SY26-3859-0
File No. S370-30

Systems

OS/VS2 VIO Logic

VS2.03.807

IBM

The minor revision incorporates the OS/VS2 MVS Supervisor Performance #2 Selectable Unit VS2.03.808.

First Edition (August 1976)

This is a reprint of SY26-3834-1 incorporating changes released in the following Technical Newsletters and Selectable Unit Newsletters:

SN26-0798 (dated July 28, 1975)
SN26-0838 (dated May 28, 1976)

This edition applies to Release 3.7 of OS/VS2 and to any subsequent releases of that system unless otherwise indicated in new editions or technical newsletters.

Significant system changes are summarized under "Summary of Amendments" following the list of illustrations.

Information in this publication is subject to significant change. Any such changes will be published in new editions or technical newsletters. Before using the publication, consult the latest *IBM System/370 Bibliography*, GC20-0001, and the technical newsletters that amend the bibliography, to learn which editions and technical newsletters are applicable and current.

Requests for copies of IBM publications should be made to the IBM branch office that serves you.

Forms for readers' comments are provided at the back of the publication. If the forms have been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California 95150. All comments and suggestions become the property of IBM.

PREFACE

The information contained in this book is intended to enable programming systems representatives and system programmers to acquire a general understanding of VIO (virtual input/output) processing and to diagnose logic-level problems by identifying segments of code that are executed under given sets of circumstances. It is assumed that the audience is thoroughly familiar with channel programming, paging I/O, and in general with the OS/VS2 virtual storage environment.

This book is designed to be used in conjunction with assembled or compiled VIO processor source code. Detailed logic, as represented by the code, is only generally reflected in this book.

The manual is divided into six sections.

- The "Introduction" section describes, in general terms, VIO functions, the subcomponents that comprise VIO, and the VIO operating environment.
- The "Method of Operation" section describes the operations (or process steps) that combine to perform a VIO function.

  Diagrams are used to relate input and output of each function to the process steps. Notes to each diagram describe the processes in more detail and provide a cross-reference to the segments of code associated with the processes.

  The diagrams are tutorial; they give you an overall view of VIO processing; they are diagnostic aids only to the extent that they enable you to refresh your understanding of the broader implications of a segment of VIO logic.
- The "Program Organization" section shows the interrelationships of modules that combine to perform a VIO function.

  The sequential flow charts in this section enable you to determine, at any point in the processing of a function, which modules have been entered and which modules will be entered.

  A table is also provided to show all the callers of each VIO module and all the modules called by each VIO module.
- The "Directory" section relates VIO modules, VIO external procedures, non-VIO modules invoked by VIO, and non-VIO modules that invoke VIO routines to diagrams within the book that refer to them. This section also includes the "Macro Where-Issued Report" that lists the action and mapping macros that are issued by individual VIO modules.
- The "Data Areas" section describes the fields and depicts the format of each VIO data area. Each VIO data area has a table that shows you the modules that modify fields in the individual blocks.

  An overview chart shows you how non-VIO data areas relate to the VIO data area structure.
- The "Diagnostic Aids" section describes the ABEND completion codes issued by VIO modules, the information written by the VIO recovery routines to the SYS1.DUMP data set, and information moved to the STAE diagnostic work area (SDWA) for the recovery/termination manager (R/TM) to write to the SYS1.LOGREC data set. This and other information in this section may assist you in isolating a module whose processing detects an error condition.

Prerequisite Reading

For information about OS/VS2:

- *OS/VS2 Planning Guide for Release 2,* GC28-0667.

For information about job management, task management, real storage management (RSM), virtual storage management (VSM), and recovery/termination management (R/TM) and about their support of VIO processing:

- *OS/VS2 System Logic Library,* SY28-0713 (Volume 1), SY28-0714 (Volume 2), SY28-0715 (Volume 3), SY28-0716 (Volume 4), SY28-0717 (Volume 5), SY28-0718 (Volume 6), and SY28-0719 (Volume 7).

For information about I/O appendages:

- *OS/VS2 System Programming Library: Data Management,* GC26-3830.

For information about the EXCP processor's interface with VIO:

- *OS/VS2 I/O Supervisor Logic,* SY26-3823.

Related Reading

For information about data management routines that interface with VIO:

- *OS/VS2 DADSM Logic,* SY26-3828.
- *OS/VS2 Open/Close/EOV Logic,* SY26-3827.
- *OS/VS2 Checkpoint/Restart Logic,* SY26-3820.

For information about access methods supported by VIO:

- *OS/VS2 BDAM Logic,* SY26-3831.
- *OS/VS2 SAM Logic,* SY26-3832.

For VIO performance estimates and information on how to optimize VIO processing:

- *OS/VS2 System Programming Library: Initialization and Tuning Guide,* GC28-0681.

For information about SMF:

- *OS/VS2 System Programming Library: System Management Facilities (SMF),* GC28-0706.

For data area layouts of intercomponent control blocks:

- *OS/VS2 Data Areas,* SYB8-0606.

For information about the SYS1.DUMP data set:

- *OS/VS2 System Programming Library: Service Aids,* GC28-0674.

For information about the SYS1.LOGREC data set:

- *OS/VS2 System Programming Library: SYS1.LOGREC Error Recording,* GC28-0677.

For extended descriptions of VIO error completion codes and programmer responses:

- *OS/VS Message Library: VS2 System Codes,* GC38-1008.

For information about messages issued by VIO:
- *OS/VS Message Library: VS2 System Messages,* GC38-1002.

For information about channel commands and about sense-byte information posted as a result of normal or abnormal termination of a channel program, see the appropriate device manual for the device that is simulated by VIO in external page storage.

CONTENTS

ILLUSTRATIONS

Figures

Diagrams

SUMMARY OF AMENDMENTS

OS/VS2 MVS Supervisor Performance #2 (VS2.03.807)

Technical Changes

- Module IDAVBPJ1 has been deleted.
- The DSPCT Extension has been added which contains the VCBs (VIO Control Blocks).
- The LGCB (Logical Group Control Block) has been deleted.
- The ACA (ASM [Auxiliary Storage Management] Control Area) has been added.
- The figure VIO Virtual Buffer has been added.
- The JNLPARM (Journal Write Parameter List) has been added.

Release 3

Two new fields have been added to the DSPCT data area-VBPHDEL and VBPHJRNP.

INTRODUCTION

The VIO Processor's Purpose

The VIO (virtual input/output) processor enables system-named temporary data sets to reside in *external page storage* (also referred to as *system paging space*) and to be processed using paging I/O. Conventionally, in a non-VIO environment, temporary data sets are processed using user-controlled I/O blocking factors and user-defined auxiliary storage volumes. To support paging I/O, VIO's blocking scheme is necessarily based on 4K-byte pages. This is transparent to the user and does not affect the user's blocking scheme.

I/O requests issued against a VIO temporary data set by conventional access methods (BSAM, QSAM, BDAM, BPAM) or by XDAP or EXCP macro instructions are intercepted before they are executed. VIO interprets and simulates the channel programs associated with the requests; the channel programs are not translated and scheduled in a conventional fashion by the EXCP processor. (See *OS/VS2 I/O Supervisor Logic* for details about the EXCP processor, module IECVEXCP.)

