SY26-3823-5
File No. S370-30

Systems

OS/VS2 I/O Supervisor Logic

Includes Selectable Units:

Supervisor Performance #1	VS2.03.805
Supervisor Performance #2	VS2.03.807
MSS Enhancements	5752-824
3838 Vector Processing Subsystem Support	5752-829
3895 Device Support	5752-830
MVS Processor Support	5752-851
Hardware Recovery Enhancements	5752-855
Processor Support 2	5752-864

Sixth Edition (December, 1978)

This is a major revision of and obsoletes SY26-3823-4. See the Summary of Amendments following the Contents for a summary of the changes that have been made to this manual. A vertical line to the left of the text or illustration indicates a technical change made in this edition; revision bars are not used, however, to indicate changes made in previous editions, technical newsletters, or supplements.

This edition with Technical Newsletter SN28-4683 applies to release 3.8 of OS/VS2 and to all subsequent releases of OS/VS2 until otherwise indicated in new editions or Technical Newsletters. Changes are continually made 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.

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

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

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Publications Development, Department D58, Building 706-2, PO Box 390, Poughkeepsie, N.Y. 12602. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Preface

The purpose of this manual is to help you troubleshoot the I/O supervisor, the VS2 component responsible for communicating with the system's I/O devices. To accomplish this purpose, the manual contains two paths to the microfiche listings: a tutorial path, to make the listings meaningful to the reader lacking a basic understanding of the I/O supervisor's function and design; and a diagnostic path, to get the knowledgeable reader (detecting an error in a dump, message, or error code) to the source of error as quickly as possible.

The Tutorial Path

The tutorial path comprises this sequence of chapters:

1. *I/O Supervisor Introduction:* Classifies the callers of the I/O supervisor into groups, according to the common services they request. Names the callers that belong to each group and explains how they communicate service requests to the I/O supervisor. Introduces the terminology of input/output processing.
2. *EXCP Processor Introduction:* Describes the function of EXCP as an interface between the I/O supervisor and certain access methods and system services. Lists the programs that qualify as "access methods". Describes the access method interface. Defines "related requests".
3. *EXCP Processor Method of Operation:* Divides the EXCP processor into services and detailed text how processing occurs in the EXCP modules.
4. *EXCP Processor Program Organization:* Shows through overview flow diagrams and detailed text how the processing occurs in the EXCP modules.
5. *I/O Supervisor Method of Operation:* Divides the I/O supervisor into groups of services, each corresponding to a group of callers, and describes the processing that performs the services.
6. *I/O Supervisor Program Organization:* Shows through overview flow diagrams and detailed text how processing occurs in the I/O supervisor modules.
7. *Data Areas:* Shows the connections between the principal data areas used by the I/O supervisor. Indicates in table which data areas are created, used by, and modified by the I/O supervisor and by EXCP.

The Diagnostic Path

The diagnostic path comprises this sequence of chapters:

1. "Diagnostic Aids": Associates diagnostic output – a dump, a message, or an error code – with the symbolic name of the procedure (module part) that provides it. Diagnostic aids are described for EXCP and for IOS.
2. "Directory": Lists the symbolic names of all the procedures in the EXCP processor and the I/O supervisor, and for each procedure, gives the name of the microfiche cards that contain its code.

If this short route fails to isolate a problem, you can extend the search in these ways:

Find the section in "Program Organization" that describes the procedure identified in "Diagnostic Aids," and, using the flow-of-control information there, investigate the procedures that might have had control earlier.

Refer to the data activity table in "Data Areas." It shows, for each data area that IOS and EXCP uses, which module creates it and which modules refer to it and modify it. The table can be used to find the data dependencies that exist between modules.

Directory Usage

For both the tutorial and diagnostic paths you should make use of the directory which cross references the IOS and EXCP method of operation functions with the actual modules and procedures that perform the functions. Frequent use of the directory can make it easier to reference information between the two sections.

The key to cross referencing in the book is the name of the procedure. With that name you can move from ABEND codes to Directory to MO/PO, for example.

EXCP Processor Logic

The EXCP processor communicates requests for I/O operations from some IBM access methods to the I/O supervisor. Its logic is similar to a part of the VS2, Release 1, I/O supervisor called the EXCP supervisor. The EXCP processor, however, is an independent component of VS2, as it shares interfaces to the I/O supervisor with many VS2 components.

Overview of Error Recovery Processing

When the I/O supervisor is notified of an I/O error, it communicates with an ERP (error recovery procedure). Which ERP – there are many – depends on which device was used.

The appendix explains in general the processing done by ERPs, tells how they differ as to residence and organization, and gives in a table the modules in each ERP and the I/O device that is supported. Additionally, the appendix contains descriptions of the ERP service modules – modules that interpret errors, update statistics, write messages, and act as go-betweens for ERPs.

Data Area Conventions

Labels inside data areas name fields if they appear in bold letters:

UCB

UCBFLD1
UCBFLD2
UCBFLD3

Labels name bits within fields if they appear in italics:

UCB

UCBFLD1
UCBBIT1
UCBBIT2
UCBBIT3

The order in which labels appear in a data area does not necessarily correspond to the actual order of fields or bits.

How to Read Decision Tables

This manual uses decision tables to relate the tests a program makes to the actions that result from those tests. How a decision table relates tests to actions is shown in this example:

TESTS	Does Condition A exist?	Yes	Yes	No	No
	Does Condition B exist?	Yes	No	No	Yes
ACTIONS	Take Action 1.	X			
	Take Action 2.			X	X
	Take Action 3.	X	X		
	Take Action 4.			X	

The decision table represents a hypothetical program that tests for Conditions A and B. It shows these relationships:

- If Conditions A and B both exist, the program takes Actions 1 and 3.
- If only Condition A exists, the program takes Action 3.
- If neither Condition A nor Condition B exists, the program takes Actions 2 and 4.
- If only Condition B exists, the program takes Action 2.

Related Publications

Most of the VS2 components and routines that communicate with the I/O supervisor or EXCP processor—enumerated in the I/O supervisor "Introduction" and in "Appendix A" under "Introduction"—are documented in logic manuals bearing their names. Those that don't fall into this category are listed below (with their references sources):

- ABP, the actual block processor: *OS/VS2 Virtual Storage Access Method (VSAM) Logic*, SY26-3825
- ACR, alternate CPU recovery: *OS/VS2 System Logic Library,* SY28-0713 through SY28-0719
- the checkpoint SVC routine: *OS/VS2 Checkpoint/Restart Logic*, SY26-3820
- MIH, the missing interrupt handler: *OS/VS2 System Logic Library,* SY28-0713 through SY28-0719
- program fetch: *OS/VS2 System Logic Library,* SY28-0713 through SY28-0719
- the region control task: *OS/VS2 System Logic Library,* SY28-0713 through SY28-0719
- RMF, the resource measurement facility: *OS/VS2 Resource Measurement Facility,* LY28-0923-1

- the task-close routine: *OS/VS2 Open/Close/EOV Logic*, SY26-3827
- MSSC, the 3850 mass storage system communicator: *OS/VS2 Mass Storage System Communicator (MSSC) Logic,* SY35-0013
- VTAM, virtual telecommunications access method: *OS/VS2 VTAM Logic,* SY28-0621
- *OS/VS2 JES3 Program Logic,* SY28-0612-0
- *OS/VS2 OLTEP Logic,* SY28-0676-1

Contents

Figures

Summary of Amendments for SY26-3823-5 As Updated by SN28-4683 OS/VS2 Release 3.8

- Changes have been made throughout this publication in support of the 3033 attached processor.
- Diagnostic aids information has been deleted from this publication. It can now be found in the following books: *OS/VS2 System Programming Library: MVS Diagnostic Techniques, OS/VS Message Library: VS2 System Messages,* and *OS/VS Message Library: VS2 System Codes.*
- A considerably expanded index has been included.
- Minor technical and editorial corrections and additions have been made.

Summary of Amendments for SY26-3823-5

Changes have been made throughout this publication to support Processor Support 2 (SU64).

Notes:

- The date for this publication is December 29, 1978. Only supplements and TNLs with dates later than December 29, 1978 apply to this publication.
- SY26-3823-5 is a major revision of the OS/VS2 MVS I/O Supervisor Logic manual. This major revision obsoletes SY26-3823-4.

Summary of Amendments for SY26-3823-4

Changes have been made throughout this publication to reflect service updates and the following SUs:

- Supervisor Performance # 1 (SU5)
- Supervisor Performance # 2 (SU7)
- MSS Enhancements (SU24)
- 3838 Vector Processing Subsystem Support (SU29)
- 3895 Device Support (SU30)
- MVS Processor Support (SU51)
- Hardware Recovery Enhancements (SU55)

Note: SY26-3823-4 is a major revision of the OS/VS2 MVS I/O Supervisor Logic manual with all outstanding SU TNLs and system library supplements incorporated. This major revision obsoletes SY26-3823-3 and SY28-0757-0.

Introduction

This section describes the general functional operation of the I/O Supervisor (IOS). Callers of IOS are highlighted, and an overview of the interrelationship between IOS and any one of its callers is presented.

Specific terminology is used to discuss the operation of IOS. The following discussion of this terminology is presented to provide a basis for understanding IOS.

Basic IOS Definitions

What Is an I/O Operation?

I/O operation is a broad way of referring to any of the following system activities:

- The transfer of data from real storage across a channel to a control unit and I/O device (commonly known as a *write* operation).
- The transfer of data from an I/O device and control unit across a channel to real storage (commonly known as a *read* operation).
- The positioning of read/write mechanisms or mounted volumes (commonly known as a *control* operation). Examples of control operations are spacing or skipping lines on a printer, backspacing a tape volume, or seeking on a direct-access device.
- The manipulation of an I/O device without the transfer of data or control information (commonly known as an *immediate* operation). Examples of immediate operations are rewinding a tape volume or recalibrating an access arm (fully retracting it).

To start an I/O operation, three elements are needed: a start-I/O instruction; a channel program, made up of one or more CCWs (channel command words); the address of the first CCW. IOS stores the address of the first CCW in a location called the CAW (channel address word) and issues a start-I/O instruction that specifies the channel, control unit, and device to be used. When the channel receives this information, it finds the address of the channel program in the CAW and begins executing it, one CCW at a time (assuming that the channel, control unit, and device are fully operational and are not busy).

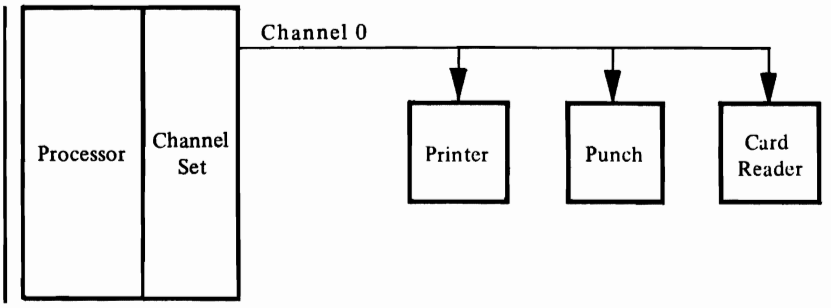
What is a Channel Set?

A channel set is a collection of up to 16 channels which may be accessed by a processor. In multiprocessor configurations which have Channel Set Switching or CRH, the channel set(s) in the configuration may be switched through special hardware, between the processors (such as when one processor has failed); in these cases, the channel sets have IDs which are independent of the processor address to which the channel set is currently attached.

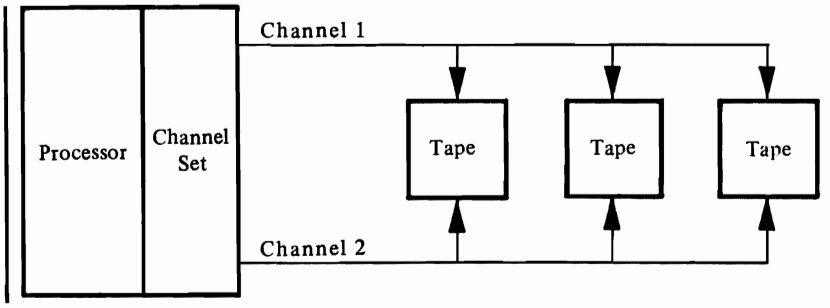
IOS always assumes that channel sets exist even on processors which do not have the hardware switching feature; in these cases, the channel sets have IDs which are equal to the processor address.

The Concept of Logical Channels

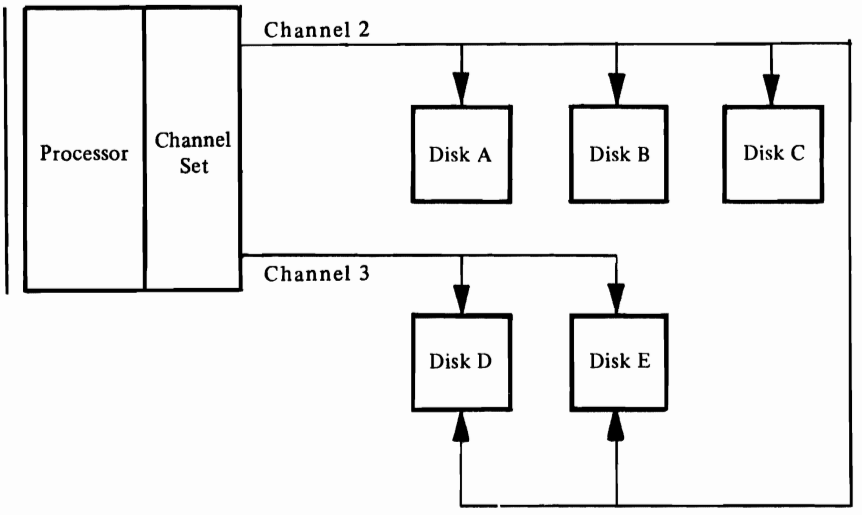
A *logical* channel is a set of physical channels which provide a path to a device or group of devices. For example, in this configuration –



the set of channels is only *one* channel, channel 0; therefore, the configuration shows one logical channel for one physical channel. But in this configuration–



two physical channels, 1 and 2, are in the set; they comprise one logical channel (since they serve the same group of devices). A third relationship is shown in this configuration:



There are two logical channels here: one composed of channels 2 and 3, by which disks D and E can be reached; the other consisting of channel 2, by which disks A, B, and C can be reached.

IOS maintains a queue, called a logical channel queue, for each logical channel in the system's device configuration. Should IOS be unable to start an I/O operation because a device, channel, or control unit is temporarily busy, it puts a record of the request in the appropriate queue. Waiting I/O requests are organized in this way so that they can be quickly located when an I/O operation ends or a device becomes "ready." (The operator makes a device ready by pressing a START button, throwing a switch to START, or—in the case of direct-access devices—inserting an address plug.)

For more information about the structure and use of logical channel queues, see "Starting an I/O Operation" in the chapter "Method of Operation."

What Is an I/O Event?

An *I/O event* is any incident in the system's I/O resources—channels, control units, devices—that causes status information to be stored in the CSW (channel status word). Examples of I/O events are:

- The completion of a read, write, or control operation. These I/O events, in addition to storing status information, cause an I/O interruption, which temporarily stops whatever processing is in progress so that IOS can look at the status information.
- The execution of a CCW with the PCI (program-controlled interruption) bit on. This I/O event also causes an I/O interruption.
- The completion of an immediate operation. In this case, status information is stored without an I/O interruption. (IOS makes tests following the start of an I/O operation to determine whether status information is stored.)

The above are known as *solicited* I/O events: they result from I/O requests. There is also a class of I/O events called *unsolicited*, all of which cause I/O interruptions. Examples of unsolicited I/O events are:

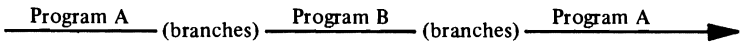
- A terminal user pressing the ATTN key
- The operator pressing the request key on his console
- The operator readying a device
- The device or channel detecting a hardware malfunction not associated with a specific I/O request

If an I/O event occurs that calls for an I/O interruption on a processor presently unable to receive I/O interruptions, the channel "holds" the interruption and status information until the processor is able to receive them.

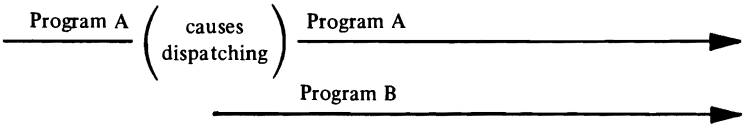
The Concept of Asynchronous Processing

If a program—call it program A—causes another program—program B—to be dispatched, then program B is said to be running *asychronously* to program A. (To put it another way, programs A and B would be running *concurrently*.)

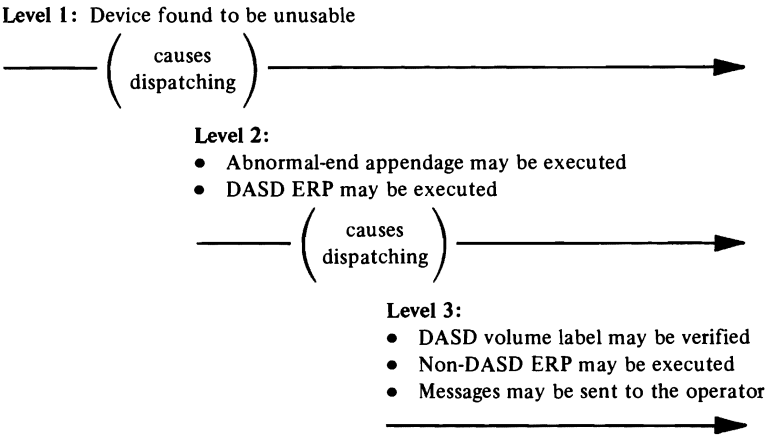
The converse of asynchronous processing is *synchronous* processing. Program A in this case relinquishes control to program B and cannot reacquire it until program B is through. Diagramatically, synchronous processing looks like this:



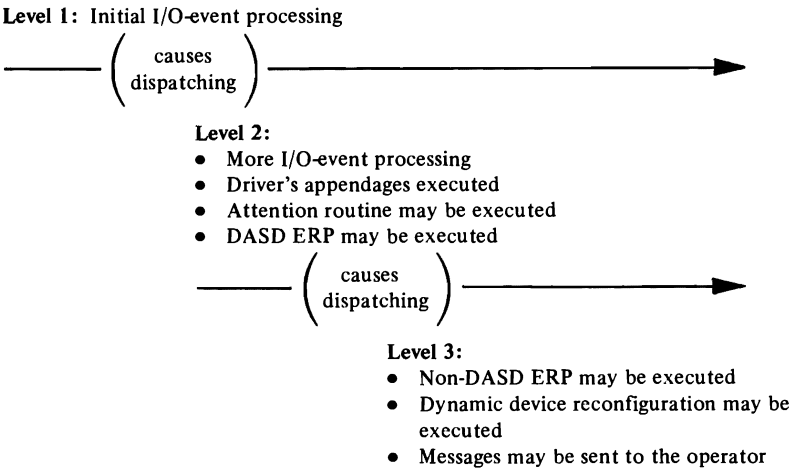
And asynchronous processing looks like this:



The concept of asynchronous processing becomes important in understanding how IOS is designed to handle I/O requests. Specifically, when IOS receives an I/O request, an IOS procedure finds out whether the device allocated for the I/O operation can be used. If the device can't be used (for some reason other than its being busy), the procedure causes other procedures to be dispatched, which in turn cause still other procedures to be dispatched. The result is three "levels" of asynchronous processing:



The same design is used in processing I/O events:



For more information about how IOS handles I/O requests and I/O events, see "Starting an I/O Operation" and "Responding to an I/O Event," respectively, in the "Method of Operation" chapter. Also, see "How the SRB Is Used," in this chapter under "Group 1 Callers and Their Interfaces," to learn how IOS causes asynchronous processing to be dispatched.

What Is a Purge Operation?

When a caller requests a purge operation, it asks IOS to perform one of two mutually exclusive functions, as specified in a parameter list:

- Halt the processing of I/O requests associated with one or more data sets, a TCB, or an address space, *and* notify the drivers that sent those requests to destroy records of similar I/O requests that they are preparing to send.
- Finish processing I/O requests associated with one or more data sets, a TCB, or an address space, *and* notify the drivers that sent those requests not to send similar I/O requests but to keep track of them (in a record or chain of records) and return the address of the record or chain.

The first alternative is called a *halt* operation; the second, a *quiesce* operation. The abnormal termination of a job is an instance that requires a halt operation; the swap out of an address space is an instance that requires a quiesce operation.

What Is a Restore Operation?

A restore operation is the process of returning to each driver the address of the record or chain of records it gave to IOS during a quiesce operation. With the address, each driver can reconstruct and submit I/O requests that it previously withheld from IOS.

I/O Supervisor Introduction

The input/output supervisor, called *IOS* for short, is the VS2 component responsible for communicating with the system's I/O devices.

IOS serves five groups of callers, each of which requests one of five basic services:

Group 1: Callers that want IOS to start an I/O operation. They are known as *drivers* of IOS.

Group 2: Callers that want IOS to respond to an I/O event.

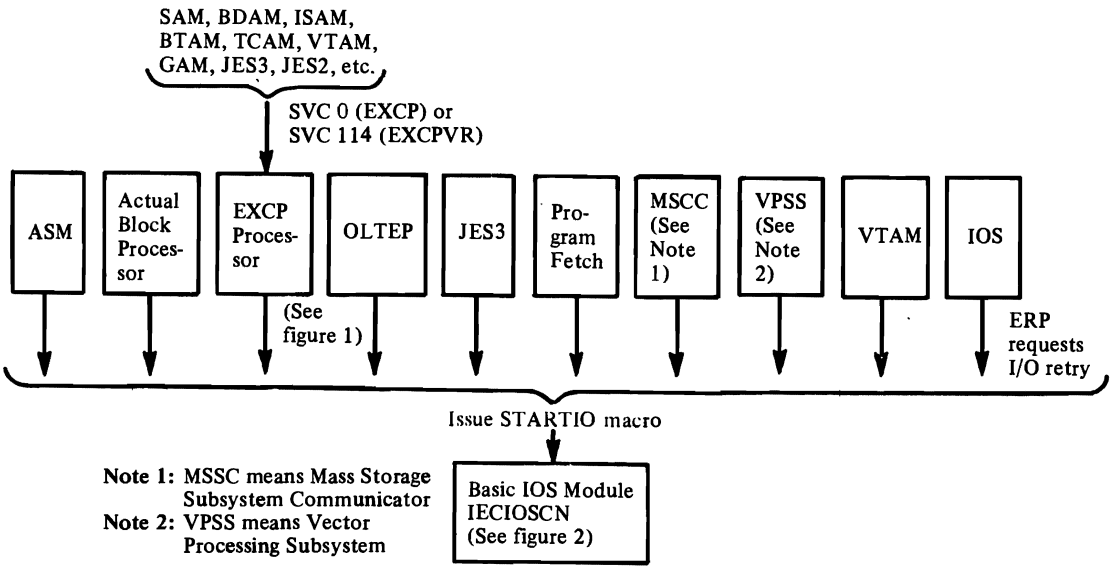
Group 3: Callers that want IOS to restore the availability of I/O resources (channels, control units, devices).

Group 4: Callers that want IOS to do a purge or restore operation.

Group 5: Callers that want IOS to halt a teleprocessing operation.

Group 1 Callers and Their Interfaces

Group 1 callers, the drivers of IOS, ask IOS to start I/O operations. Belonging to this group are the following VS2 components.



How the I/O Supervisor Is Invoked by Its Callers

ASM (the auxiliary storage manager). It calls IOS to satisfy I/O requests it receives from RSM or VBP.

ABP (the acutal block processor). It calls IOS to satisfy I/O requests it receives from VSAM.

EXCP processor. It calls IOS to satisfy I/O requests from some IBM access methods: SAM, BDAM, ISAM, BTAM, TCAM, VTAM, and GAM, PAM, etc., plus JES2 and JES3.

OLTEP (the online test executive program). It reqiests I/O operations to determine the usability of devices.

JES3. It calls IOS to perform spool I/O.

Program fetch. It requests I/O operations to read programs from a partitioned data set into virtual storage.

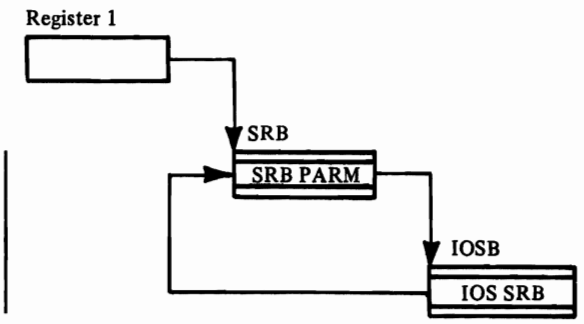
MSSC (mass storage system communicator). It calls IOS to satisfy I/O requests for the 3850 Mass Storage System.

VPSS (Vector Processing Subsystem). It calls IOS to perform I/O operations with the 3838 array processor.

VTAM (virtual telecommunications access method). It calls IOS to perform the I/O operations between VTAM and the VTAM telecommunications network.

IOS itself. IOS acts as its own driver when (a) it wants a previous I/O operation to be retried or (b) the label of a direct-access volume must be read and verified.

Drivers request the starting of an I/O operation by branching to IOS with register 1 initialized as illustrated in figure below. This interface to IOS consists of a data area, called an SRB (service request block) and a data area, called an IOSB (I/O supervisor block) – pointed to by the SRB. In this manual, the SRB and IOSB are often referred to collectively as the SRB/IOSB.



The driver executing the STARTIO macro must supply an SRB/IOSB in fixed global storage. Furthermore, the driver as well as the control blocks, CCWs, etc. must be fixed in storage until the driver has been notified that the operation was completed.

How the SRB Is Used

IOS causes the dispatching of asynchronous processing by using SRBs in two ways:

Method 1: Scheduling Asynchronous Processing. For every interruption that is a completion of an I/O request, IOS chains an SRB to a queue called an SPL (service priority list) by issuing a SCHEDULE macro. When the dispatcher checks the SPL, it gives control to the procedure addressed in the top SRB on the queue.

Method 2: Using Exit Effectors. For writing messages to the operator, writing records to SYS1.LOGREC, and calling error recovery procedures (ERPs), IOS calls a system routine, the *stage 2 exit effector* to chain an SRB to an *asynchronous exit queue.* Another routine, the *stage 3 exit effector,* on locating the queued SRB, finds the TCB and ASCB (address space control block) associated with it and marks the ASCB to show that the address space contains a "dispatchable" TCB. The dispatcher subsequently "dispatches" the TCB by giving control to the procedure addressed in the top request block on the TCB's request-block chain.

Both of these methods are used in processing I/O events, causing up to three levels of synchronous code to process I/O events concurrently. See "Responding to an I/O Event" in the chapter "Method of Operation" to understand how an I/O event is processed and where in the processing these methods are used.

How the IOSB Is Used

The IOSB contains all the information needed to start an I/O operation. In it IOS finds:

- The address of the channel program to be used.
- The address of a UCB (unit control block), which contains information about the device that has been selected for the I/O operation.
- Fields referenced in building the CCWs that prefix the driver's channel program. (These CCWs can activate hardware options on tape devices; they position the access arm and set the file mask on direct-access devices.)

The IOSB also contains information used by IOS in the course of processing a solicited I/O event, such as:

- The address of the driver's DIE procedure.
- Entries in the driver through which channel-end, abnormal-end, and PCI appendages receive control.
- The address of the driver's termination procedure. (IOS gives control to this procedure after it has done all the I/O-event processing that follows the completion of the driver's channel program.)

Not all the information in the IOSB is put there by the driver; some is filled in by IOS and presented to the driver when its DIE procedure, appendages, or termination procedure receive control. For example:

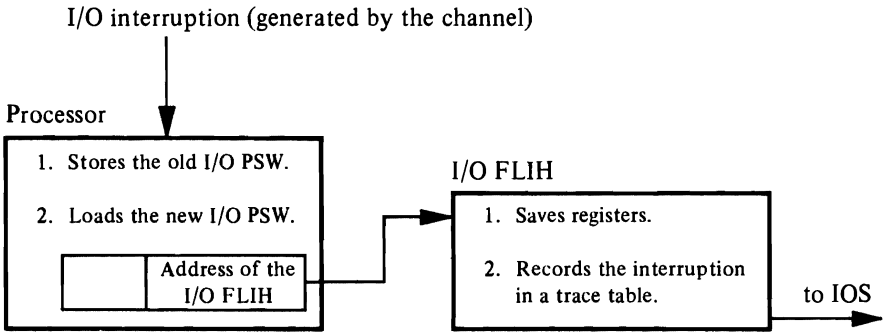
- A completion code, giving the status of the requested I/O operation
- The CSW that was stored as the result of an I/O event
- Sense information (data about the status of the device on which the operation was attempted), if the unit-check bit is set in the CSW

This is only a selection of the information in an IOSB. The "Method of Operation" chapters for both the EXCP Processor and IOS describe in more detail the names of fields and bits in the IOSB and how this information is used.

Group 2 Callers and Their Interfaces

Group 2 callers branch to the part of IOS that processes I/O events. Belonging to this group are the following VS2 components:

I/O FLIH (the input/output first-level interrupt handler). I/O FLIH branches to IOS when an I/O interruption occurs. The path of control into and out of I/O FLIH looks like this:



IOS itself. IOS is its own caller when:

- Tests show that an immediate operation completed. (Immediate operations don't cause the channel to generate an I/O interruption, but they do cause the channel to store status information—as does any I/O event—in the CSW.)
- It receives control from MIH (the missing interrupt handler). Although no I/O event occurred, IOS acts as though one has. This "simulation" permits the reallocation of I/O resources previously allocated to an uncompleted I/O operation. Status information that IOS stores in the CSW controls how IOS processes this "simulated" I/O event.

For more information about callers within IOS, see "Responding to the Condition Code Setting" and "Simulating an I/O Event" in the IOS "Method of Operation" chapter.

Group 3 Callers and Their Interfaces

Group 3 callers branch to the part of IOS that attempts to restore the availability of I/O resources (i.e., channels, control units, devices). Belonging to this group are the following VS2 components:

ACR (alternate CPU recovery). It branches to IOS if a processor becomes unusable.

CCH (the channel check handler). It branches to IOS if a channel error can't be corrected for lack of data about the last operation on the channel. IOS tries the requests again or terminates them for the failing channel. CCH can also schedule IOS as an SRB if a channel encounters a hung interface condition or if the channel becomes permanently or temporarily unusable.

CCH operates in two modes: mainline CCH and CCH MCH exit. Mainline CCH branch-enters IOS when a stored CSW indicates errors. The MCH branch-enters the CCH MCH exit if the machine check interruption code indicates that external damage has occurred. The CCH MCH exit determines if the external damage machine check occurred on a processor that is signalling a channel(s) that has become unusable.

MIH (the missing interrupt handler). It branches to IOS if the completion of an I/O operation is overdue.

Each of these callers uses register 1 as a parameter register. In it, IOS finds a code identifying the caller, MIH, and the address of the UCB (unit control block) for the device being used. If the caller is ACR, a code identifying the unusable processor is passed. If the caller is CCH, the code identifies the function IOS is to perform.

The processing performed by IOS for these callers is described under "Restoring the Availability of I/O Resources" in the IOS Method-of-Operation chapter.

Group 4 Callers and Their Interfaces

Group 4 callers ask IOS to purge I/O requests. Two callers, marked with an asterisk (*) in the following list, ask for a quiesce operation and subsequently ask IOS to restore the I/O requests.

The callers in Group 4 are:

The checkpoint SVC routine (SVC 63).* It asks IOS to quiesce I/O requests so that it can write records showing the status of a job step.

RTM (the recovery termination manager). It requests a halt operation for one of two reasons: (a) a system or user routine wants a halt operation to be done before recovery processing begins or (b) a job, task, or address space is terminating abnormally, and its resources are being returned to the system.

The region control task.* It requests a quiesce operation to prevent I/O requests, about to be passed to IOS, from being processed when the requestor's address space is being swapped out.

The task-close routine (a resource manager). It asks IOS to halt I/O requests associated with a data set that has been closed.

The I/O and path mask update routine (IECVIOPM). It requests a halt operation associated with a "data set identifier". This is done after a fixed interval during which no response was received from the path verification I/O request.

To request a purge operation, these callers issue a PURGE macro, which expands into an SVC 16 instruction. This causes IOS to receive control via the SVC interrupt handler.

On receiving control, IOS expects to find the address of a PPL (purge parameter list) in register 1. In the PPL the caller must have supplied:

- An indicator telling IOS to halt or quiesce I/O requests.
- An indicator telling IOS which requests to purge—those associated with a specific data set, those for a specific task, or those in a specific address space.
- Either a "data set identifier" (the address of a data area that identifies a data set) *or* a TCB address *or* an address space identifier. One of these is the *search argument*, the field that will be used in comparison tests to find I/O requests to be purged.
- The address of a fullword into which IOS stores the address of the PIRL (purged I/O restore list). (The PIRL is created during a quiesce operation and initialized with pointers to the interrupted work of each IOS driver.)

A restore operation is requested when a RESTORE macro is issued, resulting in an SVC 17 instruction. In this case, register 1 provides IOS with the address of the PIRL area containing the pointers to the driver's interrupted work.

Group 5 Callers and Their Interfaces

BTAM or TCAM calls IOS when it wants to halt a currently running channel program. The call is made with an IOHALT macro, which contains SVC 33 instruction in its expansion.

Depending on the contents of register 1, IOS either halts the channel program with an HDV (halt-device) instruction or branches to the EXCP processor, which modifies a CCW to halt the channel program. Register 1 also tells IOS where it can find the UCB for the teleprocessing device; register 0 points to the CCW to be modified, if that's how the caller wants the channel program to be halted.

EXCP Processor Introduction

The EXCP processor, called EXCP for short, is a VS2 component. It resides on the SYS1.NUCLEUS data set and executes in the resident area of real storage.

EXCP communicates information between the IBM access methods (plus VTAM, JES2, and JES3) and IOS (the input/output supervisor). Its 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 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 appendages by (a) gaining control at the IOS exits of each of them 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 causing a code to be put in its ECB (event control block).

As one of the drivers (Group 1 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.

The processing done by EXCP in purge and restore operations is explained in more detail in the EXCP Method of Operation chapter.

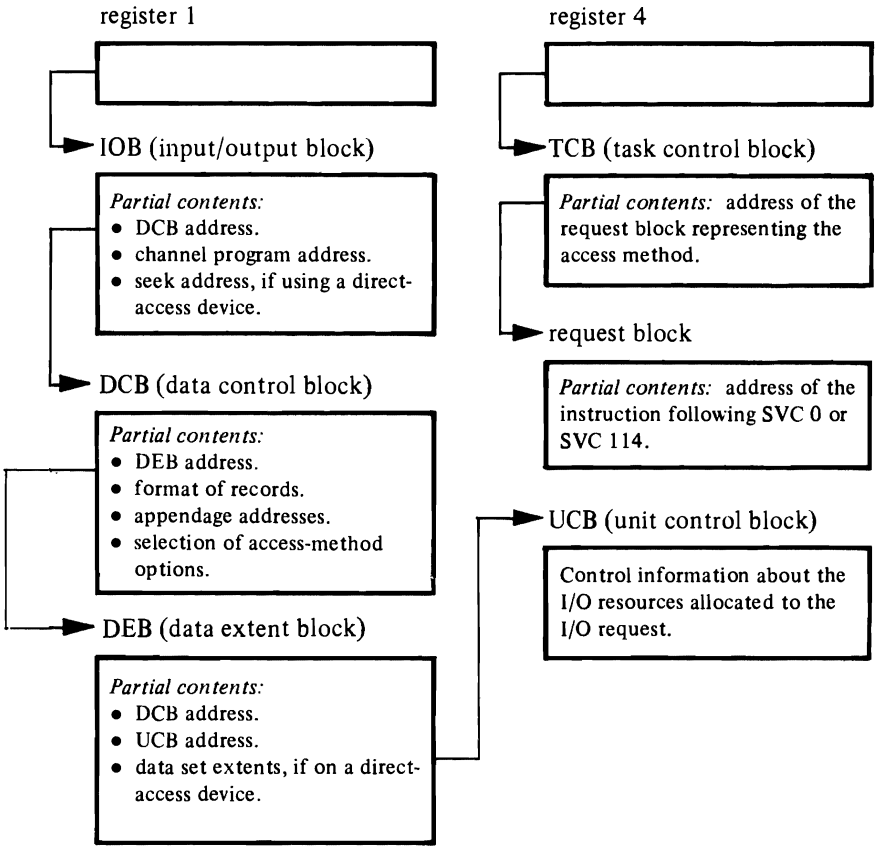
What Programs Qualify as Access Methods?

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 – SAM, BDAM, ISAM, BTAM, TCAM, VTAM, GAM, 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).

What Is 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:



These control blocks, taken together, constitute the access-method interface. It contains 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

See "Preparing to Go to IOS" in the EXCP Method of Operation chapter to learn more about the uses of the access-method interface.

What Are 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 needn't 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 currently exist:

Type 1. 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 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.

Type 3. The I/O operation for this type must complete before the next related request can be sent to IOS. (If the CSW 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.)

Refer to "Making a Record of the Request" in the EXCP Method of Operation chapter to learn how EXCP keeps track of the order and progress of related requests.

EXCP Processor Method of Operation

This chapter contains a simplification of EXCP code, divided into sections that correspond to basic EXCP operations. Basic EXCP operations consist of:

- Preparing to go to IOS (with an I/O request)
- Giving an I/O request to IOS
- Going to the PCI, CHE, and ABE appendages
- Purging and restoring I/O requests
- Telling the access method what happened (to its I/O request)
- Reusing the access-method interface
- Halting a teleprocessing operation

Each section is divided into topics that deal with functionally distinct parts of an operation.

The flow of control between labeled parts of EXCP is not stated in these sections. Rather, an order of events is implied by the order of topics within a section. If you want flow-of-control information, look at Figure 1 and the descriptions of the basic EXCP module and miscellaneous module in the "EXCP Program Organization" chapter.

Preparing to go to IOS

Preparing to go to IOS with an I/O request requires up to seven steps:

1. EXCP examines the access-method interface for irregularities that might cause I/O errors or jeopardize the security of the system.
2. EXCP makes a record of the request and puts it in a queue if it is a related request.
3. EXCP finds out if a VIO (virtual input/output) data set will be used and, if it will, does not go to IOS with the request but to the VIO component instead.
4. EXCP Puts into an SRB (service request block) and IOSB (I/O supervisor block) all the information IOS needs to process the request.
5. If tests justify it, EXCP calls the access method's PGFX (page-fix) and EOE (end-of-extent) appendages.
6. EXCP calls the access method's SIO (start-I/O) appendage.
7. If the access method uses virtual storage addresses, EXCP calls a system routine that fixes buffers, copies the channel program in fixed storage, and substitutes real storage addresses for virtual ones.

Validating the Access-Method Interface

Module: IECVEXCP
Procedure: XCP000

EXCP checks the control blocks it has been given for irregularities that might cause I/O errors or jeopardize the security of the system. Some of the irregularities checked for are:

- Conflicting DCB pointers.
- An invalid UCB.

- An invalid DEB.
- An IOB, ECB, or DCB that is not in the protection key of the caller.

The last two checks are only done if the caller has a "user" protection key. (User protection keys range from 8 to 15.)

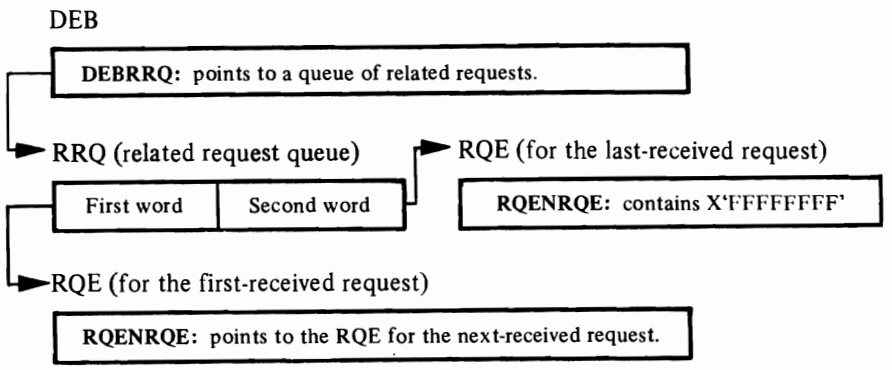
Making a Record of the Request

Module: IECVEXCP
Procedure: XCPRQE

EXCP fills a record, called an RQE (request queue element), with information, such as the addresses of the TCB, UCB, IOB, and DEB, that are needed for later processing. (The address of the SRB is also put in the RQE. See "Consolidating Information in an SRB/IOSB" in this chapter to learn more about this data area.)

If the IOB shows that the I/O request is a *related* request (the IOBUNREL bit is off), EXCP marks the RQETYPE field, using the IOBFLAG2 field as input, to show what type of related request it is. The RQETYPE field is later used with the RQEFLAG field–it shows the progress of the request–to overlap the processing of this related request with the next related request, if there is one. (See "What Are Related Requests?" for an explanation of related request types.)

Each time EXCP builds an RQE for a related request, it puts the RQE at the end of this pointer structure:

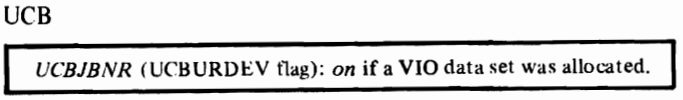


By these means, EXCP keeps track of the order in which related requests are received.

Determining If a VIO Data Set Was Allocated

Module: IECVEXCP
Procedure: XCPVAM

EXCP examines the UCB to find out if the object of the request is a VIO data set:



If a VIO data set was allocated, EXCP goes to the VIO component, using the WIEXCP macro. (The VIO component either simulates the transfer of data or uses another driver of IOS, the auxiliary storage manager, to read or write data. See *OS/VS2 VIO Logic* for more information about VIO processing.)

Consolidating Information in an SRB/IOSB

Module: IECVEXCP
Procedure: XCP050

EXCP obtains a data area for an SRB and an IOSB, in which it puts all the information that IOS needs to start an I/O operation. They are referred to collectively as *SRB/IOSB.*

In the table below, the lefthand column lists information that EXCP puts in the SRB/IOSB; the middle column shows where the information comes from, and the righthand column shows where it goes:

TCB address	RQETCB	SRBPTCB
UCB address	RQEUCB	IOSUCB
channel program address	IOBST	IOSRST*
seek address	IOBSEEK	IOSEEKA
access-method options	IOBFLAG1 DCBIFLG	IOSFLA IOSOPT
file mask	DEBDVMOD	IOSFMSK
DEB address	RQEDEB	IOSDSID

* The address of a copy of the channel program is stored in IOSRST. See "Copying and Translating a Channel Program" in this chapter for more information.

In addition, EXCP initializes the IOSB with information not found elsewhere:

IOSB

IOSDRVID: contains X'02', identifying EXCP as the driver that created the IOSB.

IOSNRM: points to EXCP code that moves data from the IOSB to the IOB and calls the access method's channel-end appendage.

IOSABN: points to EXCP code that moves data from the IOSB to the IOB and calls the access method's abnormal-end appendage.

IOSPCI: points to EXCP code that moves data from the IOSB to the IOB and calls the access method's PCI appendage.

IOSDIE: points to EXCP's DIE procedure.

IOSPGAD: points to the EXCP code that's entered when IOS is finished processing an I/O event.

Going to the PGFX, EOE, and SIO Appendages

Module: IECVEXCP
Procedures: IECVEXTC
XCP110

EXCP finds out if the access method has a PGFX appendage by examining the DEB:

DEB

DEBSIOAB: the high order bit "on" in the high order byte (DEBPGFX) means a PGFX appendage exists.

EXCP gives a PGFX appendage control if the RQE shows the access method either issued an EXCPVR macro or uses virtual storage addresses:

RQE

RQETYPE *RQE114:* on if EXCP was entered with an EXCPVR macro. *RQEVIRT:* on if the caller uses virtual storage addresses.

Pages in the list returned by the PGFX appendage are fixed if EXCP was entered by an EXCPVR macro. They are not fixed if the caller uses virtual storage addresses. (The buffers used by such callers are subsequently fixed by the processing described under "Copying and Translating the Channel Program.")

For requests from a V=R address space, EXCP checks whether the DEB has been fixed. If not, EXCP does a pagefix, using the TCB address in the DEB. (Note: This is a TCB-associated pagefix.)

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. Otherwise, the EOE Appendage is not entered.

IOB

IOBSEEK: first byte is an index to the data set extent entry in the DEB. Remaining bytes contain the seek address: the cylinder and track to which the direct-access volume will be positioned.

DEB

DEBDVMOD: the beginning of an area containing a 14-byte entry for each extent in the data set. Each entry gives the bounds in which the seek address must fall.

Upon return from the EOE appendage, EXCP performs one of the following functions as indicated by the appendage.

- Tells the access method about an "out-of-extent" error (by putting X'42' in the IOBECBCC field of the IOB) and calls the abnormal-end appendage.
- Turns on the RQEPURGE bit to indicate RQE is to be purged without further appendage processing.
- Rechecks the seek address, and if it still doesn't fall within the extent boundaries, reenters the EOE appendage.

EXCP also goes to the EOE appendage if, after IOS tries to start an I/O operation, the direct-access ERP alters the seek address (to cause a track or cylinder switch) and wants the new seek address to be verified.

EXCP enters the SIO Appendage unconditionally. Using different return addresses, the appendage can tell EXCP to continue processing the request or terminate it.

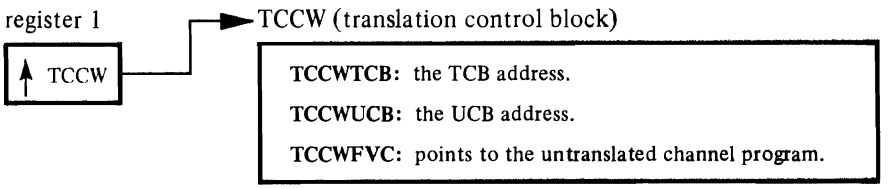
Copying and Translating the Channel Program

Module: IECVEXCP
Procedure: XCP115

If the access method uses V=R storage addresses, or if it enters EXCP with an EXCPVR macro, the channel program is ready to be executed: the channel program and the buffers reside in fixed storage, and the buffer addresses in the channel program's CCWs are real addresses.

EXCP can, in this case, convert the contents of the IOBST field to a V=R storage address and use that address to initialize the IOSRST field. (IOS assumes that the IOSRST field contains the V=R storage address of the channel program.)

The channel programs of other callers must be copied in a fixed area, the buffers must be fixed, and virtual storage addresses must be translated into real ones. This is all done by the IOS CCW translator module, IECVTCCW. On receiving control, the CCW translator module finds:



The CCW translator module returns the starting address of a fixed, translated copy of the channel program; EXCP stores the starting address in the IOSRST field.

Giving an I/O Request to IOS

Module: IECVEXCP
Procedure: XCP145

EXCP gives an I/O request to IOS by calling the IOS code that starts I/O operations. The 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 driver after a solicited I/O event occurs.) In both cases, IOS gets control with the address of the SRB in register 1.

Going to the PCI, CHE, and ABE Appendages

After receiving an I/O request from EXCP, IOS calls EXCP one or more times to communicate with a:

- PCI (program-controlled interruption) appendage
- CHE (channel-end) appendage
- ABE (abnormal-end) appendage

Before EXCP invokes one of these appendages, it transfers information from the IOSB to the IOB so that the appendage can examine the IOB and know the status of the I/O request.

If the appendage is a CHE or ABE appendage, EXCP executes options for it (such as moving data or setting flags) before returning to IOS. Otherwise no options are executed and control returns to IOS.

Entering the Disabled Interrupt Exit (DIE) Procedure

Module: IECVEXCP
Procedures: XCPDIE
XCPMAP

Normally, disabled procedures are kept to a minimum since the system cannot respond to other interrupts while disabled. However, under certain conditions, EXCP uses the disabled interrupt exit (DIE) procedure to enter an appendage itself, rather than wait for IOS to branch to the exit. The IOS code that gives control to the DIE procedure executes (synchronously) before the IOS code that branches to the exit addresses in the IOSB (which executes asynchronously).

For the normal V=V address space, the DIE procedure is not entered. It is entered only under the following two conditions:

1. If the access method is running in a V=R address space, or if it called EXCP with an EXCPVR macro, the DIE procedure branches to the PCI appendage, first setting up the IOB as described under "Transferring Status Information to Appendages" in this chapter. (EXCP assumes that fixed callers – TCAM, for instance – require better performance and want the chance to modify an active channel program as soon after a PCI interruption as possible.)
2. If the access method has given EXCP a type 3 related request, the DIE procedure checks to see if the next request element can be started. If so, the DIE initializes the IOSB with information from the request queue element (RQE) about the next request to be started. Then the DIE passes the IOSB/SRB to IOS.

Transferring Status Information to Appendages

Module: IECVEXCP
Procedures: XCPPCI
XCPCHE
XCPABE
XCPMAP

When IOS branches to one of the appendage addresses in an IOSB created by EXCP, EXCP is entered instead of the appendage. At each of these entrances, EXCP prepares to go to the appropriate appendage by transferring information about the status of the I/O event from the IOSB to the IOB.

In the table below, the lefthand column shows status information in the IOSB, the middle column shows where it is located, and the righthand column shows where EXCP puts it in the IOB.

sense information	IOSSNS	IOBSENS0 IOBSENS1
completion code	IOSCOD	IOBECBCC
channel status word	IOSCSW	IOBCSW

Additionally, if the "exceptional-condition" bit, IOSEX, is on, EXCP turns on an error bit in the IOBFLAG1 and DCBIFLGS fields.

Executing Appendage Options

Module: IECVEXCP
Procedures: XCPCHE
XCPMAP

The CHE and ABE appendages return to EXCP at any of several addresses; each return causes EXCP to execute a different set of appendage options before returning to IOS. Depending on where it is entered, EXCP takes one or both of the following actions:

- Transfers the status information, whether altered by the appendage or not, back to the IOSB.
- Sets bits in the RQEFLAG field controlling the EXCP code that IOS enters when it finishes processing the I/O event.

Purging and Restoring I/O Requests

Purging I/O requests consists of these steps:

1. If a halt operation was requested, EXCP frees the SRB/IOSBs that IOS passes and frees associated data areas that EXCP created.
2. Regardless of the type of purge operation, EXCP frees RQEs that match a search argument IOS passes.
3. If a quiesce operation was requested, EXCP saves the IOBs for requests that haven't been given to IOS and chains the IOBs together. (The chain is needed if a restore operation is subsequently requested.)

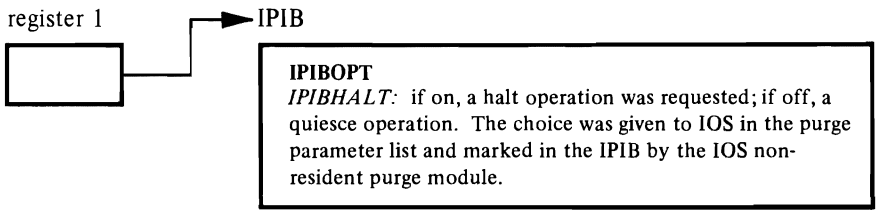
To restore I/O requests, EXCP issues an SVC instruction–SVC 0, 92, or 114–for each IOB (I/O request) in the IOB chain. The SVC instruction causes the I/O request to be reprocessed by the EXCP code that gives I/O requests to IOS.

Freeing Data Areas Known to IOS

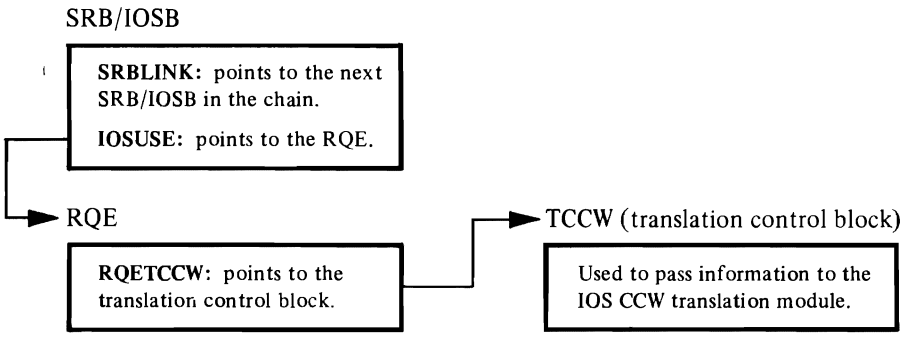
Module: IECVEXPR
Procedure: IECVXPUR

Module: IECVEXCP
Procedure: XCPTERM

On receiving control from the IOS nonresident purge module, EXCP checks the IPIB to determine if the issuer of the PURGE macro asked for a halt or quiesce operation.



If a halt operation was specified, EXCP finds in the IPIBSRB field the address of the first SRB/IOSB in a chain of SRB/IOSBs that IOS collected for EXCP's disposal. Using the following pointers, EXCP frees each SRB/IOSB and the associated RQEs and translation control blocks:

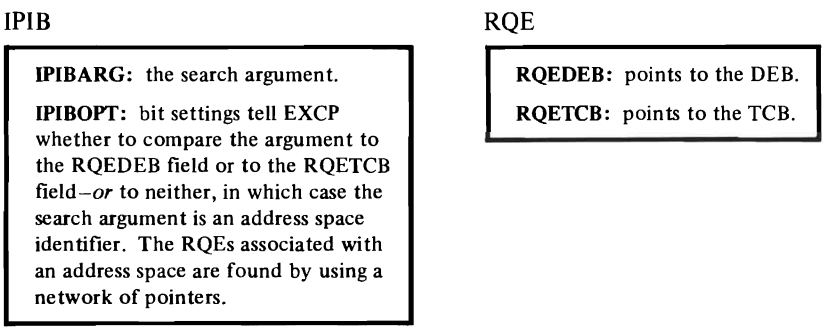


Comparing RQEs to the Search Argument

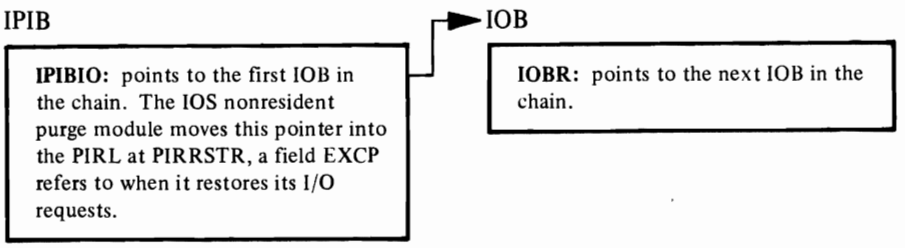
Module: IECVEXPR
Procedures: IECVXPUR
IECVRCHN

Module: IECVEXCP
Procedure: XCPTERM

There are some I/O requests for which IOS has no internal records (no SRB/IOSBs), as is the case if (a) EXCP hasn't given the I/O request to IOS yet or (b) IOS has finished processing the request, and EXCP has freed the SRB/IOSB. If IOS has no SRB/IOSB for an I/O request, it has no way to identify the RQE, should the RQE be associated with the purge operation. EXCP finds such RQEs by comparing the search argument in the IPIB to the designated field of every RQE that still exists.



RQEs that match the search argument, and any translation control blocks they point to, are freed if a halt operation was requested. (ex: if the RQE points to a SRB/IOSB and TCCW blocks, the SRB/IOSB and TCCW blocks are freed before the RQE.) If a quiesce operation was requested, matching RQEs are also freed, providing they represent I/O requests that haven't been sent to IOS, and the associated IOBs are put in a chain.

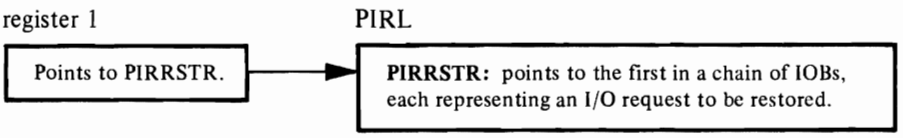


A count of all the other matching RQEs is added to the IPIBCNT field, which shows the total of I/O requests that have reached IOS but haven't been completely processed. (When the system eventually passes these RQEs to EXCP for disposal, EXCP finds them marked with an IPIB address and decreases the IPIBCNT count. The quiesce operation is only complete when an IOS driver, not necessarily EXCP, decreases the count to zero.)

