM070: 15D

TCCW
TCCWMODB

INDEX

MVS/Extended Architecture
System Logic Library:
EXCP Processor

READER'S
COMMENT
FORM

LY28-1685-0

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.

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.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

____

What is your occupation? ______________________________

How do you use this publication? ______________________________

Number of latest Newsletter associated with this publication: ____________________

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.)

MVS/Extended Architecture System Logic Library: EXCP Processor

LY28-1685-0 S370-36

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 D58, Building 921-2
PO Box 390
Poughkeepsie, New York 12602

Fold and tape Please Do Not Staple Fold and tape

Printed in U.S.A.

IBM®

LY28-1685-00

MVS/Extended Architecture System Logic Library: EXCP Processor

LY28-1685-0 S370-36

Printed in U.S.A.