VIO offers the following enhancements to conventional processing of system-named temporary data sets:

- VIO data sets are processed using paging I/O, thus eliminating channel program translation and page-fixing by the I/O supervisor for each EXCP request.
- DASD space management is more efficient because ASM (auxiliary storage management) dynamically allocates page-size physical blocks (auxiliary-storage page slots) to a VIO data set as the blocks are needed.
- DADSM allocation and scratch I/O overhead is reduced because a VTOC is not updated and because the format-1 DSCB is maintained in virtual storage in the VIO data set control block (VDSCB).
- Many I/O operations can be satisfied by moving data between the VIO buffer and a user's buffer via move-character-long (MVCL) instructions; this reduces the channel and device activity associated with normal channel-program request processing when the user's buffer size is less than the tracksize of the device being simulated.
- Because of the reclaim function (see "Reclaiming Page Frames" in this section), paging I/O to access frequently referenced pages may be substantially reduced. Because of the system's ability to reclaim pages, a VIO data set being processed on a lightly-loaded system becomes essentially an "in-core" data set.

Because of paging overhead, VIO processing may be less efficient than conventional processing if large block sizes are used. Also, if the block sizes used for the device being simulated are not optimized, the result may be a waste of space on the auxiliary storage or may cause additional paging activity (see "The VIO Data Buffer" in this section).

Prerequisites to VIO

There are certain system generation and JCL prerequisites to VIO processing that must be satisfied by the user as stated in *OS/VS2 JCL,* GC28-0692, and *OS/VS2 System Programming Library: System Generation Reference,* GC26-3792. Briefly, they are as follows:

- At system generation, a unitname must be defined via the UNITNAME macro as eligible for VIO processing. The first eligible device in the group of devices defined by the UNITNAME macro is simulated by VIO in external page storage.
- The unitname defined during system generation must be coded in a user's JCL statement that defines a system-named temporary data set.
- The user must be processing with either the BSAM, QSAM, BDAM, or BPAM access method or with either XDAP or EXCP macro instructions. Indexed access methods are not supported.
- A volume serial number for the VIO data set must not be specified on a user's DD statement.

  Note: If a volume serial number is specified, an attempt is made to mount a real device. If the device is successfully mounted, normal (non-VIO) processing occurs.

Note that the term "user" as used in this manual includes VIO-supported access methods, user problem programs, and system programs, such as the linkage editor or program fetch. A "user" is anyone who uses virtual I/O and meets the previously listed requirements.

Characteristics of VIO Data Sets

Because VIO data sets reside in external page storage, their physical organization is unlike that of conventional temporary data sets.

VIO Data Set Organization

VIO data sets are page-formatted data sets whose pages are scattered across the volumes that comprise external page storage. ASM (auxiliary storage management) dynamically allocates page-sized physical blocks to the VIO data set as page-size blocks of user data are written to auxiliary storage. To increase the efficiency of I/O operations, ASM tries to balance and reduce channel activity among the devices assigned to external page storage by grouping the I/O requests of the current users of the system. Even though all the pages of each VIO buffer are grouped together, the pages in a VIO data set, as a whole, are generally noncontiguous.

VIO data set pages are released by ASM when the VIO data set is deleted, and the paging resources are thereby immediately available for other paging needs.

Device Simulation in External Page Storage

The characteristics of the first device in the list of devices associated with a VIO unit name (see "Prerequisites to VIO") are used by VIO both in CCW processing and in initializing control information in VIO data set tracks. (Logically, a VIO track in auxiliary storage is a contiguous entity; however, as described under "VIO Data Set Organization," the pages of a VIO track in auxiliary storage are not necessarily physically contiguous; the pages in the track are dynamically assigned to auxiliary-storage page slots.) The characteristics of the first device defined by the VIO unit name are used, and bear no relationship with the auxiliary storage devices assigned to external page storage.

I/O Operations in the VIO Environment

VIO interprets channel programs, initiates paging I/O to bring page-size blocks into a VIO buffer to access desired data (as necessary), and simulates execution of the channel program by moving and comparing data between the user's buffer(s) and the VIO buffer.

The VIO Data Buffer

VIO employs a data buffer, called a VIO buffer, whose size is based on the tracksize of the simulated device rounded-up to a page boundary. For example, if a 2314 (tracksize=7294 bytes) is being simulated, the size of the VIO buffer is 8192 bytes, or 2 pages. I/O is simulated by moving data between the VIO buffer and the user's buffer, as shown in Figure 1, and actual I/O is performed by moving pages of data between the VIO buffer and external page storage. Note that the size and position of user buffers are not changed by VIO; VIO merely moves data into and out of the user buffers as directed by channel programs.

The track format is shown in the "Data Areas" section under "VTRACK—VIO Buffer." The amount of data that can be written on a track of a real device and on the simulated device is the same. The interrecord gaps are eliminated and the count, key, and data fields are packed to the left after the control information. Hence, if the last few pages of the track do not contain any data, then they are not paged in and out. If only a fraction of the last few pages have data, then this will necessitate paging I/O for the pages and will cause wasted space on auxiliary storage. See "Optimizing VIO Performance" in this section.

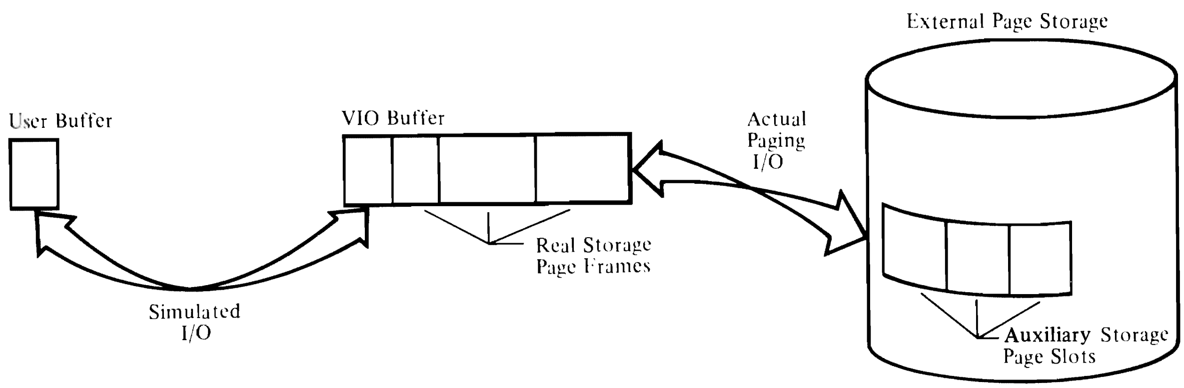


Figure 1. Moving Data in and out of the VIO Buffer

Defining Types of I/O in the VIO Environment

In response to a channel program associated with either an EXCP or XDAP request, VIO may request virtual or actual I/O, or may simulate I/O activity. The following descriptions will assist you in understanding the types of I/O that occur in the VIO environment.

- *Virtual I/O* is performed by RSM (real storage management) in response to requests by VIO's VBP (virtual block processor) subcomponent. It involves the manipulation of entries in page tables that logically connect page slots in auxiliary storage to virtual page frames.
- *Actual I/O,* that is, the physical transfer of data to and from auxiliary storage, involves the building and actual execution of channel programs with associated channel and device activity. Actual I/O is related to VIO processing in the following ways:

  Actual output (page-out) operations are initiated by a call to RSM by VBP. VBP issues the request when the VIO buffer contains new or modified data and the contents of the buffer cannot satisfy a CCW operation being simulated by EIP (EXCP intercept processor), a VIO subcomponent. Note that EIP requests VBP to write out only those pages in the VIO buffer up to and including the page that contains the last byte of data. This prevents empty (unused) pages at the end of the buffer from being written to auxiliary storage during a write operation. When RSM returns control to VBP, the virtual output operation is complete.