Restoring I/O Requests

Module: IECVEXPR
Procedure: IECVXRES

On receiving control from the IOS restore module, EXCP finds:



Restoring an I/O request means *resubmitting* an I/O request. To resubmit an I/O request, EXCP must recreate the access-method interface by:

- Putting the address of the IOB in register 1.
- Issuing an SVC 0 or an SVC 114 instruction, which causes EXCP's entry code to get control with the address of the *restore* TCB (the one the issuer or the RESTORE macro is running under) in register 4.

EXCP performs these steps for each IOB in the chain, varying the procedure in only one case: if the issuer of the PURGE macro requested that the I/O request be reprocessed under a *target* TCB (a TCB other than the restore TCB), EXCP puts the address of the target TCB in register 0 and issues an SVC 92 instruction, again giving control to its entry code.

A data area called the EPCB (EXCP purge control block), built when EXCP's I/O requests were quiesced, gives EXCP the information it needs to reprocess each of the IOBs (I/O requests) on the IOB chain:

EPCB entry (one per IOB)

EPCBIOB: contains the address of an IOB on the IOB chain.
EPCBTCB: contains X'F4' if EXCP is to issue an SVC 114 instruction to reprocess the IOB addressed in the EPCBIOB field.
EPCBTCB+1: contains zeros if the IOB addressed in the EPCBIOB field is to be reprocessed under the restore TCB. Otherwise, contains the address of the target TCB under which the IOB is to be reprocessed.

Purging Dependent I/O Requests

Module: IECVEXCP
Procedures: XCPTERM
XCPPUR

If IOS returns to EXCP a related request whose IOSB is marked with an error indication, EXCP purges all the I/O requests that depend on the successful completion of the related request; that is, all the I/O requests that follow it on the related request queue are purged. (See "Making a Record of the Request" in this chapter for how the queue is located and structured.)

DCB

DCBIFLGS: if the first two bits are on, an uncorrectable I/O error was encountered.

The purge of dependent I/O requests is a limited version of what EXCP does to complement an IOS purge operation. It includes these steps:

- Freeing the SRB/IOSBs, RQEs, and the translation control blocks belonging to related requests.
- Chaining the IOBs of dependent requests together.
- Telling the access method what happened by putting an X'48' in each ECB.

EXCP lets the access method decide whether to resubmit the I/O requests.

Telling the Access Method What Happened

Module: IECVEXCP
Procedure: XCPTERM

The process of telling the access method what happened to its I/O request is called *posting*. A one-byte completion code is put (posted) in the ECB for the access method's inspection when:

- The EOE appendage returns to EXCP with an "out-of-extent" error.
- The EOE appendage directs EXCP to ignore the I/O request and return to the access method.
- The I/O request is purged, unless the IPIB shows that the purged request should not be posted.

IPIB

IPIBOPT
IPIBPOST: if off, the request is not posted. (Set by the IOS nonresident purge module (IGC0001F) in accordance with options in the purge parameter list.)

- IOS finishes processing the request, unless the RQE shows that the request should not be posted.

RQE

RQEFLAG
RQENOPST: if on, the request is not posted. (Set by EXCP at the direction of an appendage.) if on, the request is posted.

The system routine that does the posting also finds the TCB under which the access method is running, decreases the "wait" count, and if the count becomes zero, marks the TCB "dispatchable." The access method, waiting to learn about the status of its I/O request, can then get control and examine the ECB.

When EXCP gives control to the posting routine, it passes the completion code and the ECB address in registers 10 and 11, respectively.

IOB

IOBECBCC: the completion code is taken from this field. If the I/O request was processed by IOS, the code was moved here from the IOSCOD field of the IOSB. To learn what the codes are and what they mean, see "The IOSCOD Field" in the IOS "Diagnostic Aids" chapter.

IOBECBPT: points to the ECB.

Reusing the Access-Method Interface

Module: IECVEXCP
Procedure: XCPTERM

After IOS finishes processing an I/O request, EXCP frees the RQE, unless the RQE shows that *re-EXCP* processing is requested:

RQE

RQEFLAG
RQERETRY: the "re-EXCP" bit; if on, it tells EXCP to reuse the access method interface. (EXCP would have turned it on earlier if directed to by the CHE or ABE appendage.)

An appendage requests "re-EXCP" processing as a quick way of executing the same channel program or a new one—quick because EXCP doesn't have to revalidate the access-method interface or create a new RQE. (If an appendage wants a new channel program to be executed, it must additionally change the channel-program pointer in the IOB or modify the original channel program.)

EXCP initiates the processing of the "new" I/O request by returning to the code that compares the seek address to the limits of the data set extent.

Halting a Teleprocessing Operation

Module: IECVEXPR
Procedure: SVC33

EXCP gets control from a teleprocessing access method by this route:

1. A teleprocessing access method issues an IOHALT macro, which generates an SVC 33 instruction directing IOS and EXCP to halt a teleprocessing operation.
2. The SVC interrupt handler goes to the IOS halt-I/O code.
3. IOS, finding that EXCP was chosen to halt the operation (by examining register 1), branches to EXCP's halt-I/O code.

On receiving control, EXCP finds a pointer to an untranslated CCW in register 0 and:

- Stores a "no-op" operation code, X'03', into the translated CCW that corresponds to the untranslated channel program.
- Turns off the command-chaining bit in the translated CCW.

These actions cause the channel program to end.

EXCP Processor Program Organization

This chapter is organized by object module name and by procedure name within each object module.

The following reference features are provided to help you move quickly within and between the chapter sections.

- The sections appear in the alphabetical order of their titles. (The titles are the names of the modules.)

- Procedure subtitles are assigned numbers within each module.

  When a procedure name and number is referenced, simply locate the numbered procedure within the module.

  When a module name and number is referenced, first locate the module, then locate the numbered procedure within that module.

- Place markers, printed at the top of each page, give the name of the module and the numbers of the procedures described on the page. To find the description of a given module or procedure, you can scan the place markers.

EXCP is made up of the basic EXCP module (IECVEXCP) and the miscellaneous module (IECVEXPR). Both are link-edited at system generation into the nucleus load module, IEANUCxx.

The object modules are the program units that perform the operations described in the "Method of Operation" chapter. This table shows which of these modules perform which services:

Operation	Module
Preparing to go to IOS	Basic EXCP Module
Giving an I/O request to IOS	Basic EXCP Module
Going to the PCI, CHE, and ABE appendages	Basic EXCP Module
Purging I/O requests	Miscellaneous Module (but the Basic EXCP Module does the purging described under "Purging Dependent I/O Requests")
Restoring I/O requests	Miscellaneous Module
Telling the access method what happened	Basic EXCP Module
Reusing the access method interface	Basic EXCP Module
Halting a teleprocessing operation	Miscellaneous Module

This part is divided into two sections, "Basic EXCP Module (IECVEXCP)" and "Miscellaneous Module (IECVEXPR)." Each section tells what the module does by describing the module's procedures. Each shows the flow of control into, out of, and within the module by identifying the calls made by the module's procedures, and the entrances to and exits from them.

The basic EXCP module, the larger and functionally more important module, is also represented in flow-of-control diagrams, Figure 1. The diagram shows a simplification of the module's processing and control flow. The number next to each block corresponds to the procedure number assigned to the procedure descriptions that follow the diagrams. Use the place markers at the top of each page to reference the desired procedure description.

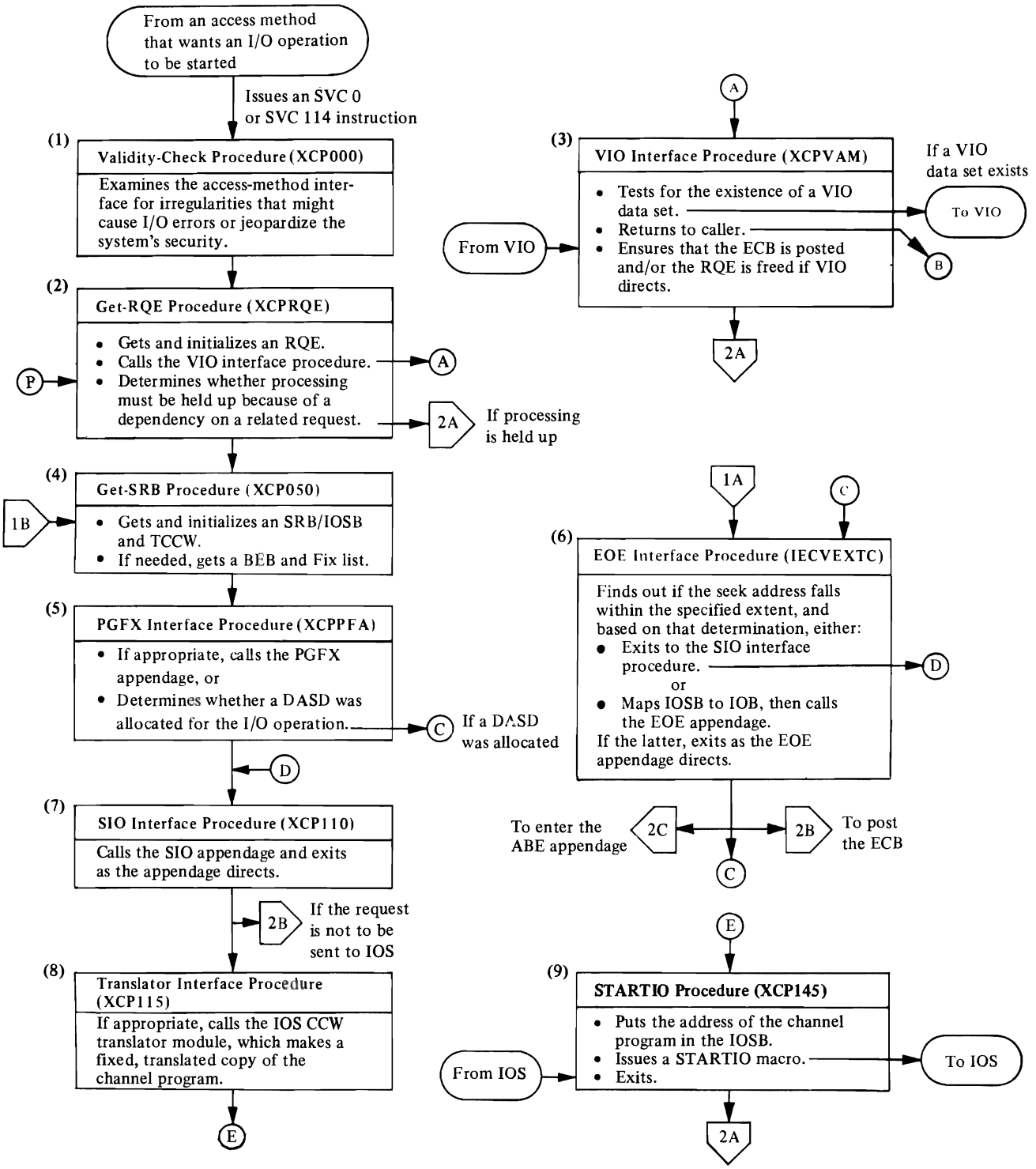


Figure 1. Flow of Control in the Basic EXCP Module (IECVEXCP) (Part 1 of 2)

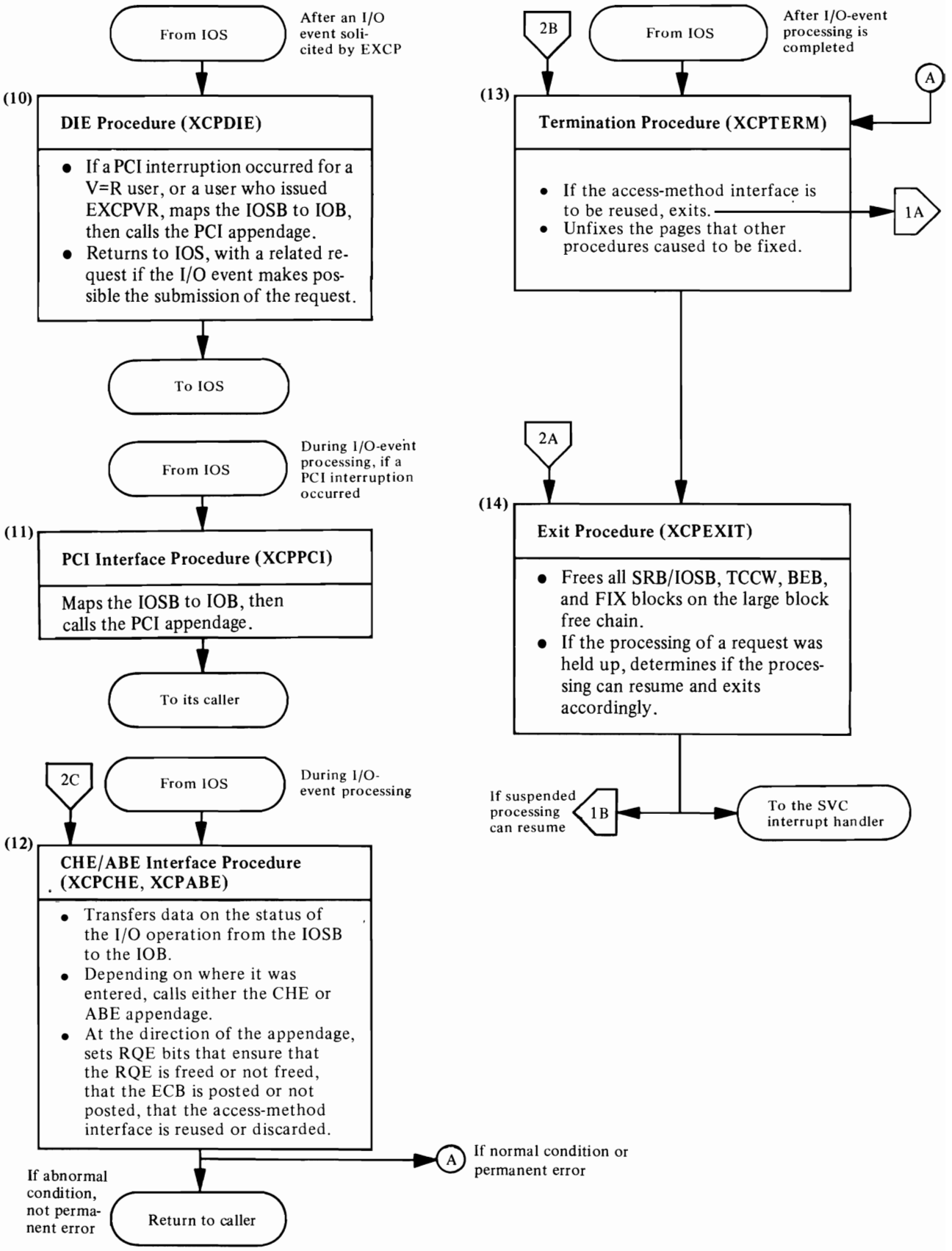


Figure 1. Flow of Control in the Basic EXCP Module (IECVEXCP) (Part 2 of 2)

Basic EXCP Module (IECVEXCP)

1. The Validity-Check Procedure (XCP000)

- Entered by the SVC interruption handler, which was entered by a caller issuing an SVC 0, SVC 92, or SVC 114 instruction. (EXCP is actually entered at IGC000, IGC092, or IGC114, depending on the SVC instruction, where it does some preliminary processing.)
- If entered via an SVC 92 instruction, determines if the caller is in supervisor state. If not, issues an ABEND macro with X'15C' code.
- If entered via an SVC 114 instruction, determines whether the caller is in supervisor state, or is authorized by the authorized program facility, or has a system protection key (0 through 7). If none of these is true of the caller, issues an ABEND macro with a X'172' code.
- Compares the pointers to the DCB in the IOBDCB and DEBDCB fields. If they aren't identical, issues an ABEND macro with a X'400' code.
- If the caller's protection key is greater than 7 (as are all user protection keys), does these things:
  (a) Verifies that the IOB, ECB, and DCB are in the caller's key. If a program check occurs, module IECVEXPR issues system completion code X'200'.
  (b) Calls the system's DEB validity-checking routine, IFGDEBCK. If notified that the DEB is invalid, issues an ABEND macro with a X'300' code.
- Compares the number of extents the IOB says a direct data set has to the number the DEB says it has. If the IOB's number is greater, issues an ABEND macro with a X'300' code.
- If the DCB pointers in the IOB and the DEB do not match, issues an ABEND macro with a X'400' code.
- Checks for a valid UCB. (The DEBUCBAD field in the DEB must point to an area whose third byte is X'FF'—a UCB). If it finds an invalid UCB, issues an ABEND macro with a X'500' code.
- If the DEB has multiple extents (such as ISAM), the DEBEXSCL field in the DEB is multiplied by the IOBM field in the IOB to get the correct extent. Then the UCB is checked as described above.
- Exits to the *get-RQE procedure (2).*

2. The Get-RQE Procedure (XCPRQE)

- Entered by the *validity-check procedure (1).*
- Calls the IOS storage manager module to get storage for an RQE. Initializes the RQE and chains it to a related request queue if the IOBUNREL bit is off.
- Exits to the *VIO interface procedure (3)* to find out if a VIO data set was allocated (if none was, control is returned); to the *get-SRB procedure (4)* if processing needn't be held up by a dependency on a related request; to the *exit procedure (14)* if processing must be held up.

3. *The VIO Interface Procedure (XCPVAM)*

- Entered by the *get-RQE procedure (2).*
- Tests the UCBJBNR bit for the existence of a VIO data set and, if it's on, enters the system's VIO component with a WIEXCP macro. If it's off, returns to the *get-RQE procedure (2).*
- Does the processing associated with the address the VIO component returns to. These are the possible return addresses and the associated processing:
  (a) *register 14+0:* Calls the *termination procedure (13)* to post the ECB and free the RQE.
  (b) *register 14+4:* Calls the *termination procedure (13)* to free the RQE.
  (c) *register 14+8:* Makes no call.
- Exits to the *exit procedure (14).*

4. *The Get-SRB Procedure (XCP050)*

- Entered by the *exit procedure (14)* when that procedure finds that a previously-delayed related request can proceed. Entered by the *get-RQE procedure (2)* if:
  (a) It's processing an unrelated request.
  (b) It's processing a type-2 or type-3 related request and three or less other such requests are ahead of it in the related request queue.
- Calls the IOS storage manager module (IECVSMGR) to get storage for an SRB/IOSB and a TCCW (translation control block). Also, for a virtual request, calls IECVSMGR for a BEB (beginning-end block, which contains the translated channel program) and a fix list. Initializes the SRB/IOSB.
- Determines whether the request is a type-3 related request or whether it is associated with a PCI appendage, which handles V=R requests for EXCPVR. If it is, puts the address of the *DIE procedure (10)* in the IOSDIE field. Otherwise, puts zeros there.
- Exits to the *PGFX interface procedure (5).*

5. *The PGFX Interface Procedure (XCPPFA)*

- Entered by the *get-SRB procedure (4).*
- If the EXCP was issued from a V=R address space, this procedure does not enter the page-fix appendage. (Page fixing is not needed, since buffers, CCWs, etc. are already in real storage.)
- Checks for the presence of a page-fix appendage and branches to it if the access method issued an EXCPVR macro or uses virtual storage addresses.
- If the access method issued an EXCPVR macro, calls the system's page-fixing routine to fix the pages in the appendage's fix list.
- Issues an ABEND macro with an X'800' code if the page-fixing routine returns with an error indication; otherwise, exits to the *EOE interface procedure (6),* if the I/O request is for a direct-access device, or to the *SIO interface procedure (7).*
- Fixes the DEB if it is not already fixed for V=R requests.
- Exits to the *EOE interface procedure (6).*

6. *The EOE Interface Procedure (IECVEXTC)*

- Entered by the *PGFX interface procedure (5)* at ECPEXT if the I/O request is for a direct-access device; and by the direct-access ERP if the ERP altered the seek address and wants the new one to be verified. For other devices, calls the *SIO interface procedure (7).*
- Finds out if the seek address falls within the specified extent. If not, goes to the EOE appendage, and when the appendage returns, does the processing associated with the address it returns to. These are the possible return addresses and the associated processing:
  (a) *register 14+0:* Puts X'42' in the IOBECBCC field and exits to the *CHE/ABE interface procedure (12)* so that the ABE appendage will be entered.
  (b) *register 14+4:* Exits to the *termination procedure (13)* so that the ECB will be posted and processing of the request terminated.
  (c) *register 14+8:* Branches to itself to recompare the seek address with the specified extent.
- Exits to the *SIO interface procedure (7).*

7. *The SIO Interface Procedure (XCP110)*

- Entered by the *EOE interface procedure (6)* and the *PGFX interface procedure (5).*
- Goes to the SIO appendage, and when the appendage returns, does the processing associated with the address it returns to. These are the possible return addresses and the associated processing:
  (a) *register 14+0:* Exits to the *translator interface procedure (8)* to continue processing the request.
  (b) *register 14+4:* Exits to the *termination procedure (13)* to prevent the request from reaching IOS.
  (c) *register 14+8:* Same as the register 14+0 return.

8. *The Translator Interface Procedure (XCP115)*

- Entered by the *SIO interface procedure (7).*
  If the access method is not running in a V=R address space or did not issue an EXCPVR macro, this procedure goes to the IOS CCW translator module (IECVTCCW), which makes a fixed, translated copy of the untranslated channel program.
- Issues an ABEND macro with a X'800' code if the CCW translator module returns an error code in register 15. Otherwise, exits to the *STARTIO procedure (9).*

9. The STARTIO Procedure (XCP145)

- Entered by the *translator interface procedure (8).*
- Puts the address of the channel program to be executed in the IOSB.
- Branches to the *exit procedure (14)* if the request must be held up by a related request. Otherwise, issues a STARTIO macro, giving the request to IOS.
- Exits to the *exit procedure (14)* when the IOS returns.

10. The DIE Procedure (XCPDIE)

- Entered by the basic IOS module (IECIOSCN).
- Goes to the PCI appendage only if the access method uses real storage addresses (i.e., execution is in a V=R address space) or if EXCP was entered with an EXCPVR macro. (First calls the *IOSB-to-IOB mapping procedure (15)* so that the PCI appendage can find information on the status of the I/O event in the IOB.)
- If a type-3 related request just completed without error, gives the request dependent on that completion, if any, to the basic IOS module to be started.
- Exits to the *basic* IOS module (IECIOSCN).

11. The PCI Interface Procedure (XCPPCI)

- Entered by the IOS post-status module (IECVPST).
- Calls the *IOSB-to-IOB mapping procedure (15)* to move information on the status of the I/O event from the IOSB to the IOB. Goes to the PCI appendage.
- Exits to IECVPST.

12. The CHE/ABE Interface Procedure (XCPCHE,XCPABE)

- Entered by IECVPST at XCPCHE under either of these conditions:

  (a) The IOSCSW field contains no status information other than a PCI, channel-end, device-end, attention, unit-exception, or wrong-length-record indication.

  (b) An ERP turned off IOSEX, the "exceptional-condition" bit, and IOSERR, the "retry" bit, in the IOSB it was processing.

  Entered by the IOS post-status module at XCPABE under either of these conditions:

  (a) The IOSCSW field contains a unit-check, channel-data-check, channel-control-check, or interface-control-check indication.

  (b) An ERP turned off the IOSERR bit, but left the IOSEX bit on.

  Entered by the *EOE interface procedure (6)* if so directed by an EOE appendage.
- Calls the *IOSB-to-IOB mapping procedure (15)* so that information on the status of the I/O event will be transferred from the IOSB to IOB.

- Depending on where it's entered, goes to the CHE or ABE appendage and does the processing associated with the address the appendage returns to. These are the possible return addresses and the associated processing:

  (a) *register 14+0:* Moves the IOB fields back to the IOSB.

  (b) *register 14+4:* Turns off the IOSEX bit and turns on the RQENOPST bit, telling the *termination procedure (13)* not to post the ECB.

  (c) *register 14+8:* Turns off the IOSEX bit and turns on the RQERETRY bit, telling the *termination procedure (13)* to ensure that the access-method interface is reused.

  (d) *register 14+12:* Turns off the IOSEX bit and turns on the RQENOPST and RQENOFRE bits, telling the *termination procedure (13)* not to post the ECB or free the RQE. (CHE and ABE appendages use this return if they called the exit effectors to schedule an asynchronous access-method routine. The RQE cannot be freed here because it must be available to the asynchronous routine when the routine is dispatched. The RQE is subsequently freed by the *SVC 3 interface procedure (17).*)

- Exits to the IOS post-status module if an exceptional condition (IOSEX flag on) and permanent error are not indicated.
- Exits to the *termination procedure (13)* if IOSEX is off, or a permanent error is indicated.

13. The Termination Procedure (XCPTERM)

- Entered by the following procedures:

  (1) The *CHE/ABE interface procedure (12)* if the request is ready for termination (i.e., the IOSEX flag is off, or the IOSEX flag is on and a permanent error is indicated).

  (2) The *EOE interface procedure (6)* to post the ECB with X'7F' if the EOE appendage disregarded an extent error.

  (3) The *SIO interface procedure (7)* is the SIO appendage wants to pervent the request from reaching IOS.

  (4) The *miscellaneous module (IECVEXPR) purge procedure (1)* so that data areas associated with a purged request will be freed.

  (5) The *post status* module after an error occurs during IOS processing resulting in an IOSCOD of X'45'.

- For tape devices, updates the block count in the DCB so that the system's close and EOV routines can use the block count in writing trailer labels for output data sets.

- If the access-method interface is to be reused, exits to the *EOE interface procedure (6)* without posting the ECB or freeing the RQE. (Exception: If the RQE represents a request that's being quiesced, calls the *miscellaneous module (IECVEXPR) restore chain procedure (2)* instead of exiting to the *EOE interface procedure (6).*)

- Ensures that the ECB is posted (unless the RQENOPST bit is on). If the system's posting routine returns with an error indication, issues an ABEND macro with an X'700' code.

- Frees the RQE (unless the RQENOFRE bit is on).
- Calls the *exit procedure (14)* to free the SRB/IOSB, TCCW, BEB, and FIX list.
- Calls the system's page-fixing routine, IEAVPSIB, to unfix pages.
- Enters RTM with a CALLRTM macro, if the completion code in the IOSCOD field is X'45', so that the access method can try to recover from an error that occurred while IOS was processing the I/O request. The completion code is obtained from the XDBA. If no XDBA exists then the completion code X'E00' is used.
- Exits to the *exit procedure (14)* if the access-method interface will not be reused, or to the *EOE interface procedure (6)* if it will; exits to the *related-request purge procedure (16)* if a related request resulted in an unsuccessful I/O operation.

14. The Exit Procedure (XCPEXIT)

- Entered by the *termination procedure (13)* if the access-method interface is not reused; by the *get-RQE procedure (2)* if the request is held up by a type-1 related request; by the *VIO interface procedure (3)*, following a return from the VIO component; by the *STARTIO procedure (9)*.
- Determines if the processing of a request dependent on a related request can proceed.
- Calls the IOS storage manager module to free the SRB/IOSB, TCCW, BEB, and FIX list.
- Exits to the *get-SRB procedure (4)* if a dependent request can continue to be processed; to the *related-request purge procedure (16)* if the DCBIFLGS field shows that a related request was marked in error; to the SVC interrupt handler (or to the dispatcher if entered by the *termination procedure (13)*) if no more requests can be passed to the *get-SRB procedure (4)*.

15. The IOSB-to-IOB Mapping Procedure (XCPMAP)

- Entered by the *DIE procedure (10)* and the *PCI interface procedure (11)* before they branch to the PCI appendage; by the *CHE/ABE interface procedure (12)* before it branches to the CHE or ABE appendage.
- Moves information about the status of the I/O event from the IOSB to the IOB for examination by a PCI, CHE, or ABE appendage.
- Exits to the return address in register 14.

16. The Related-Request Purge Procedure (XCPPUR)

- Entered by the *termination procedure (13)* if the DCB indicates that a related request failed. Also entered by ABE interface (12) if a related request failed.
- Looks at the related-request queue for requests that depend on the successful completion of the current one.

- Calls the *termination procedure (13)*, which ensures that ECBs are posted with X'48'.
- Exits to the *exit procedure (14)* when there are no dependent requests or after all dependent requests are quiesced.

17. The SVC 3 Interface Procedure (IECVX025)

- Entered by the system's SVC 3 routine, which gains control when an asynchronous access method routine issues an SVC 3 instruction.
- Since the appendage requested that the RQE not be freed at termination time, this procedure now frees the RQE that was passed as a parameter in scheduling the asynchronous routine.
- Returns to the SVC 3 routine.

Miscellaneous Module (IECVEXPR)

1. The Purge Procedure (IECVXPUR)

- Entered by the IOS nonresident purge module to process a request for either halt or quiesce.
- If a halt operation was specified, calls the *basic EXCP module (IECVEXCP) termination procedure (13),* which ensures that SRB/IOSBs, RQEs, TCCWs, BEBs, and FIX lists are freed, that fixed pages are unfixed, and that the ECBs of the purged requests are posted with X'48'.
- If a quiesce operation was specified, does the following:
  (a) Maintains a count of requests that haven't been completely processed.
  (b) Calls the *restore chain procedure (2)* each time it finds an I/O request to be quiesced (that is, each time it finds an RQE that matches the search argument in the IPIB).
- Exits to the IOS nonresident purge module.

2. The Restore Chain Procedure (IECVRCHN)

- Entered by the *purge procedure (1)* to extend a chain of IOBs that represent quiesced I/O requests. Entered by the *basic EXCP module (IECVEXCP) termination procedure (13)* or an ABE appendage if either wants an IOB to be added to a chain of IOBs.
- Creates a PIRL if none exists.
- Creates an EPCB if none exists.
- If the IOB, pointed to by the RQE, passed to it is the first, this procedure chains the IOB to the IPIB. Otherwise, chains the IOB to the end of the existing chain.
- Puts the address of the IOB in the EPCB, along with the IOB's protection key and information indicating under which TCB the request is to be restored. If the EPCB contains no free space, creates another EPCB and chains it to the last-created EPCB.
- Exits to the return address in register 14.

3. The Restore Procedure (IECVXRES)

- Entered by the IOS restore module, which was entered via a RESTORE macro.
- Examines the EPCB to find out (a) how each request was originally submitted to EXCP (whether with an EXCP or EXCPVR macro), (b) whether a request should be restored under the restore TCB or under a target TCB, and (c) under which key to restore a request. Based on this information, sets the appropriate key in the PSW and issues an SVC 0, 92, or 114 instruction for each IOB in the chain.
- Exits to the IOS restore module.

4. *The Halt-I/O Interface Procedure (SVC33)*

- Entered by the IOS nonresident halt-I/O module, which is entered when an IOHALT macro is issued.
- Confirms that the access method did not issue an EXCPVR macro and does not use real storage addresses by testing the RQETYPE field.
- Calls the IOS CCW translator module to get the address of the translated CCW corresponding to the untranslated CCW whose location was passed. If the CCW translator module returns an error code in register 15, exits to the IOS nonresident halt-I/O module with a return code of X'18' in register 15.
- Changes the command code of the translated CCW to a "no-op" and turns off its command-chaining bit.
- Exits to the IOS nonresident halt-I/O module.

5. *The Functional Recovery Procedure (XCPFRR)*

- Entered by RTM if another EXCP procedure issued an ABEND macro, an appendage took a program check, or another EXCP procedure took a program check.
- Gets storage for a debugging area (XDBA) and puts diagnostic data in it. If the system is disabled on entry, this storage is not obtained.
- If this is not the first recovery procedure to be called by RTM, puts an ABEND code of X'700' in the SDWA. Otherwise, puts one of the following codes in the SDWA:
  (a) *X'A00'*, if an appendage took a program check.
  (b) *X'200'*, if the *basic EXCP module (IECVEXCP) validity-check procedure (1)* took a protection check in determining whether an IOB, ECB, and DCB were in the caller's key.
  (c) *X'B00'*, if another EXCP procedure took an indeterminate program check.
- Sets bits in the SDWA directing RTM to free the local lock and issue a user dump (a SYSUDUMP, SYSMDUMP, or SYSABEND dump).
- Directs RTM to record the X'700', X'800', X'A00', and X'B00' abends or to record all abends (when the XDBA is absent).
- Uses the variable recording area of the SDWA to store diagnostic data.
- Exits to RTM.

Note: The format and contents of the debugging areas (XDBA and SDWA variable recording area) are described in the "Diagnostic Aids" chapter.

I/O Supervisor Method of Operation

This "Method of Operation" chapter contains a simplification of IOS code, divided into sections that correspond to basic IOS services:

- Starting an I/O operation
- Responding to an I/O event
- Restoring the availability of I/O resources
- Purging and restoring I/O requests
- Halting a teleprocessing operation
- Channel reconfiguration/channel set switching support

Each section is divided into topics that deal with functionally distinct parts of a service.

The flow of control between labeled parts of IOS is not stated in these sections. Rather, an order of events is implied by the order of topics within a section. If you want flow-of-control information, look at Figures 2-13 in the "Program Organization" chapter.

This table shows, for each IOS service, which object module performs the service and which figure shows the flow of control in the module:

Service	Module
Starting an I/O Operation (Figure 2 in the "Program Organization" chapter)	Basic IOS Module (IECIOSCN) Device Dependent SIO Modules Unit record (IECVXURS) 2305 (IECVXDRS) 2314 (IECVXSKS) 3330V (IECVXVRS) DASD (IECVXDAS) 2400 Tape (IECVXT2S) 3400 Tape (IECVXT3S)
Responding to an I/O event (Figures 3, 4 in the "Program Organization" chapter)	Basic IOS Module (IECIOSCN) DAVV Module (IECVDAVV) Device Dependent Trap Modules DASD (IECVXDAT) 2305 (IECVXDRT) Graphics (IECVXGRT) Tape (IECVXTAT) Teleprocessing (IECVXTPT) Unit record (IECVXURT) 3330V (IECVXVRT) Device Dependent Sense Modules 3851/3838 MSS (IECVXMGN) 2314 (IECVXSKN) Device Dependent End of Sense Modules 3211/3800 EOS (IECVXPRE) 2314 (IECVXSKE) Device Dependent Unsolicited Interruption Modules DASD (IECVXDAU) 3330V (IECVXVRU) Post Status Module (IECVPST)

Service	Module
Restoring the availability of I/O resources (Figures 5-8 in the "Program Organization" chapter)	I/O-Restart Modules (IECVRSTI) (IECVIRST) Build Reserve Table Module (IECVBRSV) Hot I/O Detection Module (IECVHDET) Hot I/O Recovery Module (IECVHREC) Re-drive I/O Module (IECVRDIO) Re-Reserve Devices Module (IECVRRSV) Special SIO Module (IECVESIO)
Purging I/O requests (Figure 9 in the "Program Organization" chapter)	Nonresident Purge Module (IGC0001F)
Restoring I/O requests	Restore Module (IGC0001G)
Halting a teleprocessing operation (Figure 10 in the "Program Organization" chapter)	Nonresident Halt-I/O Module (IGC0003C) Resident Halt-I/O Module (IECIHIO)
Channel reconfiguration/Channel Set Switching support (Figures 11-13 in the "Program Organization" chapter)	CRH/CHS Module (IECVCINT)

How to Use This Chapter

Before using this chapter for the first time, you should know:

- The conventions for representing data areas (see the preface).
- How to read decision tables (see the preface).
- The conventions for associating labeled pieces of IOS code with a particular service.

Each descriptive functional heading is followed by a list of the modules and procedures that perform the function. For example, this notation

Module: IECIOSCN
Procedure: ETCH1

under the heading "Finding a Path for the I/O Operation" indicates that the procedure beginning at ETCH1 in module IECIOSCN finds a path for I/O operations.

Starting an I/O Operation

Starting an I/O operation is a five-step process:

1. IOS first tests the status of the allocated device and control unit to determine if an I/O operation can be started.
2. If the I/O operation can be started, IOS then finds a path for the operation; that is, it determines what channel set and channel to use.
3. If the device is a tape or direct-access device, the device-dependent SIO module next prefixes CCWs to the driver's channel program. These CCWs activate hardware features or position I/O-device parts.
4. IOS then attempts to start I/O activity with a start-I/O instruction.
5. Last, IOS examines the condition code set by the start-I/O instruction and routes control based on the setting.

Testing the "Startability" of an I/O Operation

Module: IECIOSCN
Procedure: IECHNSCH

IOS tests the status of the device and control unit, as shown in the UCB, to find out whether to go ahead with the I/O operation or postpone it:

IOSB

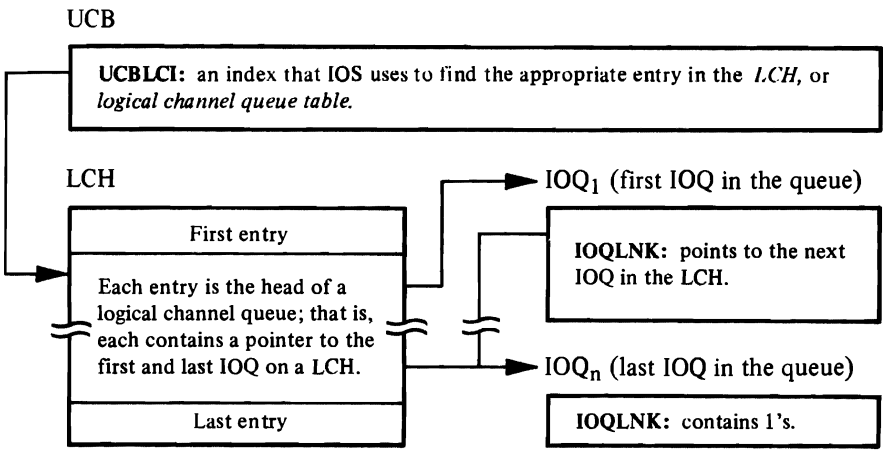
IOSUCB: points to the UCB for the device that has been allocated.

UCB

UCBFLA
UCBBSY: if on, the device is busy.
UCBNRY: if on, the device is not ready.
UCBPST: if on, IOS has associated the device with an earlier I/O request and hasn't yet disassociated it.
UCBPSNS: if on, a sense operation must be done to the device before it can be reused.
UCBCUB: if on, the control unit is busy.
UCBSAP: if on, the access arm of the device is being positioned for another I/O request.
UCBACTV: if on, a channel program is active on the device.
UCBQISCE: if on, use of the device is temporarily restricted. (IOS turns the bit on so that no driver can use the device until IOS determines that the correct direct-access volume is mounted.)

UCBJBNR
UCBURINP: if on, use of the device is restricted while unconditional reserve recovery is attempted.

If any status bit is on, IOS puts a record of the request, called an IOQ (input/output queue element), on the appropriate LCH (logical channel queue):



The other status bits tested are:

UCB

UCBFLB
UCBIORST: if on, and UCBBOX in field UCBJBNR is also c the device has been "boxed". This means that the device is unavailable for use, so the I/O request must be marked in error. This condition can be due to physical considerations, such as no paths to the device, or logical considerations, such as the possible loss of a device reservation.

If UCBBOX is off, and CRH is active, the last path to this device was from an inoperative processor.

UCBFLC
UCBITF: if on, a device notified IOS of a hardware error related to an I/O operation that IOS no longer knows about (because it returned the SRB/IOSB to the driver). IOS must use the SRB/IOSB of this request to process the error.
UCBUDE: if on, an unsolicited I/O event occurred for the device, probably because it was just readied by the operator. If it's a direct-access device, the volume label of a mounted volume must be read and verified before the device can be used. IOS creates an SRB/IOSB for the read operation.

If any of these is on, IOS schedules asynchronous processing, first setting up the IOSB to direct the subsequent flow of control. (The asynchronous processing that results from the UCBIORST or UCBITF bit being on is described under "Doing Asynchronous Processing with Driver-Created IOSBs." The asynchronous processing that results from the UCBUDE bit being on is described under "Doing Asynchronous Processing with IOS-Created IOSBs.")

Finding a Path for the I/O Operation

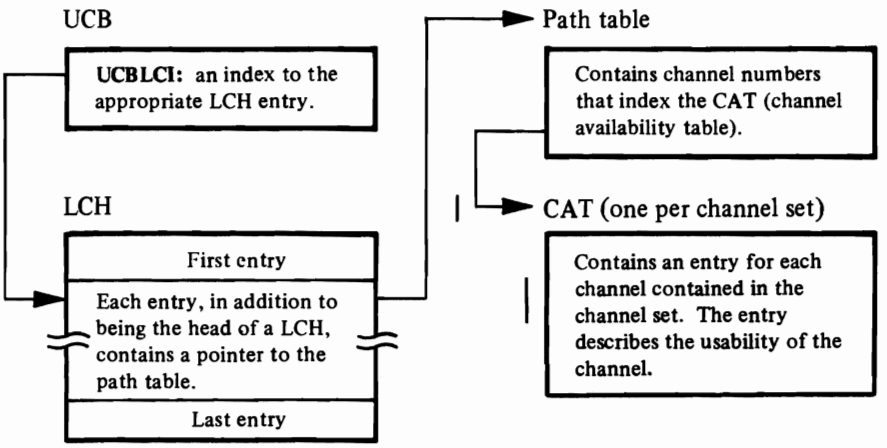
Module: IECIOSCN
Procedure: ETCH1

If there is nothing about the status of the device to warrant a postponement of the I/O operation, IOS looks for a path—that is, a processor (with channel set) to channel to control unit to device route—on which to start the I/O operation. It may find that the driver requires a specific path, as shown below:

IOSB

IOSPATH
IOSGDP: if on, the driver is requesting IOS to use a specific path, called a *guaranteed device path.* The device address is in the IOSPATH field; the channel set that is used to start the I/O operation is identified in the IOSAFF field.

If IOS is not given a path, it finds one using this chain of pointers:



Having obtained a path, IOS tests its availability with a TCH (test-channel) instruction. This instruction causes the condition code to be set according to the status of the channel. The following table explains what each condition code setting means and what action is taken:

Code	Meaning	Action
CC=0	The channel is available.	A start-I/O instruction is issued.
CC=1	The processor is unable to receive I/O interruptions; as soon as it's enabled, the channel tested will generate an interruption.	To prevent the interruption from being delayed, this is treated the same as CC=2.
CC=2	The channel is busy.	Alternate paths are tested; if they are busy, or if there are none, the IOQ is chained to the appropriate LCH to wait until the channel is not busy.
CC=3	The channel is not operational.	A start-I/O instruction is issued nevertheless.

Module: IECVXDRS (2305 SIO Module)
IECVXSKS (2314 SIO Module)
IECVXVRS (3330V SIO Module)
IECVXDAS (DASD SIO Module)
IECVXT2S (2400 Tape SIO Module)
IECVXT3S (3400 Tape SIO Module)

Before it issues a start-I/O instruction, IOS prefixes a CCW with a set-mode command—unique for each combination of density, parity (odd or even), data conversion (on or off), and EBCDIC/BCD translation (on or off)—to tape channel programs. This CCW activates hardware features that will be used in the I/O operation. Following this CCW is a transfer-in-channel CCW directed to the driver's program.

Three CCWs are prefixed to direct-access channel programs: a seek CCW, which positions the access mechanism to the specified cylinder and head; a set-file-mask CCW, which directs the device to accept or reject specified channel commands; and a transfer-in-channel CCW to the driver's channel program.

If the device is a 3330V, five CCWs are prefixed to the channel programs: a seek CCW; a transfer-in-channel CCW to the fourth CCW; a NOP CCW to handle cylinder faults; a set-file-mask CCW; and another transfer-in-channel CCW to the driver's channel program.

If the device is a 2314, a "stand-alone" seek CCW precedes the three-CCW prefix. The "stand-alone" seek CCW is not actually part of the prefix; it's a separate, one-CCW channel program. The fact that it "stands alone" from the prefix allows the control unit to disconnect from the channel when it receives the CCW, which lets the channel do other work while the access arm is positioned.

IOS can tell what kind of prefix to build and what to put in it by checking the UCB and IOSB:

UCB

UCBFL5
UCBSASK: if on, IOS builds a "stand-alone" seek CCW.

IOSB

IOSFMSK: contains the set-mode command or file mask.

IOSEEKA: contains the seek address.

IOSRST: contains the real storage address of the channel program.

A prefix to a direct-access channel program can contain additional CCWs, depending on whether the device is shared by both processors, whether it is currently reserved to one of them, and whether a system or user program has asked to reserve it. This information is found in the UCB:

UCB

UCBTBYT2
UCBRR: if on, the device is shared by loosely-coupled processors.

UCBFLB
UCBRESVH: if on, the device is currently reserved to a processor in this system.

UCBCRHRV: if on, this device is reserved to the inoperative processor.

UCBFL4
UCBRRP: if on, a reserve or release is pending.

UCBRESVP: has meaning only if UCBRRP is on; if on, a reserve pending; if off, a release is pending.

UCBSQC: the count of requests to reserve the device.

If the device is a 2314, shared, and not reserved, a reserve CCW is put in front of the "stand-alone" seek CCW. This ensures that the other processor, in handling another I/O request, does not reposition the access arm in the interval between the end of the seek and the start of the driver's channel program. Additionally, if the count of requests to reserve the device is zero, a release CCW is put in front of the driver's channel program.

For direct access devices (not 2314 or 2305), the SIO Module puts a reserve CCW at the beginning of the prefix if the device is not reserved (UCBRESVH=0, or UCBRESVH=1 with UCBRRP=1 and UCBRESVP=1) but the UCBSQC field shows that some program wants it to be reserved. If the SIO module finds the opposite indication in the UCB (UCBRESVH=1 and UCBSQC=0), it puts a release CCW at the beginning of the prefix.

Note: Some devices and device features cause IOS to build as "stand-alone" CCWs certain CCWs that are usually prefixed to a driver's channel program. For instance, a set-mode CCW for some tape devices is executed separately from the driver's channel program, as is a reserve or release CCW for a 2305 device. These "stand-alone" CCWs allow the channel to disconnect itself.

Starting I/O Activity

Module: IECIOSCN
Procedure: ESIO1

IOS issues one of two start-I/O instructions: SIO or SIOF (start-I/O fast release). SIOF is issued unless:

- The I/O request is for a shared, not reserved, direct-access device.

or

- The channel program is a "stand-alone" seek, reserve, or release CCW.

The following table explains what the condition codes set by SIO and SIOF mean:

Code	SIO Meaning	SIOF Meaning
CC=0	The I/O operation started.	The channel accepted the instruction and released the processor to do other processing. (If the channel can't start the I/O operation, it notifies IOS by setting the deferred-condition-code in the CSW to 01 and generating an I/O interruption.)
CC=1	Either an immediate operation completed *or* the channel has status information to present *or* the device or control unit is busy.	Some condition on the path to the control unit, excluding "channel busy," prevented the channel from accepting the request. (For example, the channel may have status information to present or may have found that the control unit is busy.)
CC=2	The channel is busy.	The channel is busy.
CC=3	The channel or device is not operational.	The channel or device is not operational.

Responding to the Condition Code Setting

Marking I/O Resources "Busy"

Module: IECIOSCN
Procedure: EPOSTIO1

If the condition code is 0, IOS turns on these UCB bits:

UCB

UCBFLA
UCBBSY: turned on to show the device is busy.
UCBPST: turned on to show the device is associated with an I/O request.
UCBACTV: turned on to show a channel program is active on the device.

Additionally, the channel is marked busy in the CAT only if it is a selector channel.

Module: IECIOSCN
Procedure: EPOSTIO1

If the condition code is 1, IOS makes the tests and takes the actions shown in this table:

TESTS	Is the channel holding error data and unable to store it until the processor is enabled for interruptions? (Is the channel-logout-pending bit on in the CSW?)	Yes	No	No	No	No	No
	Is the busy bit on in the CSW?	N/A	Yes	No	Yes	Yes	Yes
	Is the status-modifier bit on in the CSW?	N/A	No	N/A	Yes	No	N/A
	Does the CSW contain status information other than a busy, status-modifier, or channel-logout-pending indication?	N/A	Yes	Yes	N/A	No	Yes
ACTIONS	Tries to start the I/O request on another path.	X					
	Puts the I/O request on a logical channel queue and branches to the code that processes I/O events. (The processing of I/O events is described under "Responding to I/O Events.")		X				
	Branches to the code that processes I/O events. (An immediate operation ended.)			X			X
	Tests for the existence of an alternate control unit. If there is one, tries to start the I/O request on a path that includes the control unit. Otherwise, puts the I/O request on a logical channel queue.				X		
	Puts the I/O request on a logical channel queue.					X	

Trying to Start on Another Path

Module: IECIOSCN
Procedure: ETCH1

IOS tries to start an I/O operation on another path if:

- The condition code is 1 because of a pending channel logout.
- The condition code is 1 because of a busy control unit, but an alternate control unit exists.
- The condition code is 2 (channel is busy) or 3 (channel is not operational).

To see whether there is another online path from the same processor to the device, IOS examines the UCB:

UCB

UCBCHM: contains a subfield, four bits long, representing up to four paths to the device (two from each channel set). Each zero-bit stands for an available online path.

If the I/O operation can't be started on another online path from the same processor, IOS examines the CAT that belongs to the other channel set. If the CAT shows that the other channel set has a free channel to the device, IOS puts the I/O request on a logical channel queue and issues an RPSGNL macro, asking the other processor to try the I/O operation. (This transfer to the other processor is called a *shoulder tap.*)

If the condition code is 3, in addition to looking for another path, IOS checks the UCB to find out if the operator was told that the path is inoperative:

UCB

UCBPMSK: contains four bits, corresponding to the bits in UCBCHM. A one-bit means a message (IEA001I) was sent to the operator, telling him the path is inoperative.

If the operator hasn't been informed, IOS creates an SRB/IOSB and schedules the asynchronous processing that causes message IEA001I to be issued. (The asynchronous processing is described under "Doing Asynchronous Processing with IOS-Created IOSBs.") IOS also invokes procedure EDETECT1 in IECIOSCN to determine whether unconditional reserve recovery is needed.

Handling Channel Errors

Module: IECIOSCN
Procedure: ESTATUS1

If IOS finds an indication of a channel error in the CSW, it creates an ERP work area and gives control to CCH, which examines the error and puts its findings in the ERPIB field of the ERP work area. These CSW bits indicate a channel error occurred.

- The channel-data-check bit.
- The channel-control-check bit.
- The interface-control-check bit.

IOS also invokes procedure EDETECT1 in module IECIOSCN to determine whether unconditional reserve recovery is needed.

Handling Attention Interruptions

Module: IECIOSCN
Procedures: ESTATUS1
EATTENT1

If the attention bit is on in the CSW, IOS ensures that attention processing exists by examining the UCB:

UCB

UCBATI: if nonzero, an index to the entry in the attention table exists for the interrupting device. Zero means there is no entry for the device; that is, there is no provision for attention processing.

If the device is not represented in the attention table, IOS does no further attention processing. Otherwise, IOS makes the tests and takes the actions shown in this table:

TESTS	Was an I/O operation started on the device? (Is the UCBPST bit on?)	Yes	Yes	No
	Does the attention routine owner choose to do attention processing in a channel-end or abnormal-end appendage rather than in a separate attention routine? The flag tested is in the attention table entry.	Yes	No	N/A
ACTIONS	Does no further attention processing.	X		
	Creates an SRB/IOSB and uses the SRB to schedule the asynchronous processing that gives control to the appropriate attention routine. (The asynchronous processing is described under "Doing Asynchronous Processing with IOS-Created IOSBs.")		X	X

Note: Some devices cause the device-end bit in the CSW to be turned on when they require attention processing. Thus, if IOS receives a device-end indication and knows the device was doing no work (the UCBBSY bit is off), it handles the indication as it would an attention interruption, with this addition: the UCBUDE bit is turned on to signify that device-dependent processing might be required when an I/O request is next received for the device.

For the 3330V device, module IECVXVRU (the 3330V unsolicited interruption module) gets control when the attention occurs for device dependent processing in order to detect the resolution of a cylinder fault.

Handling Unit-Check Interruptions

Module: IECIOSCN
Procedures: ESTATUS1
ESENSE1

A unit-check indication in the CSW signifies a hardware-detected error. If IOS finds this indication, it makes the tests and takes the actions shown in this table:

TESTS	Did the error occur while a channel program was being executed? (Is the UCBPST bit on?)	Yes	No	No
	Is the error related to a solicited I/O event that has already been processed?* (Is the UCBBSY bit on?)	N/A	Yes	No
ACTIONS	Gets an ERP work area and reads sense information into it.	X	X	X
	Discards the ERP work area after sense information has been read into it.**			X
	Chains the ERP work area to the IOSB and uses the SRB to schedule asynchronous processing. (The asynchronous processing is described under "Doing Asynchronous Processing with Driver-Created IOSBs.")	X		
	Chains the ERP work area to the UCB and turns on the UCBITF bit to indicate that asynchronous processing must be scheduled when an I/O request (SRB/IOSB) is next received for the device.		X	

* The answer is "yes" in both these circumstances: (a) a driver requested an I/O operation for which two I/O events occurred, the first showing a channel-end indication, the second (the current one) showing device-end and unit check indications; and (b) the driver's SRB/IOSB no longer exists, having been used to process the first I/O event.

** The object in obtaining sense information, in this case, is not to provide data for an ERP, but merely to break the *contingent connection*—a connection between the control unit and device that prevents access to any other device attached to the control unit.

Note: Certain devices require device dependent processing before and/or after the sense operation.

- The 2314 has two modules: IECVXSKN, which gets control before the sense, and IECVXSKE, which gets control after the sense operation.
- The 3851 and 3838 use module IECVXMGN to do device-dependent processing before the sense operation.
- The 3211 and 3800 use module IECVXPRE for device-dependent processing after the sense operation.

Going to the Driver's DIE Procedure

Module: IECIOSCN
Procedures: ESTATUS1
EDIEINT1

IOS examines the UCB to find out if the I/O event is solicited:

UCB

UCBPST: if on, shows that an I/O operation was started for a driver—the I/O event is solicited. If off, indicates an unsolicited I/O event.

Having decided that a driver's I/O request is responsible for the I/O event, IOS branches to the appropriate trap module (IECVXxxT), which is pointed to by the device descriptor table (DDT) for this UCB. The trap module does device-dependent processing and returns. Then IOS branches to the driver's DIE procedure. The DIE procedure may return to IOS with a new I/O request (a new SRB/IOSB).

Using Channels That Are Free

Module: IECIOSCN
Procedure: ERSTART1

IOS, by examining the "channel mask" field of the IRT, IRTCHMSK, determines which channels are free to be used in I/O operations. The channel that last presented a channel-end indication to IOS is represented as free in the channel mask, as well as any channel found to be busy from one processor, during a test of available paths from that processor, but found to be free from the other processor.

IOS locates the queues of waiting I/O requests associated with the free channels. It then enters the code that finds paths for I/O operations, with the purpose of starting I/O operations for requests on those queues. The requests are handled in this order:

1. Requests for "stand-alone" seek operations.
2. Requests for sense operations. (The IOQs for these are on *physical* channel queues—one for each channel in the system—rather than logical channel queues.)
3. Requests for data-transfer operations.

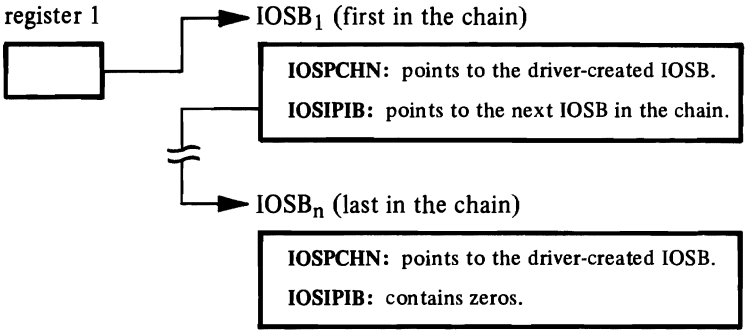
Doing Asynchronous Processing with IOS-Created IOSBs

IOS examines the IOSB pointed to by register 1. If the IOSIOSB bit is on, IOS knows (a) that the IOSB was created by another part of IOS code and (b) that processing can continue on one of four paths—to a PCI appendage, to an attention routine, to the stage 2 exit effector, or to unconditional reserve recovery. The path taken is governed by the contents of an IOSB field, IOSPROC.