  Actual input (page-in) operations occur when an attempt is made to access data in a virtual page that was involved in a prior virtual input operation. (For a description of "virtual input" operations, see the description of the *assign* function under "VBP's Relationship to RSM.") The reference to the virtual page causes a page-fault interruption, and the referenced page is paged into the VIO buffer. Note that when a request is made by EIP to VBP to read a previously written virtual track, VBP requests RSM to read only those pages in the virtual track that were previously written.
- *Simulated I/O* is performed by EIP. It involves the moving of data between the VIO buffer and the user's buffer via a move-character-long (MVCL) instruction to satisfy the operation specified by a channel program.

The situations in which these types of I/O operations occur are further described in the "Method of Operation" section of this manual; however, realizing the general distinctions between the types of I/O is important to an understanding of VIO processing.

Fundamentals of VIO Processing

VIO relies heavily on the services of RSM (real storage management). RSM initiates the I/O operations that are requested by VIO, and it maintains the page tables for virtual, auxiliary, and real storage that make data accessible in a virtual storage environment. RSM maintains the contents of the VIO buffer based on requests issued by VIO. Disregarding RSM's support processing, VIO processing can be generally described as follows:

Create Processing: When a VIO data set is being created, VIO interprets channel programs and simulates I/O activity by moving records from the user's buffer into the VIO buffer. When the VIO buffer is full, its contents are paged-out to auxiliary storage in 4K-byte blocks, or pages. This process repeats itself until create processing is completed.

Update/Retrieval Processing: When an existing record in a VIO data set is being retrieved, VIO determines whether the desired record resides in its buffer, and

- If the record is in the buffer, VIO interprets the channel program and simulates any associated I/O activity by moving records between its buffer and the user's buffer. In a non-VIO environment, channel and device activity would be involved in this operation; whereas, with VIO, the operation is performed in real storage via move-character-long (MVCL) instructions.
- If the record is not in the VIO buffer, VBP releases the current page frames associated with its buffer (for simple retrieval processing) and causes the pages to be written to auxiliary storage (for update processing). It then positions the VIO buffer to the virtual track in auxiliary storage that contains the desired record. Subsequently, when EIP attempts to access the VIO buffer, a page-fault interruption occurs and the appropriate page in the auxiliary-storage page slots is paged into the real storage page frame that comprises the accessed page of the VIO buffer.

An important consideration related to update/retrieval processing is VIO's ability to *reclaim* page frames.

Reclaiming Page Frames

VIO maintains the RSNs (real storage numbers) of page frames that have been paged out of the VIO buffer in DSPCT (data set page correspondence table) page-map entries. When a request is made to read a previously written page, the page's identifier and its RSN are passed to RSM. If the PFTE (page frame table entry) corresponding to the RSN indicates that the page frame has not been reused (that is, that it still contains the desired data), RSM reconnects the page to the VIO buffer by modifying the PFTE. In this way, a page fault and resultant I/O is avoided.

Accessing Virtual Pages in a VIO Data Set

The access method or user's problem program that issues either an EXCP or XDAP request supplies a DASD seek address for the track on which I/O processing is to start. VIO extracts an RBA (relative byte address) from the seek address and then converts the RBA into an RPN (relative page number) using the following equation:

RPN = RBA/4096 + number of pages of DSPCT page-map entries

An RPN is established for each page in the VIO buffer, the values are set in VCBs (VIO control blocks), and the VCBs for a particular I/O operation are then chained together. (Note that the first track of an extent reduces to an RBA of zero and that VIO data sets with standard labels have one extent whose size is calculated from the primary and secondary allocations specified in the user's JCL. When user labels are specified, a VIO data set has two extents: the first extent contains user labels and the second extent contains the data set.)

VIO invokes RSM and passes a pointer to a chain of VCBs that contain parameters used in performing the paging I/O.

Note: There is one VCB for each page in the VIO buffer that is to be paged in or out, and RSM processes each VCB independently in the sequence in which they are chained.

RSM uses the virtual storage address in the VCB (VCBVSA field) to access the PGTEs and XPTEs for the pages involved in the I/O operation. It uses the RSN in the VCB to access the PFTE. Using these tables, proper linkage between real-storage page frames and auxiliary-storage page slots can be obtained for a VIO read or write operation.

Describing I/O Processing in Terms of VIO Subcomponents

VIO is divided into two subcomponents: EIP (EXCP intercept processor) and VBP (virtual block processor). The I/O-related processing performed by the subcomponents is depicted in Figure 2. The figure and the following text do not describe processing that occurs when the user's first I/O request is issued to a new or passed VIO data set. When this condition exists, VIO open processing occurs (see Diagram 3 in the "Method of Operation" section for details about VIO open processing).

When an I/O request is issued to a VIO data set, EIP receives control from the EXCP processor, module IECVEXCP. (For information about the EXCP processor, see *OS/VS2 I/O Supervisor Logic.*) EIP refers to addresses in the IOB and interprets the channel program to determine whether the desired record is in the pages assigned to the VIO buffer. If the contents of the buffer satisfy the request, EIP simulates execution of the channel program by moving or comparing data between the VIO buffer and the area specified by the channel program.

When the data to be accessed by the EXCP request's channel program does not reside in the pages in the VIO buffer, EIP calls VBP via the VREADWR macro to either connect (input) or disconnect (output) a page or a sequence of pages in a page-formatted data set to or from the VIO buffer. VBP handles EIP's request by specifying either the assign or move-out functions, or a combination of these functions, and then invoking RSM. When RSM's operation is completed, VBP returns control to EIP so that it can interpret and simulate execution of the channel programs. (When EIP attempts to access these virtual pages, a page fault occurs and the virtual pages are paged into real storage.)

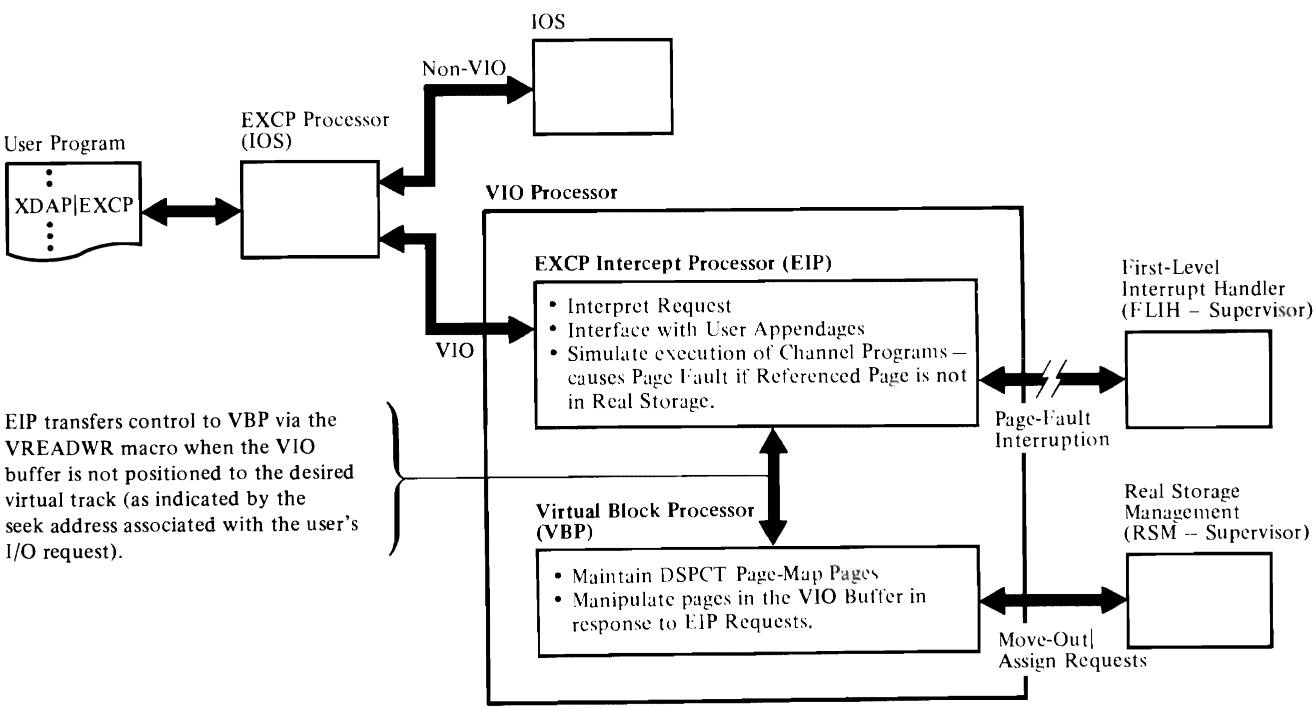


Figure 2. Handling I/O Requests in a VIO Environment

VIO Interfaces with Other Components

VIO is entered by or passes control to the OS/VS2 components shown in Figure 3.

For a description of the conditions under which calls are made to VIO and the subsequent processing that is performed by VIO, see the "Program Organization" section of this book.

Interrelationships that are generated by VIO calls to RSM and ASM are described under the headings that follow.

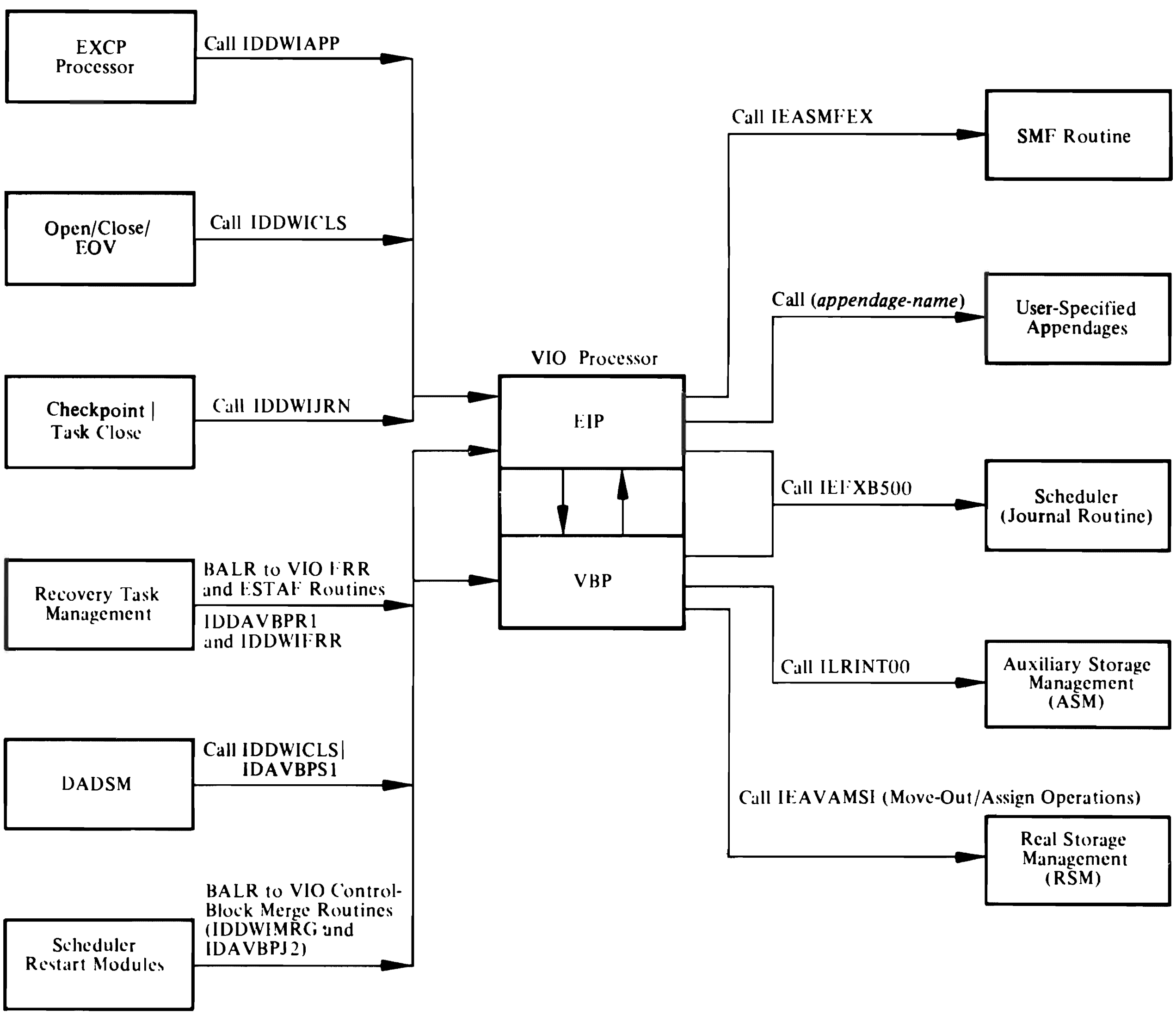


Figure 3. Overview of Component Relationships

VBP's Relationship to RSM

The VIO-services function of RSM manipulates page table entries in the page table and external page table in response to requests by VBP to connect (input) or disconnect (output) VIO data set pages to or from the VIO buffer. For detailed information about RSM, see *OS/VS2 System Logic Library.*

VBP requests the changes to the tables by passing VCBs (VIO control blocks) to RSM. VCBs are initialized during VIO open processing to support the various types of requests that are issued to RSM. Multiple VCBs representing different operations can be chained together. The VCBs are passed to RSM, and RSM performs the operations in the sequence in which the VCBs are chained. VCBs are pointed to by the DSPCT (data set page correspondence table) and are built in the LSQA (local system queue area). (See the description of the DSPCT in the "Data Areas" section of this book for information about the types of VCBs.)

The following VBP requests to RSM are supported:

Assign: The assign (virtual read) operation connects a VIO data set page to the VIO buffer. It causes page table entries to be manipulated so that when a subsequent attempt is made to access data in the VIO buffer, a page fault occurs and the page is paged into the VIO buffer in real storage.

Assign-null: Assign-null (virtual allocate) processing is requested by VBP when a virtual page is needed either for the VIO buffer or for new DSPCT page-map pages. The parameters passed are the same as those passed for the assign operation except that the LPID (logical page identifier) field is set to zero. To identify a virtual page as a page in a VIO data set, the assign-null function sets a VIO flag in the XPTE (external page table entry) associated with the virtual page.

Move-out: Move-out (write and disconnect) processing causes a page-out to occur unless the page was stolen. RSM creates a PCB (page control block) for the page-out operation and sets an indicator in the PFTE to indicate that a paging operation is in progress. The page-out operation is performed asynchronously to continued VIO processing. The pages in the VIO buffer are disconnected from the virtual storage by invalidating page table entries and setting them to zero. Then the RSNs (real storage numbers) for the page frames are saved in their corresponding DSPCT page-map entries so that the page frames may be reclaimed by a subsequent assign (read) operation involving the disconnected page frames.

If a page is stolen by the operating system before a move-out request, a page-out is not attempted. Instead, a "transfer page" operation is issued by RSM. This operation instructs ASM to change its tables to indicate that the previously stolen page in auxiliary storage belongs to the VIO data set.