Going to the PCI Exit

Module: IECVPST
Procedure: PSTIOSB

If the IOSPROC field contains IOSAPCI (X'04'), IOS enters the driver's PCI exit as many times as it finds IOS-created IOSBs in this chain:



Going to an Attention Routine

Module: IECVPST
Procedure: PSTIOSB

If IOS is processing an IOS-created IOSB like the following one, it branches to an attention routine:

IOSB

IOSPROC: contains IOSATTN (X'08'), directing IOS to enter an attention routine.
IOSPGAD: contains the address of the attention routine.

Going to the Stage 2 Exit Effector

Module: IECVPST
Procedure: PSTIOSB

Values of IOSADAVV (X'10') and IOSAWTO (X'14') in the IOSPROC field direct IOS to enter these respective parts of the system:

- Another part of IOS code—one that ensures that the correct direct-access volume is mounted on a device. (See "Verifying That the Correct Direct-Access Volume Is Mounted" below for more information.)
- The ERP message writer, to inform the operator that an I/O path, assumed to be online from UCB indicators, was found to be offline.

IOS cannot communicate with these parts directly, but a system routine called the ERP loader (IECVERPL) can. IOS ensures that the ERP loader is dispatched by branching to the stage 2 exit effector (IEA0EF00). The stage 2 exit effector begins the process of getting the ERP loader dispatched by putting the SRB on an asynchronous exit queue; then it returns to IOS. Later, the stage 3 exit effector will requeue the SRB to an SIRB (supervisor interruption request block) and cause the dispatcher to give control to the ERP loader.

Going to Unconditional Reserve Recovery

Module: IECVPST
Procedure: PSTIOSB

If the IOSPROC field contains IOSUR (X'20') and detection has not been done (IOSUSE is 0), IOS enters the unconditional reserve detection routine IECVURDT. If IECVURDT indicates that recovery is not to continue, the device is released and queued requests are redriven. If recovery is to continue, IOS enters the unconditional reserve decision routine IECVDURP.

Verifying That the Correct Direct-Access Volume Is Mounted

Module: IECVDAVV
Procedure: IECVDAVV

IOS, if entered via the exit effectors and ERP loader, checks the UCB for the label of the volume that was mounted when the device was last used:

UCB

UCBVOLID: if a volume label is here, the system expects that the volume is being used.

IOS builds a channel program to read the label of the volume currently mounted, and issues an SVC 15 instruction—causing another part of IOS code to issue a STARTIO macro for the channel program.

ERP work area

EWDSAVS: the current volume label is read here.

If the contents of UCBVOLID and EWDSAVS don't match, or if the current volume serial number can't be read because of a hardware problem, IOS asks the operator to demount the EWDSAVS volume and mount the UCBVOLID volume. Should the device lose its ready status before the read operation takes place, IOS asks the operator to restore the ready status with an "intervention required" message.

Doing Asynchronous Processing with Driver-Created IOSBs

If the IOSB pointed to by register 1 was created by a driver, IOS routes control to one or more of three driver exits and, if necessary, to an ERP. The path taken is governed by the contents of the CSW (stored in the IOSCSW field) and by the IOSEX and IOSERR settings in the IOSB.

Calling the PCI, NRM, and ABN Exits

Module: IECVPST
Procedure: IECVPST

The following table shows how IOS decides which of the driver's exits to call:

TESTS	Does the IOSB indicate a requested I/O operation was never started? (Does the IOSCOD field contain IOSMIHCA (X'51') or IOSFINTC (X'7E')*	Yes	No	No	No	No
	Is the PCI bit on in the IOSCSW field?	N/A	Yes	No	Yes	No
	Does the IOSCSW field contain any status information other than a PCI, channel-end, device-end, attention, unit-exception, or wrong-length-record indication?	N/A	Yes	Yes	No	No
ACTIONS	Calls the PCI exit.		X		X	
	Calls the NRM (channel-end) exit.				X	X
	Calls the ABN (abnormal-end) exit.	X	X	X		

* If the code that tests the startability of an I/O operation finds the UCBIORST or UCBITF bit on, it considers the operation unstartable and puts X'51' or X'7E', respectively, in the IOSCOD field.

Going to an ERP

Module: IECVPST
Procedure: IECVPST

IOS goes to an ERP if (a) the NRM or ABN exit uses the return address given it in register 14 and (b) the IOSB looks like this:

IOSB

IOSFLA
IOSEX: on, signifying that IOS or an exit detected an "exceptional condition" (a possible error condition).

IOSOPT
IOSNERP: off, signifying that IOS has the driver's permission to route control to an ERP.

IOSCOD: contains some value other than IOSMIHCA (X'51'). (Since X'51' indicates that the device is inaccessible, there is no point in doing ERP processing.)

If no ERP work area exists—the case if the CSW showed no indication of a unit check or channel error—IOS creates one before going to the ERP.

If the I/O operation was done on a direct-access device, the ERP for direct-access devices is entered directly, with a branch instruction. All other ERPs are entered indirectly: IOS branches to the stage 2 exit effector (IEA0EF00), which ensures that the appropriate nondirect-access ERP is given control asynchronously via the ERP loader (IECVERPL).

Going to the NRM or ABN Exit after ERP Processing

Module: IECVPST
Procedure: IGC015

If an ERP returns to IOS with IOSERR, the "retry" bit, off, it is telling IOS that it either corrected an error (ex: intervention required) or found a permanent error (ex: data check-parity error). To determine which of these happened, IOS tests the "exceptional-condition" bit, IOSEX. If the bit is off, the ERP corrected an error; IOS branches to the driver's NRM exit. If the bit is on, the ERP found a permanent error; IOS branches to the driver's ABN exit.

Reusing the STARTIO Interface

Module: IECVPST
Procedure: IGC015

IOS issues a STARTIO macro, requesting that an I/O operation be started, under any of these conditions:

- The NRM exit, ABN exit, or direct-access ERP returned with an indication that it wants IOS to retry or present a new I/O operation. (Register 0 points to the SRB that IOS is to use.)
- A nondirect-access ERP turned on IOSERR, the "retry" bit, in the IOSB it was processing and entered IOS with an SVC 15 instruction. (The IOSSRB field points to the SRB that IOS is to use.)
- Another part of IOS code (described under "Verifying That the Correct Direct-Access Volume Is Mounted") issued an SVC 15 instruction in order to read a direct-access volume label. (IOS uses the SRB it created when it found the UCBUDE bit on while testing the startability of an I/O operation.)

Restoring the Availability of I/O Resources

The availability of I/O resources is restored for the following conditions:

1. An ACR condition occurs for which CRH or CHS is not available or cannot be activated.
2. A channel or several channels become lost or unusable to the system.
3. A Hot I/O condition (continuous I/O interruptions from a device, control unit or channel) occurs due to a hardware malfunction.
4. A missing interruption condition occurs.

The recovery actions required for these conditions involve some common functions which are provided by generalized service routines as follows:

- Identifying devices reserved to a particular path.
- Re-reserving devices for which reserves have been reset.
- Redriving I/O requests.
- Calling the special SIO routine to perform synchronous I/O to devices without normal IOS services.

Restoring the Availability of I/O Resources for an ACR Condition

Module: IECVRSTI
Procedure: ACRPROC

A SIGP instruction with the "initial program reset" option is issued to stop on-going communication between the failing processor and all devices connected to it.

Then a table is built of all devices reserved on channels that are no longer available.

The re-reserve device service routine is used to reserve devices using available channels if possible, or to box them (force them offline). Devices which have lost their last path are boxed. All I/O which was active on the failing processor's channels is redriven causing alternate paths to be used, if available, or the request to be terminated as a permanent error.

Recovering from Lost or Unusable Channels

Module: IECVRSTI (for lost channel with reset)
Procedure: LOSTCHAN

Module: IECVIRST (for lost channel without reset)
Procedure:

When a channel suffers an error which makes it unavailable to the system, the error is reported as an external damage machine check or as a channel check with an indicator in the channel logout. These conditions are detected by the channel check handler (CCH) which will schedule the appropriate IOS recovery routine.

When a channel is lost, the recovery action is based on whether a system reset to the channel has been performed by the hardware prior to reporting the error. A system reset on the channel causes the loss of reserves which are active on that channel.

- Lost channel with reset – Devices which have lost their last path are forced offline. Devices which were forced offline cannot be used by the system unless varied online by the operator. I/O which was active on the failing channel is redriven causing alternate paths to be used, if available, or the request to be terminated as a permanent error.

- Lost channel without reset – The build reserve table service routine is invoked to find all devices reserved on the failing channel(s). IOS issues the clear channel instruction (CLRCH) to reset the channel. If the channel(s) is usable after the reset, it is marked available in the channel availability table (CAT).

  The re-reserve device service routine (IECVRRSV) is invoked to reserve devices that were identified by the build reserve table service routine (IECVBRSV). I/O that had been active on the failing channel(s) is redriven by invoking the redrive I/O service routine (IECVRDIO).

Restoring the Availability of I/O Resources after A Hot I/O Condition

Module: IECVHREC
Procedure:

Hot I/O conditions are detected by an IOS detection routine (IECVHDET) which checks for the occurrence of continuous I/O interruptions and determines that recovery is required based on threshold values for channels, control units, and devices. It will schedule Hot I/O Recovery (IECVHREC) indicating the probable source of the hot interruptions.

Hot I/O Recovery consists of:

- Identifying devices which are reserved on the channel involved in the hot I/O condition, by calling the build reserve table service routine (IECVBRSV).

- Communicating with the operator to inform him that hot I/O recovery is in progress. The operator may be required to remove a device, control unit, or channel from the I/O configuration in order to allow the system to recover. In this case the operator must respond indicating the action taken.

- Boxing the affected devices and terminating in error I/O to those devices if a non-DASD device was the source of the hot interruptions and if the operator indicates that recovery consists of removing the device or the control unit.

- Performing recovery on a channel basis if the channel or a DASD device was the source of the hot interruptions or if the operator indicates a hot device or control unit is to be recovered by removing the channel. If the channel can be reset (via the CLRCH instruction or the operator), devices that were reserved are re-reserved by invoking the re-reserve devices service routine (IECVRRSV). I/O is then redriven by invoking the redrive I/O service routine (IECVRDIO). The channel is then re-enabled if it is the first occurrence of hot I/O on that channel and the channel was reset.

Recovery from Missing Interruption Condition

Module: IECVRSTI
Procedure: MIHPROC

When entered by MIH, IOS stops any on-going communication between a channel and device by issuing HDV (halt-device) and CLRIO (clear I/O) instructions. Field UCBCHAN identifies the channel used in the last I/O operation to the device. This is the channel to which the HDV and CLRIO instructions are issued.

IOS sets bits in the CSW and branches to the code that processes I/O events, thereby simulating an I/O event. The unit check bit is set in the CSW if a sense operation was the last to be started to the device. The device end bit is set in the CSW if a sense operation is planned but not started to the device. Turning the device-end bit on insures the execution of the IOS code that starts a planned sense operation to the device. Turning on the unit-check bit as well insures the execution of the IOS code that builds a sense CCW and starts the sense operation. This is, in effect, a retry of the sense operation for which there was no I/O interruption, but with one difference: IOS does not try to read the sense data; it only tries to satisfy the hardware requirement that a sense operation be completed.

IOS sets the channel control check bit and the interface control check bit if neither the unit check bit or device end bit were set. The channel control check bit and the interface control check bit in the CSW direct the IOS code that processes I/O events to move the temporary work area to the ERPIB.

Services Used in Restoring the Availability of I/O Resources

Modules: IECVBRSV
IECVRRSV
IECVRDIO
IECVESIO

Build Reserve Table Routine (IECVBRSV) – This routine builds a reserve table RESVTAB containing all the devices which are reserved on a given channel.

Re-reserve Device Routine (IECVRRSV) – This routine attempts to box or reserve, based on the input parameter, all devices that are in the reserve table. If reserve is requested, devices that cannot be re-reserved are boxed. The re-reserve device routine uses the Special SIO Routine (IECVESIO) to issue the I/O to reserve the devices.

Redrive I/O Routine (IECVRDIO) – This routine scans all the UCBs on a given channel and simulates an interruption to re-drive those devices which are marked as active on the specified channel. Devices which no longer have an online path are boxed so that I/O requests will be posted in error.

The Special SIO Routine (IECVESIO) – This routine performs a synchronous I/O operation to a device without using normal system services. It will, however, use the RISGNL macro, if necessary, to initiate the I/O on a processor with a path to the device.

Purging and Restoring I/O Requests

Purging I/O requests is a two-step process:

1. IOS first locates the SRB/IOSBs representing requests the caller wants purged and then:
   - Puts the SRB/IOSBs into queues—one for each driver—if the caller requests a halt operation.

   or

   - Counts and marks the SRB/IOSBs if the caller requests a quiesce operation.

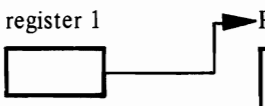
2. IOS then communicates with each of its drivers. (Each driver disposes of the SRB/IOSBs that IOS collected for it—the case when a halt operation is requested—or passes to IOS an address needed to restore I/O requests.)

To restore I/O requests, IOS points each driver to the address that it passed during the purge operation.

Comparing SRB/IOSBs to the Search Argument

Module: IGC0001F
Procedures: IGC016, SIRBPURG, LCHPURG, UCBPURG, DDRPURG, SPLPURG, IPIBPURG, PURAPLSR, BASICPRG

IOS determines what kind of purge operation the caller wants by looking in the PPL (purge parameter list):



PPLOPT1 and PPLOPT2: bit settings tell IOS (a) whether to do a halt or quiesce operation and (b) whether the operation applies to one data set, a group of data sets, a task, or an address space.

PPLDSIDA: a data-area address or the first in a chain of data area addresses if the operation applies to one data set or a group of data sets. The address or chain is used as the search argument.

PPLTCB: the address of a TCB if the operation applies to a task. The address is used as the search argument.

PPLASID: the identifier of an address space if the operation applies to an address space. The identifier is used as the search argument.

IOS looks at all the system data areas that SRB/IOSBs might be chained to and, on finding an SRB/IOSB, compares the search argument (or arguments) to one of three fields:

SRB/IOSB

SRBPTCB: to this field if the search argument is a TCB address.

IOSDSID: to this field if the search argument is a data-area address.

IOSASID: to this field if the search argument is an address space identifier.

SRB/IOSBs that match the search argument are called *applicable SRB/IOSBs.* IOS unchains these and organizes them into queues, one for each driver, if the PPL indicates a halt operation. If the PPL indicates a quiesce operation, IOS leaves the applicable SRB/IOSBs where they are, but increases a counter in the IPIB (I/O supervisor purge interface block) by the total of applicable SRB/IOSBs and puts the address of the IPIB in each one.

(Leaving applicable SRB/IOSBs where they are chained allows IOS and IOS-related code—ERPs and attention routines, for example—to continue processing the I/O requests they represent. When the processing of such an I/O request ends, the driver's termination procedure sees the IPIB address in the IOSB and decreases the counter in the IPIB. When the counter reaches zero, the quiesce operation is finished; the requestor of the quiesce operation can continue its processing with no risk of interfering with I/O operations.)

Communicating with the Drivers' Purge Procedures

Module: IGC0001F
Procedures: IGC016
DVRPURG

If a halt operation is specified, IOS stores the address of one of the SRB/IOSB chains in the IPIB and branches to the purge procedure of the driver that created the SRB/IOSBs.

IPIB

IPIBSRB: address of the SRB/IOSB chain stored here.

IOSB

IOSDVRID: identifies the driver that created it. IOS uses it as an index into the VOID (vector of IOS drivers) table, where the addresses of the drivers' purge procedures are stored.

If a quiesce operation is specified, IOS still branches to the drivers, each of which is responsible for returning to IOS an address—called a *restore* address—that the driver needs later to restore purged I/O requests.

IPIB

IPIBIO: drivers return a restore address here.

PIRL (purged I/O restore list)

PIRRSTR: IOS moves the restore addresses into the PIRL, beginning here.

Pointing Drivers to Their Restore Addresses

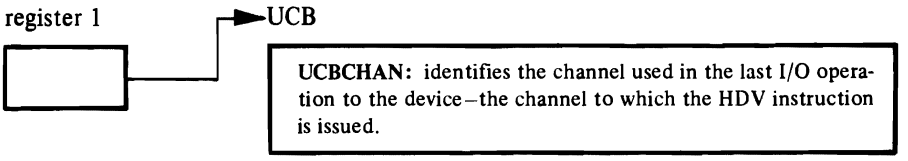
Module: IGC0001G
Procedure: IGC017

When a RESTORE macro is issued, IOS gives to each driver a pointer to the part of the PIRRSTR field that contains its restore address.

Halting a Teleprocessing or Channel-to-Channel Operation (CTC)

Module: IGC0003C
Procedure: IGC0003C

IOS examines the high-order byte of register 1 to determine whether the caller wants IOS or the EXCP processor to terminate the channel program. If the former, it issues an HDV (halt-device) instruction to the channel. If the latter, it branches to the EXCP processor.



In the case of a CTC device, the HDV instruction is issued regardless of whether an operation is active for the specified device.

Overview of Channel Reconfiguration (CRH) Support

(*Note:* For detailed logic information, refer to the descriptions of the individual CRH procedures, via the procedure names in the index.)

The Channel Reconfiguration Hardware (CRH) is a hardware feature of the System/370 168MP which, under system control, enables the operative processor to access the channels of the inoperative processor. These channels are accessed through the interface for channel 6 on the operative processor.

Using the Channel Reconfiguration Hardware

The CRH software issues a Diagnose instruction to connect the channel-6 interface to a specified channel of the inoperative processor. If, for example, IOS issues an SIO to address 672 after a Diagnose instruction for channel 2, the device at address 272 on the inoperative processor is started. When the hardware connection is broken, an SIO to address 672 accesses device 672 on the operative processor.

A CRH event (interruption) is solicited (a channel of the inoperative processor is "polled") through this sequence of events:

- Issuance of the Diagnose instruction to make the CRH hardware connection to the desired channel.
- Setting up of control register 2 so that interruptions are allowed only through the channel-6 interface on the operative processor.
- Enabling for I/O interruptions.

If channel 2 on the inoperative processor is polled, an interruption address of 672 (in the previous example) is stored in low storage at location FLCIOAA (X'B9' of the PSA). Procedure IECVCINT (the CRH interruption handler) changes this address to 272 because of bit settings that indicate that the CRH connection has been made to channel 2.

Activating the CRH Program

The CRH program is activated when one of the following conditions occurs:

- A hardware failure on one processor causes the alternate CPU recovery (ACR) facility to take the failing processor offline.
- The operator issues the first VARY channel online command for a channel attached to an offline processor.
- The operator issues a VARY processor offline with the KEEPCHAN option.

When one of these conditions occurs, IECVCRHA, the CRH activation procedure, is called. IECVCRHA indicates that CRH is active by setting a CVT pointer (CVTCRCA) to the CRH communication area (CRCA). IECVCRHA also activates the hooks in mainline IOS and sets a pointer that causes the CRH SLIH (IECVCINT) to execute after the I/O FLIH and before the I/O SLIH.

Passing Control to CRH on a Start I/O Request

(See Figure 11 in "Program Organization.")

When IOS prepares to start an I/O operation that needs a channel of the inoperative processor, the CRH hook module (IECVCRHH) receives control through hooks in the IOS mainline (IECIOSCN). The hooks are set up by IECVCRHA when CRH is activated. The CRH hook module gets control from three locations in IOS mainline:

- *In ETCH1 Procedure:* When no path is available from the operative processor, the test channel procedure can't shoulder tap to the other processor. Instead, the test channel hook procedure (IECVCRH1) is called to find a path.
- *In ESIO1 Procedure:* As soon as the SIO or SIOF instruction is issued, the SIO hook procedure (IECVCRH2) is called to determine whether the start I/O was issued through CRH. If so, IECVCRH2 does some additional CRH processing. (For details, see the "IECVCRH2 Procedure" in the Program Organization section.)
- *In ESENSE1 Procedure:* Just before the SIO for the sense operation is issued, the sense hook procedure (IECVCRH3) is called to determine whether the sense operation is to be done through CRH. The sense hook procedure issues the sense SIO, using the CRH connection if needed.

Passing Control to CRH on an I/O Event

(See Figure 12 in "I/O Supervisor Program Organization.")

When CRH is active and an I/O interruption occurs, the I/O FLIH calls the CRH SLIH (IECVCINT). The CRH SLIH processes the I/O event and then branches to the I/O SLIH (IECINT) for further processing of the I/O interruption. When IECINT returns control, the CRH SLIH polls (solicits pending interruptions from) the enabled channels of the inoperative processor, one at a time, and passes each interruption to IECINT. The CRH SLIH polls by enabling, and allowing a possible I/O interruption to be fielded by the I/O FLIH, the CRH SLIH, and the SIO SLIH. The process is repeated until all channels of the inoperative processor have been checked for pending interruptions.

Preventing Line Drops on TP Lines

The CRH facility includes a timer pop procedure to force the CRH SLIH to poll the channels of the inoperative processor often enough to prevent TP line drops. This ensurance is necessary because the CRH SLIH normally gets control only when an I/O interruption is taken on the operative processor. In a worst-case situation, when the inoperative processor's channels are handling most of the I/O, they would be polled infrequently. In this situation TP line drops would be possible. The CRH timer pop procedure solves the problem by causing an interruption to be taken on the operative processor at least once every two seconds. This frequency is sufficient to prevent a TP line drop.

Recovering from Errors

The CRH activation and deactivation procedures (IECVCRHA and IECVCRHD) use the FRRs that try to deactivate CRH. If deactivation is not possible, the FRR causes the RTM to percolate the error to the next FRR in the system.

The CRH SLIH (IECVCINT) uses an FRR to route control to the IOS mainline FRR (IECFRR) to post the IOS driver with a post code of X'45'. This post code indicates that the I/O request terminated abnormally in IOS because of a program check, or machine check.

The hook procedures use the IOS mainline FRR.

Deactivating CRH

CRH is deactivated in either of two cases:

1. The operator varies offline the last channel of the inoperative processor, or
2. The operator varies online the offline processor.

In either case the VARY processor (IEEVCPU) calls the CRH deactivation procedure (IECVCRHD). This procedure clears the CRH pointer in the CVT (CVTCRCA). CRH is then no longer active, since the CRH SLIH, the CRH hook procedures, and the CRH timer pop procedure check whether CVTCRCA is zero. If the field is zero, no CRH processing is done. The CRH deactivation procedure calls the BACKOUT subroutine in module IECVCINT, to remove the setup done by the CRH activation procedure.

Overview of Channel Set Switching (CHS) Support

(*Note:* For detailed logic information, refer to the descriptions of the individual CHS procedures, via the procedure names in the index.)

Channel Set Switching (CHS) is a feature under system control which allows the operative processor to access the channels of the inoperative processor. These channels are accessed through their normal interface, but on the operative processor.

Using Channel Set Switching

IOS issues a connect channel set instruction to connect the complete set of channels from any processor. If, for example, IOS issues an SIO to address 272 after a connect channel set instruction for the inoperative processor's channel set, the device at address 272 on the inoperative processor is started. When this channel set is disconnected and the operative processor's channel set is connected, an SIO to address 272 will access device 272 on the operative processor, if the device is symmetrically attached. Otherwise, the condition code from the SIO is 3 (not operational).

Channel polling as performed by CRH is not necessary with CHS. If the channel set is connected with the system enabled, interruptions are processed in the normal non-CHS way. On each interruption, however, the other channel set is connected prior to re-enabling.

Activating CHS

CHS is activated when one of the following conditions occurs:

- A hardware failure on one processor causes the Alternate CPU Recovery (ACR) facility to take the failing processor offline.
- The operator issues the first VARY channel online command for a channel attached to an offline processor.
- The operator issues VARY processor offline with KEEPCHAN parameter.

When one of these conditions occurs, IECVCRHA, the CRH/CHS activation procedure, is called. IECVCRHA indicates that CHS is active by setting a CVT pointer (CVTCRCA) to the CRH communication area (CRCA). A bit in the CRCA (CRCACSSA) indicates that CHS is active, rather than CRH. IECVCRHA also activates the hooks in mainline IOS and sets a pointer that will cause the CHS SLIH (IECVCSSI) to be executed after the I/O FLIH and before the I/O SLIH.

On an AP system, or on an MP with all channels in one channel set offline, CHS is not actually activated. Rather, the single channel set is connected to the operational processor. The operational processor then executes all I/O with no additional overhead.

Passing Control to CHS on a Start I/O Request

(See Figure 11 in "Program Organization".)

When IOS prepares to start an I/O operation that needs a channel of the inoperative processor, the following procedures are used in IOS mainline:

- In the ETCH1 Procedure: When no path is available from the operative processor, the test channel procedure can not shoulder tap the other processor. Instead, it connects the necessary channel set and then re-executes ETCH1.
- In the ESENSE1 Procedure: The sense procedure issues the sense SIO after connecting the necessary channel set.

Passing Control to CHS On an I/O Event

(See Figure 13 in "I/O Supervisor Program Organization.")

When CHS is active and an I/O interruption occurs, the I/O FLIH calls the CHS SLIH (IECVCSSI) instead of the I/O SLIH (IECINT). The CHS SLIH branches to IECINT for processing of the I/O interruption. When IECINT returns control, the CHS SLIH connects the other channel set. It then enables and passes the interruption to IECINT. The process is repeated until all channel sets have been cleared of pending interruptions.

Preventing Line Drops on TP Lines

CHS includes a timer pop procedure to force the CHS SLIH to connect the disconnected channel set at least every 2 seconds to prevent TP line drops. This is necessary because the CHS SLIH normally gets control only when an I/O interruption is active. The disconnected channel set could be connected infrequently and TP line drops would be possible.

Recovering from Errors

The CRH/CHS activation and deactivation procedures (IECVCRHA and IECVCRHD) use the FRRs that try to deactivate CHS. If deactivation is not possible, the FRR causes the RTM to percolate the error to the next FRR in the system.

The CHS SLIH (IECVCSSI) uses an FRR to route control to the IOS mainline FRR (IECFRR) to post the IOS driver with a post code of X'45'. This post code indicates that the I/O request terminated abnormally in IOS because of a program check or machine check.

Deactivating CHS

CHS is deactivated in either of two cases:

1. The operator varies offline the last channel of either channel set, or

2. The operator varies online the offline processor.

In either case the VARY processor (IEEVCPU) calls the CRH/CHS deactivation procedure (IECVCRHD). This procedure clears the CRH pointer in the CVT (CVTCRCA). CHS is then no longer active, since the CHS SLIH and the CRH/CHS timer pop procedure check whether CVTCRCA is zero. If the field is zero, no CHS processing is done. The CRH/CHS deactivation procedure calls the BACKOUT subroutine to remove the setup done by the CRH/CHS activation procedure.

Connect Channel Set Procedure

The connect channel set procedure (IECCONCS) issues the connect channel set instruction which causes the requested channel set to be connected to the issuing processor. This procedure is used by all system components which require the connecting of channel sets.

I/O Supervisor Program Organization

This chapter is organized by object module name and by procedure name within each object module.

This chapter has the following reference features to help you find information quickly.

- Each module is a section. The sections are arranged in alphabetical order, by module name.
- Each module (section) contains numbered procedures. Within any procedure, two types of cross-references are made: module and procedure. The type of cross reference is indicated by the word *module* or *procedure* within the procedure you are reading.

  When a procedure name and number is referenced, simply locate the numbered procedure within the module you are currently reading.

  When a module name and number is referenced, first locate the module, then locate the numbered procedure within that module.
- Place markers, printed at the top of each page, give the name of the module and the numbers of the procedures described on the page. To find the description of a given module or procedure, you can scan the place markers.

IOS is made up of 33 object modules. Three of these – those beginning with the characters IGC – are also load modules and execute from the link pack area. The other modules are link-edited at system generation into the nucleus, load module IEANUCxx.

The object modules are the program units that perform the services described in the "Method of Operation" chapter. This table shows which modules perform which services:

Service	Module*
Starting an I/O Operation (Figure 2)	Basic IOS Module (IECIOSCN) Device Dependent SIO Modules Unit record (IECVXURS) 2305 (IECVXDRS) 2314 (IECVXSKS) 3330V (IECVXVRS) DASD (IECVXDAS) 2400 Tape (IECVXT2S) 3400 Tape (IECVXT3S)

*Two modules, the CCW translator module (IECVTCCW) and the storage manager module (IECVSMCR), perform supporting operations.

Responding to an I/O event (Figures 3, 4)	Basic IOS Module (IECIOSCN) DAVV Module (IECVDAVV) Device Dependent Trap Modules DASD (IECVXDAT) 2305 (IECVXDRT) Graphics (IECVXGRT) Tape (IECVXTAT) Teleprocessing (IECVXTPT) Unit record (IECVXURT) 3330V (IECVXVRT) Device Dependent Sense Modules 3851/3838 MSS (IECVXMGN) 2314 (IECVXSKN) Device Dependent End of Sense Modules 3211/3800 EOS (IECVXPRE) 2314 (IECVXSKE) Device Dependent Unsolicited Interruption Modules DASD (IECVXDAU) 3330V (IECVXVRU) Post Status Module (IECVPST)
Restoring the availability of I/O resources (Figures 5–8)	I/O-Restart Modules (IECVRSTI and IECVIRST) Build Reserve Table Module (IECVBRSV) Hot I/O Detection Module (IECVHDET) Hot I/O Recovery Module (IECVHREC) Re-drive I/O Module (IECVRDIO) Re-Reserve Devices Module (IECVRRSV) Special SIO Module (IECVESIO)
Purging I/O requests (Figure 9)	Nonresident Purge Module (IGC0001F)
Restoring I/O requests	Restore Module (IGC0001G)
Halting a teleprocessing operation (Figure 10)	Nonresident Halt-I/O Module (IGC0003C) Resident Halt-I/O Module (IECIHIO)
Channel reconfiguration/ Channel Set Switching support (Figures 11-13)	CRH/CHS Module (IECVCINT)

This part of "Program Organization" is divided into sections, each bearing the name of a module. Each section tells what the module does by describing the module's functional pieces, or *procedures*. Each shows the flow of control into, out of, and within the module by identifying the calls made by the module's procedures, and the entrances to and exits from them. (The microfiche listings call the parts of modules *routines* or *subroutines*. *Procedure* is used here to avoid an unnecessary distinction.)

Some modules – those that are large and functionally the most important – are also represented in flow-of-control diagrams (see Figures 2-13). The diagrams don't have the detailed information that the individual chapter sections have. Rather, they show a simplification of processing and flow of control.

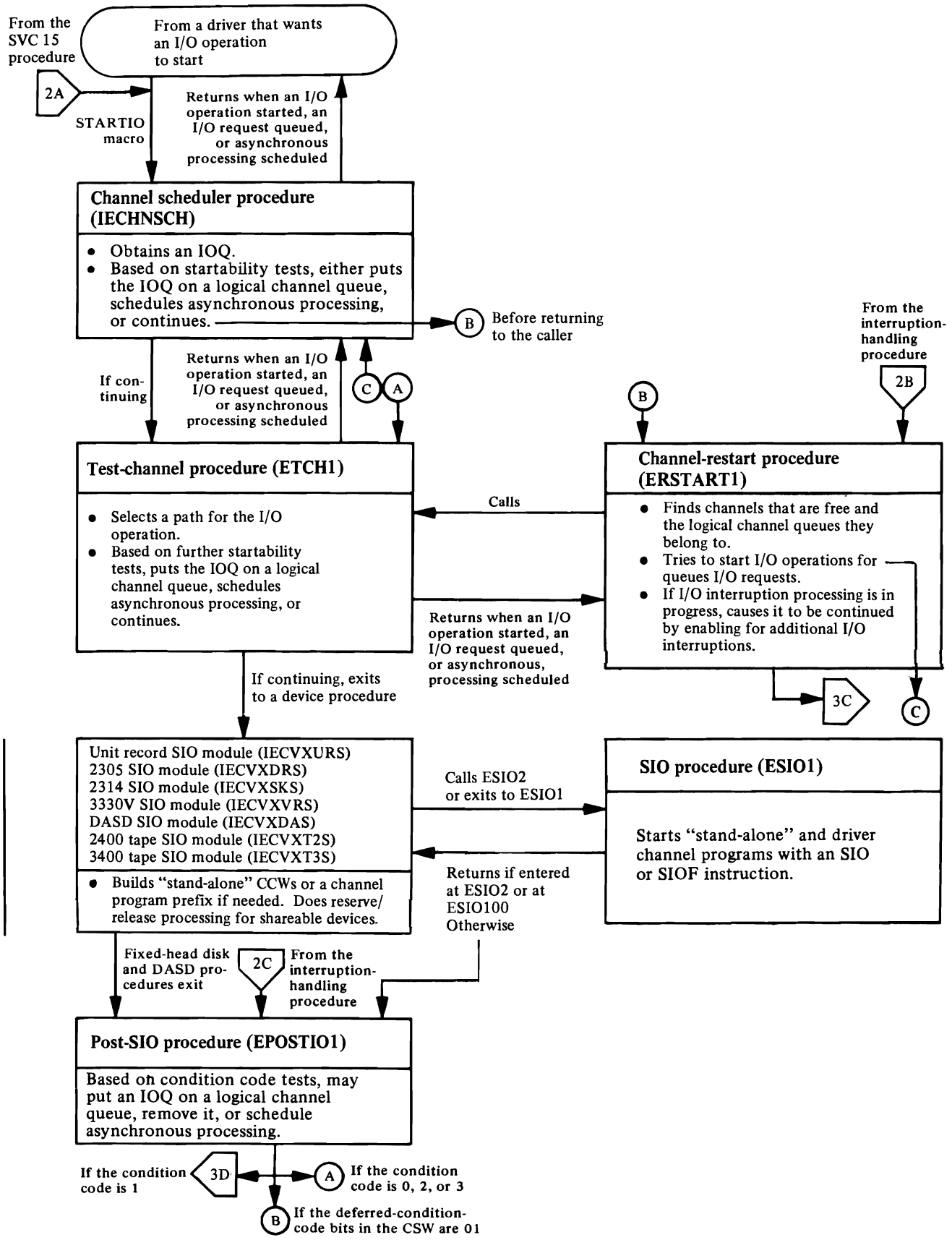


Figure 2. Starting I/O in the Basic IOS Module (IECIOSCN)

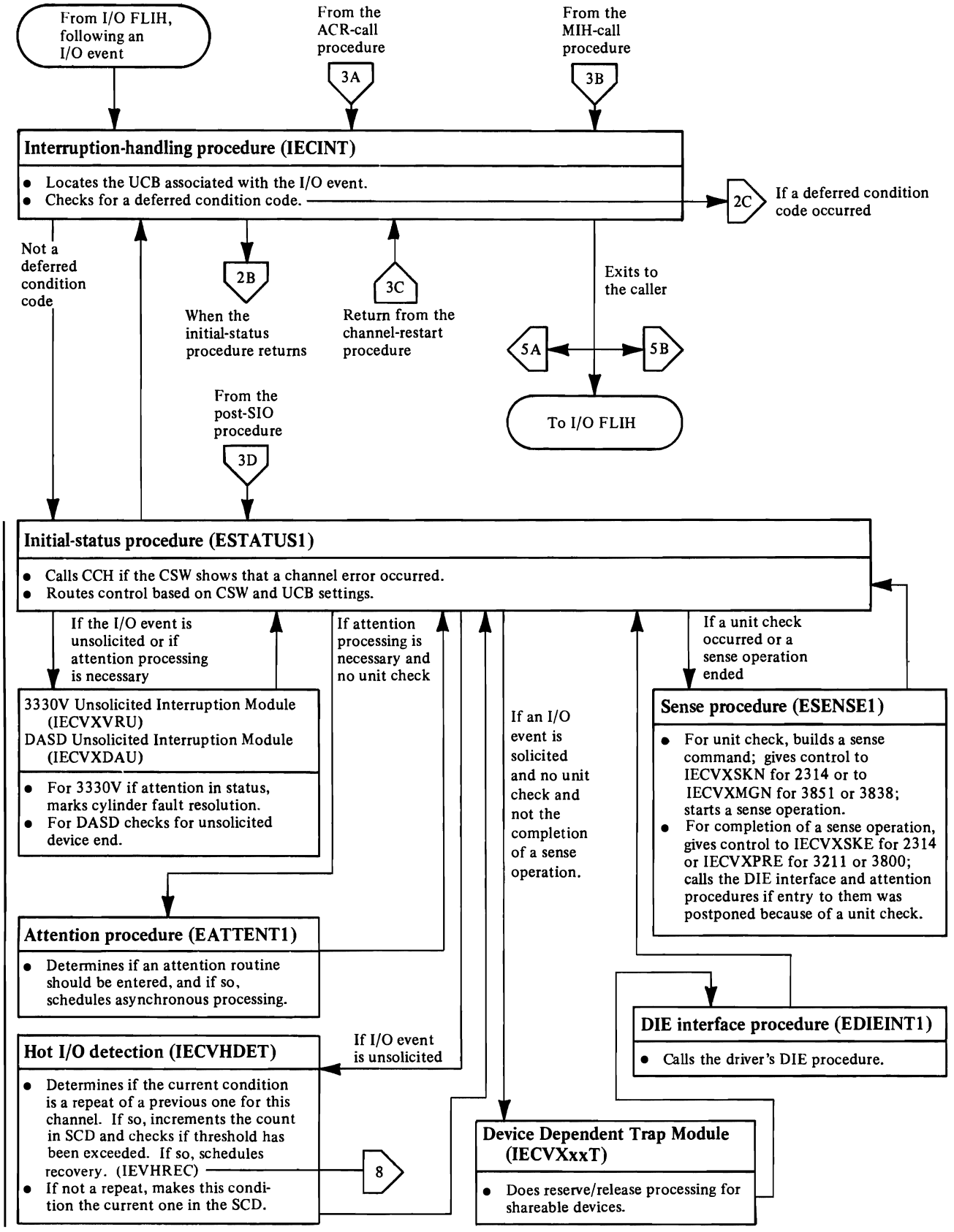


Figure 3. Responding to an I/O Event in the Basic IOS Module (IECIOSCN)

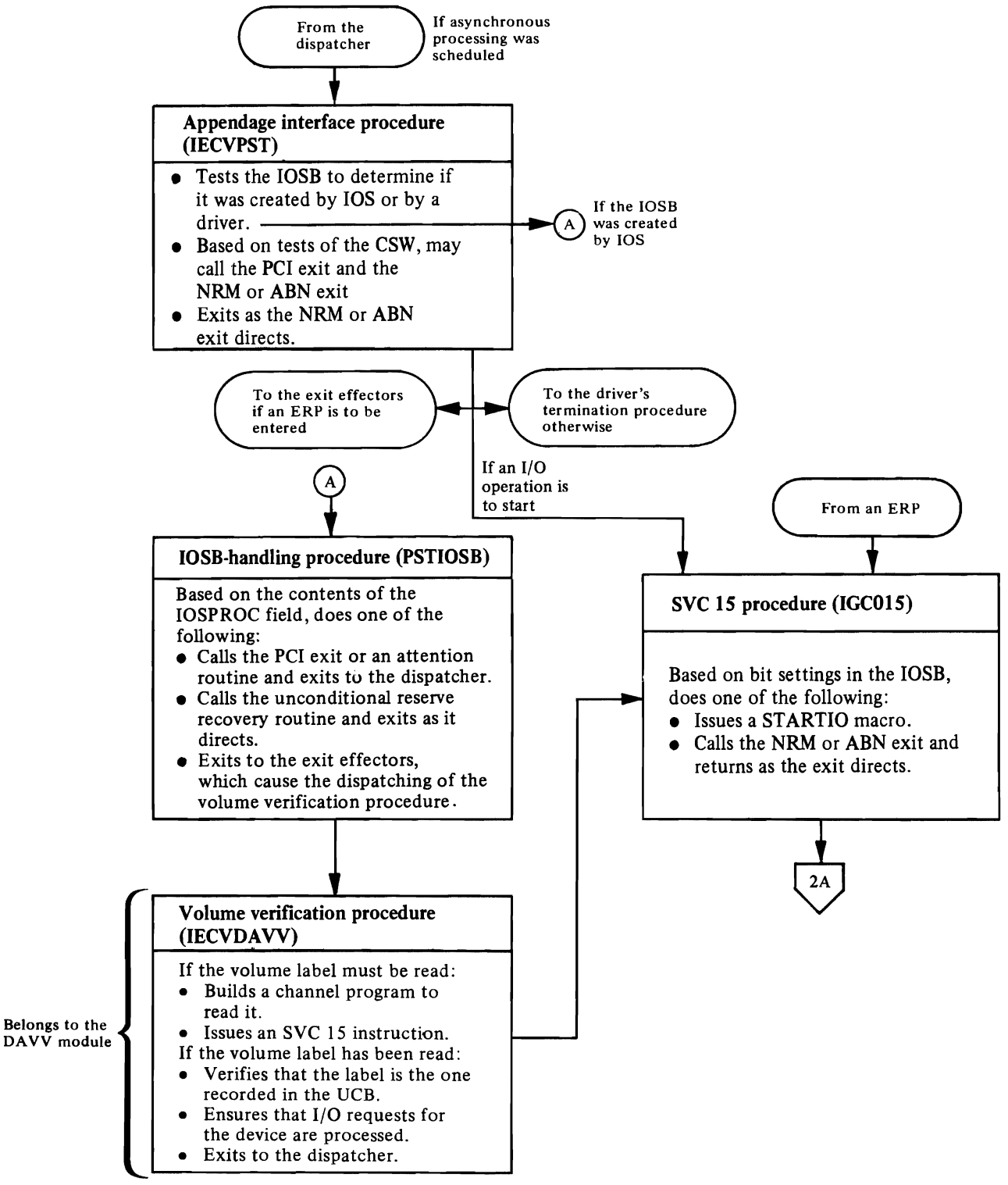


Figure 4. Responding to an I/O Event in the Post-Status Module (IECVPST) and DAVV Module (IECVDAVV)

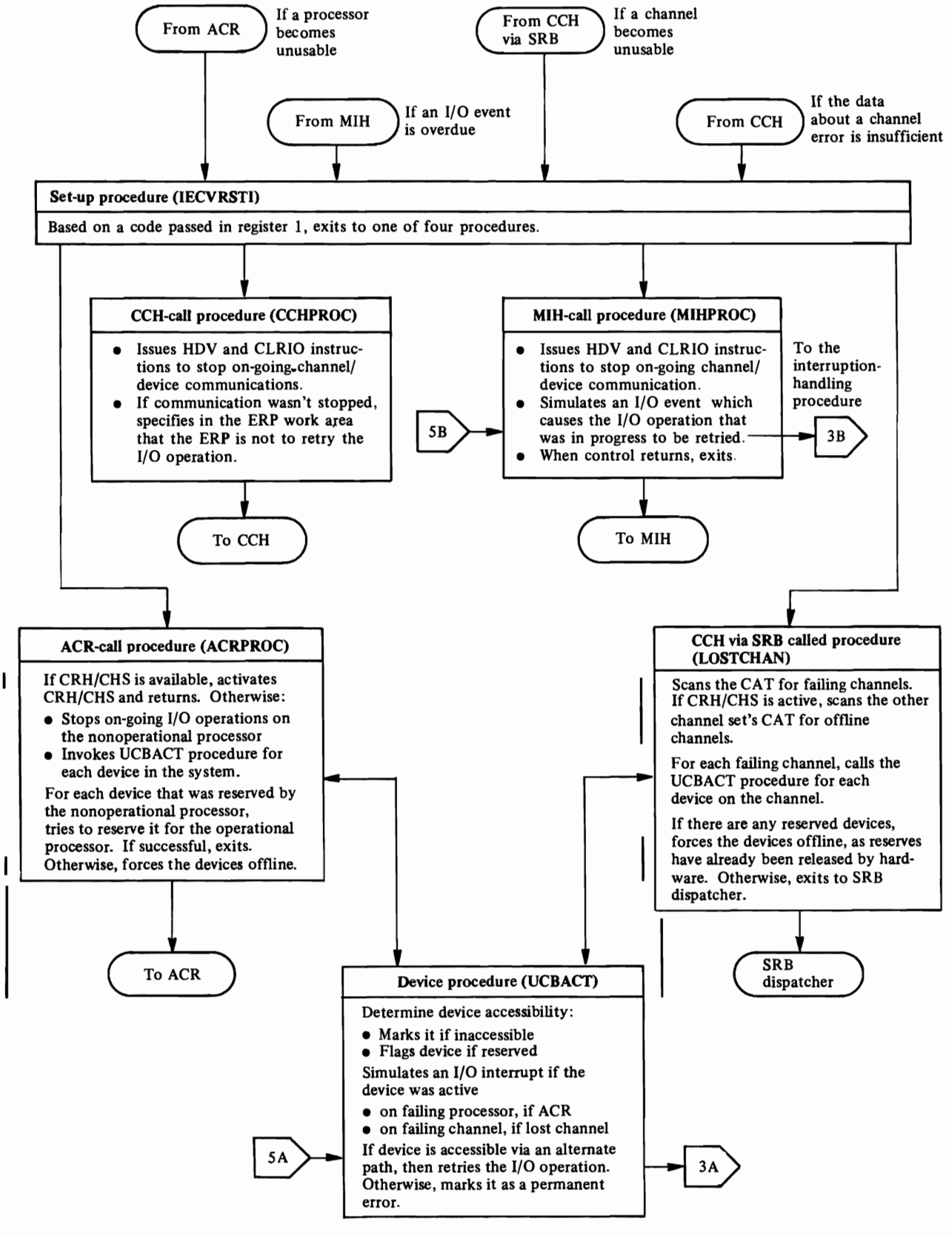


Figure 5. Restoring the Availability of I/O Resources in the I/O-Restart Module (IECVRSTI)

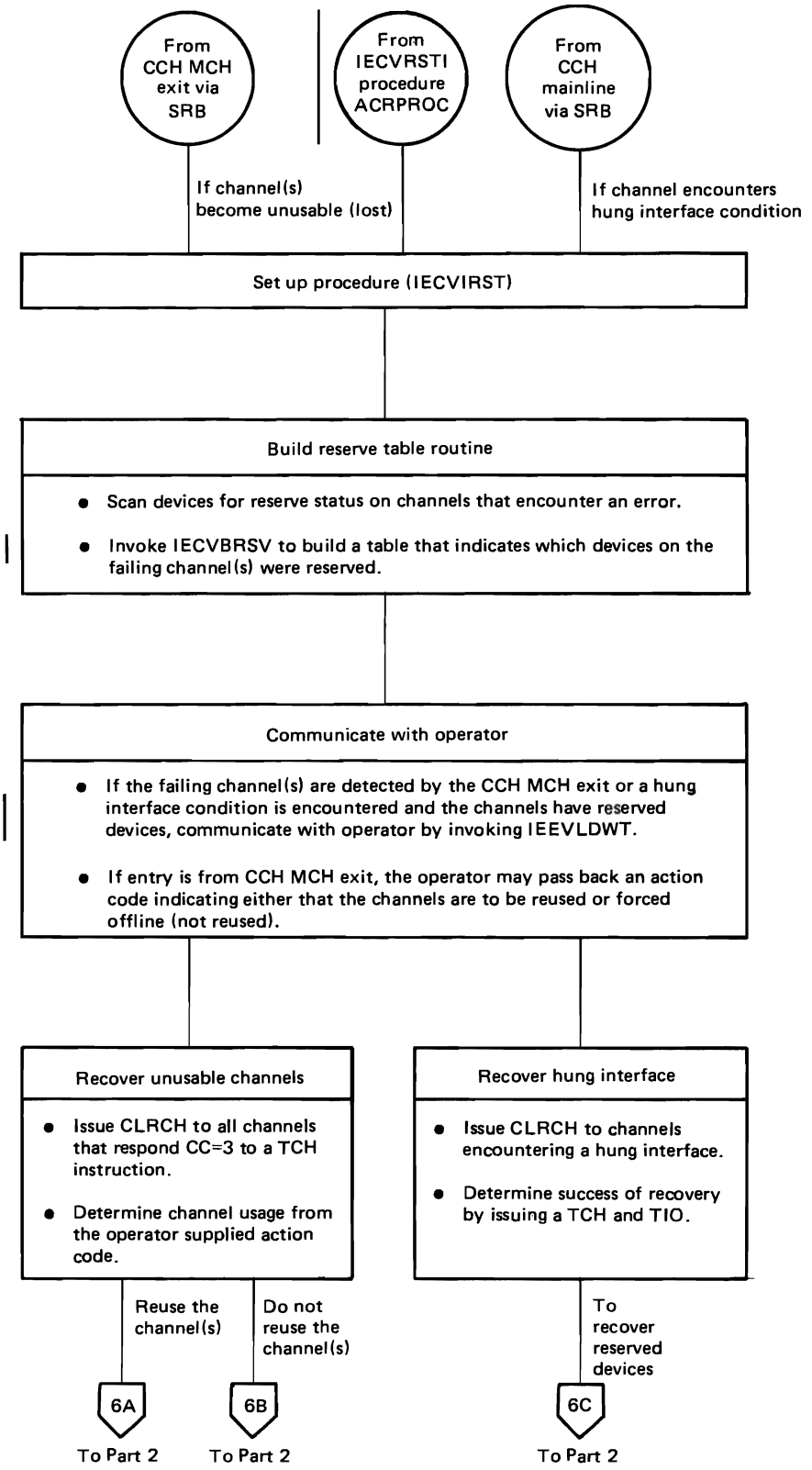


Figure 6. Restoring the Availability of I/O Resources in the I/O Restart Module (IECVIRST) (Part 1 of 2)

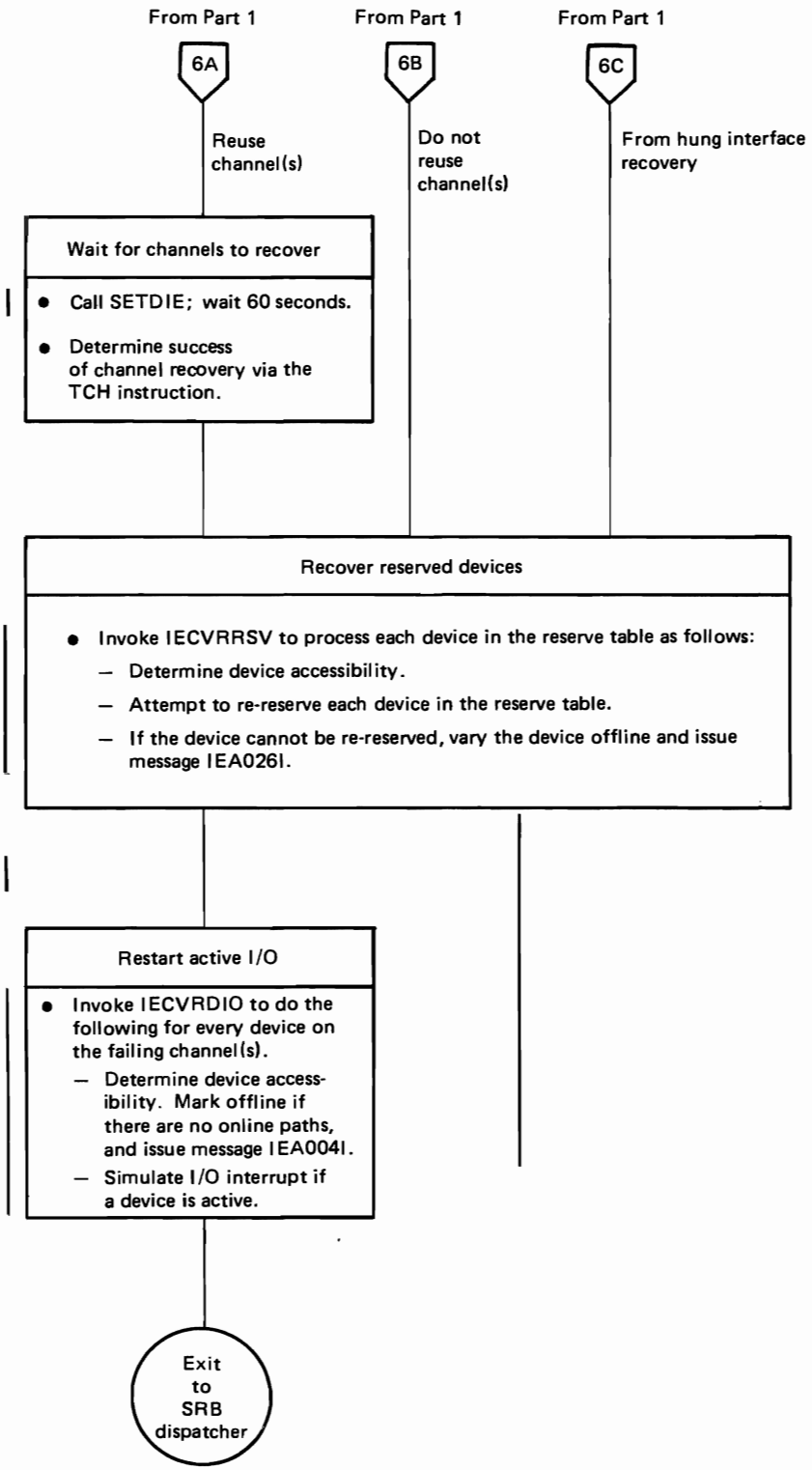


Figure 6. Restoring the Availability of I/O Resources in the I/O Restart Module (IECVIRST) (Part 2 of 2)