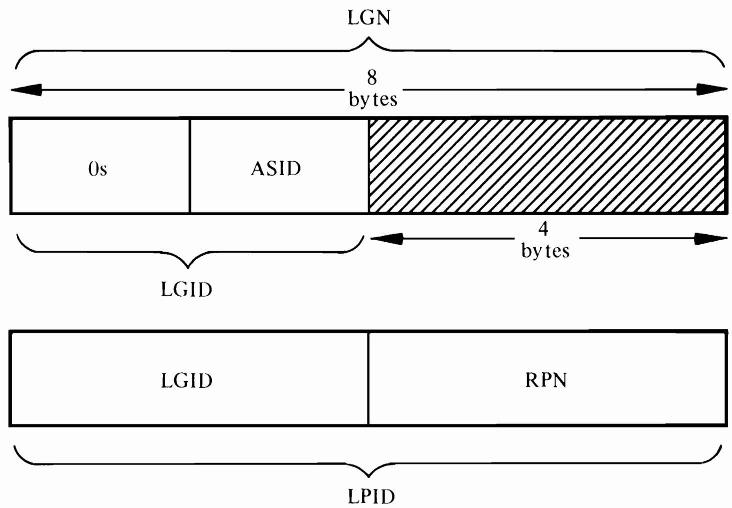
Move-out-null: Move-out-null (disconnect) processing clears the VIO buffer by disconnecting the pages in the VIO buffer from virtual storage. The pages are disconnected by modifying the page table entries associated with the pages in the VIO buffer. Then the RSNs for the page frames are saved in their corresponding DSPCT page-map entries so that the page frames may be reclaimed by a subsequent assign (read) operation.

VBP's Relationship to ASM

VBP specifies one of the following functions in the ACA (ASM control area) and then calls ASM, module ILRINT00.

Assign-logical-group: During VIO open processing, which occurs after the first EXCP is issued, VBP calls ASM to assign an LGN (logical group number) to an individual VIO data set. The LGID is a location-independent value which—together with an RPN (relative page number)—is used to establish a relationship between a page frame in real storage and a page slot in auxiliary storage.

VBP stores the LGID in fields in the DSPCT (data set page correspondence table).



VIO does not modify the LGID; however, it does adjust the RPN values in order to refer to specific pages within the data set.

Release-logical-group: If a VIO data set is being scratched, VBP calls ASM to release all auxiliary storage and control information associated with the VIO data set (or logical group). If the data set has been journaled, the storage-locator symbol is used to release the logical group; otherwise, the LGN is used. The release-logical-group function is also used at abnormal system termination to release the logical group if it cannot be accessed after a restart. (This situation occurs when the data set was not journaled by the save-logical-group function.)

Save-logical-group: If VIO is creating journal entries for a VIO data set at checkpoint or step termination, VBP calls ASM to journal the status of the VIO data set and to generate a storage-locator symbol (or "S" symbol), which is used by ASM to reactivate a VIO data set in a restart situation.

Note: ASM (Auxiliary Storage Manager), via an interface from scheduler at job termination time, releases all logical groups that are not journaled.

Activate-logical-group: If processing of a VIO data set is resumed after an automatic step or checkpoint restart, VBP passes the storage locator symbol generated by the save-logical-group function to ASM. ASM uses the symbol to regenerate control information and to regain access to the VIO data set in auxiliary storage as it existed when the most recent journaling operation was performed against the data set.

VIO Operating Considerations

VIO System Residence Requirements: The VIO load module, IDDWI (aliases IDAVBPP1 and IDAVBPS2), resides in the pageable link pack area.

Protection Key Status: VIO operates in the supervisor's key (key 0) except when I/O simulation is performed by module IDDWICPI. During I/O simulation, VIO operates in the protection key of the user.

Local Lock Retention: For processing associated with an EXCP request, the local lock is acquired by the EXCP Processor (IECVEXCP) before it passes control to VIO. This local lock is held throughout ensuing VIO processing. (See the *OS/VS2 System Logic Library* for a discussion about locks.)

When VIO processing is invoked by either task close or checkpoint routines, VIO modules set the local lock and then release it before returning to their callers.

The local lock is also acquired in the journal merge routine IDAVBPJ2 while obtaining storage for the DSPCT.

Data Set Size: The size of a VIO data set cannot exceed one volume of the device that VIO simulates in system paging space. If a size parameter is specified in JCL, the data set size is limited by the value specified—up to the maximum size of one volume. The unit-count subparameter of the unit parameter is ignored for VIO.

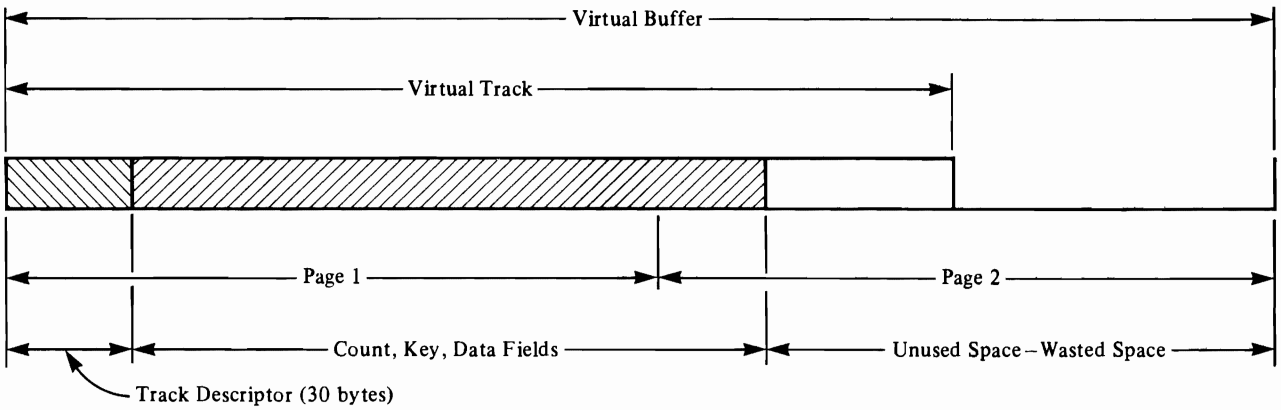
Optimizing VIO Performance

VIO performance can be improved in two main areas: virtual storage space utilization and decreasing the amount of time required to access a given number of records by VIO.

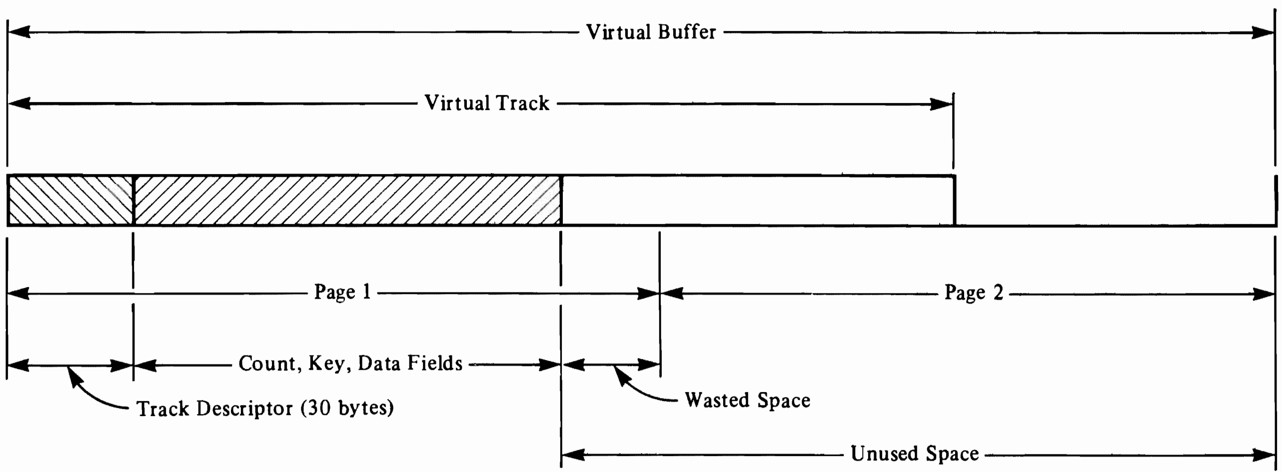
Virtual Storage Space Utilization

The VIO track size varies according to the associated virtual DASD device you specify in the UNIT parameter of the JCL statement. If UNIT=SYSDA is specified in the JCL statement, VIO will base its track size on the first eligible device in the group of devices defined for SYSDA by the system generation macro, UNITNAME.