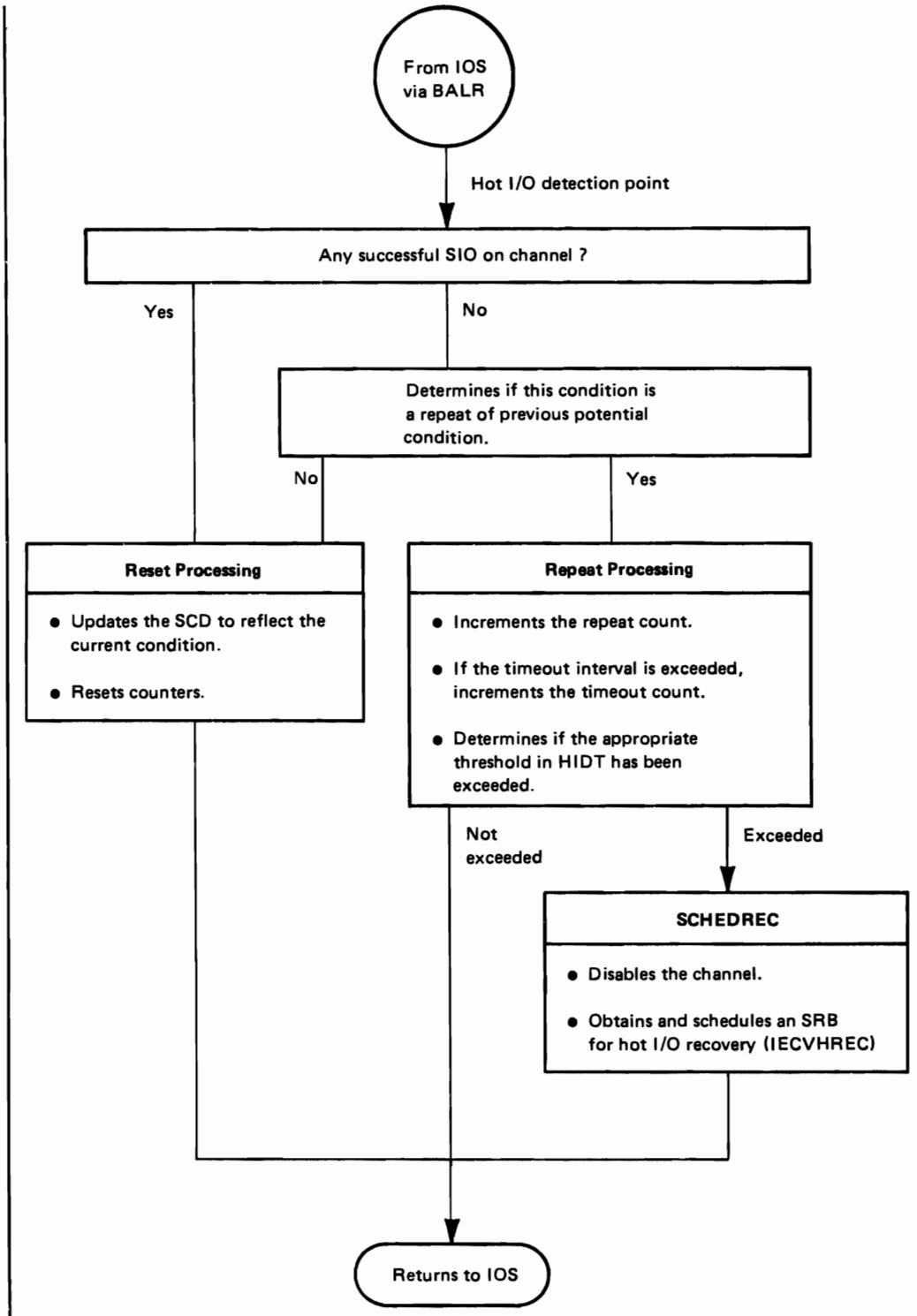


Figure 7. Hot I/O Detection

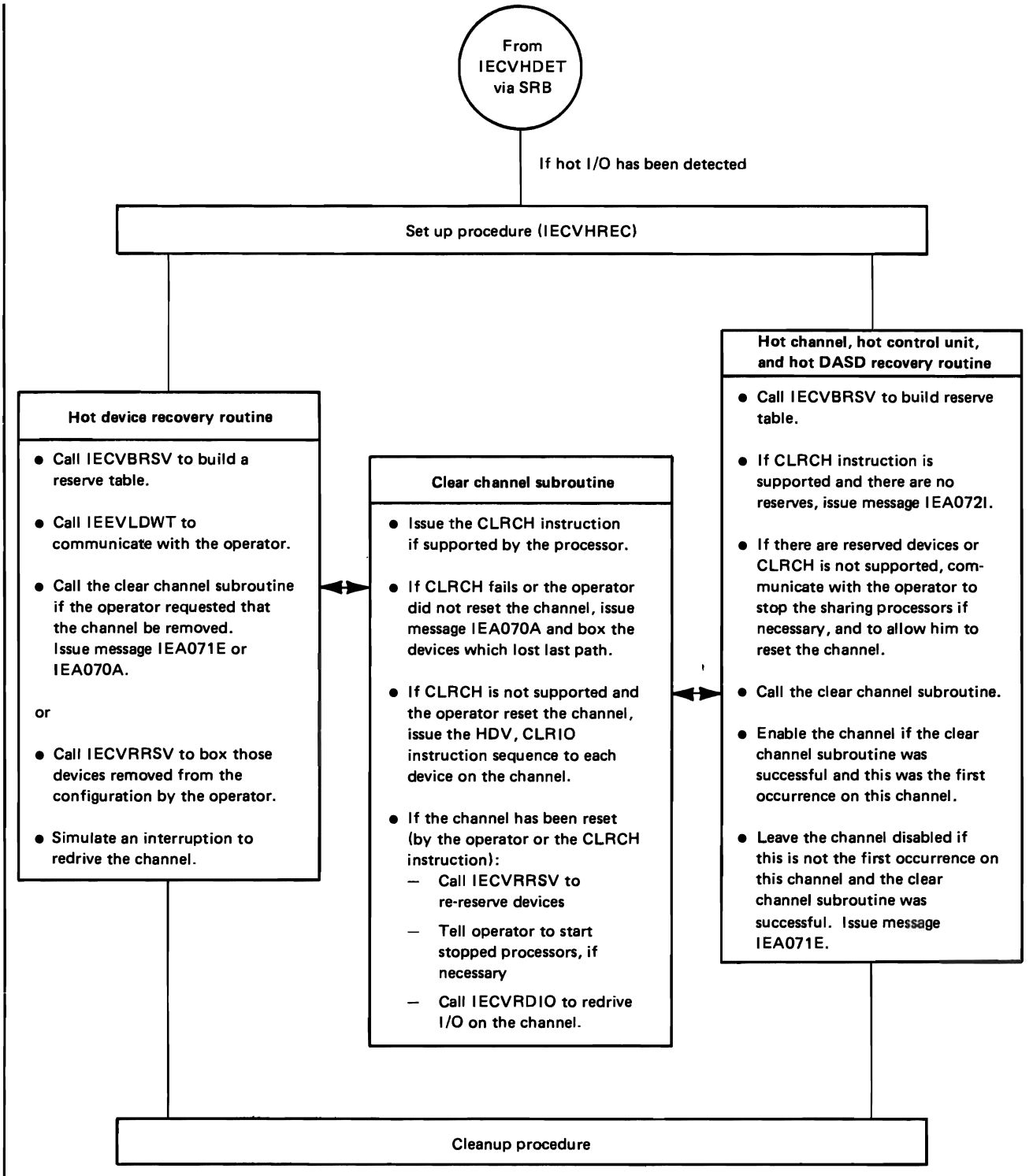


Figure 8. Recovering from a Hot I/O Event in Module IECVHREC

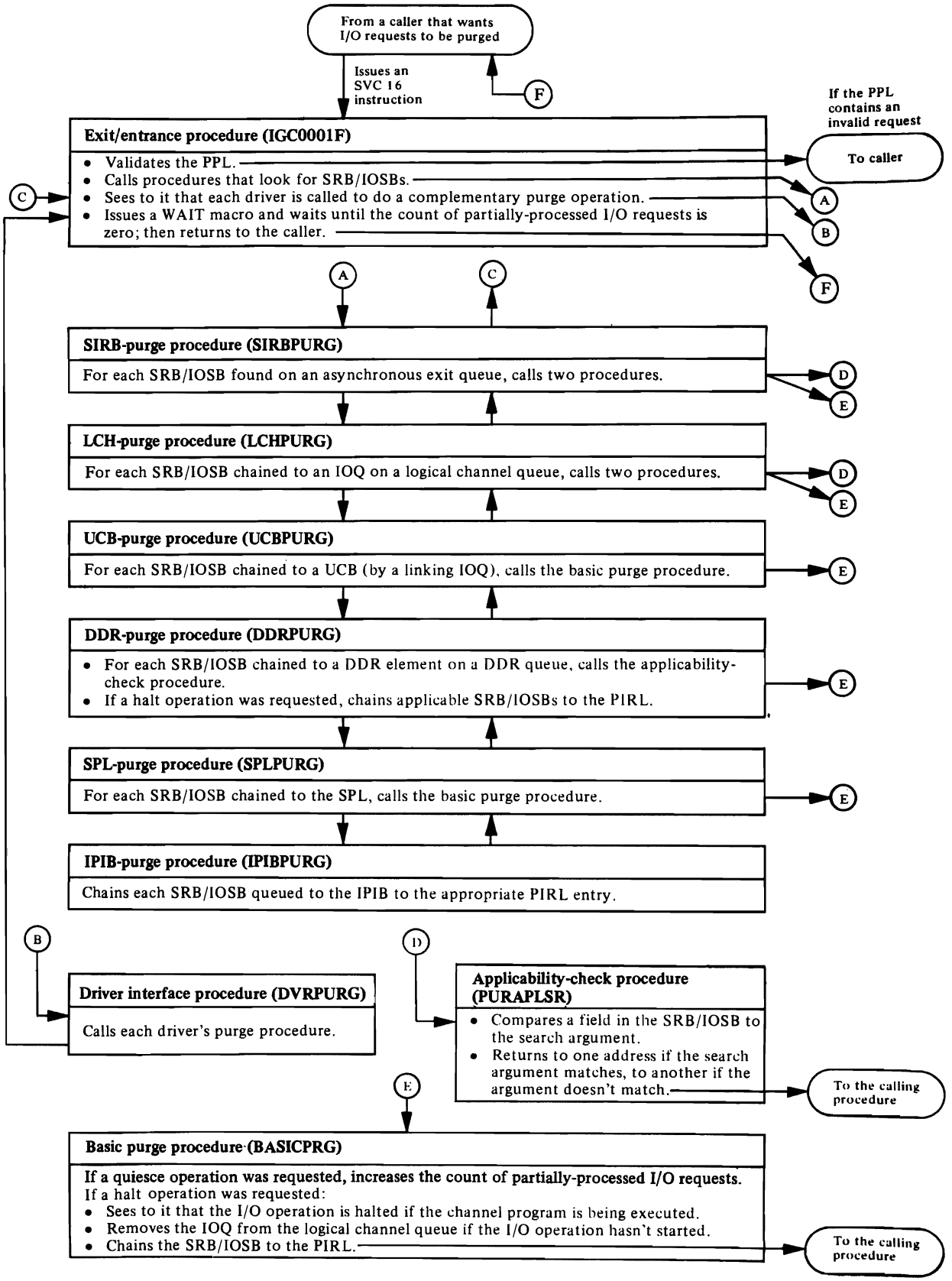


Figure 9. Purging I/O Requests in the Nonresident Purge Module (IGC0001F)

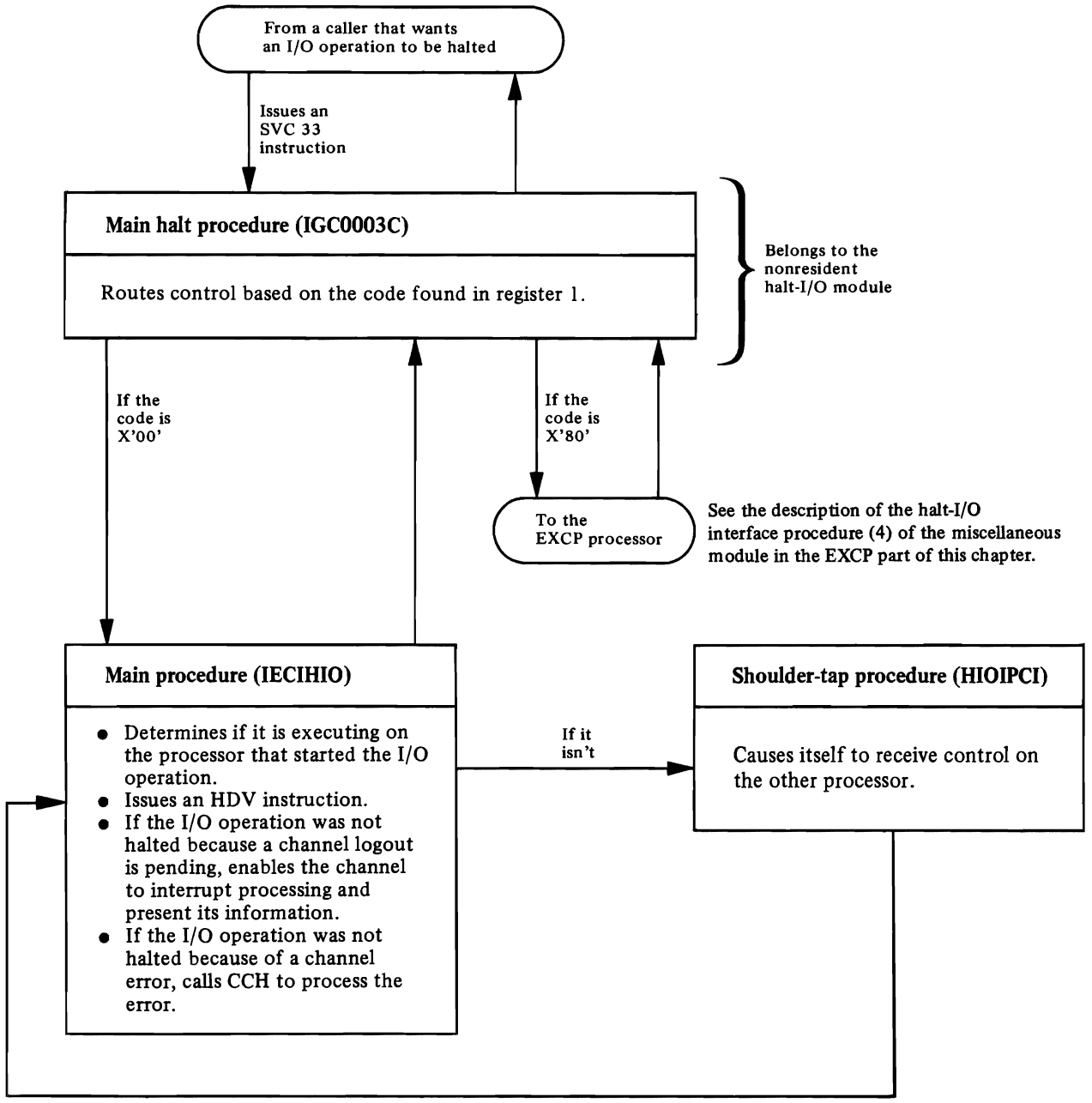
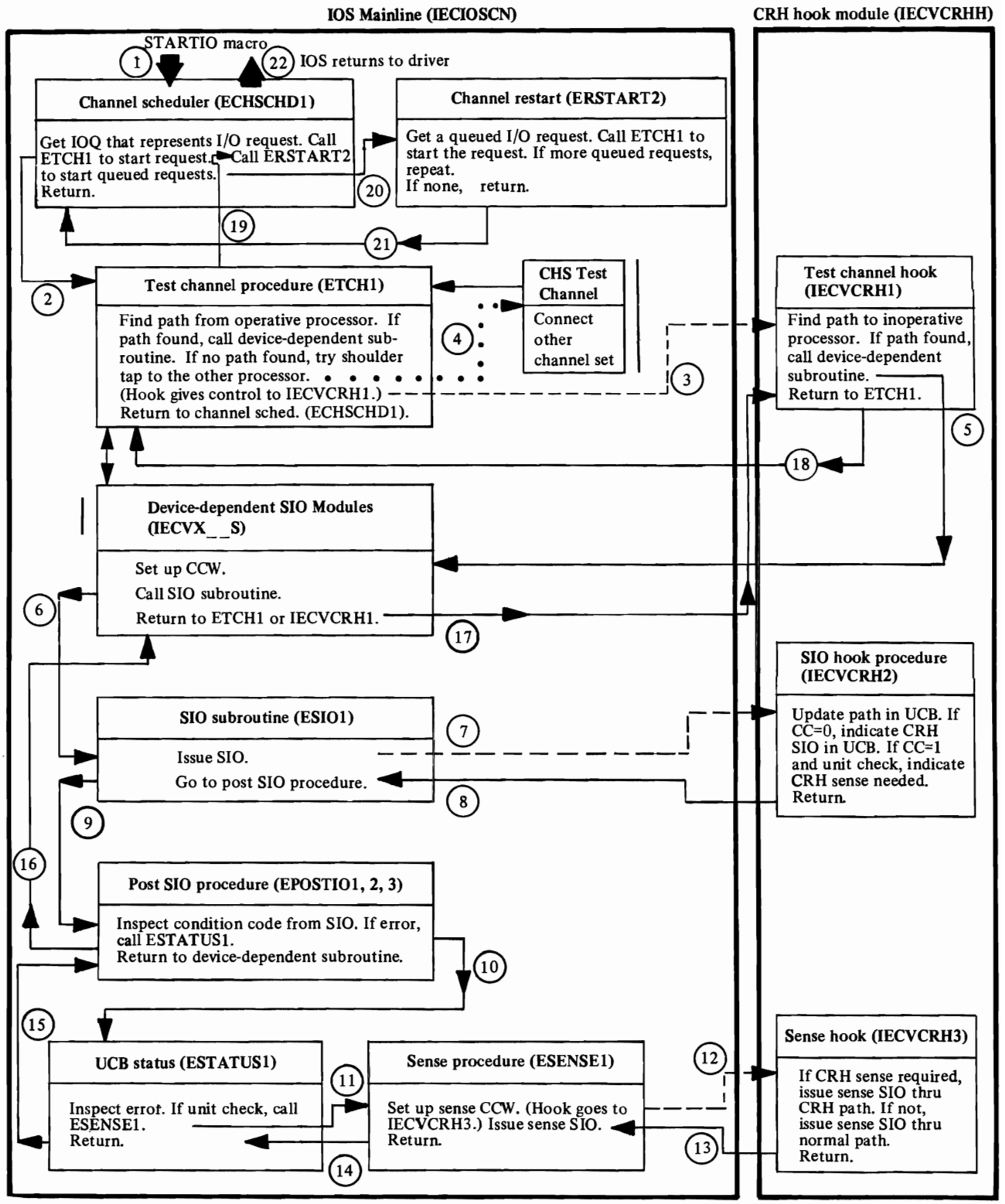


Figure 10. Halting an I/O Operation in the Nonresident Halt-I/O Module (IGC0003C) and Resident Halt-I/O Module (IECIHIO)



(An SIO is being issued to a device accessible only through the channel reconfiguration hardware. *See keyed statements that follow.*)

Legend:

- (n) = sequence no. and description
- ——→ = normal IOS flow
- – – –→ = Branch via CRH hooks to procedures in CRH module (IECVCRHH)
- • • • •→ = Branch via CHS hooks to procedures in IOS Mainline (IECIOSCN)

Figure 11. (Part 1 of 2). Channel Reconfiguration Hardware (CRH) Hook Module Interface and CHS Interface with IOS Mainline

(1) IOS mainline is called for an SIO request by an IOS driver.

(2) Channel scheduler calls ETCH1 to find a path to a device.

(3) ETCH1 cannot find a path on the operative processor, nor can it shoulder tap to the other processor. Instead, a CRH hook branches to IECVCRH1.

(4) ETCH1 cannot find a path from the connected channel set. Instead, the other channel set is connected and control returns to the beginning of ETCH1.

(5) If IECVCRH1 finds a path to the device on the inoperative processor through CRH, it calls an IOS mainline device-dependent subroutine. (**Note**: IECVCRH1 has connected the channel-6 interface on the operative processor to the desired channel on the inoperative processor.)

(6) The IOS mainline device-dependent subroutine calls ESIO1 to issue SIO, using the path found by IECVCRH1.

(7) After ESIO1 has issued SIO, a CRH hook branches to IECVCRH2. (**Note**: The SIO was issued to address 6xx. IECVCRH2 stores the actual address of the device in the UCBCHAN field.)

(8) IECVCRH2 returns to ESIO1 at the next instruction after the hook. (No such hook exists for CHS.)

(9) ESIO1 calls EPOSTIO1, 2, or 3, depending upon the condition code from the SIO, to determine if the SIO was successful.

* (10) If there was a channel error, unit check, or attention, EPOSTIO1 calls ESTATUS1 to process the error.

* (11) If there was a unit check, ESATUS1 calls ESENSE1 to issue SIO for a sense operation.

* (12) Before the SIO for sense is issued in ESENSE1, a CRH hook branches to IECVCRH3 which will issue SIO for the sense operation.

* (13) IECVCRH3 returns to ESENSE1, bypassing the sense SIO ESENSE1.

* (14) ESENSE1 returns to ESTATUS1.

* (15) ESTATUS1 returns to EPOSTIO1, 2, or 3.

(16) EPOSTIO1, 2, or 3 returns to the device dependent subroutine which called ESIO1.

(17) Device dependent subroutine returns to IECVCRH1.

(18) IECVCRH1 returns to ETCH1. (**Note**: IECVCRH1 has broken the CRH connection.)

(19) ETCH1 returns to the channel scheduler (ECHSCHD1).

(20) ECHSCHD1 calls channel restart (ERSTART2). Channel restart finds a queued I/O request and calls ETCH1 to find a path. Steps (3) through (18) are repeated. ETCH1 returns to channel restart (ERSTART2).

(21) After trying to start all the queued I/O requests, ERSTART2 returns to ECHSCHD1.

(22) ECHSCHD1 returns to the IOS driver.

***Note:** Steps 10 - 15 occur only if there are SIO errors.

Figure 11. (Part 2 of 2). Channel Reconfiguration Hardware (CRH) Hook Module Interface and CHS Interface with IOS Mainline

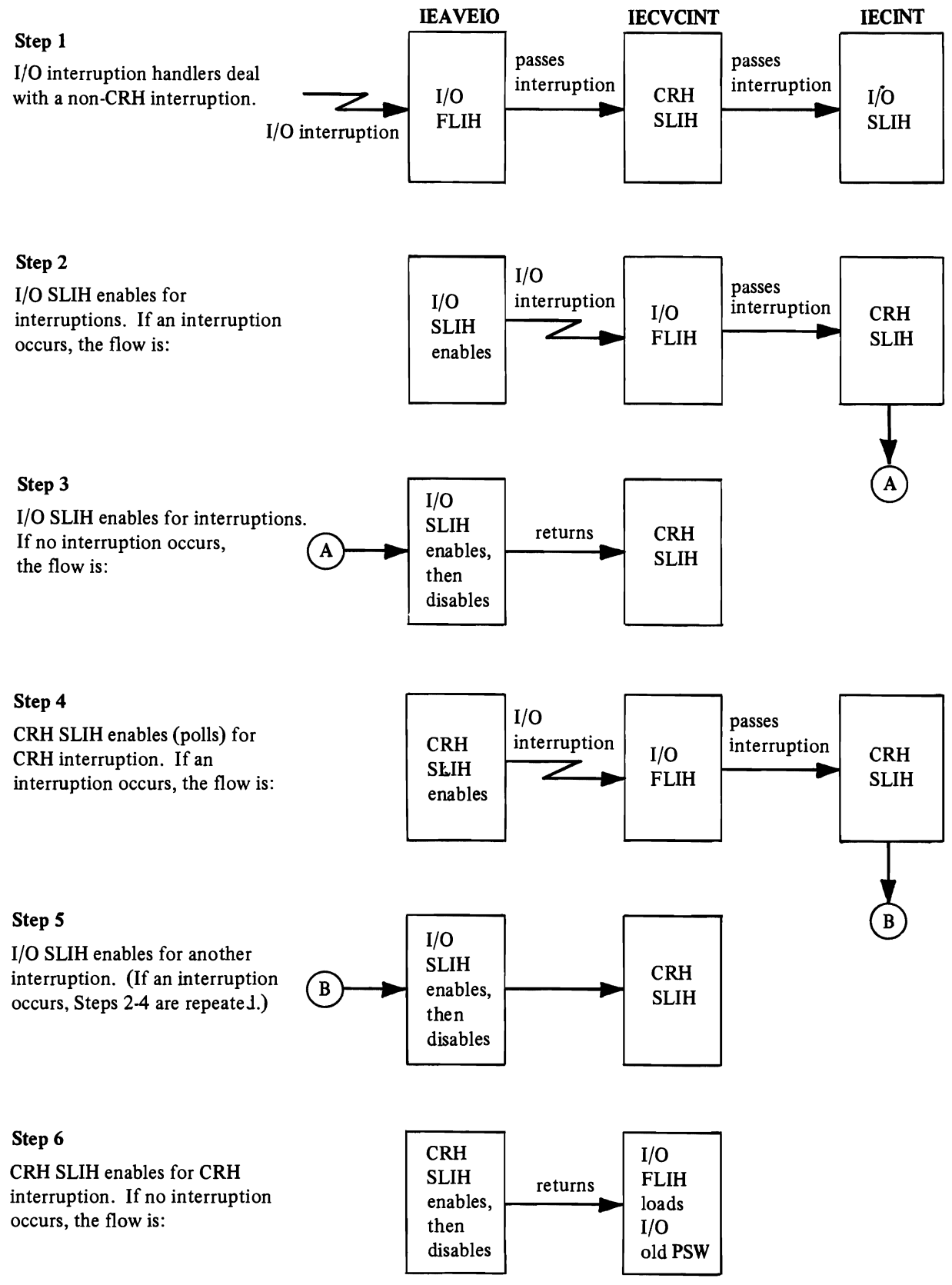


Figure 12. The Processing of Interruptions When Channel Reconfiguration Hardware (CRH) is Active

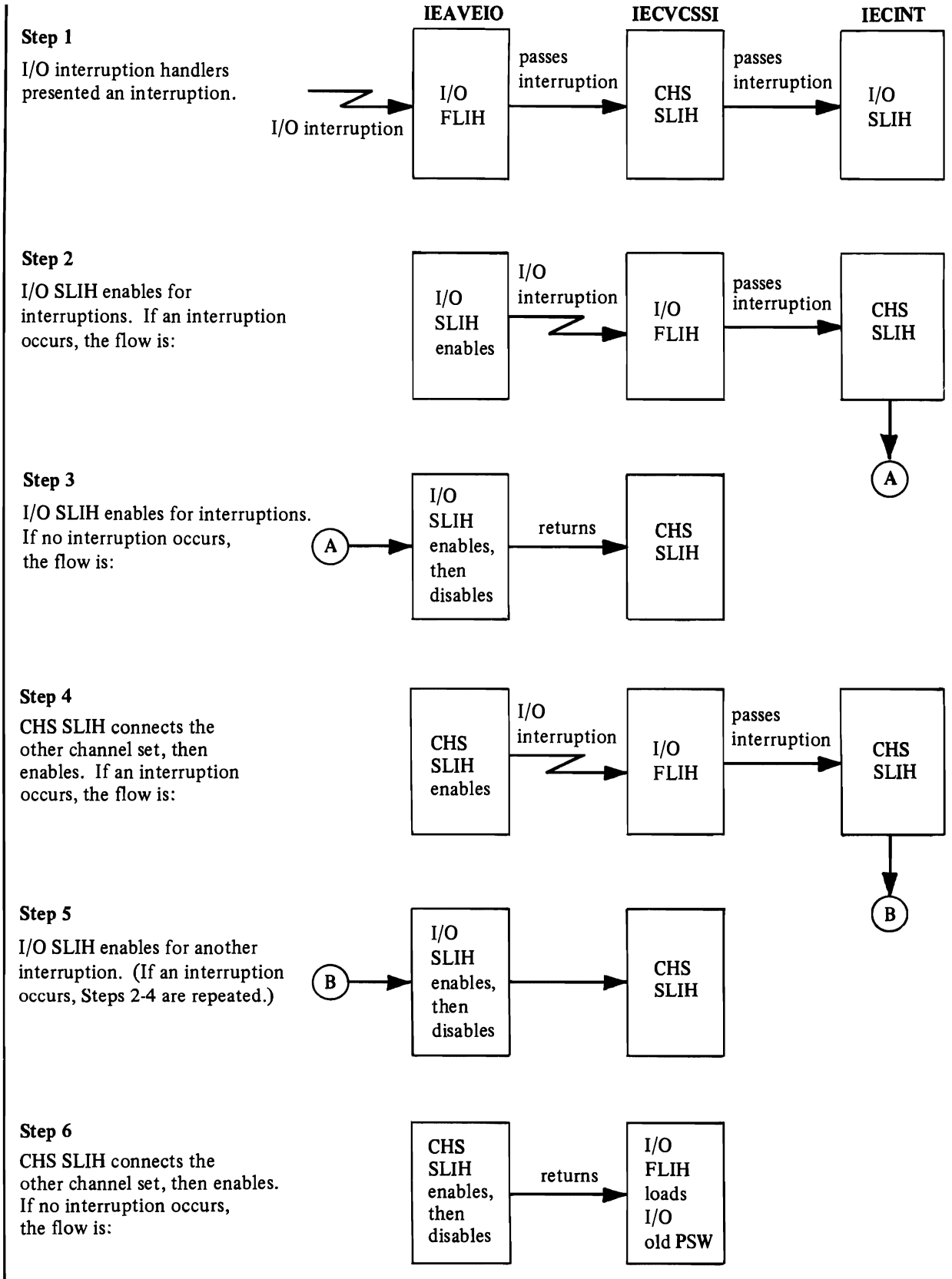


Figure 13. The Processing of Interruptions When Channel Set Switching (CHS) is Active

Basic IOS Module (IECIOSCN)

1. The Channel Scheduler Procedure (IECHNSCH)

- Entered, via a STARTIO macro, by a driver and the *post-status module (3).* (The module is actually entered at label IECVSTIO, where it does some preliminary processing: the TCB address is put in the SRB, and the address of the SRB and IOSB are put in registers 0 and 1, respectively. This procedure is then given control.)
- Disables to prevent the processor from receiving I/O and external interruptions while in control.
- Gets the UCB lock for the device in question.
- Calls the *storage manager module (1)* to get an IOQ. Initializes the IOQ and chains it to the IOSB.
- If UCB flags show that the device is not currently startable, calls the *enqueue procedure (5)* to put the IOQ on a logical channel queue and exits to the issuer of the STARTIO macro.
- Calls the *SRB scheduling procedure (7)* to schedule the *post status module* (1) if the UCBFLA flag shows "not ready" or "quiesced," and I/O is for a duplexed paging request.
- Calls the *test-channel procedure (2)* to (a) select a path to the device and (b) start an I/O operation or put it on a logical channel queue.
- Calls the *channel-restart procedure (13)* to find out if any waiting I/O requests can be started.
- Exits to the issuer of the STARTIO macro.

2. The Test-Channel Procedure (ETCH1)

- Entered by the *channel scheduler procedure (1)* and the *channel-restart procedure (13)* to select a path to the device and start an I/O operation.
- If the device is reserved, ensures that the I/O operation starts from the processor reserving the device.
- Combines the channel mask, UCBCHM, with IOSAPMSK, a mask created by the *post-status module (3),* to ensure that a path that had an error is not reselected for an I/O request being retried.
- Calls the *SRB-scheduling procedure (7)* to schedule the *post-status module (1)* if (a) the UCBFLB and UCBJBNR fields indicate that the device is currently inaccessible or (b) a guaranteed device path is unavailable.

- Selects a path from the LCH and uses a TCH instruction to test the availability of the channel. If none of the paths in the LCH is available, does the following:
  (a) Calls the *enqueue procedure (5)* to put the IOQ on a logical channel queue.
  (b) Determines whether to shoulder tap the other processor or to use CRH/CHS, if it is active. (For shoulder tapping to be expedient, there must be an available path from the other processor, and IOS must not be running in the other processor.) If shoulder tapping is expedient, turns on the bit in the IRT "channel mask" that represents the available path and shoulder taps the other processor with an RPSGNL macro. Otherwise, exits to the *channel scheduler procedure (1).*

  When CHS is active, the other channel set is connected and control returns to the beginning of this procedure to try this newly connected channel set.
- According to the type of device allocated, puts the number of the selected path (channel) in register 6 and calls the appropriate device dependent SIO Module: IEDVXDRS for 2305; IECVXSKS for 2314; IECVXVRS for 3330V; IECVXDAS for DASD; IECVXT2S for 2400 Tape; IECVXT3S for 3400 Tape; IECVXURS for unit record, teleprocessing and graphics.
- Exits to the *channel scheduler procedure (1)* via register 4; to the *channel-restart procedure (13)* via register 4 or register 4+4. (The former exit means there is an available path for another I/O request on the same logical channel queue; the latter means there is not.)

3. The SIO Procedure (ESIO1)

- Entered by the device-dependent SIO modules to start an I/O operation.
- Puts the address of the channel program and its protection key in the CAW.
- Issues an SIO or an SIOF instruction as requested by the device-dependent SIO module.
- If system trace is active and the system is being initialized, branches to the tracing routine pointed to by the CVTTRACE field. At other times, issues a HOOK macro which calls GTF to trace the results of the SIO or SIOF instruction.
- Places the condition code set by the start-I/O instruction in the IOSCC field.
- Stores the number of the channel and channel set that were used in starting the I/O operation in the UCBCHAN and UCBCPU fields, respectively.
- Updates the MF/1 or RMF counts of UCB and channel usage.
- Exits to the address in register 14. (Register 14 will contain either a return address or the address of the *post-SIO procedures (4);* the caller selects one by using the appropriate entry point.)

4. *The Post-SIO Procedure (EPOSTIO1)*

- Entered by a device-dependent SIO module and the *SIO procedure (3)* to act on the results of the start-I/O instruction issued by the *SIO procedure (3).*
- If the condition code is 0, does the following things:
  (a) Marks the channel busy in the CAT if a selector channel is in use.
  (b) Marks the UCBFLA field, showing that an I/O operation has started on the device.
  (c) Calls the *dequeue procedure (6),* to dequeue the IOQ if the IOQ is on a LCH.
- If the condition code is 1, does one of these things:
  (a) Calls the *initial-status procedure (10)* if the CSW indicates something other than a busy device or control unit (such as the completion of an immediate operation or a pending channel logout).
  (b) Calls the *enqueue procedure (5)* to put the IOQ on a logical channel queue.
- If the condition code is 2, does these things:
  (a) Marks the channel busy in the CAT.
  (b) Turns on the UCBCUB bit, which prevents requests for the device from processing if they haven't first been queued. (The bit is turned off when the *test-channel procedure (2),* entered by the *channel-restart procedure (13),* finds a path to the device that isn't busy.)
- If the condition code is 3, checks the UCBPMSK field to see if a message has been sent about the unavailability of the device. If a message hasn't been sent, does these things:
  (a) Calls the *storage manager module (5)* to get storage for an SRB/IOSB. Initializes the SRB/IOSB, indicating to the *post-status module (IECVPST) (1)* that a message must be sent to the operator.
  (b) Calls the *SRB-scheduling procedure (7)* to schedule the *post-status module (IECVPST) (1).*
  (c) If SIO was for guaranteed device path (GDP), sets a completion code of X'4D', to indicate an inoperative channel or device on the guaranteed device path, and schedules the *post status module* (IECVPST).
  (d) If SIO was for duplexed paging I/O, sets completion code of X'43', to indicate a quiesced or not-ready device on which a permanent error had occurred previously, and schedules the *post status module* (IECVPST).
  (e) If SIO was not for guaranteed device path (GDP), call the *unconditional reserve scheduling procedure (17).*
- Exits to the *test-channel procedure (2),* using an offset from the address in the IRTDDSV field to tell that procedure what to do. These are the offsets and the associated actions:
  (a) *X'00':* Return to the procedure that called without doing other processing.
  (b) *X'04':* See to it that the IOQ is put on a logical channel queue.
  (c) *X'08':* Try to start the I/O operation on another path.

5. *The Enqueue Procedure (EQUEE1)*

- Entered by the *channel scheduler procedure (1),* the *test-channel procedure (2),* the *post-SIO procedure (4),* the *DIE interface procedure (11),* and the *sense procedure (14).*
- Chains the IOQ to a LCH in FIFO order.
- Exits to the return address in register 4.

6. *The Dequeue Procedure (EQUED1)*

- Entered by the *SIO procedure (3),* the *SRB-scheduling procedure (7),* the *channel-restart procedure (13),* and the *sense procedure (14).*
- Removes the IOQ from its logical channel queue.
- Exits to the return address in register 4.

7. *The SRB-Scheduling Procedure (ESCHDIO1)*

- Entered by the *channel scheduler procedure (1),* the *test-channel procedure (2),* the *post-SIO procedure (4),* the *unsolicited device-end procedure (8),* the *DIE interface procedure (11),* the *channel-restart procedure (13),* the *sense procedure (14),* the *attention-handling procedure (15),* the *SIO module for 2305,* the *SIO module for 2314,* and the *SIO DASD module.*
- Calls the *dequeue procedure (6)* to remove the IOQ from its logical channel queue and, unless entered by the *channel-restart procedure (13),* calls the *storage manager module (2)* to free the IOQ.
- Initializes the SRB in preparation for a scheduling operation.
- Issues a SCHEDULE macro, causing the *post-status module (1)* to be dispatched asynchronously.
- Exits to the return address in register 4.

8. *The Unsolicited Device-End Procedure (EDEVEND1)*

- Entered by the *SIO module for 2314,* the *SIO module for DASD,* and the *SIO module for 3400 tape device* if they find that the device is ready (the UCBUDE bit will be on).
- If the device is a 3400 tape drive, turns on the error bits in the IOSFLA field and calls the *SRB-scheduling procedure (7),* which schedules the *post-status module (IECVPST) (1).* The ERP then gets control and informs the operator that the tape volume mounted before the device-end interruption must still be mounted.)
- If the device is a demountable DASD, calls the *storage manager module (5)* to get storage for an SRB/IOSB, sets bits in it to direct the way it will be used, and calls the *SRB-scheduling procedure (7),* to schedule the *post-status module.* (The *DAVV module (1)* then gets control and finds out if the volume mounted before the device-end interruption is still mounted.)

- If the *DAVV module* is to get control, turns on the UCBQISCE bit, to prevent the device from being used for a driver's I/O request before the *DAVV module* executes.
- Exits to the *test-channel procedure (2)*, using the address in register 4, or, if it's necessary to queue the IOQ, the address in register 4+4.

9. The Interruption-Handling Procedure (IECINT)

- Entered by the I/O FLIH, the *I/O-restart module (2, 3,* and *4),* the *resident halt-I/O module (3),* the *redrive I/O module (IECVRDIO),* the *re-reserve module (IECVRRSV),* the *special SIO module (IECVESIO),* the *unconditional reserve detection module (IECVURDT),* and the CRH and CHS interruption handlers.

  (*Note:* This procedure is frequently referred to as the I/O SLIH.)
- Finds the address of the UCB associated with the interrupting device and puts the address in register 7 for use by the procedures it calls.
- Calls CCH to process channel errors associated with an invalid device address. (Exception: If entered by the *I/O-restart module (4)*, CCH is bypassed.)
- If entered by the I/O FLIH because of an SIOF deferred condition code interruption, puts a return address in the IRTDDSV field and calls the *post-SIO procedure (4).* When control returns, exits to the I/O FLIH.
- Calls the *initial-status procedure (10)* to analyze the interruption's status and route control appropriately.
- Calls the *channel-restart procedure (13)* to start I/O operations for requests queued on the logical channel queue associated with the interrupting device. (Exception: If entered by the *I/O-restart module (4),* the *resident halt-I/O module (3),* the *redrive I/O module,* the *re-reserve module,* the *special SIO module,* the *unconditional reserve detection module (IECVURDT),* or if the *channel-restart procedure (13)* is bypassed.)
- Exits to the return address in register 4.

10. The Initial-Status Procedure (ESTATUS1)

- Entered by the *interruption-handling procedure (9)* and the *post-SIO procedure (4).*
- If the CSW shows that a channel error occurred, calls the *storage manager module (5)* to get storage for an ERP work area. Initializes the ERP work area and calls CCH. Then calls the *unconditional reserve scheduling procedure (17).*
- If PCI alone is indicated, invokes the *PCI DIE interface (12).*
- If the "active-sense" bit, UCBASNS, is on, calls the *sense procedure (14),* to process the completed sense operation.
- If the I/O event is unsolicited or if attention processing is necessary, for 3330V or DASD, calls the appropriate unsolicited interruption module to do device-dependent processing. (For 3330V – IECVXVRU, for DASD – IECVXDAU).
- If the CSW shows that an attention interruption occurred, calls the *attention-handling procedure (15).*

- If the CSW contains a unit-check indication, calls the *storage manager module (IECVSMGR) (5)* to get storage for an ERP work area. Initializes the ERP work area and calls the *sense procedure (14)* to obtain sense information. If the I/O event is unsolicited, calls the *storage manager module (IECVSMGR) (6)* to free the ERP work area.
- If the I/O event is solicited and no error occurred, ESTATUS1 calls the *appropriate trap module (IECVXxxT)* for any device dependent interrupt processing. (For 3330V devices, this routine checks the CSW CCW address to determine if a cylinder fault occurred. If it did, the IOQ is enqueued on the LCH to be held until the cylinder fault is resolved).
- If the I/O event is solicited, calls the *DIE interface procedure (11).*
- Using a HOOK macro, calls GTF to trace the I/O event.
- Exits to the return address in register 4.

11. The DIE Interface Procedure (EDIEINT1)

- Entered by the *initial-status procedure (10)* and the *sense procedure (14).*
- Calls the driver's DIE procedure (with the IOSB as input,) first issuing a TRAS macro to establish addressability to the address space the driver specified in the IOSASID field. (Following an I/O interruption, IOS doesn't necessarily receive control in the address space the DIE procedure wants to use.) When the DIE procedure returns, issues another TRAS macro to restore addressability to the address space that was being used.
- If the driver's DIE procedure submitted a new I/O request, puts an IOQ for the new request on a logical channel queue, unless requests of the kind submitted are being purged. If so, chains the SRB/IOSB for the request to the IPIB.
- Calls the *storage manager module (5)* to get storage for an SRB/IOSB if a solicited PCI interruption occurred without other status information. Calls the *SRB-scheduling procedure (7)* to schedule the *post-status module (1),* which enters the PCI exit. (If a channel program generates PCI interruptions faster than the PCI exit can process them, the SRB/IOSBs are chained to the one being processed; the *post-status module* is not rescheduled.)
- Calls the *SRB-scheduling procedure (7)* to schedule the *post-status module (IECVPST) (1).*
- Exits to the return address in register 4.

12. The PCI DIE Interface Procedure (EDIEINT2)

- Entered by the *initial-status procedure (10).*
- Checks the IOSDIE field of the IOSB for the address of a driver disabled interrupt exit (DIE) (address $\neq 0$).
- If there's no DIE routine address, schedules the PCI SRB to cause asynchronous execution of the driver's PCI exit routine.
- If there is a DIE routine address, branches to it to handle a PCI condition in the driver's code.

- Schedules the PCI SRB (as described above), if requested by the DIE routine.
- Enqueues the new SRB, if requested by the DIE routine. The SRB/IOSB is queued directly on the LCH with an IOQ. The DIE can schedule new work in IOS via a return vector. This is a performance path, since the DIE doesn't have to issue a STARTIO macro.
- Exits to the return address in register 4.

13. The Channel-Restart Procedure (ERSTART1 and ERSTART2)

- Entered by the *channel scheduler procedure (1)* and the *interruption-handling procedure (9).*
- Calls the *test-channel procedure (2)* to start I/O operations for queued I/O requests associated with (a) the channel that generated an interruption (giving notice that it's free to do more work) and (b) channels identified in the IRTCHMSK field.
- If entered for processing associated with an I/O interruption, changes the system mask to allow the I/O FLIH to receive another I/O interruption. If no interruption occurs, restores the system mask to its former setting.
- Exits to the return address in register 4.

14. The Sense Procedure (ESENSE1)

- Entered by the *initial-status procedure (10)* if it finds a unit check in the CSW or an indication that the I/O event resulted from a sense operation (the UCBASNS bit will be on).
- If entered because of a unit check, does these things:
  (a) Calls the *storage manager module (IECVSMGR) (5)* to get storage for an ERP work area, unless one has already been obtained.
  (b) If CRH is active, calls the *CRH sense hook routine (IECVCRH3)* to issue the diagnose instruction and do the sense SIO. If CHS is active and the specified channel set is not connected, exits to the *initial status procedure (10).* In this case the sense is issued on the next entry when the specified channel set is connected.
  (c) Builds a sense command. For 2314, calls IECVXSKN to build the read-home-address CCW, the read record zero CCW, and possibly the release CCW. For 3851 or 3838, calls IECVXMGN to mark the UCB and IOSB so no I/O is done to the device between the sense operation and the ERP retry.
  (d) Starts a sense operation to break the contingent connection and read sense information that an ERP will use in attempting error recovery.
  (e) If a unit check repeatedly prevents the sense information from being read, starts a sense operation to suppress data transfer and a wrong-length-record indication, trying, at a minimum, to break the contingent connection.

(f) If the channel or device is busy when the sense operation starts, calls the *enqueue procedure (5)* to put the IOQ on the physical channel queue used for sense requests. Should there be no IOQ (the unit-check indication was unsolicited), calls the *storage manager module (1* and *5)* to get storage for an IOQ and SRB/IOSB, chains the SRB/IOSB to the IOQ, and then calls the *enqueue procedure (5).*

(g) If a channel error or condition code 3 (non-operational) occurs on sense instruction, calls the *unconditional reserve scheduling procedure (17).*

(h) If no device-end indication accompanied the unit check, calls the *DIE interface procedure (11).*

(i) If entered by I/O restart (IECVRST1, 4b) because of a pseudo unit check that it sets in the CSW to get control, clears the pending sense flag and the active sense flag (UCBPSNS and UCBASNS). This is done to indicate that another sense operation need not be started.

- If entered because a sense operation completed, does these things:

  (a) Turns off the "active-sense" bit, UCBASNS, and the "pending-sense" bit, UCBPSNS.

  (b) Calls the *DIE interface procedure (11).*

  (c) Calls CCH if the status information in the CSW shows a channel error. Then calls the *unconditional reserve scheduling procedure (17).*

  (d) If an attention indication or an unsolicited device end accompanied the unit check, calls the *attention-handling procedure (15).*

  (e) For 2314, calls IECVXSKE to check for successful completion of a release initiated by IECVXSKN. For 3211 and 3800, calls IECVXPRE to handle the cancel key.

- Exits to the *initial-status procedure (10).*

15. The Attention-Handling Procedure (EATTENT1)

- Entered by the *sense procedure (14)* and the *initial-status procedure (10)* if (a) the CSW shows an attention interruption or (b) the CSW shows a device-end interruption and the device wasn't busy (the UCBBSY bit is off).
- If the attention table index (UCBATI) is non-zero, calls the *storage manager module (IECVSMGR) (5)* to get storage for an SRB/IOSB. Puts the address of the appropriate attention routine in the IOSPGAD field and X'08' in the IOSPROC field, which tells the *post-status module (IECVPST) (2)* what to do when it gets control. Calls the *SRB-scheduling procedure (7)* to schedule the *post-status module (IECVPST) (1).* (Note: This processing isn't done if (a) the attention interruption occurred as part of a solicited I/O event and (b) the attention table shows that the attention routine owner wants to do attention processing in its NRM or ABN exit.)
- Calls the *SRB-scheduling procedure (7)* to schedule the *post-status module (1)* if it determines that the *DAVV module (1),* which gets control via the *post-status module,* is waiting to process the I/O event (the UCBWDAV bit will be on).
- Exits to the return address in register 4.

16. *The Functional Recovery Procedure (IECFRR)*

- Entered by RTM if any of the procedures of the *basic IOS module* took a program check.
- Turns on the bits in the IRT channel mask (IRTCHMSK), for all system-generated channels, which will cause the *channel-restart procedure (13)* to try to start I/O operations on the channels represented by those bits.
- Checks if it was entered before, and if so, does these things:
  (a) Frees any locks held and any storage areas obtained for the I/O request being processed (if in fact an I/O request was processing at the time of the error).
  (b) Sets bits in the SDWA (via the SETRP macro) that direct RTM to write the SDWA in the SYS1.LOGREC data set and continue with termination processing.
- Ensures that the IECFRR code is operating in the right address space. (Necessary because the error might have occurred in the driver's DIE procedure, in which case the *DIE interface procedure (11)* wouldn't have had a chance to restore addressability to the address space used to process the I/O event.)
- Issues a SETFRR macro that gives RTM the address of this procedure, thereby enabling itself to be reentered if the same or another error occurs during recovery processing.
- If it can acquire the SDUMP buffer, puts diagnostic data in the buffer and issues an SDUMP macro to write the contents of the buffer in the SYS1.DUMP data set.
- If the *storage manager module* was in control when the error occurred (shown in the IRT), does these things:
  (a) Issues a SETFRR macro to delete the address of this procedure from RTM's stack of functional recovery procedures.
  (b) Exits to the *storage manager module (10)*.
- If a UCB lock is held and a channel program is active on the device, issues an HDV instruction to halt the channel program and turns off the status bits in the UCB that indicate an I/O operation is in progress. Releases the UCB lock if no I/O operation is in progress.
- If an I/O request was being processed when the error occurred (shown in the IRT), does these things:
  (a) Puts X'45' in the IOSCOD field and calls the *SRB-scheduling procedure (7)* to schedule the *post-status module (1)*.
  (b) Returns the IOQ to the *storage manager module (2)*.
- If CRH is active and a CRH connection is outstanding, issues a diagnose instruction to break the CRH connection.
- If the error occurred in a sense module (IECVXSKN for 2314 or IECVXMGN for 3851 and 3838), in an end of sense module (IECVXSKE for 2314 or IECVXPRE for 3211 and 3800), or in an unsolicited interruption module (IECVXDAU for 2314 and DASD or IECVXVRU for 3330V), IECFRR sets up to retry at the instruction following the call of the module. For abends in sense modules the sense channel program is also rebuilt.

- If the error occurred in the *channel scheduler procedure (1)* in a different address space (due to ACR or the RESTART key being pressed) and the IOQ and IOSB haven't been initialized, does these things:

  (a) Frees any locks held and any storage areas obtained for the I/O request being processed.

  (b) Issues a SETRP macro that causes RTM to write the SDWA in the SYS1.LOGREC data set and continue with termination processing.

  In other cases, sets bits in the SDWA (via the SETRP macro) that direct RTM to write the SDWA to the SYS1.LOGREC data set and gives control to the IOS procedure that last had control (shown in the IRT).

- Exits to RTM.

Note: This procedure puts diagnostic data in the SDUMP buffer and in the variable area of the SDWA. The data is described in the "Diagnostic Aids" chapter under "Output of the Basic IOS Module (IECIOSCN)."

17. The Unconditional Reserve Scheduling Procedure (EDETECT1)

- Entered by the *initial status procedure (10)* if (a) a condition code 3 occurred on SIO and a message was not previously issued, (b) channel errors occurred on SIO, or (c) channel errors occurred on an interruption.
- Entered by the *sense procedure (14)* if (a) a condition code 3 occurred on sense SIO, (b) channel errors occurred on sense SIO, or (c) channel errors occurred at the end of sense.
- If not a direct access device, returns to caller.
- If channel recovery is in progress, returns to caller.
- If the error occurred on non-sense SIO to a device which is not reserved, returns to caller.
- Calls the device validation routine (IECVDVAL) to determine whether the device type can support unconditional reserve. If it cannot, returns to caller.
- Calls the *storage manager module (IECVSMGR) (5)* to obtain an IOSB/SRB, an ERP workarea, a workarea and a savearea for unconditional reserve recovery.
- Schedules the *post status module (IECVPST)* with the IOSPROC value set to the value for unconditional reserve recovery.
- Returns to caller to continue normal handling of the error.

Build Reserve Table Module (IECVBRSV)

1. The Set Up Procedure

- Entered by I/O Restart Module IECVIRST and the hot I/O recovery module IECVHREC.
- Establishes a functional recovery routine.
- Issues the GETMAIN macro to obtain storage for a work area.

2. The Build Reserve Table Routine

- Entered at the completion of the *set up procedure (1).*
- Scans the input reserve table chain to point to the last reserve table segment.
- Scans all devices on the specified path for reserve status.
- Builds a segmented table that contains an entry for each reserved device on the specified path.
- Returns to caller.

3. The Functional Recovery Routine

- RTM enters the FRR routine when an error is encountered in IECVBRSV.
- Returns resources to the system.
- Returns to RTM and percolates.

CCW Translator Module (IECVTCCW)

1. The Routing Procedure (IECVTCCW)

- Entered by IOS drivers and other system components that require one of the following:
  (a) A copy of a channel program in fixed storage, the fixing of buffers to which the channel program points, and the substitution of real storage addresses in the copied CCWs for virtual storage addresses. (This service is known as *channel-program translation*.)
  (b) The address of a translated CCW (one containing a real storage address) that corresponds to a specified untranslated CCW (one containing a virtual storage address).
  (c) The address of an untranslated CCW that corresponds to a specified translated CCW.
  (d) The unfixing of pages that were fixed in translating a channel program.
- Exits, as directed by the TCCWOPTN field of the TCCW (translation control block), to another procedure, where the appropriate processing is done. These are the possible values in the TCCWOPTN field and the associated exits:
  (a) *X'00'*: Goes to the *CCW translation procedure (2)*.
  (b) *X'04'*: Goes to the *address retranslation procedure (9)*.
  (c) *X'08'*: Goes to the *unfix-and-free procedure (10)*.
  (d) *X'0C'*: Means that the caller entered before, without giving this module enough storage to work with, and is reentering with supplementary storage. Goes to the address in the TCCWSAVE field, the address at which the lack of storage was detected.
  (e) *X'10'*: Goes to the *single-address translation procedure (8)*.

2. The CCW Translation Procedure (TCCWI100)

- Entered by the *routing procedure (1)* to translate a channel program.
- For each CCW in the channel program, does one of the following:
  (a) Obtains the pointer to the CCW operation table from the DDT (device descriptor table) associated with the device.
  (b) Calls the *main TIC procedure (4)* if the command code indicates the CCW is a TIC (transfer-in-channel CCW).
  (c) If the CCW contains a virtual storage address, calls the *page-fix procedure (3)* to fix the page (or pages) containing this address for a length specified in the CCW count field. Copies the CCW into the BEB (beginning-end block). a block of fixed storage supplied by the caller of the *routing procedure (1)* and pointed to by the TCCWBEB field. If the CCW contains a virtual storage address, puts the corresponding real storage address in the copy.
- Calls the *IDAL procedure (7)* if a buffer crosses a page boundary.
- Calls the *TIC insertion procedure (5)* if the BEB is one CCW short of being full and more than one CCW remains to be translated.

- Calls the *unfix and free procedure (10)* if there is an error in the channel program being copied.
- Exits to the *TIC resolution procedure (6)* when the last CCW has been processed, allowing that procedure to resolve addresses that it wasn't able to resolve during previous calls from the *main TIC procedure (4)*.

3. *The Page-Fix Procedure (TCCWM000)*

- Entered by the *CCW translation procedure (2)* to fix the page (or pages) containing a buffer.
- Examines a list—called a ***fix list***—of pages previously fixed. If the buffer is within one or more of those, returns to the *CCW translation procedure (2)*. Otherwise, does the following:
  (a) Adds the virtual storage address of the page (or pages) to be fixed to the list.
  (b) Calls the system's page-fix routine to do the page fixing.
- If a new entry can't be added to the fix list for lack of space, exits to the caller of the *routing procedure (1)* with a return code of X'0C' in register 15, requesting additional storage.
- If the system's page-fix routine indicates that it could not fix a page, calls the *unfix-and-free procedure (10)*.
- Exits to the *CCW translation procedure (2)*.

4. *The Main TIC Procedure (TCCWM100)*

- Entered by the *CCW translation procedure (2)* each time it encounters a TIC in the channel program it's translating.
- Copies the TIC from the original channel program into the BEB.
- Exits to the *TIC insertion procedure (5)* if (a) the TIC is a "no-op" (it points to the CCW that follows it in the channel program) and (b) there is room for one more CCW in the BEB but more than one to be copied.
- Exits to the caller of the *routing procedure (1)* with a return code of X'0C' in register 15, requesting another BEB, if (a) the TIC is preceded by a status-modifier CCW (one that may cause the channel to skip the TIC and execute the next CCW) and (b) there is room for two more CCWs in the BEB but more than two to be copied.
- If the TIC is either a "no-op" or "resolvable" (one that points to an already-copied CCW), does the following:
  (a) Changes the pointer in the TIC to point to the real storage address of the copy.
  (b) Calls the *TIC resolution procedure (6)*, which determines whether any CCWs have been copied since it was last entered that would allow "unresolved" TICs to be resolved.
- Otherwise, builds an "unresolved TIC list," if one doesn't already exist, and enters in it the address of the TIC.
- Exits to the *CCW translation procedure (2)*.

5. *The TIC Insertion Procedure (TCCWM300)*

- Entered by the *CCW translation procedure (2)* and the *main TIC procedure (4)* if they find that the BEB is one CCW short of being full and more than one CCW remains to be translated.
- Exits to the caller of the *routing procedure (1)* with a return code of X'0C' in register 15, requesting another BEB. When control returns (from the *routing procedure (1)*, does one of the following:
  (a) Fills the last CCW-space with a TIC that points to the BEB, if the caller was the *CCW translation procedure (2)*.
  (b) Overlays the last TIC in the old BEB with one that points to the new BEB, if the caller was the *main TIC procedure (4)*.
- Exits to the *CCW translation procedure (2)*.

6. *The TIC Resolution Procedure (TCCWM200)*

- Entered by the *CCW translation procedure (2)* and the main *TIC procedure (4)*.
- Scans the list of "unresolved" TICs–those that still point to CCWs that had not yet been copied when this procedure was last entered. Determines which, if any, of the unresolved TICs can now point to CCWs in the BEB, and changes pointers accordingly.
- Exits to the *main TIC procedure (4)* if that is its caller, or to the caller of the *routing procedure (1)*. In the latter case, channel program translation is assumed to be complete.