The VIO buffer size equals the VIO track size, rounded up to a page boundary. The amount of data on the VIO track is the same amount that can be written on the real device being simulated. The data on the VIO track is packed towards the beginning of the track, eliminating the interrecord gaps that are required on a real device. Hence, the space between the end of data and the end of the buffer is unused and is not written out to auxiliary storage unless there is data on the page. The diagram that follows illustrates the wasted and unused space in the VIO buffer.



In this case both pages are read (written).



In this case, one page is read (written).

Figure 3.1 VIO Virtual Buffer

I/O Operations

Taking the preceding explanation into consideration when you specify a block size for your VIO data set, you should select a block size which is optimum for the device being simulated. Also for a given block size, you should select a device which is optimum for the block size given. Small block sizes would cause a greater number of EXCPs to be issued to transfer data between the VIO buffer and the real device. Large block sizes would cause a lesser number of EXCPs to be issued to transfer data between the VIO buffer and the real device. Since no physical I/O is required if the EXCP request can be satisfied by data movement between the VIO buffer and the access method buffer, EXCPs for smaller block sizes would be faster than for larger block sizes. However, a common overhead is involved for each EXCP issued since the end of block modules, the EXCP, and the appendages will be executed. Hence, the block sizes should not be extremely small.

You can specify the number of access method buffers through the BUFNO parameter in the JCL statement or DCB macro. For optimum performance while using VIO, specify BUFNO=1 if possible.

METHOD OF OPERATION

An MO (method-of-operation) diagram describes a function performed by segments of programming code. The individual logical steps that comprise a function are enumerated in process statements in the main body of a diagram. Control blocks that contain important information are shown as input or output to the process steps.

Notes that are numerically keyed to the process steps are provided on the page that faces the diagram. The notes provide detailed information about processes and connect logic-level information to physical segments of code; one or more module names precede the notes for each process step. Sometimes, if a process statement on the diagram is self-explanatory, corresponding notes are omitted, but the module that performs either the test or process is usually named in the notes section. Also, when a process step represents a transfer of control from one module to another, the calling and called module names are provided at the head of the note for the process statement. This provides immediate flow-of-control information for a particular process; however, note that the "Program Organization" section of the book is best used for determining an inclusive flow pattern for an entire function.

Hints on How to Read Method of Operation Diagrams

When reading MO diagrams, don't be disturbed by the graphic details presented in the input and output sections of the diagrams. Work into the charts gradually by reading only the chart's process statements. Then, if the logic is unclear to you or if you require more information, read the notes and review the inputs and outputs that correspond to particular statements that are unclear to you.

Compendiums, in the "Program Organization" section of the manual, show the flow of control among modules in performing functional operations. Referring back and forth between an MO diagram and a related compendium can give you an integrated picture of both the logic and the program structure that relate to a VIO function.

To understand the graphic symbols used in the diagrams, see Figure 4.

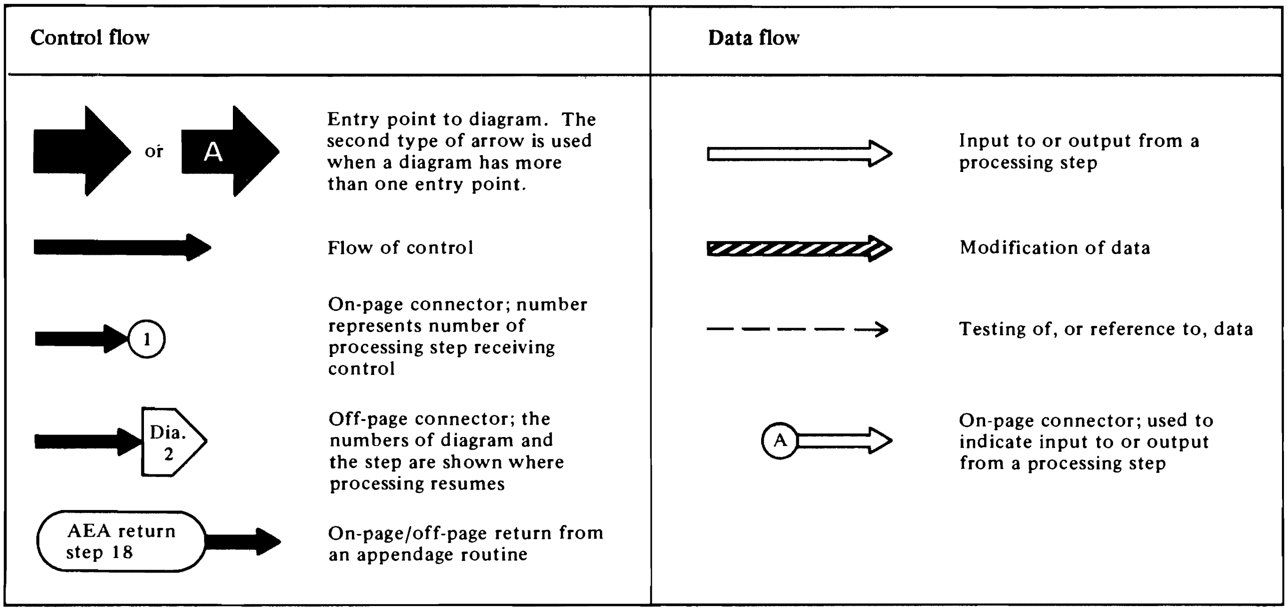


Figure 4. Graphic Symbols Used in Method of Operation Diagrams

Diagram 1. Method of Operation Contents (Tree-Structured)

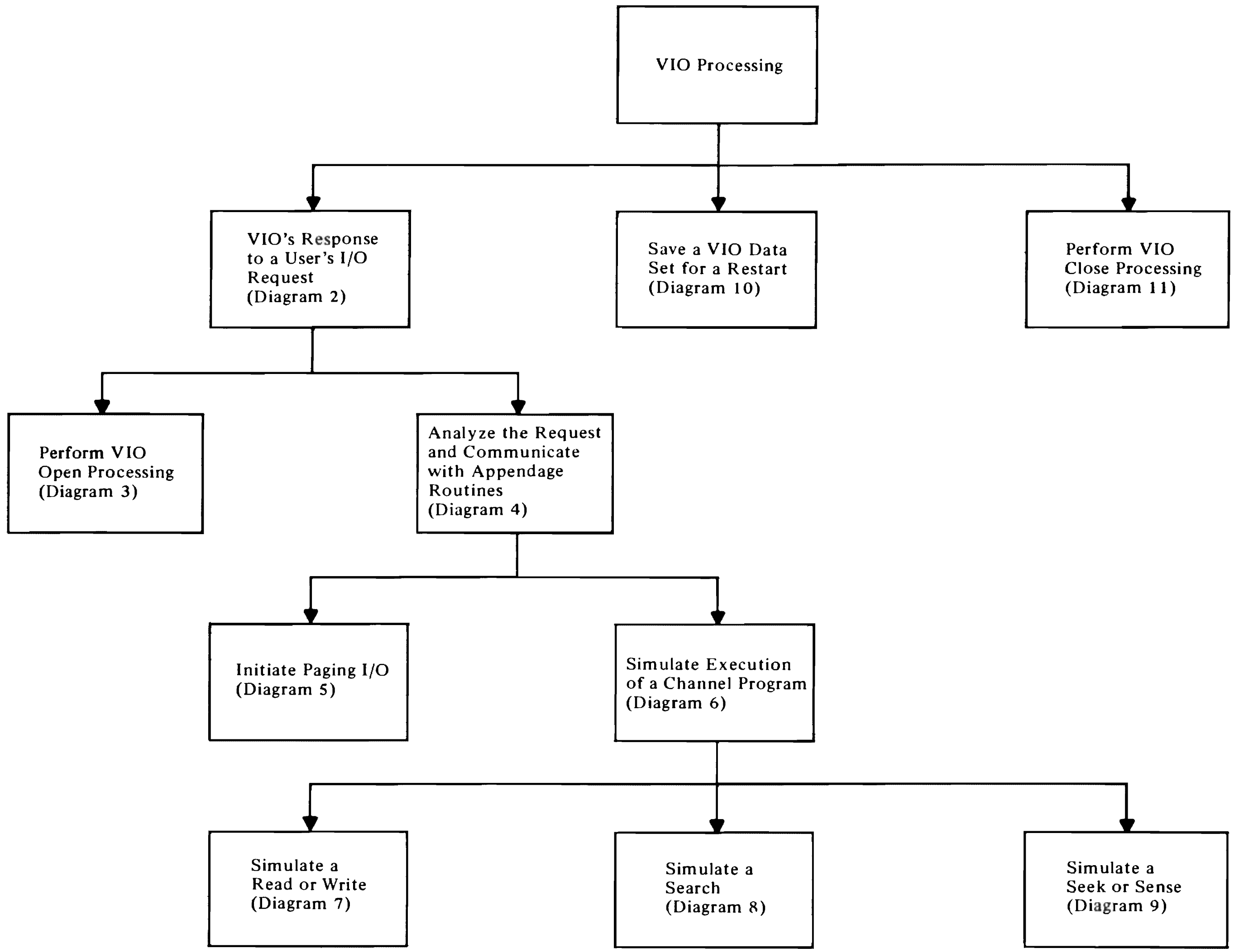
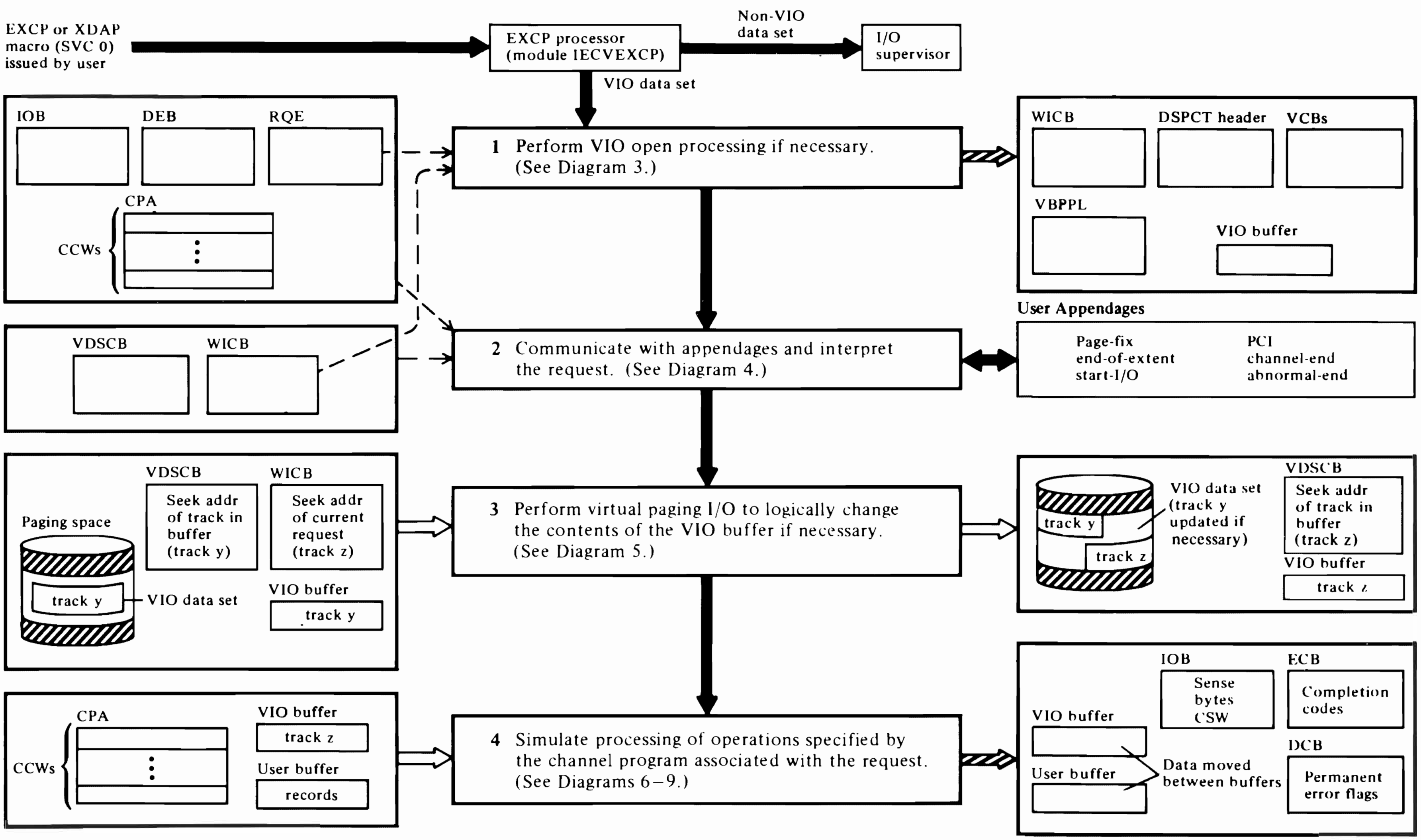