7. *The IDAL Procedure (TCCWM400)*

- Entered by the *CCW translation procedure (2)* if it finds that a buffer crosses a page boundary.
- Puts into the IDAL ("indirect-address" list) an entry containing the real storage addresses of the buffer and the pages it crosses.
- Replaces the buffer address in the CCW with the address of the IDAL entry and turns on the "indirect" bit in the CCW, indicating that the CCW points to an IDAL entry.
- When it's necessary to get storage for an IDAL–the first entry is being created or storage allocated to the IDAL has been used up–exits to the caller of the *routing procedure (1)* with a return code of X'0C' in register 15, requesting more storage.
- Exits to the *CCW translation procedure (2)*.

8. *The Single-Address Translation Procedure (TCCWX000)*

- Entered by the *routing procedure (1)* to obtain for its caller the virtual storage address of a translated CCW. (The virtual storage address of the corresponding untranslated CCW is provided in register 0.)
- Searches the BEBs that contain the translated channel program for a CCW that corresponds to the untranslated CCW.

- Exits to the caller of the *routing procedure (1)* with a return code of X'04' in register 15 if the BEBs contain no corresponding CCW, or with a return code of 0 in register 15 if the CCW was found. In the latter case, the virtual storage address of the translated CCW is returned in register 0.

9. The Address Retranslation Procedure (TCCWR000)

- Entered by the *routing procedure (1)* to obtain for its caller the virtual storage address of an untranslated CCW. (The virtual storage address of the corresponding translated CCW is provided in register 0.)
- Determines whether register 0 actually points to a translated CCW in a BEB. If it does, calculates the virtual storage address of the corresponding untranslated CCW, puts the address in register 0, and puts a return code of X'00' in register 15. Otherwise, puts a return code of X'04' in register 15.
- Exits to the caller of the *routing procedure (1)*.

10. The Unfix-and-Free Procedure (TCCWU000)

- Entered by the *routing procedure (1)* and the *page-fix procedure (3)*.
- Calls the system's page-fix routine to unfix pages that were fixed in translating a channel program.
- Chains the BEBs, FIX lists, and IDALs together and stores the address of the chain in register 0.
- Puts one of these return codes in register 15:
  (a) *X'04'*, if entered by the *page-fix procedure (3)*, indicating that an error occurred during page fixing.
  (b) *X'08'*, if entered by the *routing procedure (1)*.
- Exits to the caller of the *routing procedure (1)*.

CRH/CHS Module, Basic (IECVCINT)

Note: For overviews of CRH/CHS flow, refer to the CRH diagrams, figures 11, 12, and 13.

1. CRH/CHS Activation Procedure (IECVCRHA)

- Entered from the ACR-call procedure (ACRPROC) in the I/O restart module (IECVRSTI) on alternate CPU recovery, or from the VARY processor (IEEVCPU) when the operator issues the first VARY channel online command for a channel attached to an offline processor. Runs disabled.
- If system is not a 168 MP or a processor which supports CHS, clears the pointer to the CRH/CHS activation procedure in the IOCOM extension. Returns to the caller with an indication that the system does not support CRH or CHS.
- If CRH or CHS is already active, returns to the caller with an indication that CRH or CHS is already active (RC = 04).
- For CHS only, issues a disconnect channel set instruction in order to force the channel set attached to the inoperative processor into the disconnected state. The channel set is then connected to the operative processor.
- For CHS only, if only one channel set contains online channels, issues message IEA919I ("CHANNEL SET x SWITCHED TO PROCESSOR y"), then returns to the caller with a normal return code (RC=0).
- Initializes the CVTCRCA field of the CVT to point to the CRH communication area (CRCA) and initializes the CRH communication area. (See "Connections Between Principal Data Areas" in the "Data Areas" chapter.)
- Initializes a field in the TIMER SLIH to point to IECVCRHS.
- Changes the pointer in the I/O FLIH to point to the CRH SLIH procedure (6) (IECVCINT) or the CHS SLIH procedure (IECVCSSI) instead of to the I/O SLIH (IECINT), so that the CRH or CHS interruption handler will get control before the IOS interruption handler.
- For CRH only, activates hooks in mainline IOS so that the CRH hook procedures get control when IOS starts an I/O operation to a channel of the inoperative processor.
- For CRH only, processes all UCBs that indicate that their devices were last started for the inoperative processor. Sets the UCBCPU field to the operative processor ID, sets the UCBIORST flag to indicate that the last path started to the device was through the inoperative processor, indicates that the device is reserved to the inoperative processor if the device was reserved to that processor, and sets the "sense command needed" flag (UCBCRHSN) in the UCB if the UCB has an IOQ on the sense logical channel queue for the inoperative processor.
- Schedules an SRB which causes the CRH/CHS timer pop procedure (4) to be dispatched. This is done so that the CRH interruption handler (IECVCINT) is forced to periodically poll (solicit interruptions from) the inoperative processor's channels for pending interruptions, or so that the CHS interruption handler (IECVCSSI) is forced to periodically connect the disconnected channel set.
- Issues message IEA970I CHANNEL RECONFIGURATION HARDWARE ACTIVATED.
- Returns to the caller (ACRPROC or the VARY processor) with a normal return code (RC = 0).

2. *CRH/CHS Deactivation Procedure (IECVCRHD)*

- Entered by the VARY processor (IEEVCPU) when the operator is varying offline the last channel of the inoperative processor while CRH is active, varying offline the last channel of any channel set when CHS is active, or varying online the inoperative processor while CRH or CHS is active. This procedure runs disabled.
- Cleans up by clearing the pointer (CVTCRCA) to the CRH communication area, takes the CRH hooks out of mainline IOS, restores the I/O FLIH pointer to point to the I/O SLIH (IECINT), and for CRH clears three UCB flags associated with devices whose channels belonged to the inoperative processor. Also, for CRH, updates the UCBCPU field to reflect the processor through which the last path I/O was started. (The flags previously indicated that the devices' last path was started to the inoperative processor, that the devices were reserved to the inoperative processor, and that a sense operation was needed or issued for devices or channels connected to the inoperative processor.)
- Issues message IEA972I CHANNEL RECONFIGURATION HARDWARE DEACTIVATED.
- Returns to the caller with a normal (zero) return code.

3. *CRH/CHS STIDC Procedure (IECVCRHV)*

- Entered by the VARY processor when the operator issues a VARY CHANNEL command to place online a channel formerly connected to the inoperative processor. The procedure uses the CRH or CHS feature to access the channel and store the channel ID and condition code from the STIDC (store channel ID) instruction, so that the VARY processor can determine whether the channel can be brought online. Procedure runs disabled.
- For CRH issues the diagnose instruction to access the channel whose address VARY passed in register 1. For CHS issues the connect channel set instruction.
- Issues the STIDC instruction and puts the condition code in the high-order byte of register 1 for use by the VARY processor.
- For CRH, again issues the diagnose instruction, this time to break the CRH connection to the channel. For CHS issues the connect channel set instruction to reconnect the original channel set.
- Exits to the VARY processor to complete the processing of the VARY channel online command.

4. *CRH/CHS Timer Pop Procedure (IECVCRHT)*

- Is dispatched under an SRB scheduled by CRH/CHS activation or by IECVCRHS.
- For CRH, if an I/O interruption has not occurred in the last 2 seconds, issues an IOSINTRP macro for unit 600 to simulate an I/O interruption. Then the CRH SLIH polls for pending interruptions on the inoperative processor at least once every two seconds. The interruption is simulated on an arbitrary address (600) with zero status in the CSW. Then calls the *TIMER SLIH procedure (IEAQTE00)* to queue a TQE, causing a timer interrupt in 2 seconds.

Simulates interruptions at timed intervals if an interrupt hasn't occurred in that interval, to force the *CRH interruption handler (IECVCINT) (6)* to poll each channel of the inoperative processor, one at a time, for a pending I/O interruption. The operative processor processes each I/O interruption as it is taken.

- For CHS, if an I/O interruption has not occurred in the last 2 seconds, CHS connects the disconnected channel set. On exit from the timer pop procedure, the system will be enabled, allowing outstanding interruptions to be presented.
- If an I/O interruption has occurred in the last 2 seconds, calls IEAQTE00 to queue a TQE that causes an interruption 2 seconds after the time of the last I/O interruption.
- Exits to the dispatcher, since the procedure was executed asynchronously.

5. *CRH/CHS Schedule SRB Procedure (IECVCRHS)*

- Entered by TIMER SLIH, IEAOTI00, when a timer interruption occurs for the TQE queued by IECVCRHT.
- Schedules an SRB to dispatch IECVCRHT.
- Returns to IEAOTI00.

6. *CRH Second Level Interruption Handler (IECVCINT)*

- Entered by the I/O FLIH on an I/O interruption or by IECIHIO on a channel logout pending condition.
- Stores the current time in the CRH communication area (CRCA), in order to record the time of the latest interruption so that the *CRH/CHS timer pop procedure (4)* can determine whether to simulate another interuption (See the foregoing description of the *CRH/CHS timer pop procedure (4)*.)
- If the current interruption occurred from a channel connected through CRH (i.e., a channel belonging to the inoperative processor), updates location FLCIOAA in low storage so that it contains the actual device address. (The actual interruption, if received through the CRH, will appear to have come from channel 6.)
- If the interruption is for the operative processor, saves control register 2, prior to altering its contents.
- Verifies that the device address is valid.
- If entry was from IECIHIO, indicates that the last I/O operation for the device was through the CRH hardware connection, breaks the hardware connection, corrects the channel address at location FLCIOAA (as stated above), and calls the Halt I/O entry point of IECINT. IECINT returns control to the caller of IECIHIO.

- If the device address is invalid and the interruption was caused by a channel connected through CRH, breaks the hardware connection via the diagnose instruction, indicates that the interruption is from an invalidly addressed device and was received through the hardware connection, indicates that the connection has been broken, restores control register 2 with its saved value, and goes to the I/O SLIH to handle the invalid-address condition.
- If the interruption occurred on a path other than the expected path (condition is called a "channel burp"), indicates this condition in the CRH communication table (flag CRCACCH is turned on), clears the CSW to prevent further processing of this interruption by the I/O SLIH, and branches to the I/O SLIH to restart any possible queued I/O requests.
- Issues the Diagnose instruction to break the CRH hardware connection, indicates that the connection is not outstanding, restores control register 2 if it was altered to accept an interruption through CRH, and branches to the I/O SLIH (IECINT) to process the interruption.
- On return from IECINT, alters control register 2 to allow interruptions only from the channel-6 interface.
- Issues the diagnose instruction to each enabled channel connected to the in-operative processor and enables in order to accept any pending interruptions from those channels.
- Restores control register 2, issues the diagnose instruction to break the CRH hardware connection, and returns to the I/O FLIH.

7. *CHS Second Level Interruption Handler (IECVCSSI)*

- Entered by the I/O FLIH on an I/O interruption or by IECIHIO on a channel-logout-pending condition.
- Stores the current time in the CRH communication area (CRCA), in order to record the time of the latest interruption so that the *CRH/CHS timer pop procedure (4)* can determine whether to simulate another interruption. (See the foregoing description of the *CRH/CHS timer pop procedure (4)*).
- On return from IECINT, connects the disconnected channel set and enables in order to accept any pending interruptions from these channels.
- If no interruptions occur, disables and returns to the I/O FLIH.

8. *CRH/CHS Activation FRR Procedure (IECCRHAF)*

- Called by the RTM if an error occurred while the CRH/CHS activation procedure (IECVCRHA) was running.
- Immediately on entry, deactivates CRH function by zeroing the CVT pointer to the CRH communication area (CVTCRCA).
- If this is a reentry from an error in the FRR itself, or the retry to the entry point in IECVCRHA failed, clears the pointer to IECVCRHA (IOXCRHA) in the IOCOM extension, thus making CRH unavailable until the next IPL. Returns to RTM requesting FRR percolation.
- Otherwise, issues message IEA971I: "Unable to activate channel reconfiguration hardware."

- Calls backout procedure (11) which attempts to undo whatever IECVCRHA did before the error occurred.
- Returns to RTM with request to retry at entry point CRHARTRY in IECVCRHA.
- At entry point CRHARTRY sets up a return code "unable to activate CRH" (RC = 8) and returns to the caller of IECVCRHA.

9. CRH/CHS Deactivation FRR Procedure (IECCRHDF)

- Called by RTM if an error occurs while IECVCRHD is running.
- Immediately on entry, deactivates CRH/CHS function by zeroing the pointer (CVTCRCA) to the CRH communications area (CRCA).
- If this is a reentry from an error in the FRR itself, or if retry at CRHDEXIT failed, returns to RTM with a request for FRR percolation.
- Otherwise, calls the *backout procedure (11)* to try to complete deactivation of CRH.
- Returns to RTM with request to retry at retry entry point CRHDEXIT in IECVCRHD.

10. CRH/CHS SLIH FRR Procedure (IECCINTF)

- Is called by RTM if an error occurs while IECVCINT or IECVCSSI is running.
- Calls the IOS mainline FRR (IECFRR) to post the user with an I/O completion code of X'45'.
- IOS mainline FRR returns to RTM with retry request to IOS channel scheduler to start queued I/O requests.

11. Backout Procedure (BACKOUT)

- Called as a subroutine by the CRH/CHS deactivation procedure (IECVCRHD) or by recovery (either IECCRHAF or IECCRHDF) to deactivate CRH/CHS.
- If pointer to I/O SLIH in I/O FLIH points to CRH SLIH (IECVCINT) or CHS SLIH (IECVCSSI), restores pointer to the I/O SLIH (IECINT).
- If hooks have been activated in IOS mainline (IECIOSCN), removes hooks.
- For CHS only:
  1. Reconnects the appropriate channel set to the operational processor (processor x).
  2. Processor x issues an RISGNL macro which causes processor x to stop executing and the other processor (processor y) to try to execute. If processor y has been varied online, it will now connect its own channel set.
  3. RISGNL sends back a return code to processor x to indicate whether or not processor y is operational.
- For CRH only, scans UCBs to turn off CRH flags and update processor address field (UCBCPU) to reflect the processor through which the last-path I/O was started.
- Returns to caller.

12. *Connect Channel Set Procedure (IECCONCS)*

- Entered when a channel set should be connected and CHS is active, by way of pointer IOCCONCS in the IOS communications table (IOCOM).
- If CHS is not active or the requested channel set is not valid, control returns to the user with a non-zero return code.
- The currently connected channel set ID is saved, and control register 2 is loaded with the value specified in the CST for the requested channel set.
- The connect channel set instruction is issued for the requested channel set. If the condition code is zero, control returns to the user with a zero return code.
- If the requested channel set cannot be connected, the original channel set is reconnected and its control register 2 value is reloaded. Control returns to the user with a non-zero return code. If for some reason no channel set can be reconnected, the system is placed in a X'06B' wait state.

CRH Hook Module (IECVCRHH)

Note: Not used by CHS.

1. *The Test Channel Hook Procedure (IECVCRH1)*

- Entered when the test channel procedure (ETCH1) in IECIOSCN cannot find a path to start, and attempts to shoulder tap to the inoperative processor. The hook gives control to IECVCRH1 instead.
- If the I/O request was a guaranteed device path (GDP) request, tries to use the requested path. If GDP is unavailable or the channel is not enabled, calls the SRB scheduling procedure ESCHDIO1 (7 in the basic IOS module) to terminate the I/O request.
- If the device to be started is reserved, ensures that the I/O operation starts on the same channel as that for the last path started for that device.
- Ensures that the channel is available for the path chosen.
- Ensures that the path chosen is online.
- When IECVCRH1 finds an available path from the inoperative processor, issues a diagnose instruction to make the CRH connection to the desired channel. Issues a TCH instruction to test the availability of the path.
- If no path is available, issues a diagnose instruction to break the CRH connection and returns to IECIOSCN. IECIOSCN will attempt to start another I/O request.
- If a path is available, calls the required device-dependent SIO procedure to start the device.
- Issues a Diagnose instruction to break the CRH connection, and returns to IECIOSCN.

2. *The SIO Hook Procedure (IECVCRH2)*

- Entered via a hook after an SIO or SIOF instruction in the *basic IOS module (IECIOSCN) SIO procedure (3).*
- If the SIO was not issued across a CRH connection (indicated by CRCADIAG off), returns to IECIOSCN.
- If the CSW was stored (condition code is "1"), turns on UCBCRHSN flag in the device's UCB to indicate that if a sense command is needed, it must be issued across the CRH connection.
- If the SIO is successful (condition code is "0"), turns on UCBIORST in the device's UCB to indicate that the last path started was through the CRH connection.
- Updates UCBCHAN field to indicate the actual device address (not the address that reflects the channel 6 interface used by CRH).
- Returns to IECIOSCN.

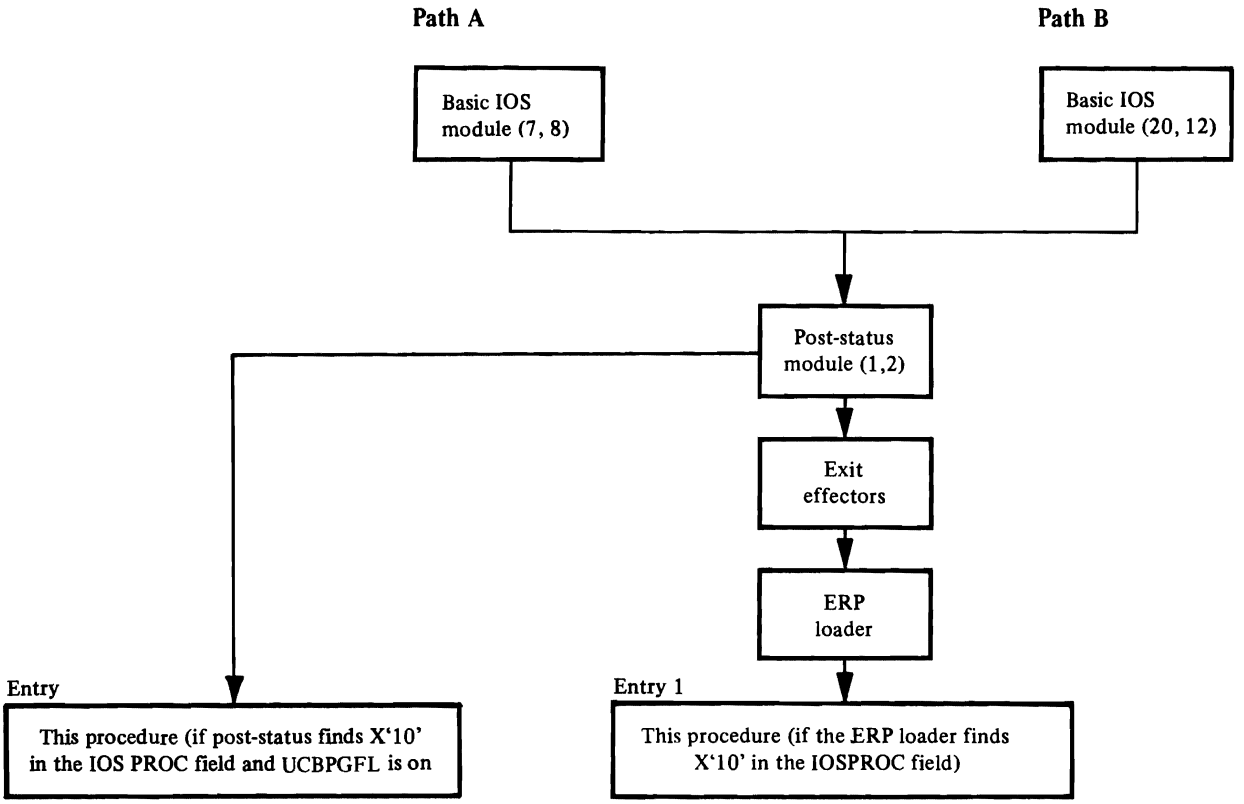
3. *The Sense Hook Procedure (IECVCRH3)*

- Entered from IECIOSCN, just before the *basic IOS module (IECIOSCN) sense procedure (14)* issues the sense SIO instruction.
- If the sense operation must be started across a CRH connection (UCBCRHSN on), and if there is no outstanding CRH connection (CRCADIAG off), issues a diagnose instruction to make the CRH connection to the required channel.
- Starts the sense operation (in place of the SIO issued by the *basic IOS module (IECIOSCN) sense procedure (14)* and saves the SIO condition code.
- If the SIO is successful (condition code is "0"), turns off UCBCRHSN to indicate a CRH sense operation is no longer required. Turns on UCBIORST to indicate that the last path started to the device was through a CRH connection.
- If a CRH connection is outstanding (CRCADIAG on), issues a diagnose instruction to break the CRH connection.
- Restores the sense SIO condition code and returns to IECIOSCN.

DAVV Module (IECVDAVV)

1. The Volume Verification Procedure (IECVDAVV)

- Entered on one of these paths:



Path A is taken if the operator readies a direct-access device and a driver subsequently submits an I/O request for that device. Path B is taken if an I/O interruption is received that resulted from an I/O request made during an earlier execution of this module on Path A—or if an I/O interruption is received because this module asked the operator to switch volumes, and in doing so, he readied the device again. Entry 2 is used if the unsolicited device end is from a page pack and runs in SRB mode. Entry 1 is the normal non-SRB mode entry.

- If entry 1, issues an ESTAE macro that gives RTM the address of the *ESTAE recovery procedure (4).*
- If entry 2, issues a SETFRR macro that gives RTM the address of the FRR recovery procedure (5).

For both entry 1 and entry 2 –

- Determines whether it was entered on Path A or B. If the latter, branches to the *interruption-handling procedure (2).*
- Builds a channel program to read the volume label on the device that was readied.

- Turns on the IOSERR bit and, if entry 1, issues an SVC 15 instruction. The SVC interrupt handler gives control to the *post-status module (3),* which issues a STARTIO macro for the channel program. If entry 2, returns directly to post-status module (3) that issues a STARTIO macro for the channel program, then exits to the dispatcher (IEAVEDS0).
- Exits to the dispatcher when the SVC interruption handler returns.

2. *The Interruption-Handling Procedure (DAVINT)*

- Entered by the *volume verification procedure (1).*
- If the *error-handling procedure (3),* during an earlier execution of this module issued messages to switch volumes, this procedure issues an SVC 87 to delete those messages from display devices and returns to the *volume verification procedure (1)* to build another channel program.
- Determines whether the volume label was read successfully. If it was, compares it to the label in the UCBVOLID field (the label of the volume the system expects to be mounted). If it wasn't read successfully, branches to the *error-handling procedure (3).*
- If the labels match, does the following:
  (a) Gets the UCB lock and turns off the UCBQISCE and UCBDAVV bits. Turns on a bit in the IRTCHMSK field that shows a channel is available to the device. (These actions cause the *basic IOS module (13)* to start driver-requested I/O operations to the device again.)
  (b) Turns off the IOSERR bit and for entry 1, issues an SVC 15 instruction. The SVC interrupt handler gives control to the *post-status module (3),* which frees the IOS-created SRB/IOSB and the ERP work area. If entry 2, returns directly to post-status module (3) that frees the IOS-created SRB/IOSB and ERP work area, then exits to the dispatcher (IEAVEDS0).
  (c) Exits to the dispatcher when the SVC interruption handler returns.
- If the labels don't match, exits to the *error-handling procedure (3).*

3. *The Error-Handling Procedure (DAVERR)*

- Entered by the *interruption-handling procedure (2).*
- If entered because the volume labels don't match, does the following:

  For entry 1 –

  (a) Issues messages using WTO macros, asking the operator to demount the currently-mounted volume and mount the volume identified in the UCB.
  (b) Acquires the UCB lock and puts the address of the SRB/IOSB in the UCBIOQ field. Turns on the UCBWDAV bit, which directs the *basic IOS module (15)* to route control to this module, using the SRB/IOSB pointed to by UCBIOQ, when the device presents the expected I/O interruption.
  (c) Exits to the dispatcher.

For entry 2 –

(a) Acquires the UCB lock and puts the address of the SRB/IOSB in the UCBIOQ field. Turns on the UCBWADV bit, which directs the basic IOS module (15) to route control to this module, using the SRB/IOSB pointed to by UCBIOQ, when the device presents the expected I/O interruption.

(b) Returns to *post-status module (6)* to load a restartable wait state, then exits to the dispatcher (IEAVEDS0).

- If entered because the volume label wasn't read successfully, does the following:

(a) Finds out if the error can be corrected with the operator's intervention. If so, and if entry 1, branches to a procedure of the ERP loader, which locates and enters the ERP message writer, IGE0025C. This module writes the "intervention-required" message to the operator. For entry 2, returns to *post-status module (6)* to load a restartable wait state, then exits to the dispatcher (IEAVEDS0).

(b) For entry 1, if the I/O operation can be retried, requests it retry with an SVC 15 instruction. If the volume label isn't read after ten such requests, issues two messages with WTO macros: one informs the operator of an uncorrectable I/O error; the other tells him to mount the volume identified in the UCB.

For entry 2, if the I/O operation can be retried, returns to post-status procedure (3). If the volume label is not read after ten such requests, returns to *post-status module (6)* to load a restartable wait state, then exits to the dispatcher (IEAVEDS0) after the restart occurs.

(c) Exits to the dispatcher.

4. The ESTAE Recovery Procedure (DAVESTA)

- Entered by RTM if any of the procedures of the *DAVV module* took a program check.
- If the *error-handling procedure (3)* issued any messages, issues an SVC 87 instruction to delete them from display devices.
- If an uncorrectable page-fault error occurred, prevents the processor it's running on from receiving interruptions and puts the system into an X'022' wait state.
- For restart of the processor if the SDUMP buffer is available, issues an SDUMP macro to write the contents of the buffer in the SYS1.DUMP data set.
- Sets bits in the SDWA (via the SETRP macro) that direct RTM to (a) record the contents of the SDWA in the SYS1.LOGREC data set and (b) return control to this procedure to avoid a continuation of termination processing.
- Branches to RTM, and when RTM returns, issues an SVC 15 instruction.
- Exits to the dispatcher with an SVC 3 instruction when the SVC interruption handler returns.

Note: This procedure puts diagnostic data in the SDUMP buffer and in the variable area of the SDWA. The data is described in the "Diagnostic Aids" chapter under "Output of the DAVV Module (IECVDAVV)."

5. *The FRR Recovery Procedure (DAVFRR)*

- Entered by RTM if any of the procedures of the DAVV module took a program check on entry 2.
- If the SDUMP buffer is available, issues an SDUMP macro to write the contents of the buffer in the SYS1.DUMP data set.
- Sets bits in the SDWA (via the SETRP macro) that direct RTM to (a) record the contents of the SDWA in the SYS1.LOGREC data set and (b) retry by returning to *post-status module (3).*
- Branches to RTM.

Note: This procedure puts diagnostic data in the SDUMP buffer and in the variable area of the SDWA. The data is described in the "Diagnostic Aids" chapter under "Output of the DAVV Module (IECVDAVV).

The Hot I/O Detection Module (IECVHDET)

1. *The Check Interruption Procedure*

- Entered from various points within the *interruption handling procedure (9) of the basic IOS module.* These points are called detection points and are locations where a particular interruption has been determined to be a type that may indicate a hot I/O condition. These types of interruptions may be as follows:
  - availability interruptions (control unit end, control unit busy, channel available)
  - interruptions without channel errors on non-SYSGENed devices
  - interruptions of unsolicited status without channel errors on SYSGENed devices
  - interruptions for channel data check, channel control check, and/or interface control check.
- Determines if an I/O operation has been successfully started on this channel since the last time hot I/O detection was entered for this channel. If so, the *reset procedure (2)* is entered. This is done to minimize false detections; as long as the channel is still usable and the system can progress, there has been no impact from the device.
- If no operation has been successfully initiated, check if the current interruption is a repeat of the previous entry for this channel. This requires that the interruption types (channel, control unit, or device) be the same, and that the device addresses are the same for device-type interruptions. If not a repeat, the *reset procedure (2)* is called.
- For repeated status, the repeat count is increased. If the time interval between this entry and the previous one for this channel exceeds the value specified in the hot I/O threshold table (HIDT), the timeout repeat count is also increased.
- The repeat count and the timeout repeat count are compared with the appropriate threshold values in the HIDT. If a threshold has been exceeded, the *schedule recovery procedure (3)* is entered. Otherwise control is returned to IOS.

2. *The Reset Procedure*

- Entered when the interruption is not a repeat or following a successful SIO on the channel.
- The status collection data area (SCD) entry for this channel is reset to reflect the new condition and the counts are set to zero.
- Control returns to *basic IOS module (IECIOSCN).*

3. *The Schedule Recovery Procedure*

- Entered when a value in the HIDT table exceeds the threshold value.
- Disables interruptions from the affected channel in the CST, CAT, and control register 2.
- Obtains an SRB from SQA and schedules the hot I/O recovery module IECVHREC.
- Returns to the *basic IOS module (IECIOSCN).*

Hot I/O Recovery Module (IECVHREC)

1. *The Set Up Procedure*

- Hot I/O detection module (IECVHDET) schedules IECVHREC as an SRB.
- Establishes an FRR.
- Obtains a workarea using the GETMAIN macro and initializes this workarea.
- Disables and sets the IOS 'super' bit to indicate disabled without a lock.

2. *The Hot Device Recovery Routine*

- Entered at the completion of the *set up routine (1)* if the status collection data area (SCD) indicates that the condition detected was a hot device and that the device was sysgened and not a direct access (DASD) device.
- Calls module IECVBRSV to build a table of all the reserved devices on the same channel as the hot device.
- Calls internal procedure SETUPBUF to set up the LOGREC buffer for recording.
- Initializes the disabled console communication block (DCCB) and the message buffers for message IEA066A (restartable wait state X'066') if there are no reserves on the same channel, or message IEA067A (restartable wait state X'067') if there are reserves on the same channel.
- Calls module IEEVLDWT to issue a message to the operator if the console is available or else load a restartable wait.

- Examines the operator response to the message or restartable wait state as follows:
  (a) Operator response indicates that nothing was removed from the configuration.
    - No action is required.
  (b) Operator response indicates that a device was removed from the configuration.
    - Device is boxed by calling module IECVRRSV.
  (c) Operator response indicates that a control unit was removed from the configuration.
    - All devices removed by the operator are boxed by calling module IECVRRSV.
  (d) Operator response indicates that the channel is to be removed from the configuration.
    - Sets indicator that the channel is not to be enabled.
    - Calls *subroutine CLRCH (5)* to re-reserve devices and re-drive I/O as required.
    - If CLRCH routine was successful (channel was reset and the I/O redriven) issues message IEA071E using the RECORD macro to indicate that the channel was forced offline and disabled.
- If the operator response was other than to remove the channel, then the channel is enabled for interruptions.
- A channel available interruption is simulated by using the IOSINTRP macro to redrive I/O for the device.
- The IOS 'super' bit is turned off since IOSINTRP will cause the enabled state.

3. The Hot Channel, Hot Control Unit, and Hot DASD Recovery Routine

- Entered at the completion of the set up routine (1) if the status collection data area (SCD) indicates a hot channel, a hot DASD, or a device which was not SYSGENed.
- If the channel was not SYSGENed, issues IEA071E message indicating that the channel is disabled and offline and exits to the *clean up procedure (4).*
- If this is the first recovery attempt for this channel, sets an indicator that the channel can be enabled after recovery; otherwise indicates that the channel cannot be enabled after recovery.
- Calls module IECVBRSV to build the reserve table.
- If processor supports the CLRCH instruction and there are no reserved devices on the channel, issues message IEA072I to inform the operator that recovery is in progress.
- If there are reserved devices or if the processor does not support the CLRCH instruction, calls module IEEVLDWT to issue a message.

  If there are reserves, message IEA069A is issued or wait state X'069' is loaded. If there are no reserves, message IEA068A is issued or wait state X'068' is loaded.

- Saves operator response indicating whether or not a channel was reset.
- Invokes *subroutine CLRCH (5)* to attempt recovery of the failing channel and the active I/O requests.
- If the CLRCH routine indicates that channel reset was not successful, clears the reason codes and counts in the SCD.
- If channel reset was successful, does the following:
  (a) If this is the first detection for the channel, enables the channel and reduces the threshold counts so a recurrence can be detected sooner.
  (b) If not the first detection for this channel, leaves the channel disabled and issues message IEA071E.

4. *The Clean Up Procedure*

- FREEMAINs the work areas.
- Enables and then turns off the IOS 'super' bit.
- Deletes the FRR and returns to the dispatcher.

5. *The Clear Channel Subroutine (CLRCH)*

- Called by the *hot device recovery procedure (2)* and the *hot channel* and *hot DASD recovery procedure (3).*
- Issues the clear channel instruction (CLRCH) if it is supported by the processor. When the CLRCH is not supported, checks to see if the operator has reset the channel and if so issues a HDV, CLRIO instruction sequence to each device on the channel.

  If either the CLRCH was successful or the operator reset the channel, calls IECVRRSV to re-reserve devices and issues message IEA421E, if necessary; also calls IECVRDIO to redrive active I/O.
- If the CLRCH fails or if the operator did not reset the channel (the processor does not support the CLRCH instruction), issues message IEA070A or loads restartable wait state X'06A'. Then devices which lost their last path are boxed.

6. *The Functional Recovery Routine (HRECFRR)*

- RTM enters the FRR whan an error is encountered in IECVHREC.
- Sets an indicator for RTM to request recording.
- Requests RTM to free SALLOC if held.
- If this is first entry to the FRR, does the following:
  (a) Establishes an FRR.
  (b) Cleans up counts in the SCD.
  (c) Calls module IECVRRSV to re-reserve devices if required.
  (d) Enables channel if required.
  (e) Takes an SDUMP.

- If this is a recursive entry to the FRR and the channel has been reset but re-reserves and boxing are not complete, the FRR calls module IEEVLDWT to issue IEA151W and to load non-restartable wait state X'04E'.
- Issues a 'start stopped processors' message if required.
- FREEMAINs work areas.
- Deletes FRR if this is first entry to FRR.
- Turns off the IOS 'super' bit and returns to RTM.

I/O - Restart Modules (IECVRSTI and IECVIRST)

Introduction to I/O Restart Modules, IECVRSTI and IECVIRST

The I/O restart function is composed of two modules: IECVRSTI and IECVIRST. ACR, mainline CCH, CCH MCH exit, and MIH enter IECVRSTI. CCH MCH exit or mainline CCH schedules IECVIRST as an SRB.

IECVRSTI is entered from CCH in one of two ways. Mainline CCH branch enters IECVRSTI when any of the following occurs:

- The channel fails to log the failure.
- The channel's CAT entry is invalid.
- NIP failed to load a required channel-dependent module.
- Mainline CCH's FRR is entered due to an error.

If the external-damage-reason-code-validity flag is zero (in the MCIC) or if the flag is one and the channel control failure flag (bit 4 in the external damage reason code) is one, the second CCH entry, CCH MCH Exit, schedules IECVRSTI as an SRB.

Mainline CCH schedules this module as an SRB if the hung interface indicator is on in the LCL (limited channel logout).

CCH MCH exit schedules the SRB if the external-damage-reason-code-validity flag (in the MCIC) is one and the channel control failure flag (bit 4 in the reason code) is zero. During ACR, if the CCH MCH exit used to schedule IECVIRST indicates (in the channel recovery work area) that IECVIRST has been scheduled but not dispatched, IECVRSTI branches to IECVIRST.

Note:

Assumptions made by CCH-IOS:

1. If the hung interface flag is set in the LCL or the external-damage-reason-code-validity flag is set in the MCIC, the hardware supports the clear channel instruction (CLRCH).

2. The hardware does not reset reserves on devices unless the external damage reason code validity flag in the MCIC is zero or the validity flag is one and the channel control failure flag is one. The hardware has already released reserved devices in these cases.

1. *The Set-Up Procedure (IECVRSTI)*

- Entered by ACR, mainline CCH, and MIH.
- Scheduled as an SRB by CCH MCH exit.
- Issues GETMAIN macro to obtain storage for a work area then calls the *storage manager module (IECVSMGR) get-large-block procedure (5)* to get a 160-byte block to be used as a message buffer.
- Exits based on the contents of register 1. The possible values in the high-order byte and the corresponding exits are:
  (a) *X'00'*: Exits to the *ACR-call procedure (2).*
  (b) *X'20'*: Exits to the *MIH-call procedure (4).*
  (c) *X'30'*: Exits to the *CCH-call procedure (7)* for lost channels.
  (d) *X'40'*: Exits to the *CCH-call procedure (3)* for channel checks.

2. *The ACR-Call Procedure (ACRPROC)*

- Entered by the *set-up procedure (1).*
- If CRH or CHS is generated in the system, make sure the inoperative processor is stopped. Calls CRH/CHS activation (IECVCRHA) routine to activate channel reconfiguration hardware or channel set switching. If successful, and IECVIRST was scheduled to run on the failing processor, branches to IECVIRST. It then returns to ACR.

  If CRH/CHS activation was not successful, it calls the *storage manager module (IECVSMGR) get-large-block procedure (5)* to get storage for a "reserve" table, a table that will show the status of all devices reserved by the nonoperational processor.
- Issues a SIGP instruction to stop on-going I/O operations.
- For each device on the system it calls the *device procedure (8).*
- If none of the devices were reserved by the nonoperational processor, it does the following:
  (a) Calls the *storage manager module (IECVSMGR) free-large-block procedure (6)* to free the 160-byte blocks it acquired.
  (b) Exits to ACR.
- Using the IOSMAP macro, ACRPROC gets a table containing information about the accessibility of each device from the operational processor and does the following:
  (a) Issues a SIGP instruction to stop on-going I/O operations.
  (b) For each active device, sets up an ERPIB (ERP interface block), which the *basic IOS module (IECIOSCN) initial status procedure (10)* moves into the ERP work area. This work area directs the ERP to retry the I/O operation, provided that an alternate path to the device is available. If no path is available to the device, no retry is indicated. Puts X'08' in register 11, turns on the channel-control-check and interface-control-check bits in the CSW, and branches to the *basic IOS module (IECIOSCN) interruption handling procedure (9).* The codes tell the *basic IOS module (IECIOSCN)* to bypass calling CCH, bypass enabling the system to receive I/O interruptions, and bypass starting I/O requests that are waiting on queues.

(c) For each inaccessible device, marks the UCBSTAT field to show that the device is offline and inaccessible to the system. Marks the UCBFLB field to identify itself as the module that set the device offline. (These fields are examined by the *basic IOS module (IECIOSCN) test channel procedure (2)* to ensure that an I/O operation can be started to the device.) Calls the *message procedure (6),* each time it has processed eight UCBs, to issue message IEA004I, which identifies the inaccessible devices.

(d) If none of the inaccessible devices are reserved by the non-operational processor, ACRPROC calls the *storage manager module (IECVSMGR) free-large-block procedure (6)* to free the 160-byte blocks it acquired prior to returning to ACR.

(e) Exits to ACR.

- If any device is reserved by the nonoperational processor, does these things:

  (a) Puts an entry in the reserve table for each device reserved to the nonoperational processor. Indicates in each entry whether there is an online path from the operational processor to the device.

  (b) Issues message IEA440A or puts the system in a 041 wait state until the operator signals that no other system is sharing a reserved device, and then issues a SIGP instruction to release all reserved devices.

  (c) Calls IECVRRSV to reserve to the operational processor or force offline the devices in the reserve table that are online to that processor.

  (d) Calls the *storage manager module (IECVSMGR) free-large block procedure (6)* to free the 160-byte blocks it acquired, and exits to ACR.

3. *The CCH Call Procedure (CCHPROC) for Channel Checks*

- Entered by the *set-up procedure (1).*
- Calls the *clear-device procedure (5)* to stop any on-going communication between the channel and device.
- Tests the IORFLAGS field of the I/O-restart work area to see if the device was successfully "cleared." If it wasn't does the following:

  (a) Sets up a field in the ERP work area that subsequently directs the ERP not to retry the I/O operation.

  (b) Turns on the channel-control-check and interface-control-check bits in the CSW—one, but not both, may already be on. These bit settings are used as a diagnostic aid, showing this module was entered.

- Exits to the *basic IOS module (IECIOSCN) interruption handling procedure (9),* where I/O-event processing is resumed.

4. *The MIH-Call Procedure (MIHPROC)*

- Entered by the *set-up procedure (1).*
- Calls the *clear-device procedure (5)* to stop any on-going communication between the channel and device.
- Tests the "pending-sense" (UCBPSNS) and "active-sense" (UCBASNS) bits, and based on their settings, alters the CSW as follows:
  (a) If a sense operation is needed but has not yet been started *(UCBPSNS on, UCBASNS off):* turns on the device-end bit, allowing the *basic IOS module (9),* when it subsequently gets control, to reuse the device.
  (b) If a sense operation was started *(UCBPSNS on, UCBASNS on):* turns on the unit-check bit, causing the *basic IOS module (IECIOSCN) interruption handling procedure (9),* when it subsequently gets control, to reissue the sense operation.
  (c) If the last I/O started to this device was not a sense operation *(UCBPSNS off, UCBASNS off):* turns on the channel-control-check and interface-control-check bits as a signal to the *basic IOS module (IECIOSCN) interruption handling procedure (9)* that control must be routed to an ERP.
- Tests the IORFLAGS field of the I/O-restart work area to see if the device was successfully "cleared." If it wasn't, sets up a field in the ERP work area that subsequently directs the ERP not to retry the I/O operation.
- Puts X'08' in register 11, telling the *basic IOS module (IECIOSCN) interruption handling procedure (9)* not to go to CCH or allow the system to receive I/O interruptions, and branches to the basic IOS module. This establishes a path to the appropriate ERP.
- When control returns from the *basic IOS module (IECIOSCN) interruption handling procedure (9),* exits to MIH.

5. *The Clear-Device Procedure (CLEARDEV)*

- Entered by the *CCH-call procedure (3)* and the *MIH-call procedure (4).*
- Gets from the UCBCHAN field the device address last used in starting an I/O operation. Uses it in HDV and CLRIO instructions to "clear" the device (stop any on-going communication between the device and channel).
- If the condition code setting for either instruction shows that the device didn't clear, calls the *message procedure (6)* to issue message IEA003I. Marks the IORFLAGS field of the I/O-restart work area to show the device didn't clear.
- Exits to the return address in register 14.

6. *The Message Procedure (RECORDIT)*

- Entered by the *ACR-call procedure (2)* to issue message IEA004I by the *clear-device procedure (5)* to issue message IEA003I; by the *device procedure (8)* to issue message IEA004I, by the *CCH-call procedure (7)* for lost channel to issue messages IEA004I and IEA410E.
- Puts the message it's passed into the list form of a WTO macro, and using the RECORD macro, passes the address of the WTO macro to the system's asynchronous recording facility. (The message is written when the message-writing procedure in the asynchronous recording facility is dispatched.)
- Exits to the return address in register 14.

7. *The CCH-Call Procedure (LOSTCHAN) for Lost Channels*

- Entered by the *set up procedure (1).*
- Calls the *storage manager module (IECVSMGR) get-large-block procedure (5)* to get 160-byte blocks for message buffers and reserve tables.
- Scans the specified channel set's CAT for lost channels. If CRH is active, scans the other channel set's CAT for lost channels.
- Calls the *message procedure (6)* to issue message IEA410E which informs the operator of the lost channel(s).
- For each device on each lost channel, it calls the *device procedure (8).*
- If there are reserved devices on the lost channel(s), it calls IECVRSSV to force offline each such device. It calls the *storage manager module (IECVSMGR) get-large-block procedure (5)* to free all acquired 160-byte storage blocks and then exits to the SRB dispatcher.

8. *The Device Procedure (UCBACT)*

- Called by the *ACR-call procedure (2)* or the *CCH-call procedure (7)* for lost channels.
- Using the IOSMAP macro, determines if an alternate path to the device exists from either the operational processor, if called by the *ACR-call procedure (2);* or from any alternate paths, if called by the *CCH-call procedure (7)* for lost channels.
- When no alternate paths are available, it marks the UCBSTAT field for each device to show that it is offline and inaccessible to the system. It marks the UCBFLB field to inform IOS that any I/O requests for that device are to be marked in permanent error. These fields are examined by the *basic IOS module (IECIOSCN) test channel procedure (2)* to ensure that an I/O operation can be started to the device. It calls the *message procedure (6)* to issue message IEA004I which identifies the inaccessible devices after they have been processed.

- If the device is reserved on the nonoperational processor (ACR) or on the lost channel (LOSTCHAN), add the device to the reserve table.
- If the device is active on the nonoperational processor (ACRPROC) or on the lost channel (LOSTCHAN), set up an ERPIB (ERP interface block) which the *basic IOS module (IECIOSCN) initial status procedure (10)* will move into the ERP work area. The ERPIB directs the ERP to retry the I/O operation if an alternate path to the device exists or to set no retry if no alternate path exists. It puts X'0A' in register 11, turns on the *channel control check* and the *interface control check* bits in the CSW and branches to the *basic IOS module (IECIOSCN) interruption handling procedure (9).* The codes tell the basic IOS module to bypass calling CCH, bypass enabling the system to receive I/O interruptions, and bypass starting I/O requests that are waiting in queues.
- Exits to caller.

I/O Restart Module - IECVIRST

1. The Set Up Procedure:

- CCH schedules IECVIRST as an SRB (see "Introduction to I/O Restart Modules IECVRSTI and IECVIRST).
- Disables for I/O and external interruptions; sets the IOS 'super flag'; establishes an FRR.
- Obtains a storage work area by issuing the GETMAIN macro.

2. The Build Reserve Table Routine:

- Entered at the completion of the *set up routine (1)* or from the restart active I/O routine (8).
- Scans the CAT to determine which channel(s) have errors (CCH sets flags in the CAT to indicate the entry type and channel(s) in error).
- Calls build reserve table module (IECVBRSV).

3. The Operator Communication Routine:

- Entered at the completion of the *build reserve table routine (2).*
- Communicates with the operator by calling IEEVLDWT which issues messages and loads re-startable wait state codes.

  If IECVIRST is scheduled from mainline CCH and no reserved devices are found and the error was a hung interface, this routine is bypassed. If reserved devices are found, the table is passed to the operator as a parameter when the wait state is loaded.

- The operator elects to re-IPL the system or restart the system (perform the restart function). If the system is restarted and CCH MCH exit scheduled IECVIRST, the operator responds with an action code indicating whether the installation wishes to reuse the failing channel(s) (attempt recovery) or not to reuse the failing channel(s) (they are forced offline). If scheduled by the CCH MCH exit, the failing channel(s) may be recovered by performing a re-IPL.

- If supplied, the action code is saved for later use.

- Enters the *recover unusable channel routine (4)* if scheduled by CCH MCH exit. Otherwise, it enters the *recover hung interface routine (6)* scheduled by mainline CCH.

4. *The Recover Unusable Channel Routine:*

- The *operator communication routine (3)* enters this routine when the CCH MCH exit schedules IECVIRST.

- Issues the CLRCH instruction to all channels that respond CC=3 to a TCH instruction. The channels that support the 'external damage reason code' are gathered into 'groups.' If the channel error is group-related, it is necessary to issue a CLRCH to every channel in the group to initiate a possible recovery. (Every channel in the group responded CC=3 to a TCH instruction).

- The action code the operator supplies in the *operator communication routine (3)* is examined. If the installation does not wish to reuse the failing channels, they are forced offline and the *re-reserve device routine (7)* is entered. If an attempt is made to reuse the failing channels, the *wait-for-channel-to-recover routine (5)* is entered.

5. *The Wait For Channels to Recover Routine:*

- The *recover unusable channel routine (4)* enters this routine when the installation elects to reuse the failing channels.

- Calls SETDIE to establish a DIE exit to get control in 60 seconds. Exits to the SRB dispatcher if SETDIE is able to establish the DIE. The DIE routine re-schedules IECVIRST to continue processing. The 60 second wait allows the channel group to re-initialize.

- Issues a TCH instruction to each failing channel to determine whether recovery was successful (failing channels are flagged in the CAT by CCH MCH exit). Channels failing to recover are forced offline.

- This routine enters the *re-reserve device routine (7)* when the *build reserve table routine (2)* finds any reserved devices. Otherwise, processing continues with the *restart active I/O routine (8).*

6. *The Recover Hung Interface Routine:*

- The *operator communication routine (3)* enters this routine when mainline CCH schedules IECVIRST.
- Issues a CLRCH instruction to a channel encountering a hung interface condition (mainline CCH flags failing channels in the CAT).
- Issues TCH and TIO instructions to determine success of recovery action. A channel recovers if the CLRCH and TCH instructions return a CC=0 and the TIO does not result in another hung interface condition. If the channel does not recover, it is forced offline and not reused.
- Enters the *re-reserve device routine (7)* if reserved devices are found in the build reserve table. Otherwise, it enters the *restart active I/O routine (8).*
- Issues message IEA410E if the channel does not recover from the hung interface condition.

7. *The Re-Reserve Device Routine:*

- Entered from the *recover unusable channel(s) routine (4),* the *wait-for-channels-to-recover routine (5),* or the *recover hung interface routine (6)* if reserved devices are found by the *build reserve table routine (2).* If no reserved devices are found, this routine is bypassed and processing continues with the *restart active I/O routine (8).*
- Invokes IECVRRSV to re-reserve each device.
- After all re-reserves are completed, issues message IEA421E.

8. *The Restart Active I/O Routine:*

- Invokes IECVRDIO to restart I/O to each device attached to the failing channels.
- If another group error occurs while IECVIRST is recovering a group error, IECVIRST will return to *build reserve table routine (2).*
- Returns resources to the system, and exits to the SRB dispatcher.

9. *The Functional Recovery Routine:*

- RTM enters the FRR routine when an error is encountered in IECVIRST.
- Causes a disabled wait state to be loaded when reserved devices are found but not re-reserved, and if the CLRCH instruction has been issued to the unusable channel(s).
- Resets the SRB to cause IECVRSTI to re-enter IECVIRST if the FRR was entered by ACR.

- Otherwise, invokes SVC DUMP to dump work areas.
- Returns resources to the system.
- Returns to RTM indicating that recording of the error and percolation should take place.

Nonresident Halt - I/O Module (IGC0003C)

1. The Main Halt Procedure (IGC0003C)

- Entered if a user or system program running under a TCB issues an SVC 33 instruction (or an IOHALT macro, which expands into an SVC 33 instruction). JES3, BTAM and TCAM are the system callers.
- Issues a GETMAIN macro for a register save area.
- Acquires the lock for the UCB pointed to by register 1, first issuing a PGFIX macro to prevent itself from being paged out.
- Issues a SETFRR macro that gives RTM the address of the *functional recovery procedure (3).*
- Determines if invoker is authorized to halt device.
- If halt is for an inactive channel-to-channel device or a selector channel, goes to *CTC halt procedure (2).*
- Tests the second byte of register 1 and takes the associated action:
  (a) *If it's X'00':* Calls the *resident halt-I/O module (1)* to halt a channel program with an HDV (halt device) instruction.
  (b) *If it's X'80':* Calls the EXCP miscellaneous module to halt a channel program on a teleprocessing device by modifying a CCW. (Register 0 contains the offset from the IOB to an untranslated CCW. Its translated counterpart is the CCW to be modified.)
- Frees the UCB lock and issues a SETFRR macro that deletes the address of the *functional recovery procedure (3)* from RTM's stack of such procedures.
- Makes pageable all fixed pages and frees save area.
- Puts one of these return codes in register 15:
  (a) *X'00':* The channel program was halted using the method specified in register 1.
  (b) *X'04':* No channel program was active on the device.

(c) *X'08'*: The device is not a teleprocessing device or graphics device.

(d) *X'0C'*: An invalid UCB address was passed or an I/O error was encountered in trying to halt the device.

(e) *X'10'*: The EXCP processor was called to halt the channel program but the original EXCP request did not need CCW translation.

(f) *X'14'*: The EXCP processor was called to halt the channel program. It found that the CCW pointed to by register 0 was not yet translated.

(g) *X'18'*: The EXCP processor was called to halt the channel program. It found that the CCW pointed to by register 0 was not part of the original channel program.

(h) *X'1C'*: The EXCP processor was called to halt the channel program, but it wasn't the driver submitting the original I/O request.

- Exits to the return address in register 14.

2. *CTC Halt Procedure (HALT3000)*

- Entered by *main halt procedure (1)* to halt channel-to-channel (CTC) devices on a selector channel if the UCB is not active, to avoid halting the wrong device.
- Schedules an SRB to cause entry to a routine under control of the desired processor. When dispatched, the SRB-controlled routine does the following steps.
- Issues test channel instruction to clear the selector channel.
- Gets the UCB lock to prevent the other processor (in a multiprocessor) from issuing I/O from or to the device.
- Issues another test channel instruction to ensure that the channel is still clear.
- Calls the *main procedure* (IECIHIO) in the resident halt I/O module (IECIHIO) to halt a channel program by means of a halt device (HDV) instruction.
- Frees the UCB lock and returns to the *main halt procedure (1)* in the nonresident halt-I/O module (IGC0003C) by posting its ECB.

3. The Functional Recovery Procedure (HALT0900)

- Entered by RTM if the *main halt procedure (1)* took a program check.
- Releases the UCB lock.
- Directs RTM to continue termination processing. If it is the first recovery procedure to be entered by RTM, sets bits in the SDWA (via the SETRP macro) that direct RTM to write the SDWA in the SYS1.LOGREC data set and issue a user dump (a SYSUDUMP, SYSMDUMP, or SYSABEND dump).
- Exits to the return address in register 14.

Note: This procedure puts diagnostic data in the variable area of the SDWA. The data is described in the "Diagnostic Aids" chapter under "Output of the Nonresident Halt-I/O Module (IGC0003C)."

Nonresident Purge Module (IGC0001F)

1. The Entrance/Exit Procedure (IGC016)

- Entered with an SVC 16 instruction by a user or system program running under a TCB. The checkpoint SVC routine, RTM, the region control task, and the task-close routine are among the system callers.

  Note: Also entered by problem-state drivers wanting IOS to decrease the count in the IPIB of partially-processed I/O requests. (The drivers differentiate themselves from other callers by passing the complement of the IPIB address in register 1.) This procedure calls the *resident purge module (1)* to decrease the count and returns to the driver.
- Gets local subpool 0 storage for a PIRL and global storage for a PWA (purge work area). Copies the PPL into the PWA.
- Issues an ESTAE macro that gives RTM the address of the *ESTAE recovery procedure (12)*.
- Issues a PGFIX macro to prevent this module and the PIRL from being paged out (a precondition for acquiring the UCB lock in the *UCB-purge procedure (4)*).

- Exits to the caller if any of these incompatibilities is found:

  (a) The caller is not in supervisor state but requested that I/O requests associated with a given address space be purged.

  (b) The caller asked that just the I/O requests to a given data set be purged, but the field that identifies the data set contains zeros.

  (c) The caller is not in supervisor state and asked that I/O requests associated with a given TCB be purged, but that TCB does not point to the job-step TCB.

  Incompatibilities (a) and (b) are denoted by a return code of X'08' in register 15. Incompatibility (c) is denoted by a return code of X'04' in register 15.

- Ignores a request to purge asynchronous processing associated with I/O requests being quiesced if the purge operation applies to the caller's own TCB.
- Initializes the IPIB.
- If a quiesce operation was requested, issues an ENQ macro to prevent this module from handling another quiesce request in the same address space until it has processed the current request. (The major and minor enqueuing names are SYSZEC16 and PURGE, respectively.)
- If the system's ENQ routine returns with a nonzero return code and the quiesce operation does not apply to an address space, returns to the caller with a code of X'14' in register 15.
- Issues a STATUS macro, stopping all processing in the address space that runs under a TCB or under a "quiesceable" SRB (one that represents processing that may be stopped).
- Indicates that a purge operation is active in the address space by (a) storing the IPIB address in the ASCBIOSP field of the ASCB and (b) increasing the count of active purge operations in the IOCPGCT field of the IOCM (IOS communications table).
- Acquires the local lock and issues a SETFRR macro that gives RTM the address of the *functional recovery procedure (11).*

  (**Note**: The caller must have turned on bit 7 of byte 0 in the PPL to receive return codes.)

- Performs DEB validity check, if validity checking is required, or if the caller is not in supervisor state (for dataset identifier (DSID) purge), by going to the system *DEB validity check* routine.
- Calls the *SIRB-purge procedure (2)*, which disposes of applicable SRB/IOSBs chained to the SIRB or to the asynchronous exit queue.
- Calls the *LCH-purge procedure (3)*, which disposes of applicable SRB/IOSBs belonging to I/O requests on logical channel queues.
- Calls the *UCB-purge procedure (4)*, which disposes of SRB/IOSBs belonging to active I/O requests.
- Calls the *DDR-purge procedure (5)*, which disposes of SRB/IOSBs belonging to I/O requests waiting to be assigned to another device by DDR.
- Releases the local lock and issues another SETFRR macro to delete the address of the *functional recovery procedure (11)* from RTM's stack of such procedures.
- Calls the *SPL-purge procedure (6)*, which disposes of applicable SRB/IOSBs waiting on the service priority list to be dispatched.
- Calls the *IPIB-purge procedure (7)*, which disposes of applicable SRB/IOSBs given to the *basic IOS Module (11)* by the drivers' DIE procedures.
- Calls the *driver interface procedure (8)*, which enters the drivers' purge procedures.
- Calls the *storage manager module-purge-free procedure (8)* to free IOS storage allocated to the address space if the caller requested that all I/O requests in the address space be halted.
- Decreases the count of active purge operations in the IOCPGCT field.
- Issues another STATUS macro so that processing in the address space can resume.
- If a quiesce operation was requested, tests the count of the partially-processed I/O requests in the IPIBCNT field. If the count is not zero, does the following:

  (a) Issues a WAIT macro, putting the module in a wait state until the count becomes zero.

  (b) When the wait ends, repeats the calls to the other procedures of this module.

  (c) Tests the count again.

  Loops on steps (a, b, and c) until the count is found to be zero at step (c).
- Calls PRGCOMP0 routine to schedule the IECVSCOM (10) routine in IECVSMGR to compress the storage manager's free queue.
- If an ENQ macro was previously issued, issues a DEQ macro to allow another quiesce operation in the address space.

- Performs DEB validity check for non-supervisor callers to check additional DEBs, by use of the system *DEB validity check routine.*
- Issues another ESTAE macro to delete the address of the *ESTAE recovery procedure (12)* from RTM's stack of such procedures.
- Uses a PGFREE macro to unfix the storage previously fixed.
- Frees the PWA, and the PIRL, unless the PIRL shows that there are I/O requests to be restored.
- Exits to the return address in register 14.

2. *The SIRB-Purge Procedure (SIRBPURG)*

- Entered by the *entrance/exit procedure (1).*
- For each SRB/IOSB chained to an asynchronous exit queue, calls the *applicability-check procedure (9)* to find out if the SRB/IOSB matches the search argument.
- Removes each applicable SRB/IOSB from the asynchronous exit queue. Calls the *basic purge procedure (10)* to dispose of them.
- Calls the *basic purge procedure (10)* to dispose of the SRB chained to the SIRB if it matches the search argument. If reentered at register 14+4 (a match was found), replaces the SRB address with the address of the ERP work area, marks the SIRB "purged," and alters the resume PSW in the SIRB to point to the *resident purge module (2).*
- Exits to the return address in register 14.

3. *The LCH-Purge Procedure (LCHPURG)*

- Entered by the *entrance/exit procedure (1).*
- Does the following for each logical channel queue that contains an I/O queue element (IOQ):
  (a) Acquires an LCH lock.
  (b) Calls the *applicability-check procedure (9)* for each I/O request found on the LCH.
  (c) If a halt operation was specified, gets the lock for the UCB associated with applicable SRB/IOSBs. Calls the *basic purge procedure (10)* to dispose of applicable SRB/IOSBs that aren't associated with a reserve, release, or pending sense operation. Chains those that are to the appropriate PIRL entry and changes the IOQs to point to replacement SRB/IOSBs, allowing the processing of reserve, release, and pending sense operations to complete. Releases the UCB lock.
  (d) If a quiesce operation was specified, increases the count of partially-processed I/O requests in the IPIBCNT field and stores the IPIB address in the IOSB.
  (e) Releases the LCH lock.
- Exits to the return address in register 14.

4. *The UCB-Purge Procedure (UCBPURG)*

- Entered by the *entrance/exit procedure (1).*
- Scans UCBs for indications of active I/O operations on associated devices. Does the following for each such UCB it finds:
  (a) Gets the UCB lock.
  (b) Calls the *basic purge procedure (10),* if a quiesce operation was requested, to dispose of an applicable SRB/IOSB.
  (c) If a halt operation was requested, calls the *basic purge procedure (10)* to dispose of an applicable SRB/IOSB that isn't associated with a pending sense operation. If an applicable SRB/IOSB is associated with a pending sense operation, chains it to the appropriate PIRL entry and changes the IOQ to point to a replacement SRB/IOSB, allowing the sense operation to start.
  (d) Releases the UCB lock.
- Exits to the return address in register 14.

5. *The DDR-Purge Procedure (DDRPURG)*

- Entered by the *entrance/exit procedure (1).*
- For each DDR element on the DDR queue (pointed to by the ASXBDDR field of the ASCB extension), calls the *applicability-check procedure (9).*
- If a quiesce operation was requested, does the following for each DDR element that matches the search argument:
  (a) Increases the count of partially-processed I/O requests in the IPIBCNT field.
  (b) Stores the IPIB address in the associated IOSB.
- If a halt operation was requested, does the following for each DDR element that matches the search argument:
  (a) Chains the associated SRB/IOSB to the appropriate PIRL entry.
  (b) Marks the DDR element "purged."
- Gets the CMS lock.
- Processes the DDR queue in the master scheduler's address space as it did the previous DDR queue.
- Releases the CMS lock.
- Exits to the return address in register 14.

6. *The SPL-Purge Procedure (SPLPURG)*

- Entered by the *entrance/exit procedure (1)*.
- For each SRB on the SPL (service priority list), does the following:
  (a) Using the PURGEDQ macro, calls the system's SPL purge module, which dequeues the SRB, if it's applicable, and gives it to the procedure whose address is in the SRBRMTR field—in this case the *resident purge module (3)*. (The *resident purge module (3)* chains the SRB/IOSB to the IPIB.)
  (b) Calls the *basic purge procedure (10)*, if a halt operation was requested, to dispose of any applicable SRB/IOSB it finds chained to the IPIB.
  (c) If a quiesce operation was requested or an SRB/IOSB, and after further checking, is not really applicable, removes the SRB/IOSB from the IPIB and puts the SRB back on the SPL with a SCHEDULE macro.
- Exits to the return address in register 14.

7. *The IPIB-Purge Procedure (IPIBPURG)*

- Entered by the *entrance/exit procedure (1)* to dispose of SRB/IOSBs that are chained to an IPIB. (An SRB/IOSB is chained to the IPIB by the *basic IOS module (10)* if a driver's DIE procedure submits the SRB/IOSB while IOS, in the other processor, is halting I/O requests with such SRB/IOSBs.)
- Gets the IOSYNCH lock.
- Zeros out the IPIB pointer in the ASCBIOSP field of the ASCB.
- Chains each SRB/IOSB queued to the IPIB to the appropriate PIRL entry.
- Releases the IOSYNCH lock.
- Exits to the return address in register 14.

8. *The Driver Interface Procedure (DVRPURG)*

- Entered by the *entrance/exit procedure (1)*.
- Finds the addresses of the driver's purge procedures in the VOID (vector of IOS drivers) table. Puts the IPIB address in register 1 and the address of the PIRL entry to which SRB/IOSBs are chained in the IPIBSRB field. Calls the drivers' purge procedures.
- When a driver's purge procedure returns, moves the IPIBIO and IPIBDVRU fields to the PIRRSTR and PIRDVRU fields, respectively. The PIRL fields will be used by the driver's restore procedure to restore I/O requests.
- Exits to the return address in register 14.

9. The Applicability-Check Procedure (PURAPLSR)

- Entered by the *SIRB-purge procedure (2)*, the *LCH-purge procedure (3)*, the *DDR-purge procedure (5)*, and the *basic purge procedure (10)*.
- Compares a field in the SRB/IOSB or DDR element to the search argument in the PPL to find out if the purge request applies to the I/O request they represent.
- Exits to the return address in register 14 if it finds a match; to register 14+4 if it doesn't.

10. The Basic Purge Procedure (BASICPRG)

- Entered by the *SIRB-purge procedure (2)*, the *LCH-purge procedure (3)*, the *UCB-purge procedure (4)*, and the *SPL-purge procedure (6)*.
- Calls the *applicability-check procedure (9)* to find out if the SRB/IOSB matches the search argument. If it doesn't, exits to the return address in register 14+4.
- If a quiesce operation was requested, increases the count of partially-processed I/O requests in the IPIBCNT field, stores the IPIB address in the IOSB, and exits to the return address in register 14.
- If a halt operation was requested, does the following:
  (a) If the caller is the *UCB-purge procedure (4)*, calls the *resident halt-I/O module (IECIHIO) main procedure (1)* to halt the I/O operation and turns off the "busy" bits in the UCBFLA field.
  (b) Removes the IOQ from its logical channel queue if the caller is the *LCH-purge procedure (3)*.
  (c) Chains the SRB/IOSB to the appropriate PIRL entry.
  (d) Calls the *storage manager module (IECVSMGR) free-small-block and free-large-block procedures (2 and 6)* to free the IOQ, and the ERP work area if one exists.
  (e) Issues an IOSGEN RESTART macro to reset the IRT "channel mask" to show that the channel is free. (When the *basic IOS module (IECIOSCN) channel-restart procedure (13)* executes again, it will examine the channel mask and try to start I/O requests on logical channel queues associated with the channel.)
  (f) Exits to the return address in register 14+8.

11. The Functional Recovery Procedure (PURGEFRR)

- Entered by RTM if any of the procedures of the *nonresident purge module (IGC0001F)* took a program check while the local lock was held.
- Sets bits in the SDWA (via the SETRP macro) which direct RTM to write the SDWA in the SYS1.LOGREC data set, release any locks held, and continue with termination processing.
- Exits to RTM.

Note: This procedure puts diagnostic data in the variable area of the SDWA, and in the SDUMP buffer if it's available for use. The diagnostic data is described in the "Diagnostic Aids" chapter under "Output of the Nonresident Purge Module (IGC0001F)."

12. The ESTAE Recovery Procedure (PRGESTAE)

- Entered by RTM if any of the procedures of the *nonresident purge module* took a program check.
- Decreases the count of active purge operations in the IOCPGCT field of the IOS communications table.
- Zeros out the IPIB pointer in the ASCBIOSP field of the ASCB.
- Issues a STATUS macro, allowing processing in the address space that runs under a TCB or SRB to be dispatched.
- Issues a DEQ macro to allow another purge operation in the address space.
- Issues an SDUMP macro, if the SDUMP buffer was acquired by the *functional recovery procedure (11)*, to write the contents of the buffer to the SYS1.DUMP data set.
- Sets bits in the SDWA (via the SETRP macro) that direct RTM to write the SDWA in the SYS1.LOGREC data set and continue termination processing.
- Exits to RTM.

Note: This procedure puts diagnostic data in the variable area of the SDWA. The data is described in the "Diagnostic Aids" chapter under "Output of the Nonresident Purge Module (IGC0001F)."

13. Compress Interface (PRGCOMP0)

- Tests for a quiesce. If there is not a purge quiesce, returns to the *entrance/exit procedure (1).*
- Tests for a minimum number of free blocks (COMPMIN) on IECVSMGR's free queue of 160-byte blocks. If not at least the *minimum number (97),* returns to the *entrance/exit procedure (1)* via register 14.
- Obtains the local lock and issues a SETFRR macro to call *storage manager module (IECVSMGR) get-large-block-procedure (5).*
- Calls the *storage manager module (IECVSMGR) get-large-block procedure (5)* for a 160-byte block to schedule *storage manager module (IECVSMGR) compress procedure (10).*
- Deletes the FRR and frees the local lock.
- Initializes the block to an SRB.
- Schedules *storage manager module (IECVSMGR) compress procedure (10)* via the SRB on the global queue.
- Returns to the *entrance/exit procedure (1).*

Post-Status Module (IECVPST)

1. *The Exit Interface Procedure (IECVPST)*

- Entered by the dispatcher if it finds an SRB that addresses this procedure on an SPL. (Both the *basic IOS module (IECIOSCN) SRB scheduling procedure (7)* and the *IOS IOSB-handling procedure (2)* put such SRBs on an SPL with a SCHEDULE macro.)
- Issues a SETFRR macro that gives RTM the address of the *functional recovery procedure (4).*
- Calls the system's real-to-virtual translation module to get the virtual storage address of the CCW pointed to by the CSW. Puts this address in the IOSCSW field, where the CSW has been saved for the inspection of exits.
- If the IOSPSLL bit is off in the IOSOPT field (meaning the driver wants the local lock to be held when an exit is entered), acquires the local lock.
- Tests the IOSB to find out if it was created by IOS. If so (the IOSIOSB bit will be on), exits to the *IOSB-handling procedure (2).*
- Calls the PCI, NRM, or ABN exit, based on settings in the CSW, or enters an ERP – via the ERP interface (5), exit effector, and ERP loader for nondirect-access ERPs – if the IOSERR bit is on (indicating an ERP is waiting to find out what happened to a retried I/O operation).

  If the NRM or ABN exit was called, does the processing associated with the address returned to.

  Note: If the driver indicated in the IOSOPT field that its termination procedure does not need the local lock, the local lock is released before exiting to the driver's termination procedure.

  These are the possible return addresses and the associated processing:

  (a) *register 14+0:* Tests the IOSEX bit, and if it's on, uses ERP interface (5) to branch to the direct-access ERP or goes to a nondirect-access ERP via the ERP interface (5), exit effectors, and ERP loader. Otherwise, exits to the driver by branching to its termination procedure.

  (b) *register 14+4:* Calls the *storage manager module (6)* to free the ERP work area, if one exists. Returns to the exit by branching to the address returned in register 15. When the exit comes back, control is given to the dispatcher.

  (c) *register 14+8:* Issues a STARTIO macro to retry the I/O operation or to start a new I/O operation.

  (d) *register 14+12:* Goes to DDR via the exit effectors and ERP loader. This return address is used only for a paging I/O request.

  (e) *register 14+16:* Bypasses branch to driver's termination routine. (*Note:* This return address is restricted and is for *EXCP drivers only.*)

2. *The IOS IOSB-Handling Procedure (PSTIOSB)*

- Entered by the *exit interface procedure (1)* if it determines that the IOSB was created by IOS.
- Determines whether I/O processing should be terminated–the case when the IOSCOD field is X'45'. If so, calls the *storage manager module (IECVSMGR) free-large-block procedure (6)* to free the SRB/IOSB and the EWA, then exits to the dispatcher.
- Does the proecssing indexed by the value in the IOSPROC field. these are the possible values and the associated processing:

  (a) *X'00: Enters the exit interface procedure (1)* to handle as a non-IOS-generated IOSB.

  (b) *X'04':* Enters the PCI exit, and reenters it as many times as there are IOS-created IOSBs chained to the original IOS-created IOSB. Calls the *storage manager module (IECVSMGR) free-large-block procedure (6)* to free the IOS-created IOSBs. If it finds the driver-created IOSB at the chain's end, issues a SCHEDULE macro, scheduling the *exit interface procedure (1)* to enter the PCI exit again. Exits to the dispatcher.

  (c) *X'08':* Gets the address of the attention routine from the IOSPGAD field and calls it. When the attention routine returns, these are the possible return addresses and the associated processing: register 14+0 exits to the dispatcher. Register 14+4 reschedules IECVPST to a new address space (indicated in IOSASID) so that the attention routine is reentered.

  (d) *X'0C':* Calls the *storage manager module (IECVSMGR) free-large-block procedure (6)* to free the SRB/IOSB, and the ERP work area if one exists. (Used by the *nonresident purge module (IGC0001F) LCH-purge procedure (3)* to free IOS-created IOSBs.) Exits to the dispatcher.

  (e) *X'10':* Goes to the *DAVV module (IECVDAVV) volume verification procedure (1)* via the exit effectors and ERP loader if the unsolicited device end is not from a page pack. If UCBPGFL is on for the UCB whose address is in IOSUCB, DAVV module (1) is called directly. (The processing done by the ERP loader is described in "Appendix B" under "ERP Service Modules.")

  (f) *X'14':* Goes to the ERP message writer via the exit effectors and ERP loader. (The processing done by the ERP message writer is described in "Appendix B" under "ERP Service Modules.") If intervention is required on a page pack, PSTIOSB passes control to the restartable wait procedure (6).

  (g) *X'20':* Goes to the *unconditional reserve recovery procedure (7).*

3. *The SVC 15 Procedure (IGC015)*

- Entered by nondirect-access ERPs with an SVC 15 instruction and by the direct-access ERP with a branch instruction through ERP interface (PSTEFF, 5).

- Based on the setting of two bits in the IOSFLA field, does one of the following:

  (a) *IOSERR on, IOSEX any setting:* Issues a STARTIO macro to retry the I/O operation, first testing the IOSTSB field to see if a channel error occurred during the original operation. If so, sets bits in the IOSAPMSK field that will prevent the *basic IOS module (IECIOSCN) test-channel procedure (2)* from reselecting the path for the retry operation unless the path is the only one available.

  (b) *IOSERR off, IOSEX on:* Changes the completion code in the IOSCOD field and branches to the ABN exit. The codes and their meanings are as follows:

IOSCOD on entry: on exit	Meaning
7F; 41	ERP can't correct error.
7E; 44	I/O operation was never started; current SRB/IOSB is used to process the error.
74; 51	I/O operation was never started; device is inaccessible from the only operational processor.
Not 7F, 7E, or 74; left as is	See the explanations of IOSCOD codes in the "Diagnostic Aids" chapter under "Informative IOSB Fields."

  When the ABN exit routine returns, does the processing associated with the return address, as described in the *exit interface procedure (1).* One exception: if the register 14+0 return is used, doesn't go to an ERP, regardless of the IOSEX setting.

  (c) *IOSERR off, IOSEX off:* Puts X'7F' in the IOSCOD field, showing a corrected error, and branches to the NRM exit. When the exit routine returns, does the processing associated with the return address, as described in the *exit interface procedure (1).*

4. The Functional Recovery Procedure (PSTFRRTN)

- Entered by RTM if any of the procedures of the *post-status module* took a program check.
- If the caller is RTM, turns off UCB settings that direct control to the *DAVV module*, puts X'45' in the IOSCOD field, and, if the SDUMP buffer is available, issues an SDUMP macro to write the contents of the buffer in the SYS1.DUMP data set.
- Sets bits in the SDWA (via the SETRP macro) that direct RTM to (a) record the contents of the SDWA in the SYS1.LOGREC data set and (b) return control to this procedure to avoid a continuation of termination processing.
- Branches to RTM for retry. When control returns to IECVPST, frees any locks and storage that IOS acquired and exits to the dispatcher.

Note: This procedure puts diagnostic data in the SDUMP buffer and in the variable area of the SDWA. The data is described in the "Diagnostic Aids" chapter under "Output of the Post-Status Module (IECVPST)."

5. *ERP Interface Procedure (PSTEFF)*

- Entered by the *exit interface procedure (1)* to cause entry to an ERP.
- If request is for a non-IBM ERP, goes to the *SVC 15 procedure (3)* to post the driver with an error code.
- If request is for an IBM-supplied ERP, gets ERP work area, if one does not already exist.
- Branches to DASD ERP, or goes to non-DASD ERP via the exit effectors and the ERP loader (IECVERPL).
- Return from a non-DASD ERP is via SVC 15. Processing is described under the *SVC 15 procedure (3).*
- On return from a DASD ERP, processes according to the return address, as follows:

  (a) *register 14+0:* Goes to *SVC 15 procedure (3)* to:

  (1) Do a retry on the same path if a guaranteed device path was specified.

  (2) Do a retry on an alternate path if a guaranteed device path was not specified, provided there is an alternate path. Marks the failing path in the IOSB.

  (b) *register 14+4:* If the driver is program fetch and a data check occurred (IOSCOD = X'71'), goes to the channel end exit belonging to PCI fetch.

  Otherwise, goes to the *SVC 15 procedure (3)* to check the IOSEX flag and process accordingly:

  (1) If the IOSEX flag is on, the error is considered permanent. In this case, sets a permanent error condition and branches to the driver's abnormal end exit.

  (2) If the IOSEX flag is off, the error has been corrected. Sets post code for normal completion (X'7F') and goes to the driver's *channel end exit* via the *exit interface procedure (1).*

  (c) *register 14+8:* Uses the exit effectors and the ERP loader to schedule the ERP message writer, IGE0025C.

  (d) *register 14+12:* Goes to abnormal end exit interface in *SVC 15 procedure (3).*

6. *Restartable Wait Procedure (PSTWAIT)*

- Entered by the *DAVV module (IECVDAVV) error handling procedure (3)* and IOS *IOSB handling procedure (2).*
- Builds message IEA073A to explain why the page data set is unavailable.
- Calls IEEVLDWT to issue message IEA073A to the console; if the message cannot be sent to the console, IEEVLDWT loads restartable wait state code 2F.
- When control returns after the operator has pushed the RESTART key, if DAVV is waiting for a device end interruption, the routine exits to return control to the dispatcher. If DAVV is not waiting, exit is to the *SVC 15 procedure (3)* to retry the I/O.

7. *The Unconditional Reserve Procedure (PSTUR)*

- Entered by the *IOS IOSB-handling procedure (2).*
- If this is the first entry, calls the *unconditional reserve detection module (IECVURDT)* to determine whether the error is permanent. If it is not, branches to free storage, redrive queued I/O, and exit.
- If this is not the first entry or if the error is permanent, calls the *unconditional reserve decision module (IECVDURP)* to communicate with the operator, if necessary. IECVDURP performs unconditional reserve recovery by calling the *unconditional reserve service module (IECVURSV).* There are three possible returns:
  (a) register 14+0 with IOSLOG off – branch to free storage and exit.
  (b) register 14+0 with IOSLOG on – go to the ERP logging routine via the exit effectors and the ERP loader.
  (c) register 14+4 – exit immediately. This routine will be re-entered to complete processing.

Redrive I/O Service Module (IECVRDIO)

1. *The Redrive I/O Service Procedure*

- Entered by the hot I/O recovery module (IECVHREC) and I/O restart module IECVIRST.
- Does the following for each device on the requested channel:
  – Indicates that volume verification is necessary if the device was inactive or was active on the requested path, i.e. channel and channel set.
  – Calls the *path check procedure (2)* to determine path availability.
  – If the box-only function was not requested, calls the *restart I/O procedure (3)* to clean up active I/O operations.
- Calls I/O SLIH to request it to redrive all channels on the next I/O interrupt or I/O request.

2. *The Path Check Procedure*

- Entered by the *redrive I/O service procedure (1).*
- Uses IOSMAP macro to determine online paths.
- If no online paths exist, the device is boxed for hierarchy reasons, and the device address is added to message IEA004I.
- Returns to caller.

3. *The Restart I/O Procedure*

- Entered by the *redrive I/O service procedure (1).*
- If an operation was active on the requested path, calls I/O SLIH, simulating status with interface-control check and channel-control check. This will clean up the UCB and send the active request back to the issuer.
- Returns to caller.

4. *The Functional Recovery Routine*

- Entered from RTM on error during IECVRDIO.
- Prepares diagnostic data.
- Indicates those locks that are held and should be freed.
- Attempts an SDUMP.
- Frees the general work area.
- Returns to RTM and percolates.

Re-reserve Module (IECVRRSV)

1. *The Set Up Procedure*

- Entered from the I/O restart modules IECVRSTI and IECVIRST, the hot I/O recovery module IECVHREC, and the unconditional reserve recovery module IECVURSV.
- Obtains the storage for a work area.

2. *The Do Reserve Procedure*

- If box-only function is not requested, processes each device in the reserve table (RESVTAB) as follows:
  - calls *check reserve procedure (3)* to determine if the device should still be reserved. If not, marks the entry complete.
  - uses IOSMAP to determine all online paths to the device.
  - calls the *special SIO module (IECVESIO)* to perform reserve, unconditional reserve, or unconditional reserve/release on each online path until successful.
  - calls the *show reserve procedure (4)* if the operation was successful.
- Calls the *box device procedure (5)* to handle any unsuccessful operations or to perform the box-only function.
- Frees the work area.
- Returns to caller with an X'00' return code if the requested function was successfully completed. A X'04' return code is returned if the function was not successfully completed.

3. *The Check Reserve Procedure*

- Entered by the *do-reserve procedure (2).*
- Determines if the device should still be reserved to this system.
- If not, gets the UCB lock and resets all reservation indicators.
- Returns to caller.

4. *The Show Reserve Procedure*

- Entered by the *do-reserve procedure (2).*
- Obtains the UCB lock.
- If release was performed, resets all the UCB reservation indicators.
- Otherwise, indicates on which path the device was reserved, and resets any pending indicators.
- Returns to caller.

5. *The Box Devices Procedure*

- Entered by the *do-reserve procedure (2)* or the *FRR procedure (6).*
- Processes each device in the reserve table if boxing is allowed.
  - If not marked complete or boxed, gets the UCB lock, boxes the device, and resets the UCB reservation indicators.
  - If the caller specified that messages be issued, this procedure places the device address in message IEA026I and issues it.
- Returns to caller.

6. *The Functional Recovery Routine*

- Entered from RTM on an error in IECVRRSV.
- Obtains diagnostic data.
- If all devices were not fully processed, calls the *box devices procedure (5).*
- If this is a recursive entry and boxing could not be completed, issues message IEA151W and loads wait state X'04E'.
- Frees work area.
- Returns to RTM requesting percolation.

Resident Halt - I/O Module (IECIHIO)

1. The Main Procedure (IECIHIO)

- Entered by the *nonresident purge module (IGC0001F) basic purge procedure (10)* and the *nonresident halt-I/O module (IGC0003C) main halt procedure (1),* with the UCB lock of the associated device held, to halt a channel program currently active on the device.
- Issues a SETFRR macro that gives RTM the address of the *functional recovery procedure (5).*
- Calls the *shoulder-tap procedure (2)* if not entered on the processor that started the I/O operation.
- Issues an HDV instruction to halt the I/O operation in progress. Calls the *channel-logout procedure (3)* if the HDV instruction is not accepted because of a pending channel logout (the condition code is set to 1, and the logout-pending bit and channel-control-check bit are "on" in the CSW). Calls the *channel error procedure (4)* if the HDV instruction is not accepted because of a channel error (the condition code is set to 1, and a channel error is indicated in the CSW).
- Issues a SETFRR macro that deletes the address of the *functional recovery procedure (5)* from RTM's stack of such procedures.
- Puts one of these return codes in register 15:

  (a) *X'00':* The I/O operation was halted.

  (b) *X'08':* The other processor was asked to halt the I/O operation but could not.
- Exits to the return address in register 14.

2. The Shoulder-Tap Procedure (HIOIPCI)

- Entered by the *main procedure (1)* if that procedure was not entered in the processor that started the I/O operation.
- Issues an RISGNL macro, which gives control to a system routine, the *RISGNL routine,* that shoulder taps the other processor with a SIGP instruction. This procedure then continues in the other processor and branches to the *main procedure (1)* at a point (HIOCPUCK) just before the HDV instruction is issued. When the *main procedure (1)* exits in the shoulder-tapped processor, it does not return to the module's caller, but to the RISGNL routine. The *RISGNL routine* returns to this procedure with a return code that tells whether the shoulder-tapped processor was able to halt the I/O operation.
- Reissues the RISGNL macro up to 512 times if the *RISGNL routine* indicates (with a return code of 8) that the shoulder-tapped processor was unable to halt the I/O operation.
- Issues an ABEND macro with an X'C0D' completion code if an error occurred while processing in the shoulder-tapped processor.
- Exits to the *main procedure (1),* passing on the RISGNL routine's return code in register 15.

3. The Channel-Logout Procedure (HIOLOP)

- Entered by the *main procedure (1)* if the HDV instruction results in a condition code of 1 and a logout-pending indication in the CSW.
- Alters the I/O new PSW to point to the part of this procedure (IECHK5) that will process the logout interruption (the interruption generated by the channel when it is allowed to present its logout information to the processor).
- Loops on an instruction (STOSM) that enables the "logout-pending" channel to interrupt processing and present its information.
- Gets control at IECHK5 when the interruption occurs, prevents the "logout-pending" channel from presenting other interruptions, and resets the I/O new PSW.
- Releases the UCB lock.
- Calls the *basic module (IECIOSCN) interruption-handling procedure (9)* to continue processing the interruption. Calls the CRH module (IECVCINT) if CRH support is active.
- Reacquires the UCB lock.
- Exits to the *main procedure (1)* at HIOHIO; or to the *RPSGNL routine* with a return code of X'08' in register 15, without executing any of the previous steps, if entered in a shoulder-tapped processor.

4. The Channel Error Procedure (HIOCCH)

- Entered by the *main procedure (1)* if a channel error, excluding a logout-pending indication, results from the HDV instruction.
- Calls CCH to process the channel error.
- Exits to HIORTY in the *main procedure (1)* to retry the HDV instruction if CCH returns with a return code of X'00' in register 15. (The HDV instruction is only retried once.) Otherwise, exits to the caller of this module with a return code of X'04'; or to the *RISGNL routine,* with a return code of X'08', if the HDV instruction was issued from the shoulder-tapped processor.

5. The Functional Recovery Procedure (HIOFRR)

- Entered by RTM if:
  (a) Any of the procedures of the *resident halt-I/O module* took a program check.
  (b) The *shoulder-tap procedure (2)* issued an ABEND macro with a X'C0D' code.
- Acquires the UCB lock to protect the processing environment of this module's caller (just in case the error occurred between the time the *channel-logout procedure (3)* released and reacquired the UCB lock.)
- If the *channel-logout procedure (3)* was entered, restores the I/O new PSW that it altered and ensures that no channel can present an interruption.
- If the SDUMP buffer is available, issues an SDUMP macro to write the contents of the buffer in the SYS1.DUMP data set.

- If the error didn't occur in a shoulder-tapped processor, issues a SETRP macro, directing RTM to record the contents of the SDWA in the SYS1.LOGREC data set and do one of the following:

  (a) Continue termination processing if the error didn't occur in the *basic IOS module (9)* or in CCH.

  (b) Retry the HDV instruction if the error occurred in the *basic IOS module (IECIOSCN) interruption-handling procedure (9)* or in CCH.

- If the error occurred in a shoulder-tapped processor, puts the ABEND code and a return code of X'0C' in the FRR work area and branches to the return address in register 14. (The *shoulder-tap procedure (2)* will eventually get control in the calling processor, find the return code, and issue an ABEND macro with a X'C0D' code, causing this procedure to again get control.) When control returns in the calling processor, overlays the X'C0D' code in the SDWA with the original ABEND code and sets bits in the SDWA (via the SETRP macro) that direct RTM to record the contents of the SDWA in the SYS1.LOGREC data set and continue termination processing.

- Exits to the return address in register 14.

Note: This procedure puts diagnostic data in the SDUMP buffer and in the variable area of the SDWA. The data is described in the "Diagnostic Aids" chapter under "Output of the Resident Halt-I/O Module (IECIHIO)."

Resident Purge Module (IECVPURG)

1. The Decrement-Count Procedure (IECVQCNT)

- Entered by a driver's termination procedure if, during a quiesce operation, IOS finished processing an I/O request submitted by that driver. (When an I/O request has been completely processed, the *post-status module (IECVPST) exit interface and SVC 15 procedures (1 and 3)* branches to the driver's termination procedure, or gives it control with a SCHEDULE macro if the request was processed by a nondirect-access ERP.)

  Note: Also entered by the *nonresident purge module (IGC0001F) entrance/exit procedure (1)* on behalf of a problem-state driver that can't enter directly.

- Decreases the count of partially-processed I/O requests in the IPIBCNT field of the IPIB (IOS purge interface block). If the count becomes zero, acquires the local lock, if not already held, and calls the *system's post routine* to post the ECB created by the *nonresident purge module (IGC0001F) entrance/exit procedure (1),* bringing that module out of a wait state.
- If it acquired the local lock, releases it.
- Exits to the return address in register 14.

2. The SIRB Clean-Up Procedure (IECVPRCU)

- Entered by the dispatcher, if an applicable SRB/IOSB was dequeued from an SIRB, after the *nonresident purge module (IGC0001F) entrance/exit procedure (1)* restarts TCB processing with a STATUS macro. (The dispatcher loads the resume PSW in the SIRB, which was modified to point to this procedure by the *nonresident purge module (IGC0001F) SIRB-purge procedure (2).*)
- Acquires the local lock and calls the *storage manager module (IECVSMGR) free-large-block procedure (6)* to free the ERP work area addressed in the SIRB.
- Exits with an SVC 3 instruction.

3. The Chain-SRB Procedure (IECVPRDQ)

- Entered by the system's SPL purge module, which was entered by the *nonresident purge module (IGC0001F) SPL-purge procedure (6)* to remove an SRB from the SPL.

  Issues an ABEND with an X'C0D' code if IPIB pointer, ASCBIOSP, is zero. (The 'C0D' code means that a QUIESCE macro was issued without a work area.)
- Puts the SRB/IOSB pointed to by register 1 on the queue of SRB/IOSBs chained from the IPIB (IOS purge interface block).
- Exits to the return address in register 14.

Restore Module (IGC0001G)

1. The Restore Procedure (IGC017)

- Entered with an SVC 17 instruction by a user or system program running under a TCB. (A purge operation, not necessarily initiated by the same program, must precede entry to this module.)
- Issues a GETMAIN macro for a save area to be used by the called drivers.
- Calls in turn the restore procedures of each driver, passing to each in register 1, the address of the field, if nonzero, in the PIRL (PIRRSTR) that points to data saved during the purge operation. (The PIRL purged I/O restore list is created during a quiesce operation and is initialized with pointers to the interrupted work of each driver.)
- Puts one of these return codes in register 15:
  (a) *X'00'*: All the drivers were able to restore their I/O requests.
  (b) *X'04'*: One or more drivers returned a code that indicated an error in restoring I/O requests.
- Frees the PIRL and the save area via a FREEMAIN macro.
- Exits to the return address in register 14.

The SIO Module for DASD Devices (IECVXDAS)

1. *DASD SIO Procedure*

Note: Descriptions of the SIO module for the 2305, 2314 and 3330V devices are presented separately on the following pages. All other DASDs are handled by this procedure.

- Entered by the *basic IOS module (IECIOSCN) test channel procedure (2).*
- If the UCBUDE (unsolicited device end) bit is on, issues the IOSCKVOL macro for volume verification.
- Builds a channel-program prefix containing seek, set-file-mask, and TIC CCWs. If tests justify it, adds a reserve or release CCW to the beginning of the prefix and issues the IOSSCP macro specifying that (a) the driver's channel program, with prefix, starts with a SIO instruction and (b) control returns here.
- If the device is not shared, or if it is shared and reserved, issues the IOSSCP macro specifying that (a) the driver's channel program, with prefix, starts with a SIOF instruction and (b) control is given to the *basic IOS module (IECIOSCN) post-SIO procedure (4).*
- If the device is shared and not reserved, issues a IOSSCP macro specifying that (a) the driver's channel program, with prefix, is started with a SIO instruction and (b) control is given to the *basic IOS module (IECIOSCN) post-SIO procedure (4).*
- Exits to the *basic IOS module (IECIOSCN) post-SIO procedure (4).*

The SIO Module for the 2305 Device (IECVXDRS)

1. 2305 SIO Procedure

- Entered by the *basic IOS module (IECIOSCN) test channel procedure (2).*
- If tests of bits in the UCB justify it, builds a stand-alone reserve or release CCW. Issues the IOSSCP macro specifying that (a) the CCW executes with a SIO instruction and (b) control returns here.
- Builds a channel-program prefix containing seek, set-file-mask, and TIC CCWs.
- Selects an exposure (a device number – a 2305 can be addressed by eight device numbers) to use in starting the I/O operation, unless the driver indicated that it wants to use a specific exposure.
- If the device is not shared, or if it is shared and reserved, issues IOSSCP macro specifying that (a) the driver's channel program, with prefix, starts with a SIOF instruction and (b) control returns here.
- If the device is shared and not reserved, issues the IOSSCP macro specifying that (a) the driver's channel program, with prefix, starts with a SIO instruction and (b) control returns here.
- Exits to the *basic IOS module (IECIOSCN) post-SIO procedure (4).*

Note: IOS selects the base exposure (the lowest device number of the eight possible exposures) for stand-alone reserve and CCWs only. The other seven exposures are selected for normal data transfer operations.

The SIO Module for the 2314 Device (IECVXSKS)

1. 2314 SIO Procedure

- Entered by the *basic IOS module (IECIOSCN) test channel procedure (2).*
- If the UCBUDE (unsolicited device end) bit is on, issues the IOSCKVOL macro for volume verification.
- Builds a stand-alone seek CCW. Issues the IOSSCP macro specifying that (a) the channel program starts with a SIO instruction and (b) control returns here. When control returns, exits to the *basic IOS module (IECIOSCN) post-SIO procedure (4)* if the condition code is nonzero; otherwise, does the following:
  (a) Turns on the UCBSAP bit, indicating a seek operation is in progress.
  (b) Issues a TIO instruction up to 256 times, testing for a condition code of 1, which indicates that status information is stored in the CSW. Starts the driver's channel program if, during the execution of this loop, status information is stored that contains a channel-end and device-end indication. Otherwise, returns to the *basic IOS module (IECIOSCN) test channel procedure (2)* with a request to put the IOQ on a logical channel queue.
- Builds a channel-program prefix containing seek set-file-mask and TIC CCWs. If tests justify it, adds a reserve or release CCW to the prefix and issues the IOSSCP macro specifying that (a) the driver's channel program, with prefix, starts with a SIO instruction and (b) control returns here.
- If the device is not shared, or if it is shared and reserved, issues the IOSSCP macro specifying that (a) the driver's channel program, with prefix, starts with a SIOF instruction and (b) control is given to the *basic IOS module (IECIOSCN) post-SIO procedure (4).*
- If the device is shared and not reserved, issues a IOSSCP macro specifying that (a) the driver's channel program, with prefix, is started with a SIO instruction and (b) control is given to the *basic IOS module (IECIOSCN) post-SIO procedure (4).*
- Exits to the *basic IOS module (IECIOSCN) post-SIO procedure (4).*

The SIO Module for the 3330V Device (IECVXVRS)

1. 3330V SIO Procedure

- Entered by the *basic IOS module (IECIOSCN) test-channel procedure (2).*
- If the request has been held because of a pending cylinder fault resolution, and no cylinder fault was resolved for the device, the IECVXVRS exits to the *basic IOS module (IECIOSCN) test-channel procedure (2)* at offset X'04' to cause the IOQ to remain queued.
- If the request has been held because of a pending cylinder fault resolution and a cylinder fault was resolved for the device, this request and all similar requests are released. This request then processes normally.
- Builds a channel-program prefix containing seek, TIC, NOOP, set-file-mask, and TIC CCWs. If tests justify it, adds a reserve or release CCW to the beginning of the prefix and issues the IOSSCP macro specifying that (a) the driver's channel program, with prefix, starts with a SIO instruction and (b) control returns here.
- If the device is not shared, or if it is shared and reserved, issues the IOSSCP macro specifying that (a) the driver's channel program, with prefix, starts with a SIOF instruction and (b) control is given to *basic IOS module (IECIOSCN) post-SIO procedure (4).*
- If the device is shared and not reserved, issues a IOSSCP macro specifying that (a) the driver's channel program, with prefix, is started with a SIO instruction and (b) control is given to the *basic IOS module (IECIOSCN) post-SIO procedure (4).*
- Exits to the *basic IOS module (IECIOSCN) post-SIO procedure (4).*

The SIO Module for the 2400 Tape Device (IECVXT2S)

1. 2400 SIO Procedure

- Entered by the *basic IOS module (IECIOSCN) test-channel procedure (2).*
- If the control unit can handle simultaneous read and write operations to two tape devices, builds a stand-alone set-mode CCW. Issues the IOSSCP macro specifying that (a) the set-mode starts with a SIO instruction and (b) control returns here.
- Builds a channel program prefix consisting of a set-mode CCW and a TIC. If the I/O request is one that the tape ERP wants retried, adds error correction CCWs to the prefix as the ERP directs.
- Issues the IOSSCP macro specifying that (a) the driver's channel program, with prefix, is started with a SIOF instruction and (b) after the SIOF instruction, control passes to the *basic IOS module (IECIOSCN) post-SIO procedure (4).*

The SIO Module for the 3400 Tape Device (IECVXT3S)

1. 3400 SIO Procedure

- Entered by the *basic IOS module (IECIOSCN) test-channel procedure (2).*
- If the UCBUDE (unsolicited device end) bit is on, issues the IOSCKVOL macro for volume verification.
- Builds a channel program prefix consisting of a set-mode CCW and a TIC. If the I/O request is one that the tape ERP wants retried, adds error correction CCWs to the prefix as the ERP directs.
- Issues the IOSSCP macro specifying that (a) the driver's channel program, with prefix, is started with a SIOF instruction and (b) after the SIOF instruction control passes to the *basic IOS module (IECIOSCN) post-SIO procedure (4).*

The SIO Module for Unit Record Devices (IECVXURS)

1. Unit Record SIO Procedure

- Used for unit record, graphic, TP, 3851, 3838, 3211 and 3800 devices.
- Entered by the *basic IOS module (IECIOSCN) test-channel procedure (2).*
- Issues the IOSSCP macro specifying that (a) the driver's channel program is started with a SIOF instruction and (b) after the SIOF instruction, control passes to the *basic IOS module post-SIO procedure (4).*

Special SIO Module (IECVESIO)

1. The Entrance/Exit Procedure

- Entered by a system program which requires an I/O operation to be performed "synchronously," without using normal system services.
- If the requested channel set is connected, calls the *SIO procedure (2).* Otherwise, if CRH or CHS is active, obtains access to the required channel, then calls the *SIO procedure (2).*
- If the required channel is connected to the other processor, and the caller allows, IECVESIO issues an RISGNL macro to obtain control on the other processor in the *SIGP entry procedure (3).*
- Returns to the caller with I/O complete, if possible, and a return code.

  The return code in register 15 will be one of the following:

  (a) X'00': the channel program completed successfully.

  (b) X'04': a unit check occurred; no sense data was returned.

  (c) X'08': a unit check occurred; sense data was returned.

  (d) X'0C': channel error occurred on SIO instruction.

  (e) X'10': condition code of 3 occurred on a SIO instruction.

  (f) X'14': condition code of 0 occurred on a TIO instruction, while waiting for device end.

  (g) X'18': maximum SIO count was exceeded.

  (h) X'1C': maximum TIO count was exceeded.

  (i) X'20': channel program completed with a unit exception, PCI, incorrect length, channel program check, protection check, or chaining check.

  (j) X'24': requested channel set not available.

  (k) X'28': enablement required.

  (l) X'2C': maximum enablement count exceeded.

  (m) X'30': channel error on TIO.

  (n) X'34': condition code of 3 occurred on TIO instruction.

  (o) X'38': attention present in CSW.

  (p) X'3C': other error.

2. *The SIO Procedure (SIORTN)*

- If requested by the caller, enters a TIO loop waiting for an attention from the required device.
- Issues the SIO instruction to the required device with the caller's channel program. Then enters a TIO loop waiting for the device end interruption.
- If a unit check occurs, a sense is issued. If the device is busy and the caller allows, the processor is enabled for all I/O interruptions which are then processed by the IOS SLIH. If the caller does not allow the processor to be enabled, the I/O request is not performed.

3. *The SIGP Entry Procedure (IECVESIG)*

- Receives control on the other processor when it is connected to the requested channel set. Then IECVESIG calls the *SIO procedure (2)* and control returns to IPC.

4. *The Functional Recovery Routine (ESIOFRR)*

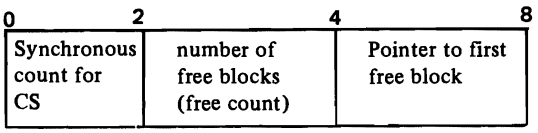
- RTM enters the FRR routine when an error is encountered in IECVESIO.
- Puts the FRR parameters into the variable field of the SDWA.
- Returns to RTM.

Storage Manager Module (IECVSMGR)

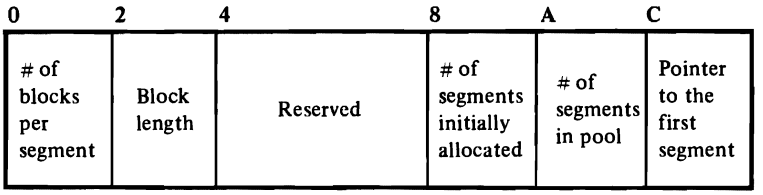
1. The Get-Small-Block Procedure (GETBLK0)

- Entered by procedures of the *basic IOS module (IECIOSCN)* that want a 12-byte block for use as an IOQ.
- Checks the 12-byte block free queue for a pointer to a free block. The free queue header has the following format:

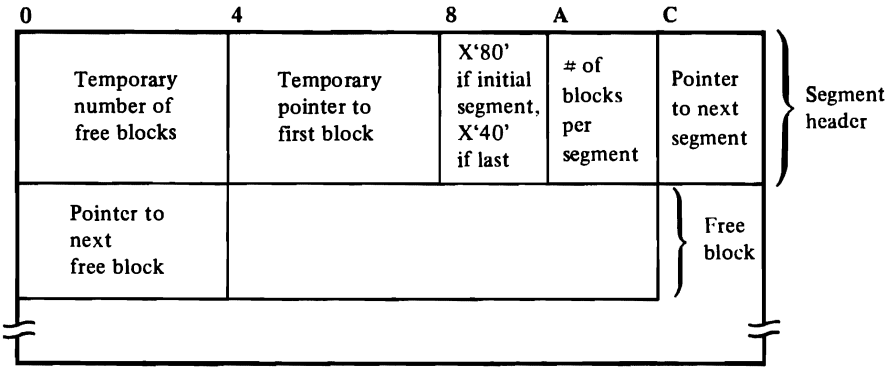
Free Queue Header:



Pool Header:



Pool Segment:



- If there are no free blocks, acquires the global SALLOC (space allocation) lock, issues a GETMAIN for a 2,048-byte segment, formats the new segment, and uses compare and swap (CS) logic to add the new chain of blocks to the free queue. (The new segment is also added to the chain of segments in a last-in, first-out manner).
- To synchronize the queue manipulation, increases the "synchronous count", decreases the free count, uses CS (compare and swap) logic to remove the first free block, and puts the address of the dequeued block in register 11.
- Exits to the return address in register 14.

2. *The Free-Small-Block Procedure (FRBLK0)*

- Entered by procedures of the *basic IOS module (IECIOSCN)* and by the *non-resident purge module (IGC0001F) basic purge procedure (10)* to return an IOQ to the "small block" pool. The IOQ address is passed in register 1.

- Tests the IOQ for the IOQALOC bit (turned on by *basic IOS module (IECIOSCN) channel scheduler procedure (1), basic IOS module (IECIOSCN) the DIE interface procedure (11), basic IOS module (IECIOSCN) the PCI DIE interface procedure (12),* or *basic IOS module (IECIOSCN) the sense procedure (14)* when the 12-byte block is acquired). If the bit is not on, issues a 'C0D' abend code because the block is invalid or is about to be freed twice. The abend causes *the functional recovery procedure (11)* to check the queues, restructure if necessary, and return to the caller of *the free-small-block procedure (2)* without putting the block on the free queue.

- If the block is valid, uses CS logic to increase the free count and the synchronous count and put the block back on the free queue in a push-down (last-in, first-out) manner.

- Exits to the return address in register 14.

3. *The Get-Medium-Block Procedure (GETBLK4)*

- Entered by the EXCP processor to get a 40-byte block for use as an RQE (request queue element).

- Looks at the 40-byte block free queue header for a pointer to a free block. The header has the same format as the 12-byte block free queue header.

- If no blocks are on the free queue:

  a) Acquires the global SALLOC lock.

  b) Issues a GETMAIN for 4K of subpool 245.

  c) Formats the page into 40-byte blocks.

  d) Updates the pool header of 4K pages and chains the new set of 40-byte blocks to the free queue.

  e) Frees the global SALLOC lock.

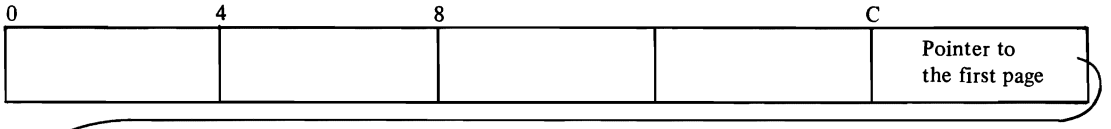
- If blocks are on the free queue, dequeues one block using CS logic. The CS logic increases the synchronous count and decreases the free count.

- Tests the block prefix to ensure that the block is marked free.

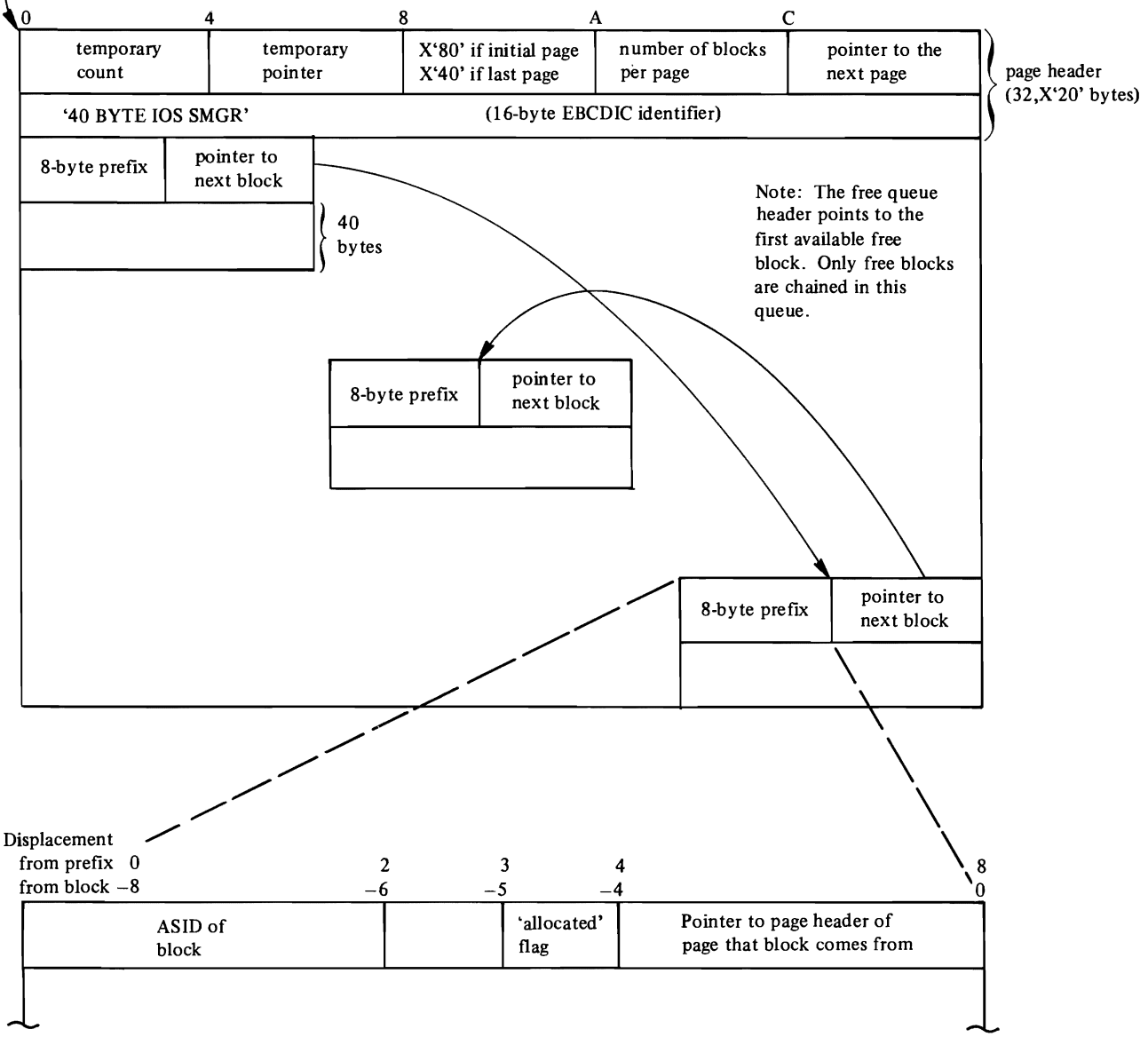
  a) If not free, issues a 'C0D' abend to cause *the storage manager module (IECVSMGR) functional recovery procedure (11)* to record the error. The FRR issues an SDUMP if the SDUMP buffer is available, validity checks the other storage manager queues, and zeros the 40-byte block free queue. A retry causes a GETMAIN to be issued for a new 4K page.

  b) If a free or valid block, puts the ASID (address-space identifier) in the prefix of the dequeued block, turns on the 'allocated' flag in the prefix, and passes the address of the block in register 11.

Pool Header:



Page of 40-byte blocks:



- If more than one block is needed, the previous steps are repeated for each block, and the blocks are chained together.
- Exits to the return address in register 14.

4. *The Free-Medium-Block Procedure (FRBLK4)*

- Entered by the EXCP processor to return 40-byte blocks to the "medium block" pool. Also entered by the *purge-free procedure (8)* to return a block belonging to a terminating job. Register 1 contains the size and address of a single block to be returned or the size and address of the first block in a chain of blocks to be returned.
- Checks the block prefix to ensure that it is marked "allocated."

  a) If not allocated, issues a 'C0D' abend to cause *the storage manager module (IECVSMGR) functional recovery procedure (11)* to record the error. This functional recovery procedure receives control from the EXCP processor functional recovery procedure in module, IECVEXPR. The FRR issues an SDUMP if the SDUMP buffer is available, validity checks the other storage manager queues, and retries by returning to the caller of *free-medium-block procedure (4)* without placing the invalid blocks on the free queue.

  b) If allocated and valid, zeros the address-space identifier in the block's prefix, sets the 'allocated' flag to indicate free, increases the free count and the synchronous count, and uses CS logic to put the block on the 40-byte block free queue in a last-in, first-out manner.

- Exits to the return address in register 14.

5. *The Get-Large-Block Procedure (GETBLK)*

- Entered by the *basic IOS Module (IECIOSCN),* the *post-status module (IECVPST),* the *I/O-restart module (IECVRSTI),* the *nonresident purge module (IGC0001F),* the direct-access ERP (IECVDERP), and a module (IECVIOPM) that's used by the VARY processor to identify online paths to a device. Each enters to get a 160-byte block for use as an SRB/IOSB or as an ERP work area. Also entered by the EXCP processor to get a 160-byte block for use as an SRB/IOSB, a TCCW, BEB, FIX list, or IDAL. Register 11 specifies the number of blocks needed and the ASID (address-space identifier) of the address space that will acquire them.

- Looks at the 160-byte block free queue header for a pointer to a free block. If no free blocks are available:

  a) Acquires the IOSYNCH lock conditionally to serialize against the *compress routine (10)* and the *purge-free procedure (8).*

  b) If the IOSYNCH lock is already held, a compressing of the subpools (by the *compress interface (13)*) of the storage manager's free queue may be in progress – again checks to see if any blocks are on the free queue – if not, repeats (a).

  c) Once the IOSYNCH lock is acquired, calls the *get-storage procedure (7)* to get a new 4K page of storage formatted into 160-byte blocks.

  d) Takes the new chain of 160-byte blocks and uses CS logic to update the free count and synchronous count and put the new blocks on the free queue.

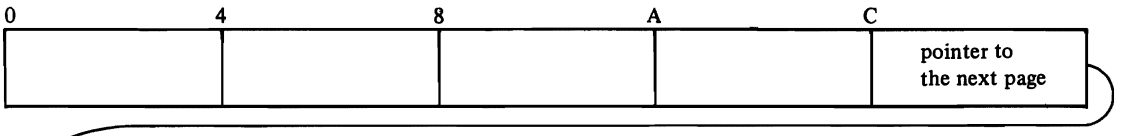
  e) Releases the IOSYNCH lock.

- When blocks are available:

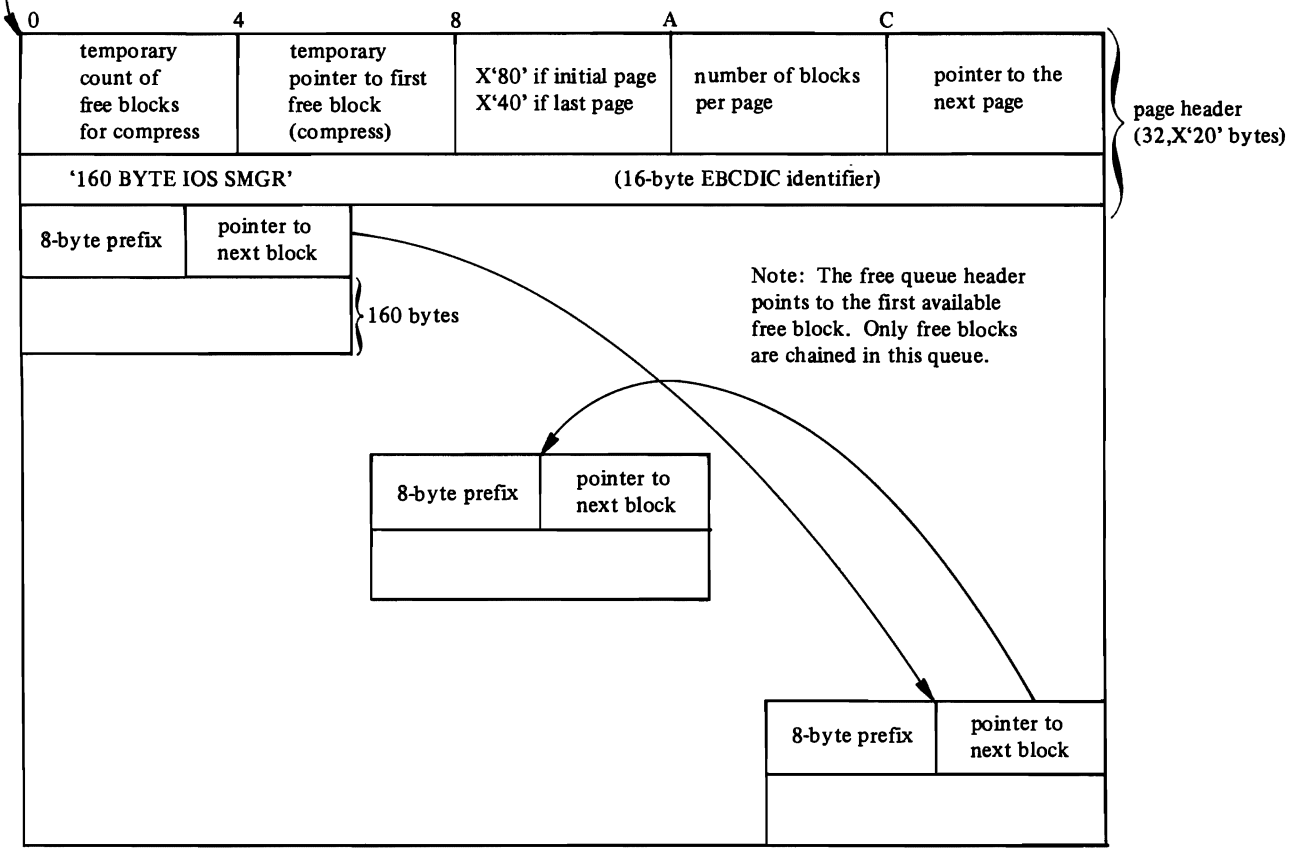
  a) Uses CS logic to dequeue an available block and update the free count and synchronous count. Checks the prefix of the block to ensure that it is marked "free". If not, issues a 'C0D' abend which causes the FRR to record the error and zero the free queue pointer. (Retry proceeds as if no free blocks were available upon entry.)

  b) Puts the ASID it received into the prefix of the dequeued block and marks the prefix "allocated". Puts the address of the block in register 11.

Pool header:



Page of 160-byte blocks:



Free queue header points to the chain of all free blocks (without regard to the page the block comes from). This page structure used mainly by *compress* (10) and *purge free* (8).

- If more than one block is needed, repeats the previous steps for each block and chains the blocks together.
- Exits to the return address in register 14.

6. *The Free-Large-Block Procedure (FREEBLK)*

- Entered by the modules, listed under the *get-large-block procedure (5)*, that want to return 160-byte blocks to the "large block" pool. Also entered by the *purge-free procedure (8)* to return a block belonging to a terminating job. Register 1 contains the size and address of a single block to be returned or the size and address of the first block in a chain of blocks to be returned.
- Issues a SETFRR macro that gives RTM the address of the *functional recovery procedure (11).*
- Checks the block to ensure that it is valid and marked allocated.
  a) If not allocated, issues a 'C0D' abend to cause *the storage manager module (IECVSMGR) functional recovery procedure (11)* to record the error. The FRR validity checks the storage manager's queues and retries by returning to the caller of *free-large-block procedure (6)* without placing the invalid blocks on the free queue.
  b) If a valid and allocated block, zeros the ASID in the prefix and marks the block "free", uses CS logic to update the free count and synchronous count, and puts the block or chain of blocks back on the free queue.
- Exits to the return address in register 14.

7. *The Get-Storage Procedure (GETCORE)*

- Entered by the *pool initialization procedure (9)* at IPL time to obtain a 2048-byte 'segment' of initial small blocks, and 4096-byte pages for 40-byte blocks and 160-byte blocks; and by the *get-large-block procedure (5)* to obtain additional 4K pages of 160-byte blocks as needed.
- Acquires the SALLOC lock and issues a GETMAIN to get either a 4096-byte page, or a 2048-byte 'segment' of small blocks. Then frees the SALLOC lock.
- Formats the segment by initializing its header and chaining the blocks within the segment together.
- Updates the pool header to show the addition of a segment to the pool.
- Exits to the return address in register 9.

8. *The Purge-Free Procedure (PRGFREE)*

- If an address space is terminating, entered by the *nonresident purge module (IGC0001F) entrance/exit procedure (1),* to return all the 40-byte and 160-byte blocks that this module previously allocated to the address space.
- Searches all the blocks in the medium-block pool for each block belonging to the address space of the terminating job, then calls the *free medium-block procedure (4).*
- Acquires the IOSYNCH lock. Searches all the blocks in the large-block pool. For each block matching the ASID of the terminating job, calls the *free-large block procedure (6).* Frees the IOSYNCH lock to open a "lock window," gets the lock again and starts the scan from the beginning of the large-block pool.

- Exits to the return address in register 14 after the search is completed.

9. *The Pool Initialization Procedure (IECVCPRM)*

- Entered by module IEAVNIP0 when the nucleus is initialized to ensure that storage for the "small block," "medium block," and "large block" pools is obtained and formatted.
- Calls the *get-storage procedure (7)*, which gets storage for the pools and formats the headers and segments.
- Exits to the return address in register 14.

10. *Compress Procedure (IECVSCOM)*

A separate entry point in IECVSMGR. Scheduled as an SRB by *non-resident purge module* (1) when purge detects a large number (one greater than COMPMIN) of 160-byte blocks on the free queue. This routine attempts to free the 4K pages of 160-byte blocks no longer being used. (Note: There is also a branch entry point in IECVBSCOM to this routine.)

- If scheduled as an SRB, returns the SRB block to the free queue of 160-byte blocks.
- Checks for a minimum number of free blocks (CMIN160) and a minimum number of total pages (MINPGES) in the large-block pool. Only proceeds if both minimums are satisfied.
- If another compress is in progress, exits to caller or dispatcher (IEAVEDS0). If not, turns on 'compress-in-progress' word flag.
- Issues a SETFRR and acquires the IOSYNCH lock to serialize against *the get-large-block procedure* (5) and *the purge-free procedure* (8).
- Dequeues the entire free queue of 160-byte blocks and uses the pointer in each block prefix to sort the block back to the page it came from. If the block came from an initial page (obtained at NIP time), it is put back on the free queue. If not from a NIP page, the block is chained temporarily to a header in the page that the block did come from. When all free blocks are thus sorted, those free blocks from pages that have allocated blocks still outstanding are put back on the free queue.
- The IOSYNCH lock is freed. If any pages contain only free blocks, acquires the global SALLOC lock and frees the 4K pages that contain only free blocks. The SALLOC lock is freed.
- Turns off the 'compress-in-progress' indicator and

  a) If entered in SRB mode through IECVSCOM, returns to the dispatcher (IEAVEDS0).

  b) If branch entered, sets register 15 to four if no pages are freed and to zero if at least one page is freed. Returns to the caller via register 14.

11. The Functional Recovery Procedure (IECVSMFR)

- Entered by RTM If any of the procedures of the *storage manager module* took a program check.
- If the SDUMP buffer is available, issues an SDUMP macro to write the contents of the buffer in the SYS1.DUMP data set.
- As a back-up measure, issues a SETFRR macro that again gives RTM the address of this procedure.
- Releases the SALLOC lock if it is held and acquires the IOSYNCH lock if it is not already held.
- Looks for invalid data in the pool header that was being processed when the error occurred. If it finds invalid data, calls the *get-storage procedure (7)* to get a new, formatted pool.
- Calls the system's *queue verification routine (IEAVEQV0)* to examine the 2048- or 4096-byte 'segments' in the pool that was being processed when the error occurred.
- If the error occurred during a get-small, -medium, or -large block request, the appropriate free queue is zeroed because the whole free queue is in doubt.
- If the error occurred during a free-small, -medium, or -large block request, the invalid blocks are discarded, that is, not returned to the free queue.
- Sets bits in the SDWA (via the SETRP macro) that direct RTM to:
  (a) Record the contents of the SDWA in the SYS1.LOGREC data set.
  (b) Continue termination processing if the error occurred during ACR processing, following a restart of the system by the operator, or as a result of invalid page tables.
  c) For a get-small, -medium, or -large block request, retry is to the point where the get routine checks for a zero free queue.
  d) Retry for a free-small, -medium, or -large block request is to restore registers, delete the FRR if any, and return to the caller of the free routine without putting the invalid blocks on the free queue.
- Exits to RTM.

Note: This procedure puts diagnostic data in the SDUMP buffer; the data is described in the "Diagnostic Aids" chapter under "Output of the Storage Manager Module (IECVSMGR)." Instead of writing in the variable area of the SDWA, it passes the area to the queue verification routine for use as a QVOD (queue verification output data area). The format and contents of the QVOD are described in the microfiche document *OS/VS2 Data Areas*, SYB8-0606.

12. The Block Verification Procedure (SMGFREVR)

- Entered by the system's *queue verification routine* (IEAVEQV0) to examine a block allocated from the "small block," "medium block," or "large block" pool.

- Tries to address the block. If it can't, exits to the return address in register 14 with a return code of X'08' in register 15.
- If a machine check occurred, finds out whether the block resides in defective storage. If so, exits to the return address in register 14 with a return code of X'08' in register 15.
- If the block is an IOQ, finds out whether the block resides in the "small block" pool. Exits to the return address in register 14 with a return code of X'00' in register 15 if the block resides in the pool; with a return code of X'08' if it doesn't.
- Validates the segment and chain that the block belongs to by checking for invalid data in the "header" field of the segment.
- If no errors are found, puts a return code of X'00' in register 15. Otherwise, uses a return code of X'04'. Exits to the return address in register 14.

The Unconditional Reserve Decision Module (IECVDURP)

1. The Main Procedure

- Entered by the *post-status module (IECVPST) unconditional reserve procedure (7).*
- If this is the first entry to the module and if the ownership of the device is established, calls the *unconditional reserve service module (IECVURSV)* to attempt path recovery.
- If this is the first entry to the module and if the ownership of the device is in question, schedules the asynchronous exit routine to issue message IEA427A. This message presents the operator with the recovery options.
- If the error was on a paging device, calls the restartable wait state service routine (IEEVDLWT) to issue message IEA427A or to load restartable wait state X'06F'.
- If this is not the first entry (the response from the operator has been received), IECVDURP checks the response to determine what options are passed to the unconditional reserve service module.
  - If the operator response was YES, recovery is requested, IECVDURP calls the *unconditional reserve service module* to issue the unconditional reserve instruction.
  - If the operator response was NO, no recovery requested, IECVDURP calls the *unconditional reserve service module* with indication to force the device offline because volume integrity may be lost.
  - If the operator response was NOOP, IECVDURP calls the *unconditional reserve service module* with indication to redrive I/O without any other processing.
- If unconditional reserve recovery is successful, IECVDURP issues message IEA428I.
- If unconditional reserve recovery is not successful, IECVDURP issues message IEA429I.
- Returns to caller.

2. The Device Validation Routine (IECVDVAL)

- Entered by the *basic IOS module (IECIOSCN) unconditional reserve scheduling procedure (17).*
- Verifies that the device on which a path failure occurred supports the unconditional reserve DASD command.
- Returns to caller at register 14+0 if unconditional reserve supported; at register 14+4 if unconditional reserve not supported.

The Unconditional Reserve Detection Module (IECVURDT)

1. The Main Procedure (IECVURDT)

- Entered by the *post-status module (IECVPST) unconditional reserve procedure (7).*
- Issues the TCH (test channel) instruction. If the condition code is 3 (channel not operational), returns to caller with indication to continue unconditional reserve recovery.
- Issues the HDV (halt-device) instruction.
  - If the condition code is 1 (CSW stored), calls the *condition code one procedure (2).*
- Issues the CLRIO (clear I/O) instruction.
  - If the condition code is 3 (not operational), returns to caller with indication to continue unconditional reserve recovery.
  - If the condition code is 1 (CSW stored), calls the *condition code one procedure (2).*
- Issues the HDV (halt-device) instruction.
  - If the condition code is 3 (not operational), returns to caller with indication to continue unconditional reserve recovery.
  - If the condition code is 1 (CSW stored), calls the *condition code one procedure (2).*
  - If the condition code is 0 (subchannel busy with another device or interruption pending) or condition code 2 (channel working), returns to the caller with indication that no recovery is to be done.
- If *condition code one procedure (2)* did not indicate that detection was to continue, returns to the caller with indication that no recovery is to be done (the error was transient).

2. The Condition Code One Procedure (CC1RTN)

- Entered by the *main procedure (1)* when a condition code 1 is received on HDV or CLRIO instruction.
- If logout-pending is indicated in the CSW, enables that channel for an I/O interruption. If the interruption is received, calls the IOS SLIH to handle the interruption and branches to the *main procedure (1).*
- If a channel check is indicated in the CSW and
  1. this is the first channel check: calls the channel check handler (CCH) and returns to the beginning of the main procedure.
  2. this is the second channel check: returns to the beginning of the main procedure.
  3. this is the third channel check: assumes that there is a permanent error. Returns to the post-status module (IECVPST) with an indication that unconditional reserve recovery is to be continued.
- If a unit check is indicated in the CSW, calls the IOS SLIH and returns to the post-status module (IECVPST) indicating no further recovery is to be done.

The Unconditional Reserve Service Module (IECVURSV)

1. Unconditional Reserve Procedure

- Entered by the *unconditional reserve decision module (IECVDURP)* to perform required cleanup for a device.
- If the request is to perform unconditional reserve recovery, calls the *re-reserve module (IECVRRSV)* indicating that unconditional reserve is to be tried and the device is to be forced offline if recovery fails.
- If the request is to force the device offline, calls the *re-reserve module (IECVRRSV)* indicating box-only function.
- In all cases, issues the IOSINTRP macro to redrive any queued requests.
- Returns to the caller.

Directory

Note: This directory is common to *both IOS* and *EXCP* procedures and modules. For a more complete directory listing for the MVS system, refer to the *OS/VS2 Directory (Microfiche)*, SYB8-0743.

The table below lists, in alphanumeric order, the symbolic name of each IOS or EXCP procedure, names the microfiche cards that contain its code, and tells which pages in this manual refer to it.

The column headings and their meanings are:

Procedure Name: the symbolic name of the procedure.

Descriptive Name: the name given to the procedure in the "Program Organization" chapter.

Module Name: the name of the object module to which the procedure belongs.

Microfiche Name: the name of the microfiche cards that contain the object module's code.

MO Page: the pages in the "Method of Operation" chapter that refer to the procedure.

PO Page: the page in the "Program Organization" chapter where the description of the procedure begins.

Procedure Name	Descriptive Name	Module Name	Microfiche Name	MO Page	PO Page
ACRPROC	ACR-Call Procedure	IECVRSTI	IECVRSTI	60	117
BACKOUT	CRH Backout Procedure	IECVCINT	IECVCINT		106
BASICPRG	Basic Purge Procedure	IGC0001F	IGC0001F	63-64	132
CCHPROC	CCH-Call Procedure	IECVRSTI	IECVRSTI	60	118
CC1RTN	Condition Code 1 Procedure	IECVURDT	IECVURDT		164
CLEARDEV	Clear-Device Procedure	IECVRSTI	IECVRSTI		119
DAVERR	Error-Handling Procedure	IECVDAVV	IECVDAVV		110
DAVESTA*	ESTAE Recovery Procedure	IECVDAVV	IECVDAVV		111
DAVFRR	FRR Recovery Procedure	IECVDAVV	IECVDAVV		112

* An entry point in the module.

Procedure Name	Descriptive Name	Module Name	Microfiche Name	MO Page	PO Page
DAVINT	Interruption-Handling Procedure	IECVDAVV	IECVDAVV		110
DDRPURG	DDR-Purge Procedure	IGC0001F	IGC0001F	63	130
DVRPURG	Driver Interface Procedure	IGC0001F	IGC0001F	64	131
EATTENT1	Attention-Handling Procedure	IECIOSCN	IECIOSAM	53	94
EDETECT1	Unconditional Reserve Scheduling Procedure	IECIOSCN	IECIOSAM		96
EDEVEND1	Unsolicited Device-End Procedure	IECIOSCN	IECIOSAM		90
EDIEINT1	DIE Interface Procedure	IECIOSCN	IECIOSAM	55	92
EDIEINT2	PCI DIE Interface	IECIOSCN	IECIOSAM		92
EPOSTIO1	Post-SIO Procedure	IECIOSCN	IECIOSAM	48, 49	89
EQUED1	Dequeue Procedure	IECIOSCN	IECIOSAM		90
EQUEE1	Enqueue Procedure	IECIOSCN	IECIOSAM		90
ERSTART1	Channel-Restart Procedure	IECIOSCN	IECIOSAM	55	93
ERSTART2					93
ESCHDIO1	SRB-Scheduling Procedure	IECIOSCN	IECIOSAM		90
ESENSE1	Sense Procedure	IECIOSCN	IECIOSAM	54	93
ESIO1	SIO Procedure	IECIOSCN	IECIOSAM	47	88
ESTATUS1	Initial-Status Procedure	IECIOSCN	IECIOSAM	52-55	91
ETCH1	Test-Channel Procedure	IECIOSCN	IECIOSAM	42, 45, 50	87
FRBLK0	Free-Small-Block Procedure	IECVSMGR	IECVSMGR		153
FRBLK4	Free-Medium-Block Procedure	IECVSMGR	IECVSMGR		156
FREEBLK	Free-Large-Block Procedure	IECVSMGR	IECVSMGR		159
GETBLK	Get-Large-Block Procedure	IECVSMGR	IECVSMGR		157
GETBLK0	Get-Small-Block Procedure	IECVSMGR	IECVSMGR		153
GETBLK4	Get-Medium-Block Procedure	IECVSMGR	IECVSMGR		154
GETCORE	Get-Storage Procedure	IECVSMGR	IECVSMGR		159

Procedure Name	Descriptive Name	Module Name	Microfiche Name	MO Page	PO Page
HALT0900*	Functional Recovery Procedure	IGC0003C	IGC0003C		126
HALT3000	CTC Halt Procedure	IGC0003C	IGC0003C		125
HIOCCH	Channel Error Procedure	IECIHIO	IECIHIO		142
HIOFRR*	Functional Recovery Procedure	IECIHIO	IECIHIO		142
HIOIPCI*	Shoulder-Tap Procedure	IECIHIO	IECIHIO		141
HIOLOP	Channel-Logout Procedure	IECIHIO	IECIHIO		142
IECCINTF	CRH SLIH FRR Procedure	IECVCINT	IECVCINT		106
IECCONCS	Connect Channel Set Procedure	IECVCINT	IECVCINT		107
IECCRHAF	CRH Activation FRR Procedure	IECVCINT	IECVCINT		105
IECCRHDF	CRH Deactivation FRR Procedure	IECVCINT	IECVCINT		106
IECFRR*	Functional Recovery Procedure	IECIOSCN	IECIOSAM		95
IECHNSCH*	Channel Scheduler Procedure	IECIOSCN	IECIOSAM	43	87
IECIHIO*	Main Procedure	IECIHIO	IECIHIO		141
IECINT*	Interrupt-Handling Procedure	IECIOSCN	IECIOSAM	51	91
IECLMSGC	Message Exit - Communications (TP)	IECLMSGC	IECLMSGC		208
IECLMSGD	Message Exit - DASD	IECLMSGD	IECLMSGD		208
IECLMSGG	Message Exit - Graphics	IECLMSGG	IECLMSGG		208
IECLMSGM	Message Exit - 3851	IECLMSGM	IECLMSGM		208
IECLMSGT	Message Exit - Tape	IECLMSGT	IECLMSGT		208
IECLMSGU	Message Exit - Unit Record	IECLMSGU	IECLMSGU		208
IECVBRSV	Build Reserve Table Module	IECVBRSV	IECVBRSV	62	
IECVCINT	CRH Interrupt Handler	IECVCINT	IECVCINT	66, 81	104
IECVCPRM*	Pool Initialization Procedure	IECVSMGR	IECVSMGR		160
IECVCRHA	CRH Activation Procedure	IECVCINT	IECVCINT	65	102
IECVCRHD	CRH Deactivation Procedure	IECVCINT	IECVCINT	67	103
IECVCRHS	CRH Schedule SRB Procedure	IECVCINT	IECVCINT		104

* An entry point in the module.

Procedure Name	Descriptive Name	Module Name	Microfiche Name	MO Page	PO Page
IECVCRHT	CRH Timer Pop Procedure	IECVCINT	IECVCINT	66	103
IECVCRHV	CRH STIDC Procedure	IECVCINT	IECVCINT		103
IECVCRH1	TCH Hook Procedure	IECVCRHH	IECVCRHH	66	107
IECVCRH2	SIO/SIOF Hook Procedure	IECVCRHH	IECVCRHH	66	108
IECVCRH3	SENSE SIO Hook Procedure	IECVCRHH	IECVCRHH	66	108
IECVDAVV*	Volume Verification Procedure	IECVDAVV	IECVDAVV	55	109
IECVDDT0	DDTs Load Module	IECVDDT0	IECVDDT0		175
IECVDDT2	DDT Table – Teleprocessing	IECVDDT0	IECVDDT0		177
IECVDDT3	DDT Table – 2305 Model 2	IECVDDT0	IECVDDT0		177
IECVDDT4	DDT Table – Graphics	IECVDDT0	IECVDDT0		177
IECVDDT5	DDT Table – Unit Record	IECVDDT0	IECVDDT0		177
IECVDDT6	DDT Table – 2955 Comm	IECVDDT0	IECVDDT0		177
IECVDDT7	DDT Table – 3704/3705 Comm	IECVDDT0	IECVDDT0		177
IECVDDT8	DDT Table – 3211 Printer	IECVDDT0	IECVDDT0		177
IECVDDT9	DDT Table – 3800 Printer	IECVDDT0	IECVDDT0		177
IECVDDTA	DDT Table – 3890 Micr	IECVDDT0	IECVDDT0		177
IECVDDTB	DDT Table – 3886 OCR	IECVDDT0	IECVDDT0		177
IECVDDTC	DDT Table – 3895 Printer	IECVDDT0	IECVDDT0		177
IECVDDTD	DDT Table – 1287/1288 OCR	IECVDDT0	IECVDDT0		177
IECVDDTE	DDT Table – 3851 MSS	IECVDDT0	IECVDDT0		177
IECVDDTF	DDT Table – 3540 Diskette	IECVDDT0	IECVDDT0		177
IECVDDTG	DDT Table – 3400 Tape	IECVDDT0	IECVDDT0		177
IECVDDTH	DDT Table – 2305 Model 1	IECVDDT0	IECVDDT0		177
IECVDDTJ	DDT Table – 2314	IECVDDT0	IECVDDT0		177
IECVDDTK	DDT Table – 3330 Virtual	IECVDDT0	IECVDDT0		177
IECVDDTL	DDT Table – 3330	IECVDDT0	IECVDDT0		177
IECVDDTM	DDT Table – 3340	IECVDDT0	IECVDDT0		177
IECVDDTN	DDT Table – 3350	IECVDDT0	IECVDDT0		177
IECVDDTO	DDT Table – 3838	IECVDDT0	IECVDDT0		177
IECVDDTQ	DDT Table – 2400 Tape	IECVDDT0	IECVDDT0		177

* An entry point in the module.

Procedure Name	Descriptive Name	Module Name	Microfiche Name	MO Page	PO Page
IECVDURP	Unconditional Reserve Decision Module	IECVDURP	IECVDURP		163
IECVERPL	ERP Loader	IECVERPL	IECVERPL	204	206
IECVESIG	SIGP Entry Procedure	IECVESIO	IECVESIO		152
IECVESIO	Special SIO Module	IECVESIO	IECVESIO	62	151
IECVEXTC*	EOE Interface Procedure	IECVEXCP	IECVEXCP	20	34
IECVHDET	Hot I/O Detection Module	IECVHDET	IECVHDET		112
IECVHREC	Hot I/O Recovery Module	IECVHREC	IECVHREC		113
IECVIRST	I/O Restart Module	IECVIRST	IECVIRST		121
IECVOPTA	TCCW Oper Table – 2955	IECVOTBL	IECVOTBL		176
IECVOPTB	TCCW Oper Table – 3704/3705 Comm	IECVOTBL	IECVOTBL		176
IECVOPTC	TCCW Oper Table – Teleprocessing	IECVOTBL	IECVOTBL		176
IECVOPTD	TCCW Oper Table – DASD	IECVOTBL	IECVOTBL		176
IECVOPTE	TCCW Oper Table – 3211 Printer	IECVOTBL	IECVOTBL		176
IECVOPTF	TCCW Oper Table – 3800 Printer	IECVOTBL	IECVOTBL		176
IECVOPTG	TCCW Oper Table – Graphics	IECVOTBL	IECVOTBL		176
IECVOPTH	TCCW Oper Table – 3890 MICR	IECVOTBL	IECVOTBL		176
IECVOPTI	TCCW Oper Table – 3886 OCR	IECVOTBL	IECVOTBL		176
IECVOPTJ	TCCW Oper Table – 3895 Printer	IECVOTBL	IECVOTBL		176
IECVOPTK	TCCW Oper Table – 1287/1288 OCR	IECVOTBL	IECVOTBL		176
IECVOPTL	TCCW Oper Table – 3851 MSS	IECVOTBL	IECVOTBL		176
IECVOPTM	TCCW Oper Table – 3540 Diskette	IECVOTBL	IECVOTBL		176
IECVOPTN	TCCW Oper Table – 3838 VPSS	IECVOTBL	IECVOTBL		176
IECVOPTT	TCCW Oper Table – Tape	IECVOTBL	IECVOTBL		176
IECVOPTU	TCCW Oper Table – Unit Record	IECVOTBL	IECVOTBL		176
IECVOTBL	TCCW Operation Table – Load Modules	IECVOTBL	IECVOTBL		176
IECVPRCU*	SIRB Clean-Up Procedure	IECVPURG	IECVPURG		144

* An entry point in the module.

Procedure Name	Descriptive Name	Module Name	Microfiche Name	MO Page	PO Page
IECVPRDQ*	Chain-SRB Procedure	IECVPURG	IECVPURG		144
IECVPST*	Appendage Interface Procedure	IECVPST	IECVPST	58	134
IECVQCNT*	Decrement-Count Procedure	IECVPURG	IECVPURG		144
IECVRCHN*	Restore Chain Procedure	IECVEXPR	IECVEXPR	24, 26	39
IECVRDIO	Redrive I/O Module	IECVRDIO	IECVRDIO	62	138
IECVRRSV	Re-reserve Module	IECVRRSV	IECVRRSV	62	139
IECVRSTI*	Set-Up Procedure	IECVRSTI	IECVRSTI		117
IECVSCOM	Compress Procedure	IECVSMGR	IECVSMGR		160
IECVSMFR*	Functional Recovery Procedure	IECVSMGR	IECVSMGR		161
IECVTCCW*	Routing Procedure	IECVTCCW	IECVTCCW		98
IECVURDT	Unconditional Reserve Detection Module	IECVURDT	IECVURDT		164
IECVURSV	Unconditional Reserve Service Module	IECVURSV	IECVURSV		165
IECVXDAS	DASD SIO Module	IECVXDAS	IECVXDAS		146
IECVXDAT	DASD Trap Module	IECVXDAT	IECVXDAT		
IECVXDAU	DASD Unsolicited Module	IECVXDAU	IECVXDAU		
IECVXDRS	2305 SIO Module	IECVXDRS	IECVXDRS		147
IECVXDRT	2305 Trap Module	IECVXDRT	IECVXDRT		
IECVXGRT	Graphics Trap Module	IECVXGRT	IECVXGRT		
IECVXMGN	3851 MSS Sense Module	IECVXMGN	IECVXMGN		
IECVXPRE	3211/3800 EOS Module	IECVXPRE	IECVXPRE		
IECVXPUR*	Purge Procedure	IECVEXPR	IECVEXPR	23, 24	39
IECVXRES*	Restore Procedure	IECVEXPR	IECVEXPR	25	39
IECVXSKE	2314 EOS Module	IECVXSKE	IECVXSKE		
IECVXSKN	2314 Sense Module	IECVXSKN	IECVXSKN		
IECVXSKS	2314 SIO Module	IECVXSKS	IECVXSKS		148
IECVXTAT	Tape Trap Module	IECVXTAT	IECVXTAT		
IECVXTPT	TP Trap Module	IECVXTPT	IECVXTPT		
IECVXT2S	2400 SIO Module	IECVXT2S	IECVXT2S		150
IECVXT3S	3400 SIO Module	IECVXT3S	IECVXT3S		150
IECVXURS	UR SIO Module	IECVXURS	IECVXURS		150
IECVXURT	UR Trap Module	IECVXURT	IECVXURT		
IECVXVRS	3330V SIO Module	IECVXVRS	IECVXVRS		149
IECVXVRT	3330V Trap Module	IECVXVRT	IECVXVRT		
IECVXVRU	3330V Unsolicited Module	IECVXVRU	IECVXVRU		
IECVX025*	SVC 3 Interface Procedure	IECVEXCP	IECVEXCP		38
IECXTLER*	XCTL Procedure in ERP Loader	IECVERPL	IECVERPL	205	207

* An entry point in the module.

Procedure Name	Descriptive Name	Module Name	Microfiche Name	MO Page	PO Page
IGC0003C*	Main Halt Procedure	ICG0003C	IGC0003C	65	124
IGC015*	SVC 15 Procedure	IECVPST	IECVPST	59	135
IGC016*	Entrance/Exit Procedure	IGC0001F	IGC0001F	63, 64	126
IGC017*	Restore Procedure	IGC0001G	IGC0001G	64	145
IGE0025C	ERP Message Writer	IGE0025C	IGE0025C		208
IPIBPURG	IPIB-Purge Procedure	IGC0001F	IGC0001F		131
LCHPURG	LCH Purge Procedure	IGC0001F	IGC0001F	63	129
LOSTCHAN	CCH-Call Procedure for a Lost Channel	IECVRSTI	IECVRSTI	60	120
MIHPROC	MIH-Call Procedure	IECVRSTI	IECVRSTI	61	119
PRGCOMP0	Compres Interface	IGC0001F	IGC0001F		133
PRGESTAE*	ESTAE Recovery Procedure	IGC0001F	IGC0001F		133
PRGFREE	Purge-Free Procedure	IECVSMGR	IECVSMGR		159
PSTEFF	ERP Interface	IECVPST	IECVPST		137
PSTFRRTN	Functional Recovery Procedure	IECVPST	IECVPST		136
PSTIOSB	IOSB-Handling Procedure	IECVPST	IECVPST	56, 57	135
PSTUR	Unconditional Reserve Procedure	IECVPST	IECVPST		138
PSTWAIT	Restartable Wait Procedure	IECVPST	IECVPST		137
PURAPLSR	Applicability-Check Procedure	IGC0001F	IGC0001F	63	132
PURGEFRR*	Functional Recovery Procedure	IGC0001F	IGC0001F		132
RECORDIT	Message Procedure	IECVRSTI	IECVRSTI		120
SIORTN	SIO Procedure	IECVESIO	IECVESIO		152
SIRBPURG	SIRB-Purge Procedure	IGC0001F	IGC0001F	63	129
SMGFREVR*	Block Verification Procedure	IECVSMGR	IECVSMGR		161
SPLPURG	SPL-Purge Procedure	IGC0001F	IGC0001F	63	131
SVC0 (see XCP000)					
SVC15 (see IGC015)					
SVC16 (see IGC016)					
SVC17 (see IGC017)					
SVC33	Halt-I/O Interface Procedure	IECVEXPR	IECVEXPR	28	40
SVC92 (see XCP000)					
SVC114 (see XCP000)					
TCCWI100	CCW Translation Procedure	IECVTCCW	IECVTCCW		98
TCCWM000	Page-Fix Procedure	IECVTCCW	IECVTCCW		99

* An entry point in the module.

Procedure Name	Descriptive Name	Module Name	Microfiche Name	MO Page	PO Page
TCCWM100	Main TIC Procedure	IECVTCCW	IECVTCCW		99
TCCWM200	TIC Resolution Procedure	IECVTCCW	IECVTCCW		100
TCCWM300	TIC Insertion Procedure	IECVTCCW	IECVTCCW		100
TCCWM400	IDAL Procedure	IECVTCCW	IECVTCCW		100
TCCWR000	Address Retranslation Procedure	IECVTCCW	IECVTCCW		101
TCCWU000	Unfix-and-Free Procedure	IECVTCCW	IECVTCCW		101
TCCWX000	Single-Address Translation Procedure	IECVTCCW	IECVTCCW		100
UCBACT	Device Procedure	IECVRSTI	IECVRSTI	59	120
UCBPURG	UCB-Purge Procedure	IGC0001F	IGC0001F	63	130
XCPABE	CHE/ABE Interface Procedure	IECVEXCP	IECVEXCP	22	35
XCPCHE	CHE/ABE Interface Procedure	IECVEXCP	IECVEXCP	22	35
XCPDIE	DIE Procedure	IECVEXCP	IECVEXCP	23	35
XCPEXIT	Exit Procedure	IECVEXCP	IECVEXCP		37
XCPFRR*	Functional Recovery Procedure	IECVEXPR	IECVEXPR		40
XCPMAP	IOSB-to-IOS Mapping Procedure	IECVEXCP	IECVEXCP	22, 23	37
XCPPCI	PCI Interface Procedure	IECVEXCP	IECVEXCP	23	35
XCPPFA	PGFX Interface Procedure	IECVEXCP	IECVEXCP	20	33
XCPPUR	Related-Request Procedure	IECVEXCP	IECVEXCP	26	37
XCPRQE	Get-RQE Procedure	IECVEXCP	IECVEXCP	18	32
XCPTERM	Termination Procedure	IECVEXCP	IECVEXCP	23, 24 26-28	36
XCPVAM	VIO Interface Procedure	IECVEXCP	IECVEXCP	18	33
XCP000	Validity-Check Procedure	IECVEXCP	IECVEXCP	17	32
XCP050	Get-SRB Procedure	IECVEXCP	IEXVEXCP	19	33
XCP110	SIO Interface Procedure	IECVEXCP	IECVEXCP	20	34
XCP115	Translator Interface Procedure	IECVEXCP	IECVEXCP	21	34
XCP145	STARTIO Procedure	IECVEXCP	IECVEXCP	21	35

* An entry point in the module.

Data Areas

The table below lists the data areas that IOS and EXCP use. It gives the acronym and mapping macro for each data area and the identifier that is used as a prefix for field and bit labels.

Data Area	Acronym	Mapping Macro	Identifier
*Beginning-end block	BEB	IECDBEB	BEB
*Channel availability table	CAT	IECDCAT	CAT
*Channel set table	CST	IECDCST	CST
Channel set channel recovery work area	CSCRWA	IECDCSWK	CSCR
*Communications vector table	CVT	CVT	CVT
CRH communications area	CRCA	IECDCRCA	CRCA
*Data control block	DCB	DCBD	DCB
*Data extent block	DEB	IEZDEB	DEB
Device descriptor table	DDT	IECDDT	DDT
*ERP work area (common segment)	EWA	EWAMAP	EWA
ERP work area (DASD segment)	EWD	EWDMAP	EWD
*EXCP debugging area	XDBA	IECDXDBA	XDBA
Fix list	FIX	IECDFIX	FIX
Hot I/O detection thresholds	HIDT	IECDHIDT	HIDT
Indirect address list	IDAL	IECDIDAL	IDAL
*Input/output block	IOB	IEZIOB	IOB
*I/O queue element	IOQ	IECDIOQ	IOQ
*I/O recovery table	IRT	IECDIRT	IRT
*I/O supervisor block	IOSB	IECDIOSB	IOS
*I/O supervisor purge interface block	IPIB	IECDIPIB	IPIB
*Logical channel queue table	LCH	IECDLCH	LCH
Purge parameter list	PPL	IECDPPL	PPL
Purged I/O restore list	PIRL	IECDPIRL	PIR
*Request queue element	RQE	IECDRQE	RQE
Reserve table	RESVTAB	IECDRESV	RESVTAB
*Service request block	SRB	IHASRB	SRB
*Status collection data area	SCD	IECDSCD	SCD
*Task control block	TCB	IKJTCB	TCB
*Translation control block	TCCW	IECDTCCW	TCCW
*Unit control block	UCB	IEFUCBOB	UCB
Vector of IOS drivers	VOID	IECDVOID	VOID

The format and contents of these data areas are described in the microfiche document *OS/VS2 Data Areas*, SYB8-0606.

*Those data areas that are preceded by an asterisk are also described in the *OS/VS2 System Programming Library: Debugging Handbook*, GC28-0632.

CCW Translation Operation Table

The CCW translation operation table communicates to IECVTCCW, the CCW translator, 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 handling consists of IECVTCCW treating a CCW as a data transfer command, translating the data address from a virtual address to a real address, and fixing the data area.

The following bits are defined in each byte of the CCW translation operation table to indicate that special handling is needed by IECVTCCW:

X'80'	Bit 0	The channel command can cause the next CCW in the channel program to be skipped.
X'40'	Bit 1	The channel command is a non-data transfer command.

The following is a list of device classes and specific devices with their corresponding CCW translation operation table CSECT names. These CSECTs are contained in module IECVOTBL.

Tape	IECVOPTT
Teleprocessing	IECVOPTC
Direct Access	IECVOPTD
Display Graphics	IECVOPTG
Unit Record	IECVOPTU
2955 Communication	IECVOPTA
3704/3705 Communication	IECVOPTB
3211 Printer	IECVOPTE
3800 Printer	IECVOPTF
3890 Document Processor	IECVOPTH
3886 Optical Character Reader	IECVOPTI
3895 Printer	IECVOPTJ
1287/1288 Optical Character Reader	IECVOPTK
3851 Mass Storage Controller	IECVOPTL
3540 Diskette I/O Unit	IECVOPTM
3838 Array Processor	IECVOPTN

Device Descriptor Table (DDT)

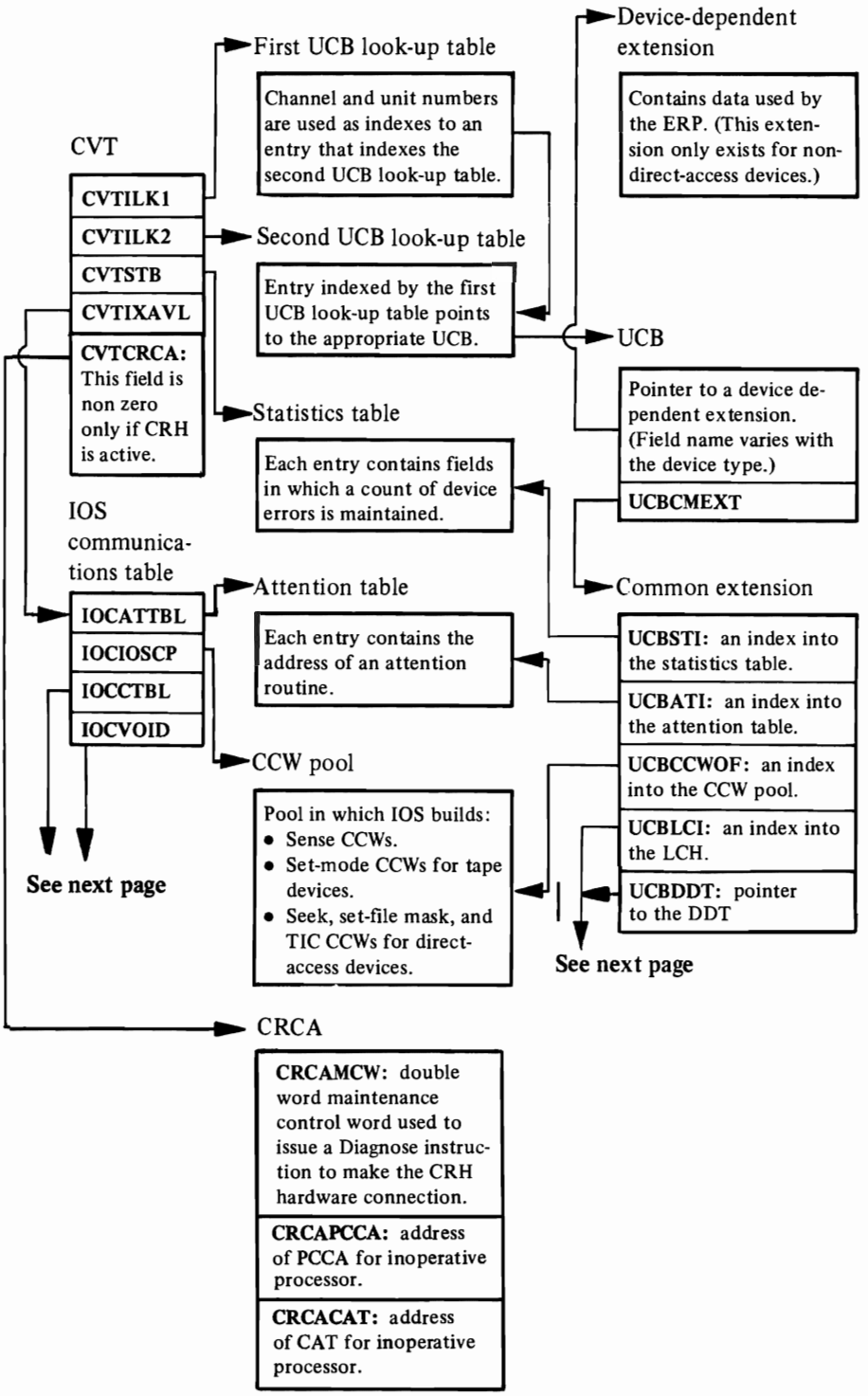
The device descriptor table (DDT) is a variable length list of four-byte fields pointed to by UCBDDT. These fields contain pointers to device dependent IOS exits, tables, and other device dependent data. A DDT will be pointed to by every UCB representing the same kind of device. Macro IECDDT describes the DDT.

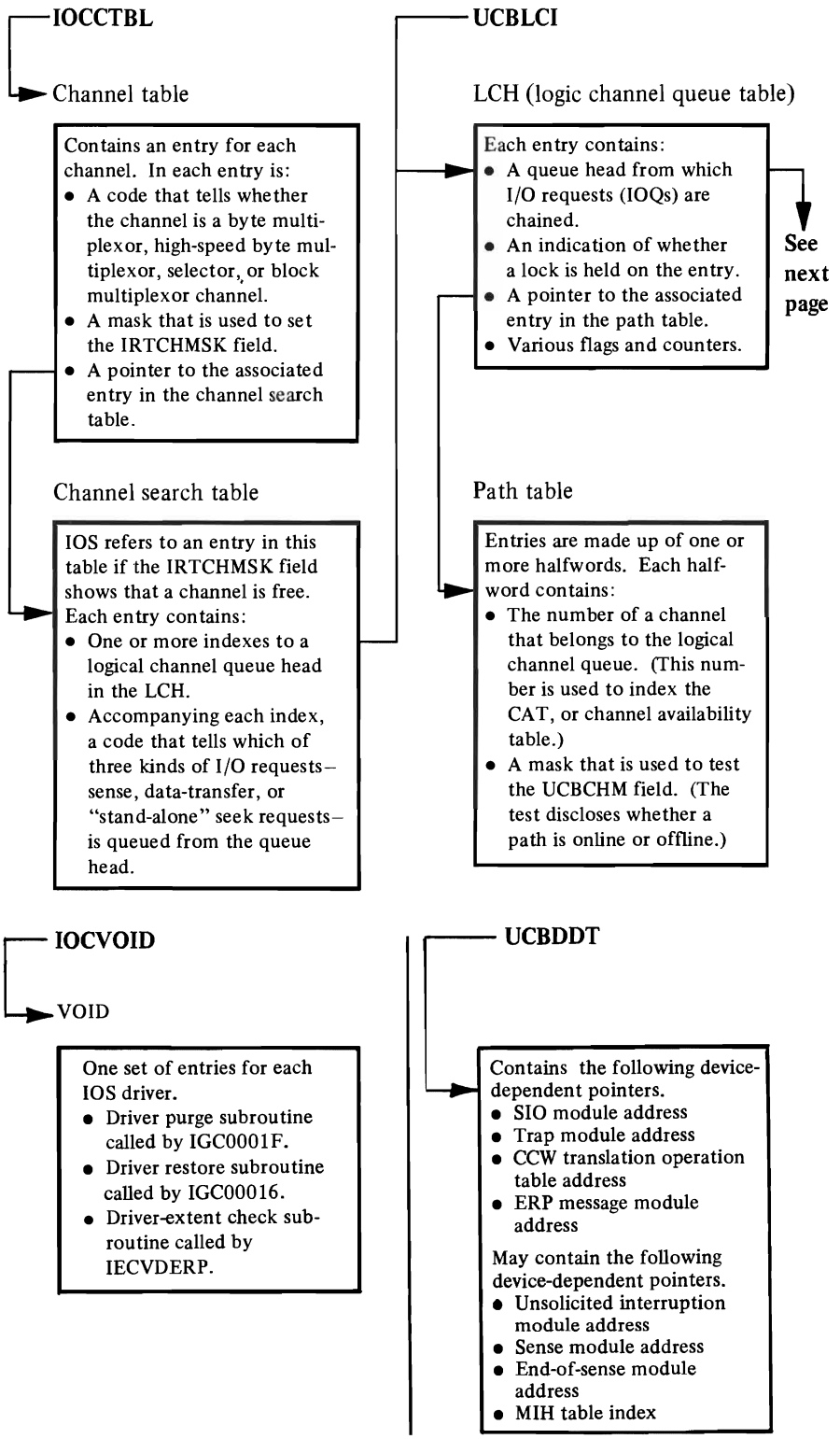
The following is a list of devices with their corresponding DDT CSECT names. These CSECTs are contained in module IECVDDT0.

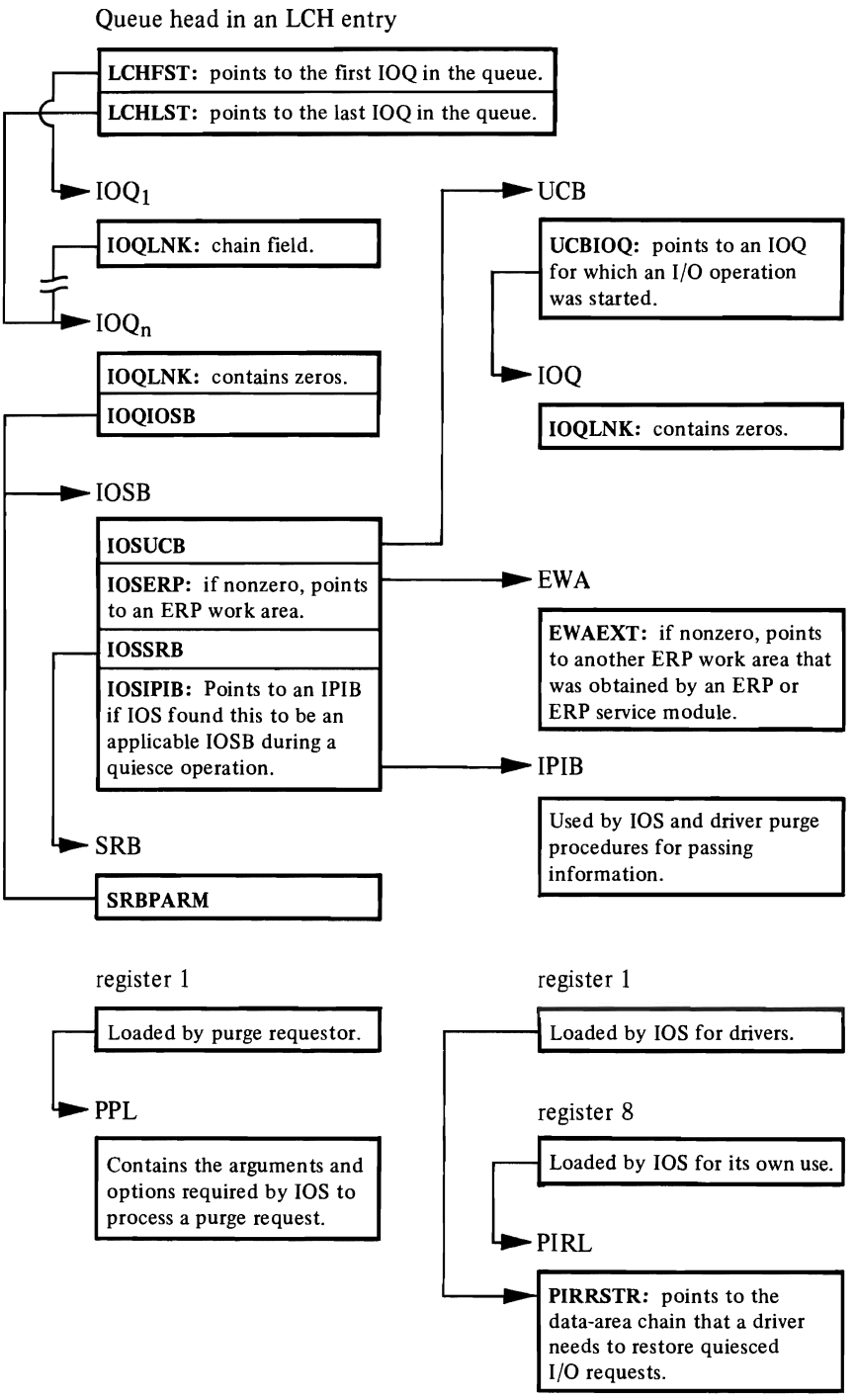
Device	DDT CSECT
Teleprocessing	IECVDDT2
2305-2 Fixed Head Storage	IECVDDT3
Graphics	IECVDDT4
Unit Record	IECVDDT5
2955 Communications	IECVDDT6
3704/3705 Communications Controller	IECVDDT7
3211 Printer	IECVDDT8
3800 Printer	IECVDDT9
3890 Document Processor	IECVDDTA
3886 Optical Character Reader	IECVDDTB
3895 Document/Inscriber	IECVDDTC
1287/1288 Optical Reader	IECVDDTD
3851 Mass Storage Controller	IECVDDTE
3540 Diskette I/O Unit	IECVDDTF
3400 Series Magnetic Tape Units	IECVDDTG
2305-1 Fixed Head Storage	IECVDDTH
2314 Direct Access Storage Facility	IECVDDTJ
3330-V Disk Storage	IECVDDTK
3330 Disk Storage	IECVDDTL
3340 Direct Access Storage Facility	IECVDDTM
3350 Direct Access Storage Facility	IECVDDTN
3838 Array Processor	IECVDDTO
2400 Series Magnetic Tape Units	IECVDDTQ

Connections between Principal IOS Data Areas

The diagram below shows the connections between data areas used by IOS in starting an I/O operation, in responding to an I/O event, and in purging and restoring I/O requests. The diagram contains a *selection* of data-area fields. No data area is shown in its entirety.




IOCCTBL
Channel table
Contains an entry for each channel. In each entry is:
• A code that tells whether the channel is a byte multiplexor, high-speed byte multiplexor, selector, or block multiplexor channel.
• A mask that is used to set the IRTCHMSK field.
• A pointer to the associated entry in the channel search table.
Channel search table
IOS refers to an entry in this table if the IRTCHMSK field shows that a channel is free. Each entry contains:
• One or more indexes to a logical channel queue head in the LCH.
• Accompanying each index, a code that tells which of three kinds of I/O requests– sense, data-transfer, or "stand-alone" seek requests– is queued from the queue head.
UCBLCI
LCH (logic channel queue table)
Each entry contains:
• A queue head from which I/O requests (IOQs) are chained.
• An indication of whether a lock is held on the entry.
• A pointer to the associated entry in the path table.
• Various flags and counters.
See next page
Path table
Entries are made up of one or more halfwords. Each halfword contains:
• The number of a channel that belongs to the logical channel queue. (This number is used to index the CAT, or channel availability table.)
• A mask that is used to test the UCBCHM field. (The test discloses whether a path is online or offline.)
IOCVOID
VOID
One set of entries for each IOS driver.
• Driver purge subroutine called by IGC0001F.
• Driver restore subroutine called by IGC00016.
• Driver-extent check subroutine called by IECVDERP.
UCBDDT
Contains the following device-dependent pointers.
• SIO module address
• Trap module address
• CCW translation operation table address
• ERP message module address
May contain the following device-dependent pointers.
• Unsolicited interruption module address
• Sense module address
• End-of-sense module address
• MIH table index


Queue head in an LCH entry
LCHFST: points to the first IOQ in the queue.
LCHLST: points to the last IOQ in the queue.
IOQ1
IOQLNK: chain field.
IOQn
IOQLNK: contains zeros.
IOQIOSB
IOSB
IOSUCB
IOSERP: if nonzero, points to an ERP work area.
IOSSRB
IOSIPIB: Points to an IPIB if IOS found this to be an applicable IOSB during a quiesce operation.
SRB
SRBPARM
UCB
UCBIOQ: points to an IOQ for which an I/O operation was started.
IOQ
IOQLNK: contains zeros.
EWA
EWAEXT: if nonzero, points to another ERP work area that was obtained by an ERP or ERP service module.
IPIB
Used by IOS and driver purge procedures for passing information.
register 1
Loaded by purge requestor.
PPL
Contains the arguments and options required by IOS to process a purge request.
register 1
Loaded by IOS for drivers.
register 8
Loaded by IOS for its own use.
PIRL
PIRRSTR: points to the data-area chain that a driver needs to restore quiesced I/O requests.

Data Area Usage Table

The data area usage table is a cross reference between the data area names and the IOS module names. It has two parts.

1. The first part lists data areas in alphabetical order. Across from each data area are all the modules which use that data area.
2. The second part lists modules in alphameric order. Across from each module are all the data areas which that module uses.

The Access column of the table contains the following symbols:

C: if the module creates the data area
R: if the module refers to the data area
M: if the module modifies the data area

Data Area Name	Usage	Access	Usage	Access	Usage	Access
CAT	IECIOSCN	(RM)	IECVCINT	(R)	IECVCRHH	(R)
	IECVHDET	(RM)	IECVHREC	(RM)	IECVIRST	(RM)
	IECVMAP	(R)	IECVRSTI	(RM)	IECVURDT	(R)
CRCA	IECIHIO	(RM)	IECIOSCN	(RM)	IECVBRSV	(R)
	IECVCINT	(RM)	IECVCRHH	(CRM)	IECVERPL	(R)
	IECVESIO	(RM)	IECVGENA	(R)	IECVIRST	(R)
	IECVRDIO	(R)	IECVRSTI	(R)	IECVURDT	(RM)
CSCR	IECVIRST	(RM)	IECVRSTI	(RM)		
CST	IECIHIO	(R)	IECIOSCN	(R)	IECVCINT	(R)
	IECVESIO	(R)	IECVGENA	(R)	IECVHDET	(RM)
	IECVHREC	(R)	IECVIRST	(R)	IECVMAP	(R)
	IECVRSTI	(RM)	IGC0003C	(R)		
CVT	IECIHIO	(R)	IECIOSCN	(R)	IECVBRSV	(R)
	IECVCINT	(RM)	IECVDAVV	(R)	IECVERPL	(R)
	IECVESIO	(R)	IECVEXCP	(R)	IECVEXPR	(R)
	IECVGENA	(R)	IECVHDET	(R)	IECVIRST	(R)
	IECVMAP	(R)	IECVPST	(R)	IECVPURG	(R)
	IECVRDIO	(R)	IECVRRSV	(R)	IECVRSTI	(R)
	IECVSMGR	(R)	IECVTCCW	(R)	IECVURDT	(R)
	IECVURSV	(R)	IGC0001F	(R)	IGC0001G	(R)
	IGC0003C	(R)	IGE0025C	(R)	IGE0025D	(R)
DDT	IECIOSCN	(R)	IECVTCCW	(R)	IGE0025C	(R)
EWA/EWD	IECIOSCN	(CM)	IECVDURP	(RM)	IECVPST	(C)
	IECVURDT	(RM)	IGE0025C	(RM)	IGE0025D	(RM)
HIDT	IECVHDET	(R)	IECVHIDT	(C)	IECVHREC	(R)
IOQ	IECIOSCN	(CRM)	IECVCINT	(R)	IECVGENA	(RM)
	IECVRSTI	(R)	IGC0001F	(RM)	IGC0003C	(R)
IOSB	IECIOSCN	(CRM)	IECVCRHH	(R)	IECVDAVV	(RM)
	IECVDURP	(R)	IECVERPL	(RM)	IECVEXCP	(C)
	IECVEXPR	(RM)	IECVGENA	(R)	IECVRSTI	(R)
	IECVURDT	(R)	IECVURSV	(R)	IECVXDAS	(RM)
	IECVXDRS	(RM)	IECVXSKS	(RM)	IECVXT2S	(R)
	IECVXT3S	(R)	IECVXURS	(R)	IECVXVRS	(RM)
	IGC0001F	(RM)	IGC0003C	(R)	IGE0025C	(RM)
	IGE0025D	(R)				

Data Area Name	Usage	Access	Usage	Access	Usage	Access
IPIB	IECIOSCN	(RM)	IECVCINT	(R)	IECVEXPR	(RM)
	IECVPURG	(RM)	IECVCRHH	(R)	IGC0001F	(CRM)
IRT	IECIOSCN	(RM)	IECVCRHH	(R)	IECVGENA	(R)
	IECVHDET	(RM)	IECVRSTI	(R)	IECVSMGR	(RM)
LCH	IECIOSCN	(RM)	IECVCRHH	(R)	IECVGENA	(RM)
	IECVMAP	(R)	IGC0001F	(RM)		
PIRL	IECVEXPR	(CM)	IGC0001F	(CM)	IGC0001G	(M)
PPL	IGC0001F	(R)				
RESVTAB	IECVBRSV	(CRM)	IECVHREC	(CRM)	IECVIRST	(CRM)
	IECVRRSV	(RM)	IECVRSTI	(CRM)	IECVURSV	(CRM)
SCD	IECVHDET	(RM)	IECVHREC	(RM)		
SRB	IECIOSCN	(CM)	IECVCINT	(RM)	IECVEXCP	(CM)
	IECVEXPR	(R)	IECVPST	(RM)	IGC0001F	(R)
	IGE0025C	(R)				
UCB	IECIHIO	(RM)	IECIOSCN	(RM)	IECVBRSV	(R)
	IECVCINT	(RM)	IECVCRHH	(RM)	IECVDAVV	(RM)
	IECVDURP	(R)	IECVERPL	(R)	IECVEXCP	(R)
	IECVGENA	(RM)	IECVHREC	(R)	IECVIRST	(R)
	IECVPST	(RM)	IECVRDIO	(RM)	IECVRRSV	(RM)
	IECVRSTI	(RM)	IECVTCCW	(R)	IECVURDT	(RM)
	IECVURSV	(RM)	IECVXDAS	(RM)	IECVXDRS	(RM)
	IECVXSKS	(RM)	IECVXT2S	(R)	IECVXT3S	(R)
	IECVXURS	(R)	IECVXVRS	(RM)	IGC0001F	(R)
	IGC0001G	(R)	IGC0003C	(RM)	IGE0025C	(R)
	IGE0025D	(R)				
VOID	IGC0001F	(R)	IGC0001G	(R)		

Module Name	Usage	Access	Usage	Access	Usage	Access
IECIHIO	CRCA	(RM)	CST	(R)	CVT	(R)
	UCB	(RM)				
IECIOSCN	CAT	(RM)	CRCA	(RM)	CST	(R)
	CVT	(R)	DDT	(R)	EWA/EWD	(CM)
	IOQ	(CRM)	IOSB	(CRM)	IPIB	(RM)
	IRT	(RM)	LCH	(RM)	SRB	(CM)
	UCB	(RM)				
IECVBRSV	CRCA	(R)	CVT	(R)	RESVTAB	(CRM)
	UCB	(R)				
IECVCINT	CAT	(R)	CRCA	(RM)	CST	(R)
	CVT	(R)	IOQ	(R)	IPIB	(R)
	SRB	(RM)	UCB	(RM)		
IECVCRHH	CAT	(R)	CRCA	(CRM)	IOSB	(R)
	IPIB	(R)	IRT	(R)	LCH	(R)
	UCB	(RM)				
IECVDAVV	CVT	(R)	IOSB	(RM)	UCB	(RM)
IECVDURP	EWA/EWD	(RM)	IOSB	(R)	UCB	(R)
IECVERPL	CRCA	(R)	CVT	(R)	IOSB	(RM)
	UCB	(R)				
IECVESIO	CRCA	(RM)	CST	(R)	CVT	(R)
	UCB	(RM)				

Module Name	Usage	Access	Usage	Access	Usage	Access
IECVEXCP	CVT	(R)	IOSB	(C)	SRB	(CM)
	UCB	(R)				
IECVEXPR	CVT	(R)	IOSB	(R)	IPIB	(RM)
	PIRL	(CM)	SRB	(R)		
IECVGENA	CRCA	(R)	CST	(R)	CVT	(R)
	IOQ	(RM)	IOSB	(R)	IRT	(R)
	LCH	(RM)	UCB	(RM)		
IECVHDET	CAT	(RM)	CST	(RM)	CVT	(R)
	HIDT	(R)	IRT	(RM)	SCD	(RM)
IECVHIDT	HIDT	(C)				
IECVHREC	CAT	(RM)	CST	(R)	HIDT	(R)
	RESVTAB	(CRM)	SCD	(RM)	UCB	(R)
IECVIRST	CAT	(RM)	CRCA	(R)	CSCR	(RM)
	CST	(R)	CVT	(R)	RESVTAB	(CRM)
	UCB	(RM)				
IECVMAP	CAT	(R)	CST	(R)	CVT	(R)
	LCH	(R)	UCB	(R)		
IECVPST	CVT	(R)	EWA/EWD	(C)	IOSB	(RM)
	SRB	(RM)	UCB	(RM)		
IECVPURG	CVT	(R)	IPIB	(RM)		
IECVRDIO	CRCA	(R)	CVT	(R)	UCB	(RM)
IECVRRSV	CVT	(R)	RESVTAB	(RM)	UCB	(RM)
IECVRSTI	CAT	(R)	CRCA	(R)	CSCR	(RM)
	CST	(RM)	IOQ	(R)	IOSB	(R)
	IRT	(R)	RESVTAB	(CRM)	UCB	(RM)
IECVSMGR	CVT	(R)	IRT	(RM)		
IECVTCCW	CVT	(R)	DDT	(R)	UCB	(R)
IECVURDT	CAT	(R)	CRCA	(RM)	CVT	(R)
	EWA/EWD	(RM)	IOSB	(R)	UCB	(RM)
IECVURSV	CVT	(R)	IOSB	(R)	RESVTAB	(CRM)
	UCB	(RM)				
IECVXDAS	IOSB	(RM)	UCB	(RM)		
IECVXDRS	IOSB	(RM)	UCB	(RM)		
IECVXSKS	IOSB	(RM)	UCB	(RM)		
IECVXT2S	IOSB	(R)	UCB	(R)		
IECVXT3S	IOSB	(R)	UCB	(R)		
IECVXURS	IOSB	(R)	UCB	(R)		
IECVXVRS	IOSB	(RM)	UCB	(RM)		
IGC0001F	CVT	(R)	IOQ	(RM)	IOSB	(RM)
	IPIB	(CRM)	LCH	(RM)	PIRL	(CM)
	PPL	(R)	SRB	(R)	UCB	(R)
	VOID	(R)				
IGC0001G	CVT	(R)	PRIL	(M)	UCB	(R)
	VOID	(R)				
IGC0003C	CST	(R)	CVT	(R)	IOQ	(R)
	IOSB	(R)	UCB	(RM)		
IGE0025C	CVT	(R)	DDT	(R)	EWA/EWD	(RM)
	IOSB	(RM)	SRB	(R)	UCB	(R)
IGE0025D	CVT	(R)	EWA/EWD	(RM)	IOSB	(R)
	UCB	(R)				

Diagnostic Aids

Diagnostic aids information, formerly found in this section, can now be found in the following books:

OS/VS2 System Programming Library: MVS Diagnostic Techniques. This book contains the entire contents of the Diagnostic Aids section: the table of EXCP ABEND codes, the EXCP debugging area, the output of IOS recovery procedures, informative IOSB fields, wait-state codes, the table of messages, and the table of IOS return codes.

OS/VS Message Library: VS2 System Codes. This book contains information from wait-state codes, the table of EXCP ABEND codes, and the table of IOS return codes.

OS/VS Message Library: VS2 System Messages. This book contains information from the table of messages.

Note: Both EXCP and IOS diagnostic aids are described in this chapter under separate headings.

Table of EXCP ABEND Codes

The table below matches ABEND codes with the symbolic names of the EXCP procedures that issue them. To find out what processing conditions cause a particular ABEND code to be issued, refer to the description of the procedure that issues it. (Use the procedure's symbolic name to find a page reference to its description in this appendix's "Directory.")

Note: For the meanings of IOS ABEND codes, refer to the identified procedures. All procedures except XCPFRR are in module IECVEXCP. Procedure XCPFRR is in module IECVEXPR.

Code	Procedure
X'15C'	XCP000
X'172'	XCP000
X'200'	XCPFRR
X'300'	XCP000
X'400'	XCP000
X'500'	XCP000
X'700'	XCPFRR,XCPTERM
X'800'	XCPPFA,XCPTERM,XCP115
X'A00'	XCPFRR, XCPTERM
X'B00'	XCPFRR
X'E00'	XCPTERM

The EXCP Debugging Area

EXCP's functional recovery procedure, XCPFRR, does not put diagnostic data in the SDUMP buffer. Instead, it gets storage for its own debugging area and puts diagnostic data there. Also, the variable recording area of the SDWA is used to contain diagnostic data.

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

1. Get the address of the CVT from location X'4C' in the dump.
2. Get the address of the TCB from the first word of the CVT.
3. Look X'C0' bytes into the TCB and get the address of the debugging area.
4. 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:

Byte	Contents
0	The ABEND code that EXCP issued.
2	A byte that shows where the error occurred. These are the possible bit settings and their meanings: *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 CHE appendage. *X'09'*: The error occurred in an ABE appendage. *X'05'*: The error occurred in an EOE appendage. *X'03'*: The error occurred in a PGFX appendage. *X'01'*: The error occurred in an SIO appendage.
3	Reserved
4	The PSW before RTM was entered. (RTM is entered when a program check occurs or when an ABEND macro is issued.)
C	Reserved
E	ABEND code at entry to the FRR.
10	The register contents before RTM was entered. If there is an RQE, and byte 25, bit 5 (RQEDIE) is 'on', the error occurred in the PCI appendage during the disabled interruption exit (DIE) processing. (EXCP PCI appendages execute as part of the DIE processing for V=R and EXCPVR requests.) The LOGREC entry and the SVC dump record initiated by IECIOSCN contains the register contents and PSW at the time of the original error.
50	Translation exception address.
54	The RQE for the I/O request that was being processed.

The remainder of the debugging area contains up to twelve 160 byte blocks involved with the EXCP request. If these blocks are present, they appear in the following sequence:

EWA
SRB/IOSB
TCCW
IDAL
FIX list
BEB
. . .

The first 160 bytes following the last block are zero. The SRB and TCCW are valid only if the address of the RQE within these blocks is valid.

The format and contents of the SDWA variable recording area are as follows:

Byte (offset in SDWA)	Contents
194	original ABEND code
196	adjusted ABEND code set by XCPFRR
198	highest lock held word from the PSA

19C	contents of the 6 word FRR parameter area
1B4	contents of the active RQE
1DC	TCCW option byte
1DD	TCCW translation flag
1DE	IOSB completion code
1DF	reserved
1E0	ASID of the EXCP request

The Output of IOS Recovery Procedures

Functional (FRR) and ESTAE recovery procedures can record their virtual storage environments by two means:

- By issuing an SDUMP macro, which causes the contents of the 4K SDUMP buffer to be written in a SYS1.DUMP data set. (There are ten SYS1.DUMP data sets, SYS1.DUMP00–09.)
- By issuing a SETRP macro, specifying RECORD=YES, which directs RTM to write the SDWA in the SYS1.LOGREC data set.

Some Facts about SYS1.DUMP Dumps

To format a dump for a SYS1.DUMP data set, use the AMDPRDMP service aid (*OS/VS2 Service Aids*, GC28-0633, tells how). If the dump contains an SDUMP buffer record that was put in the SYS1.DUMP data set by an IOS recovery procedure, each page will be titled "IOS-*module name* ERROR," where *module name* identifies the module to which the recovery procedure belongs.

To locate the SDUMP buffer record, you must:

1. Get the address of the CVT from location X'4C' in the dump.
2. Look X'24C' bytes into the CVT and get the address of the SDUMP buffer record.

The third halfword of the SDUMP buffer record tells you how much of the 4K bytes contains meaningful data; six bytes of zeros mark the end of the meaningful data.

Some Facts about SYS1.LOGREC Dumps

To get a dump of the SYS1.LOGREC data set, use the IFCEREP1 service aid (*OS/VS2 System Programming Library: SYS1.LOGREC Error Recording*, GC28-0677, tells how). IFCEREP1 formats the standard area—the first 404 bytes—of each SDWA into a series of titles, each followed by pertinent data found in the standard area. (For example, under the title *Component/Module Name/ID*, you would find the module name *IECIOSCN* if the functional recovery procedure of the basic IOS module wrote in the SDWA.) IFCEREP1 puts the variable area—the last 108 bytes—of each SDWA in a decimal or hexadecimal format, whichever you request.

The remaining topics in this section describe the output—the SDUMP buffer records and SDWA variable areas—of IOS recovery procedures. Before looking at the descriptions for the first time, note these facts:

- Offsets into SDUMP buffer records and SDWA variable areas are given in hexadecimal numbers.
- The formats of data areas listed as part of an SDUMP buffer record or SDWA variable area are shown in the microfiche document *OS/VS2 Data Areas*, SYB8-0606, unless stated otherwise.

Output of the Basic IOS Module (IECIOSCN)

The module's functional recovery procedure, IECFRR, puts one or more of these settings in byte 6 of the SDUMP buffer record:

X'80', indicating that an IRT is in the record, beginning at byte 8.

X'40', if a UCB is in the record.

X'20', if an IOQ is in the record.

X'10', if an IOSB is in the record.

X'08', if a logical channel queue, the header of the "small block" pool, and the first 2048-byte segment of the pool are in the record. (The format of the header and segment is shown in the "Program Organization" chapter under "Storage Manager Module (IECVSMGR).")

If a UCB lock was held when IECFRR was entered, the UCB associated with that lock appears in the record. If an LCH lock was held when IECFRR was entered, the LCH associated with that lock, the header of the "small block" pool, and the pool's first 2,048-byte segment is included. If an I/O request was being processed, its IOQ and IOSB appears. These data areas appear in this order: UCB, IOQ, IOSB, logical channel queue, "small block" pool header, first "small block" pool segment.

Other 4K records and the output of IECFRR follow the SDUMP buffer record in the dump. These records contain the SQA (system queue area), the system's trace tables, and the nucleus.

IECFRR puts the following data in the variable recording area of the SDWA:

At byte 0: X'80', if an IOQ is in the SDWA.
X'40', if a UCB is in the SDWA.

At byte 1: the code that was returned when IECFRR issued an SDUMP macro to write in the SYS1.DUMP data set. (X'FF' means nothing was written.)

At byte 4: an IOQ, if an I/O request was being processed when IECFRR was entered.

At byte 10: a UCB (prefix segment included), if a UCB lock was held when IECFRR was entered.

Output of the Build Reserve Table Module (IECVBRSV)

The functional recovery routine (BRSVFRR) of IECVBRSV issues an SDUMP macro requesting an SQA, nucleus, all PSAs and a summary.

BRSVFRR puts the 24-byte FRR parameter area into the SDWA variable recording area.

Output of the DAVV Module (IECVDAVV)

The module's ESTAE recovery procedure, DAVVESTA, puts the following data in the SDUMP buffer record:

At byte 6: the SRB being processed when it was entered.

At byte 32: the IOSB being processed when it was entered.

At byte 9E: the ERP work area used by DAVV. (The work area includes the common segment, EWA, and the direct-access segment, EWD.)

At byte 13E: the UCB (prefix segment included) used in the processing that preceded the error.

DAVVESTA puts the following data in the variable area of the SDWA:

At byte 0: the IOSB being processed when it was entered.

At byte 6C: X'04', a code indicating that DAVVESTA asked RTM to return control instead of continuing termination processing.

Output of the Hot I/O Recovery Module (IECVHREC)

This module's functional recovery procedure (HRECFRR) takes an SDUMP but puts no data in the SDUMP buffer.

HRECFRR puts the following data into the variable area of the SDWA:

At byte 0: A copy of the SCD.

At byte 32: A copy of the FRR work area, which contains the following:

word 0:	first base register
word 1:	second base register
word 2:	SRB pointer
word 3:	work area pointer
word 4:	SCD pointer
word 5, byte 0:	Flags
X'01'	reserved
X'02'	reserved
X'04'	FRR recursion indicator
X'08'	address of work area is valid
X'10'	reserved devices found, message IEA421E not yet issued
X'20'	channel can be enabled
X'40'	channel was reset, re-reserves not complete
X'80'	SALLOC lock held
word 5, bytes 1-3:	reserved

Output of the I/O Restart Module (IECVIRST)

The functional recovery procedures, (IRSTFRR), of this module issues an SDUMP macro requesting SQA, the nucleus, and the 4K buffer to be dumped. The work area (storage area retrieved via GETMAIN that holds the compiler's automatic data) is copied to the 4K buffer along with each reserve table segment.

IRSTFRR puts the following data in the variable area of the SDWA:

At byte 0: The 24 byte FRR parameter area returned by the SETFRR macro.

At byte 24: Halfword channel mask. Each bit in the halfword channel mask corresponds to a given channel.
(Bit 1 corresponds to channel 1
•
•
•
Bit 16 corresponds to channel 16.)
If a bit is on, the corresponding channel encountered an error.

IECVIRST loads one or more wait states. For each loaded wait state, a system termination record is written to the SYS1.LOGREC data set. *Note:* this record may not appear in the data set since the system may not be able to perform I/O operations before the wait state is loaded. It appears in the SYS1.LOGREC buffer located in the SQA in storage. The mapping macro, IHALRB, maps the system termination record. The variable area within the system termination record contains:

At byte 0: Current registers (0-15)

At byte 64: The 24 byte FRR parameter area returned by the SETFRR macro.

At byte 84: Halfword channel mask. Each bit in the halfword channel mask corresponds to a given channel.
(Bit 1 corresponds to channel 1
•
•
•
Bit 16 corresponds to channel 16.)
If a bit is on, the corresponding channel encountered an error.

At byte 86: Halfword channel set id.

Output of the Nonresident Halt-I/O Module (IGC0003C)

The module's functional recovery procedure, HALT0900, writes no SDUMP buffer record. If HALT0900 is the first recovery procedure entered by RTM, it writes the following data in the variable area of the SDWA:

At byte 0: the contents of register 0 and 1 when the module was entered to halt a teleprocessing operation.

At byte 8: the IOQ for the teleprocessing operation.

At byte 14: the UCB (prefix segment included) for the teleprocessing device.

Additionally, HALT0900 directs RTM to put trace data, task-related data areas, and all the IECIHIO code in a user dump (SYSABEND, SYSMDUMP, or SYSUDUMP), if such a dump was requested.

Output of the Nonresident Purge Module (IGC0001F)

The module's functional recovery procedure, PURGEFRR, puts data in the SDUMP buffer, but the module's ESTAE recovery procedure, PRGESTAE, writes the contents of the buffer into the SYS1.DUMP data set. PURGEFRR puts the following information into the SDUMP buffer:

At byte 6: the PPL.

At byte 16: the IPIB.

At byte 3A: a work area whose contents include a variable number of saved registers, a list of pages to be fixed, and the list forms of macros used by the module.

At byte 106: if the module holds a UCB lock, the UCB (prefix and common segments only) associated with that lock.

At byte 126: if the module holds an LCH lock, the logical channel queue associated with that lock and all the IOQs on the logical channel queue.

PURGEFRR puts the following data into the variable area of the SDWA:

At byte 0: IGC0001F (the module name).

At byte 8: IGC016 (the module's entry point).

At byte 16: PURGEFRR (the recovery procedure's name and entry point).

PRGESTAE puts the same data in its SDWA, except at byte 16, where it writes its own name.

Output of the Post-Status Module (IECVPST)

The module's functional recovery procedure, PSTFRRTY, puts at byte 6 of the SDUMP buffer record the IOSB that was being processed when the error occurred.

PSTFRRTY puts the following data in the variable area of the SDWA:

At byte 0: the IOSB that was being processed when the error occurred.

At byte 6C: IECVPST (the module name).

At byte 73: X'04', a code indicating that PSTFRRTY asked RTM to return control instead of continuing termination processing.

At byte 74: the address of the IOSB.

At byte 78: the address of the FRR work area.

At byte 7C: the contents of the base register.

Output of the Redrive I/O Service Routine (IECVRDIO)

The module's functional recovery procedure, RDIOFRR, puts at byte 6 of the SDUMP buffer record the general work area used for automatic data.

The following is placed in the variable area of the SDWA:

At byte 0: a copy of the 24-byte FRR work area pointed to by SDWAPARM.

Output of the Re-Reserve Service Routine (IECVRRSV)

The module's functional recovery procedure, RRSVFRR, writes no SDUMP buffer record. The following is placed in the variable area of the SDWA:

At byte 0: a copy of the 24-byte FRR work area pointed to by SDWAPARM.

Output of the Resident Halt-I/O Module (IECIHIO)

The module's functional recovery procedure, HIOFRR, puts at byte 6 of the SDUMP buffer record the UCB (prefix segment included) for the device on which a channel program was to be halted. Following the SDUMP buffer record in the dump are other 4K records written by HIOFRR. These contain all the IECIHIO code, the PSA or prefixed save area (the first 4K bytes of low storage), and the system's trace tables.

HIOFRR puts the following data in the variable area of the SDWA:

At byte 0: X'0C', if the error occurred in the shoulder-tapped processor; otherwise, the code that was returned when HIOFRR issued an SDUMP macro to write in the SYS1.DUMP data set. (X'FF' means nothing was written to the SYS1.DUMP data set.)

At byte 1: the UCB (prefix included) for the device on which a channel program halted.

Output of the Special SIO Module (IECVESIO)

This module's functional recovery procedure, ESIOFRR, does not use the SDUMP buffer. However, the following is placed in the variable area of the SDWA.

At byte 0: The 24-byte FRR parameters.

Output of the Storage Manager Module (IECVSMGR)

This module's functional recovery procedure, IECVSMFR, puts at byte 6 of the SDUMP buffer record the pool headers for the "small block," "medium block," and "large block" pools. (The section "Storage Manager Module (IECVSMGR)" in the "Program Organization" chapter shows the format of the header for the "small, medium, and large block" pools.)

Following the SDUMP buffer record in the dump are other 4K records written by IECVSMFR. These contain the SQA (system queue area), the system's trace tables, and the code in the IECVSMGR module.

The output of the system's queue verification routine is in the variable area of the SDWA. IECVSMFR passes the variable area to that routine for use as a QVOD (queue verification output data area). The free queue for small, medium, and large blocks is moved to SDWA.

Output of the Unconditional Reserve Detection Module (IECVURDT)

This module's functional recovery procedure does not use the SDUMP buffer. However, the following is placed in the variable area of the SDWA.

At byte 0: The 24-byte FRR parameters.

Output of the Unconditional Reserve Service Module (IECVURSV)

This module's functional recovery procedure does not use the SDUMP buffer. However, the following is placed in the variable area of the SDWA.

At byte 0: The 24-byte FRR parameters.

Informative IOSB Fields

An examination of three IOSB fields, IOSDRVID, IOSPROC, and IOSCOD, answers these questions:

1. Did IOS create the IOSB, or did one of its drivers create it? If one of the drivers, which one?
2. If IOS created the IOSB, why did it?
3. If a driver created the IOSB, what does the IOSB show about the status of the I/O request it represents?

The IOSDRVID field answers (1), the IOSPROC field answers (2), and the IOSCOD field answers (3).

The IOSDRVID Field

IOSDRVID is a one-byte field at an offset of four bytes into the IOSB. The possible contents and their meanings are:

Contents	Meaning
X'00'	IOS created the IOSB.
X'01'	The driver wants to be anonymous to IOS because it doesn't want to take part in certain kinds of I/O processing. (For example, the driver might not want to be called to dispose of the IOSB during a purge operation.)
X'02'	EXCP is the driver.
X'03'	ABP (VSAM) is the driver.
X'04'	VTAM is the driver.
X'05'	TCAM is the driver.
X'06'	OLTEP is the driver.
X'07'	Program fetch is the driver.

X'08'	JES3 is the driver.
X'09'	MSC is the driver.
X'0A'	IECVIOPM is the driver.
X'0B'	VPSS is the driver.

The IOSPROC Field

IOSPROC is a one-byte field at an offset of three bytes into the IOSB. The field is used as an index to a branch table in the post status module (IECVPST). The possible contents and what they tell about the IOSB are:

Contents	What They Tell about the IOSB**
X'00'	Indicates "normal" non-IOS generated IOSB.
X'04'	The IOSB was created by EDIEINT1 when a PCI interruption occurred without other status information. It was marked X'04' to direct PSTIOSB to enter the driver's PCI appendage.
X'08'	The IOSB was created by EATTENT1 when tests of the CSW, UCB, and attention table indicated that control should be routed to an attention routine. The IOSB was marked X'08' to direct PSTIOSB to branch to the attention routine for the device.
X'0C'	The IOSB was created by LCHPURG to replace a purged IOSB for an I/O operation that must be completed–a sense, reserve, or release operation. After the I/O operation is completed, PSTIOSB sees the X'0C' and enters FREEBLK, which frees the IOSB.
X'10'	The IOSB was created by EDEVEND1 when called by IECVXDAS because a direct-access device was readied. It was marked X'10' to direct PSTIOSB to enter IECVDAVV via the exit effectors and ERP loader.
X'14'	The IOSB was created by EPOSTIO1 when it determined that a message must be sent to the operator about the availability of the device. The IOSB was marked X'14' to direct PSTIOSB to enter, via the exit effectors and ERP loader, the ERP message writer (IGE0025C).***
X'20'	The IOSB was created by EDETECT1 when it determined that unconditional reserve recovery was needed. It was marked X'20' to direct PSTIOSB to enter IECVURDT and IECVDURP.

** To learn more about a procedure associated with a given IOSPROC value, refer to the description of the procedure in the "Program Organization" chapter. (Use the procedure's symbolic name to find a page reference to its description in the "Directory.")

*** The ERP loader and ERP message writer are described in "Appendix" under "ERP Service Modules."

The IOSCOD Field

IOSCOD is a one-byte field at an offset of five bytes into the IOSB. The possible contents, with explanations of what they mean are:

Contents	Explanation*
X'41'	An ERP, the ABE appendage, or the CHE appendage detected an I/O error and determined that it was uncorrectable. (An ERP does not put X'41' in IOSCOD. IGC015 does it if, on receiving control from the ERP, it finds the "exceptional-condition" bit (IOSEX) on, the "retry" bit (IOSERR) off, and X'7F' in IOSCOD.)
X'42'	The EOE appendage detected an extent error and directed the driver to put X'42' in IOSCOD.
X'43'	A paging I/O operation couldn't be started immediately, and the IOSB specified that I/O-request processing be terminated in such a case. ETCH1 complied by putting X'43' in IOSCOD and scheduling IECVPST. (ABP, on finding X'43', submits a new I/O request to read a duplicate page on another device.)
X'44'	ETCH1 stopped processing the I/O request because its SRB and IOSB were needed to process a hardware error on the device allocated to the request. (ETCH1 does not put X'44' in IOSCOD. IGC015 does it if, on receiving control from the ERP, it finds the "exceptional-condition" bit (IOSEX) on, the "retry" bit (IOSERR) off, and X'7E' in IOSCOD.)
X'45'	I/O-request processing was terminated abnormally. Reasons for the termination are: • The IECIOSCN or IECVPST module took a program check. • The operator pressed the RESTART key while an I/O request was being processed. • A program check occurred while a nonresident ERP or ERP service module was in control. • A program check occurred in the NRM/ABN exit processing of module, IECVEXCP. IECFRR, PSTFRRTY, or the ERP loader's ESTAE procedure (ERPLESTA) was entered by RTM.
X'48'	The I/O request was purged. The driver's purge procedure put X'48' in IOSCOD.
X'4B'	An I/O error occurred when the tape ERP requested that a volume be repositioned. The ERP put X'4B' in IOSCOD.
X'4C'	In asking for an I/O operation on a specific 2305 exposure, the driver specified an invalid exposure number in the IOSB. IECVXDRS puts X'4C' in IOSCOD and scheduled IECVPST.
X'4D'	The driver guaranteed the availability of a path to the device, but when the start-I/O instruction was issued, the condition code was set to 3 (channel or device not operational). EPOSTIO1 put X'4D' in IOSCOD and scheduled IECVPST.

* To learn more about a procedure associated with a given IOSCOD value, refer to the description of the procedure in the "Program Organization" chapter. (Use a procedure's symbolic name to find a page reference to its description in the "Directory.")

Contents	Explanation*
X'4E'	One of the following occurred: • The driver guaranteed the availability of a path to the device, but the device was reserved to another path. • The driver asked IOS to release a device, and in trying, IOS found that at least one other user of the device wanted it to be reserved. A device dependent SIO module (IECVXDAS, IECVXDRS, IECVXSKS, or IECVXVRS), put X'4E' in IOSCOD and scheduled IECVPST.
X'4F'	The driver guaranteed the availability of a path to the device, but ETCH1 found that the channel set on that path was not configured. ETCH1 puts X'4F' in IOSCOD and schedules IECVPST.
X'51'	ETCH1 determined that the device has been boxed and placed offline, and scheduled IECVPST. (ETCH1 does not put X'51' in IOSCOD if the driver is EXCP. The ERP is eventually given control, and if it enters IGC015 with the "exceptional-condition" bit (IOSEX) on, the "retry" bit (IOSERR) off, and X'74' in IOSCOD, IGC015 overlays X'74' with X'51'.)
X'71'	Set by the direct-access ERP when the sense bytes show a data check and the IOSDRVID field shows that the driver is program fetch.
X'74'	Set by ETCH1 when it determined that a device was boxed and placed offline, and the request was from EXCP. (X'74' may be altered by IGC015. See the explanation for X'51'.)
X'7E'	Set by ETCH1 when it determined that the SRB and IOSB for the request were needed to process a hardware error on the device allocated to the request. (X'7E' may be altered by IGC015. See the explanation for X'44'.)
X'7F'	Set the IECHNSCH before the I/O operation was started. If the IOSB has been returned to the driver's termination procedure, X'7F' signifies that the I/O operation completed successfully.

* To learn more about a procedure associated with a given IOSCOD value, refer to the description of the procedure in the "Program Organization" chapter. (Use a procedure's symbolic name to find a page reference to its description in the "Directory.")

Table of Messages

The table below gives the numbers of IOS and ERP messages, identifies the IOS procedures that detect a need for the message, and indicates the non-IOS module that issues it. To understand what an IOS procedure detects, refer to its description in the "Program Organization" chapter. (Use the procedure's symbolic name to find a page reference to its description in the "Directory.")

Message	Issued by	Procedure Detecting Need for Message
IEA000A	IGE0025C*	EPOSTIO1 DAVERR or an ERP**
IEA000I	IGE0025C	an ERP
IEA001I	IGE0025C	EPOSTIO1
IEA003I	IEAVTRET	CLEARDEV***
IEA004I	IEAVTRET	ACRPROC***
IEA004I	IEAVTRET	UCBACT***
IEA004I	IEAVTRET	LOSTCHAN***
IEA004I	IEAVTRET	IECVRDIO
IEA026I	IEAVTRET	IECVRRSV
IEA066A	IEEVLDWT	IECVHREC
IEA067A	IEEVLDWT	IECVHREC
IEA068A	IEEVLDWT	IECVHREC
IEA069A	IEEVLDWT	IECVHREC
IEA070A	IEEVLDWT	IECVHREC
IEA071E	IEEVLDWT	IECVHREC
IEA072I	IEEVLDWT	IECVHREC
IEA073A	IEEVLDWT	IECVPST
IEA151W	IGFPTERM	IECVRRSV
IEA151W	IEEVLDWT	IECVRSTI
IEA151W	IEEVLDWT	IECVHREC
IEA410E	IEAVTRET	LOSTCHAN***
IEA410E	IEAVTRET	IECVIRST
IEA421E	IEAVTRET	IECVIRST
IEA427A	IECVDURP	IECVDURP
IEA428I	IECVDURP	IECVDURP
IEA429I	IECVDURP	IECVDURP
IEA438A	IEEVLDWT	IECVIRST
IEA439D	IEEVLDWT	IECVIRST
IEA440A	IEEVLDWT	IECVRSTI
IEA604A	IECVDAVV	DAVINT
IEA605A	IECVDAVV	DAVINT
IEA606I	IECVDAVV	DAVINT
IEA970I	IEAVTRET	IECVCRHA
IEA971I	IEAVTRET	IECVCRHA
IEA972I	IEAVTRET	IECVCRHD
IEA989I	IEAVTRET	IECVCRHA

* This is the ERP message writer; it's described in "Appendix B" under "ERP Service Modules."

** IEA000A is issued only if an ERP module finds the "intervention-required" bit on in the sense bytes. ERP modules and the devices they support are listed in "Appendix B" under "Table of ERP Modules."

*** The message is formatted by another IOS procedure, RECORDIT. It gives control to IEAVTRER, which schedules IEAVTRET to write the message asynchronously.

Wait-State Codes

(See the Directory for the page number of the PO description of each procedure.)

Code	Issuing Procedure
X'022'	DAVESTA (ESTAE Recovery)
X'02F'	PSTWAIT
X'041'	ACRPROC (ACR Call Procedure)
X'04C'	IECVIRST
X'04D'	IECVIRST
X'04E'	IECVIRST, IECVHREC, IECVRRSV, IECVRSTI
X'066'	IECVHREC
X'067'	IECVHREC
X'068'	IECVHREC
X'069'	IECVHREC
X'06A'	IECVHREC
X'06B'	IECVCINT
X'06F'	IECVDURP

Table of IOS Return Codes

The table below matches a return code with the symbolic names of IOS procedures that exit with the code in register 15. To find out what a return code means when used by a given procedure, refer to the description of that procedure in the "Program Organization" chapter. (Use the procedure's symbolic name to find a page reference to its description in the "Directory.")

Return Code	Procedure Name
X'00'	HIOCCH
	IECIHIO
	IGC0001G
	IGC0003C
	IGC016
	SMGFREVR
	TCCWR000
	IECVCINT
	IECCONCS
	IECVESIO
	IECVRRSV
	IECVURSV
X'04'	HIOCCH
	IGC0001G
	IGC0003C
	IGC016
	SMGFREVR
	TCCWR000
	TCCWU000
	TCCWX000
	IECVCINT
	IECCONCS
	IECVESIO
	IECVRRSV
	IECVURSV
X'08'	HIOCCH
	HIOLOP
	IECIHIO
	IGC0003C
	IGC016
	SMGFREVR
	TCCWR000
	IECVCINT
	IECVESIO
X'0C'	IGC0003C
	TCCWM000
	TCCWM100
	TCCWM300
	TCCWM400
	IECVESIO
X'10'	IGC0003C
	IECVESIO
X'14'	IGC0003C
	IGC016
	IECVESIO
X'18'	IGC0003C
	IECVESIO
X'20'	IGC0003C
	IECVESIO

Return Code	Procedure Name
X'24'	IECVESIO
X'28'	IECVESIO
X'2C'	IECVESIO
X'30'	IECVESIO
X'34'	IECVESIO
X'38'	IECVESIO
X'3C'	IECVESIO

Appendix: Overview of I/O Error Recovery Processing

The Function and Characteristics of ERPs

An ERP (error recovery procedure) is a program that evaluates sense data, CSW status bits, and data in the ERP work area, and takes appropriate actions. It can be specialized to process the sense information of only one type of device, as is the ERP for the 2540 Card Read Punch, or it can be able to process the sense data of a "family" of devices, as can the ERP for tape devices or the ERP for direct-access devices.

Only one ERP, the direct-access ERP, resides in the nucleus; it is contained in one load module. Non-direct-access ERPs are paged in and out of the link pack area, and some of these are contained in two or more load modules. The first load is given control by the *ERP loader* (IECVERPL) and may, based on feedback from the *error interpreter* (IECVITRP), give control to another load. (The ERP loader and error interpreter are described under "ERP Service Modules.")

Most ERPs write messages to the operator's console by calling the *ERP message writer* (IGE0025C) via a procedure in the ERP loader. All ERPs write records in the SYS1.LOGREC data set by calling *OBR,* the outboard record routine (IGE0025F). All but the tape and direct-access ERPs update an error statistics table by calling the *error statistics recorder* (IGE0025D), this too via the ERP loader. (The ERP message writer and error statistics recorder are described under "ERP Service Modules." OBR is described in *OS/VS2 SYS1.LOGREC Error Recording Logic,* SY28-0678.)

Table of ERP Modules

The following table matches the module or pair of modules that makes up an ERP with the device or devices the ERP supports:

Modules	Devices
IECVDERP	2305 Fixed Head Storage 2314 Direct-Access Storage Facility 2319 Disk Storage 3330 Disk Storage 3330-1 Disk Storage 3340 Direct Access Storage Facility 3350 Direct Access Storage Facility
IGE0000D	1052 Printer 3210 Console Typewriter 3215 Console Typewriter
IGE0000G (first load)	3211 Printer, 1403 Printer
IGE0000H	3851 Mass Storage System Data Staging Manager (DSM)
IGE0000I (first load) IGE0100I (second load)	2400 Series Magnetic Tape Units 3400 Series Magnetic Tape Units
IGE0001A	3505 Card Reader 3525 Card Punch
IGE0001C (first load)	1442 Card Read Punch 2501 Card Reader 2520 Card Read Punch 2540 Card Read Punch
IGE0001F (first load) IGE0101F (second load)	3886 Optical Character Reader
IGE0001G	3890 Document Processor
IGE0001H	3851 Mass Storage Controller (MSC)
IGE0002A	2671 Paper Tape Reader
IGE0002B	3895 Document/Inscriber
IGE0002C	3540 Diskette Input/Output Unit
IGE0002E	Channel-to-Channel Adapter
IGE0003C	3838 Array Processor
IGE0004 }	3704 and 3705 Communications Controllers
IGE0010A	2250 Display Unit
IGE0010B	1053 Printer 2260 Display Station
IGE0010D	3066 Display Console
IGE0010E	3270 Information Display System
IGE0011A	2459 Tape Cartridge Reader
IGE0011C	1287 Optical Reader
IGE0011D	1288 Optical Page Reader
IGE0011E	1275 Optical Reader Sorter 1419 Magnetic Character Reader

ERP Service Modules

The ERP Loader (IECVERPL)

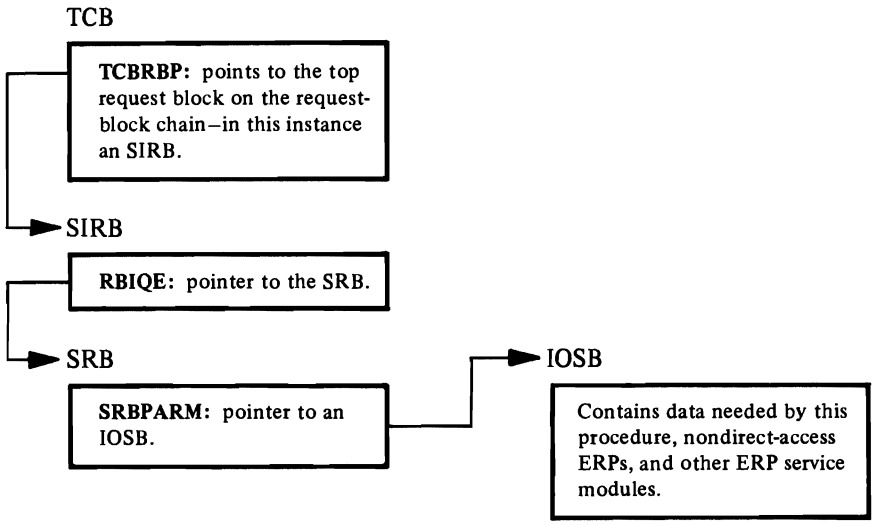
The ERP loader consists of three parts:

- A *routing procedure*, which is dispatched as the result of processing requested by the IOS post-status module (IECVPST) and carried out by the stage 2 and stage 3 exit effectors. In causing the routing procedure to be dispatched, IOS ensures that either a nondirect-access ERP, the IOS DAVV module, the ERP message writer, or DDR—whichever the processing environment calls for—eventually gets control.
- A *module location procedure*, which is used by nondirect-access ERPs and ERP service modules to get to other modules.
- An *error notification procedure*, which calls the IOS post-status module if the routing or module location procedure detects an error.

The following topics describe how the procedures are called, what they do, and how they return to their caller.

The Routing Procedure

When dispatched, the routing procedure locates the IOSB at the end of this chain:



The routing procedure puts the address of the IOSB in register 1, where nondirect-access ERPs and ERP service modules except it, and examines IOSPROC, the IOSB field that determines the remainder of the routing procedure's processing. Listed in the following table are the possible contents of the IOSPROC field and the corresponding actions taken by the routing procedure:

IOSPROC Contents	Actions
X'00'	Loads register 13 with the contents of the UCBETI field and exits to the module location procedure at ERPL3ENT.
X'10'	Exits to the IOS DAVV module.
X'14'	Puts the characters *IGE0025C*, the name of the ERP message writer, into the SIRB and exits to the module location procedure at a point following ERPL3ENT.
X'18'	Puts the characters *IGE0660A*, the name of the DDR module, into the SIRB and exits to the module location procedure at a point following ERPL3ENT.
Anything else	Exits to the error notification procedure.

The Module Location Procedure

The module location procedure is entered, with register 13 containing the hexadecimal identifier of the module to be located. Possible modules are:

- Load modules of nondirect-access ERPs wanting to give control to other load modules or to ERP service modules.
- ERP service modules wanting to give control to other ERP service modules.
- The ERP loader's routing procedure, which enters on behalf of IOS to ensure that a nondirect-access ERP is given control.

ERP and ERP service modules enter at IECXTLER. The routing procedure enters at ERPL3ENT.

The identifier in register 13 is converted to four characters, the prefix *IGE0* is added to them, and the 8-character module named thus formed is stored in the SIRB. (The routing procedure enters at this point instead of at ERPL3ENT if IOS wants IGE0025C, the ERP message writer, or IGE0660A, the DDR module, to be entered. The routing procedure will have already put the appropriate module name in the SIRB.)

Control is passed to routines that search the system's contents directory entries and link pack directory entries for the address of the module named in the SIRB. If the module can be located, it is entered. If not, the error notification procedure is entered.

The Error Notification Procedure

The error notification procedure is entered at ERPLT2 by:

- The routing procedure if the IOSPROC field contains an invalid value.
- The module location procedure if the module it was asked to locate can't be found in the system's directory entries.

The error notification procedure tells IOS about the error by turning on the "exceptional-condition" bit, IOSEX, turning off the "retry" bit, IOSERR, and calling the IOS post-status module (IECVPST) with an SVC 15 instruction. (To learn what the post-status module does when it gets control, see the description of the SVC 15 procedure in "Post-Status Module (IECVPST).")

The error notification procedure exits to the dispatcher with an SVC 3 instruction when control returns.

ERP Loader Module (IECVERPL) Detailed Processing

The ERP Loader (IECVERPL)

Invoked by the dispatcher at entry point IECVERPL, or by an ERP by means of an XCTL macro at entry point IECXTLER.

(a) IECVERPL: Dispatcher entry

- Specifies an ESTAE exit.
- Checks the IOSPROC field in the IOSB (which was set by the driver), to determine which routine is to get control, as follows:
  - (1) If IOSPROC = 00 (driver created the IOSB), uses the error index table (UCBETI) in the UCB extension to find the desired routine.
  - (2) If IOSPROC = IOSDAVV (X'10'), goes to module IECVDAVV to verify the volume.
  - (3) If IOSPROC = IOSAWTO (X'14'), goes to the ERP XCTL entry of the ERP loader (IECXTLER) to cause entry to the ERP message writer (IGE0025C).
  - (4) If IOSPROC = IOSADDR (X'18'), goes to DDR routine to evaluate the results of an I/O retry.
  - (5) If none of these IOSPROC conditions is indicated, exits to the *post status module (IECVPST) SVC 15 procedure* (3) because this entry is for an unauthorized request. The *SVC 15 procedure* terminates the associated I/O request, then issues an SVC 3.

(b) IECXTLER: ERP XCTL entry

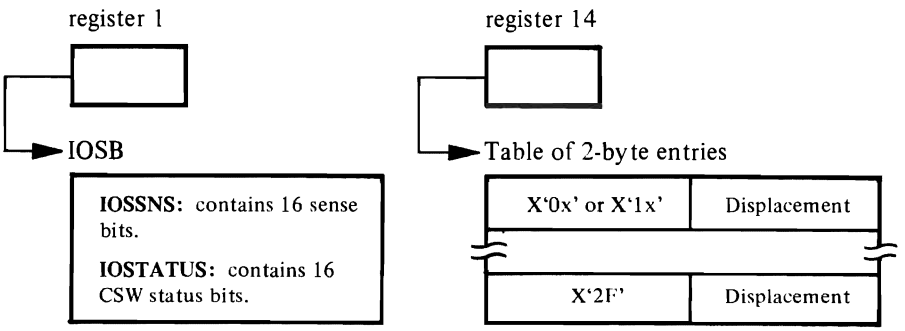
- Entered by an ERP to give control to another ERP load module or to use a service routine, such as the ERP message writer or the error statistics recorder.
- Uses the number in register 13 to build an IGE0-prefixed name to use in a search of the CDE queues and the link pack directory queues (LPDEs) to find the entry point name of the desired load module.
- Gets the local and CMS locks to prevent alteration of the queues during the search.
- Searches the CDE queues for the entry point name. If the name is not found, frees the local and CMS locks and searches the link pack directory for the entry point name, since the module may be in the PLPA.
- If the return code is 4, the entry point name does not exist. In this case, exits to the *post status module (IECVPST) SVC 15 procedure* (3) to terminate the I/O request (post, indicating "IOSB in error").
- If the entry point name is found in the CDEs, frees the locks and branches to the entry point.
- If the entry point name is found in the link pack directory, branches to the entry point.

(c) IECVERPL: ESTAE procedure entry

- Invokes SVC dump.
- Branches to the RTM, specifying retry in module (IECVPST) *SVC 15 procedure* (3). The *SVC 15 procedure* terminates the I/O request with an "IOSB in error" condition.

The Error Interpreter (IECVITRP)

The error interpreter is entered by both the direct-access ERP and the first load of nondirect-access ERPs to direct the course of error recovery processing. It receives this input:



If the first byte in a table entry is X'0x', one of the sixteen sense bits is tested. The specific bit tested depends on the low-order value of the byte. If the first byte is X'1x', one of the sixteen CSW status bits is tested. Should IECVITRP find a sense or CSW status bit on, it adds the displacement in the second byte of

the appropriate entry to register 14 and branches to that address. If IECVITRP reaches the end of the table—indicated by X'2F' in the first byte of an entry—it uses the displacement in that entry.

The error interpreter, having found an error, is not reentered by the ERP to finish testing sense and CSW bits. Thus, ERPs only deal with one error per iteration of an I/O operation.

The ERP Message Writer (IGE0025C)

The ERP message writer actually consists of IGE0025C, and a message exit. IGE0025C does message determination and builds the general part of the message. The device-dependent message exits are called to build any device dependent sections of the message, e.g. sense. If an exit is not available, IGE0025C issues the general message that it has built. Otherwise, the exit issues the WTO for the complete message, then returns to the message writer for cleanup and exit.

There are six message exits. They receive control from IGE0025C with pointers to the IOSB and a message buffer. Device-dependent information is formatted and put into the buffer and the WTO is issued. Control is then returned to IGE0025C. The message exits are:

IECLMSGD – the exit for DASD devices
IECLMSGC – the exit for communication devices
IECLMSGG – the exit for graphics devices
IECLMSGM – the exit for the 3851 device
IECLMSGT – the exit for tape devices
IECLMSGU – the exit for unit record devices

The following table shows what the ERP message writer might find and what message it writes:

Findings	Message Issued
A start-I/O instruction set the condition code to 3, and there is no other path to the device.	IEA000A, "intervention required"
A start-I/O instruction set the condition code to 3, but there is another path to the device.	IEA001I, "path inoperative"
A unit check was received with an "intervention-required" indication in the sense bytes.	IEA000A, "intervention required," unless the message has already been sent (the UCBIVRS bit will be on)
A unit check was received for a 3851 with an "intervention-required" indication in the sense bytes, and sense byte 3, bit 1 was on.	IEA000I, "intervention required," unless the message has already been sent (the UCBIVRS bit will be on).
The ERP determined that an uncorrectable I/O error occurred.	IEA000I, "permanent I/O error"

Differences in the handling of IOSBs are shown in this table:

TESTS	Is the IOSB a deferred IOSB?	Yes	Yes	No	No
	Does the ERP want data recorded in the SYS1.LOGREC data set? (Is the IOSLOG bit on?)	Yes	No	Yes	No
ACTIONS	Puts the IOSB on a queue that OBR looks at.	X			
	Calls the IOS storage manager module (IECVSMGR) to free the IOSB.		X		
	Returns to the ERP loader's module location procedure, first storing in register 13 a value that will cause the procedure to enter either the error statistics recorder or OBR.*			X	
	Issues an SVC 15 instruction, calling the IOS post-status module, and when control returns, exits to the dispatcher with an SVC 3 instruction.**				X

* The value stored in register 13 is governed by the contents of EWASTUP, the "statistics-update" field of the ERP work area.

** To learn what the post-status module does when it's called, see the description of the SVC 15 procedure in "Post-Status Module (IECVPST)."

Note: The ERP message writer does not issue messages if the nucleus is being initialized, but it performs the processing that is directed by the setting of the IOSLOG bit.

Index

OS/VS2 I/O Supervisor Logic
SY26-3823-5

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.***

Possible topics for comments are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If comments apply to a Selectable Unit, please provide the name of the Selectable Unit _____.

If you wish a reply, give your name and mailing address:

_____

_____

_____

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form. — Cut or Fold Along Line —

Please circle the description that most closely describes your occupation.

Customer	(Q) Install Mgr.	(U) System Consult.	(X) System Analyst	(Y) System Prog.	(Z) Applica. Prog.	(F) System Oper.	(I) I/O Oper.	(L) Term. Oper.	(O) Other

IBM	(S) System Eng.	(P) Prog. Sys. Rep.	(A) System Analyst	(B) System Prog.	(C) Applica. Prog.	(D) Dev. Prog.	(R) Comp. Prog.	(G) System Oper.	(J) I/O Oper.	(E) Ed. Dev. Rep.	(N) Cust. Eng.	(T) Tech. Staff Rep.

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.)

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

OS/VS2 I/O Supervisor Logic
SY26-3823-5

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.***

Possible topics for comments are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If comments apply to a Selectable Unit, please provide the name of the Selectable Unit _____.

If you wish a reply, give your name and mailing address:

_____

_____

_____

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

Cut or Fold Along Line

Please circle the description that most closely describes your occupation.

Customer	(Q) Install Mgr.	(U) System Consult.	(X) System Analyst	(Y) System Prog.	(Z) Applica. Prog.	(F) System Oper.	(I) I/O Oper.	(L) Term. Oper.	(O) Other

IBM	(S) System Eng.	(P) Prog. Sys. Rep.	(A) System Analyst	(B) System Prog.	(C) Applica. Prog.	(D) Dev. Prog.	(R) Comp. Prog.	(G) System Oper.	(J) I/O Oper.	(E) Ed. Dev. Rep.	(N) Cust. Eng.	(T) Tech. Staff Rep.

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.)

SY26-3823-5

Reader's Comment Form

Cut or Fold Along Line

Fold and tape | Please Do Not Staple | Fold and tape

First Class
Permit 40
Armonk
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department D58, Building 706-2
PO Box 390
Poughkeepsie, New York 12602

Fold and tape | Please Do Not Staple | Fold and tape

OS/VS2 I/O Supervisor Logic (S370-30) Printed in U.S.A. SY26-3823-5

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

Technical Newsletter

This Newsletter No.	SN28-4683
Date	October 25, 1979
Supplement No.	
Base Publication No.	SY26-3823-5
File No.	S370-30
Prerequisite Newsletters/ Supplements	None

OS/VS2 I/O Supervisor Logic

This newsletter contains replacement pages for *OS/VS2 I/O Supervisor Logic.*

Before inserting any of the attached pages into *OS/VS2 I/O Supervisor Logic*, read ***carefully*** the instructions on this cover. They indicate when and how you should insert the pages.

Pages to be Removed	Attached Pages to be Inserted*
Cover - Edition Notice	Cover - Edition Notice
xi - xiv	xi - xiv
19 - 24	19 - 24
29 - 32	29 - 32
37 - 38	37 - 38
41 - 42	41 - 42
67 - 68	67 - 68
101 - 102	101 - 102
105 - 106	105 - 106
185 - 200	185 - 200
I-1 - I-9	I-1 - I-12

*If you are inserting pages from different Newsletters/Supplements and ***identical*** page numbers are involved, always use the page with the latest date (shown in the slug at the top of the page). The page with the latest date contains the most complete information.

A change to the text or to an illustration is indicated by a vertical line to the left of the change.

IBM Corporation, Publications Development, Department D58, Building 706-2, PO Box 390, Poughkeepsie, New York 12602

Printed in U.S.A.

Summary of Amendments

- Changes have been made throughout this publication in support of the 3033 attached processor.
- Diagnostic aids information has been deleted from this publication. It can now be found in the following books: *OS/VS2 System Programming Library: MVS Diagnostic Techniques, OS/VS Message Library: VS2 System Messages,* and *OS/VS Message Library: VS2 System Codes.*
- A considerably expanded index has been included.
- Minor technical and editorial corrections and additions have been made.

Note: *Please file this cover letter at the back of the base publication to provide a record of changes.*