Diagram 2. VIO's Response to a User's I/O Request



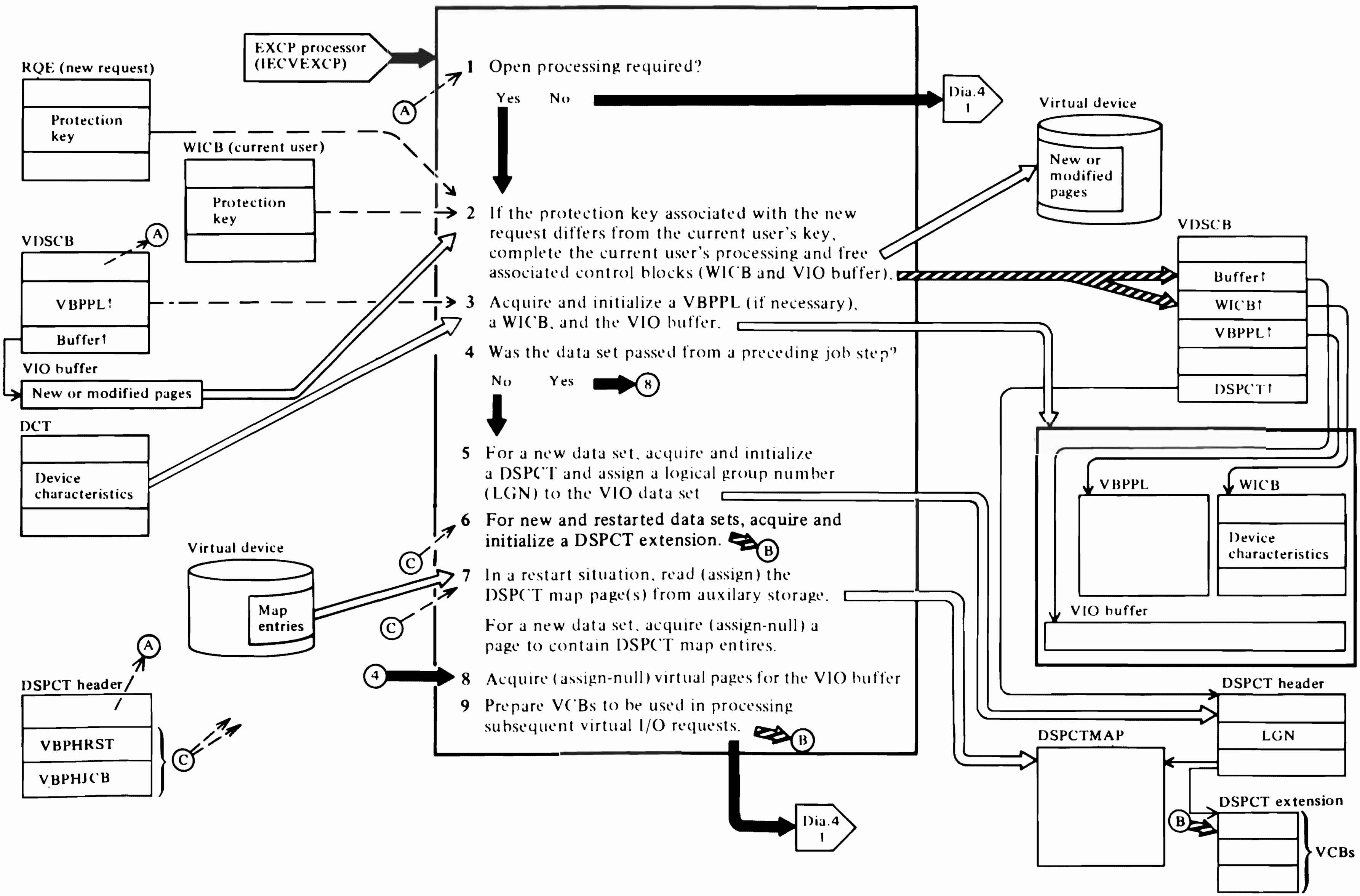
Notes for Diagram 2

Conditions at Entry to the Mainline VIO Processor

At step initiation, the scheduler determines that prerequisites for VIO processing have been satisfied. (See "Prerequisites to VIO" in the "Introduction" for information about prerequisites.) Recognizing that processing is directed to a simulated device, the scheduler does not attempt to perform space allocation on a real device; it passes control to the DADSM allocate function to construct a VDSCB for the VIO data set. The VDSCB contains a UCB, format-1 DSCB, and VIO work area. Note that because the VDSCB is built in the scheduler work area (SWA)—not in "low-core"—a 3-byte address is necessary to access the UCB in the VDSCB.

When an EXCP or XDAP macro (SVC 0) is issued by a user of VIO, control is passed to the EXCP processor, module IECVEXCP. The EXCP processor constructs a request queue element (RQE) and passes control to VIO.

Diagram 3. Perform VIO Open Processing



Notes for Diagram 3

Entry Conditions Related to VIO Open Processing

VIO open processing is independent of common open processing, which always precedes it, and occurs when a user issues the first XDAP or EXCP macro against a VIO data set under any of the following circumstances:

- A new VIO data set is being created.
- A restart operation is directed against an existing VIO data set.
- A VIO data set is being reopened after common and VIO close processing have been done.
- A user, other than the current user, attempts to access an existing VIO data set. This situation is detected when the protection keys of the users do not match. For example, when a user-created PDS (partitioned data set) contains both problem-program data (such as user-defined tables) and linkage-editor data, the key conflict occurs when the data set is accessed by both the problem program and program fetch.

The flow of control into and out of VIO open processing is as follows:

- The EXCP processor determines that the request involves a VIO data set and passes control to the VIO interface routine, module IDDWIAPP.
- The interface routine determines that a condition requiring VIO open processing exists and passes control to the VIO open routines.
- When open processing is completed, the VIO open routines return control to the interface routine, which then continues VIO processing related to the EXCP request (see Diagram 4).
- If an EXCP is issued against a scratched data set, then IDDWIAPP issues a 0E6 ABEND.

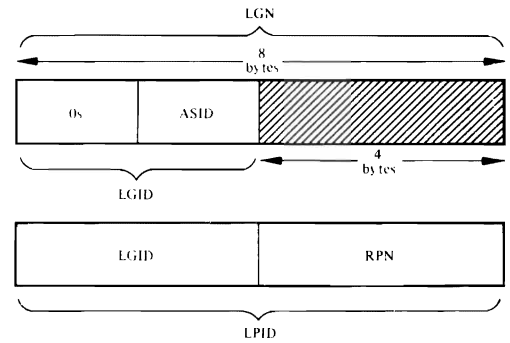
1. **IDDWIAPP:** (See the introductory comments to this figure.)
2. **IDDWIAPP:** If VIO open processing is to occur because of a key change, do the following clean-up operations:

   (a) **IDDWIAPP calls IDDWITRM:** If the VIO buffer contains new or modified data, IDDWITRM issues a VREADWR macro to cause the pages in the buffer to be written to external page storage. (For a description of processing associated with the VREADWR macro, see Figure 6, steps 4-7, in the "Program Organization" section of the manual.) Otherwise (that is, when the VIO buffer doesn't contain new or modified data), IDDWITRM simply returns to IDDWIAPP.

   (b) **IDDWIAPP:** When the buffer contents are no longer needed, free the buffer and its associated WICB, and continue open processing in the conventional fashion.
3. **IDDWIAPP:** Acquire a VBPPL if necessary. (A VBPPL is acquired for a new job step, after a restart, or for a new data set.)

   Acquire a WICB and initialize it with the device characteristics to be simulated in paging space. (For specific information about the characteristics, see the WICDVTAB field in the WICB, offset 70 (decimal), in the "Data Areas" section.) Also acquire a VIO buffer based on the track size, rounded-up to a page boundary, of the device being simulated. Usually, the VIO buffer, WICB, and VBPPL are acquired once per job step when the first EXCP or XDAP macro is issued to a VIO data set.
4. **IDDWIAPP.**
5. **IDDWIAPP calls IDAVBPO1 via VOPEN macro:** Acquire a DSPCT based on a fixed header size and a 4-byte entry for each potential DSPCT page-map page in the data set. (Potential data set size is the primary extent plus fifteen of the secondary extents described in the user's DD statement—reduced, if necessary, to the size of one full volume of the device being simulated in external page storage.) Initialize the header with control information.

   IDAVBPO1 calls ASM, module ILRINT00: Assign an LGN (logical group number) to the current logical group, or data set. The LGN is an 8-byte field. ASM (auxiliary storage management) returns an LGID (logical group identifier) in the first word of the LGN/LPID field. The returned LGN is stored in the DSPCT. The LGID is used by VIO as the "LPID generator," the LPID (logical page identifier) being a combination of the LGID value assigned to the logical group by ASM and the RPN (relative page number) that is generated by VIO and resides in the second word of the LGN field. The LPID relates a page-sized portion of a VIO data set to an auxiliary-storage page slot. If ASM is unable to assign an LGN to the VIO data set, IDAVBPO1 issues an ABEND macro with a 0E1 completion code.

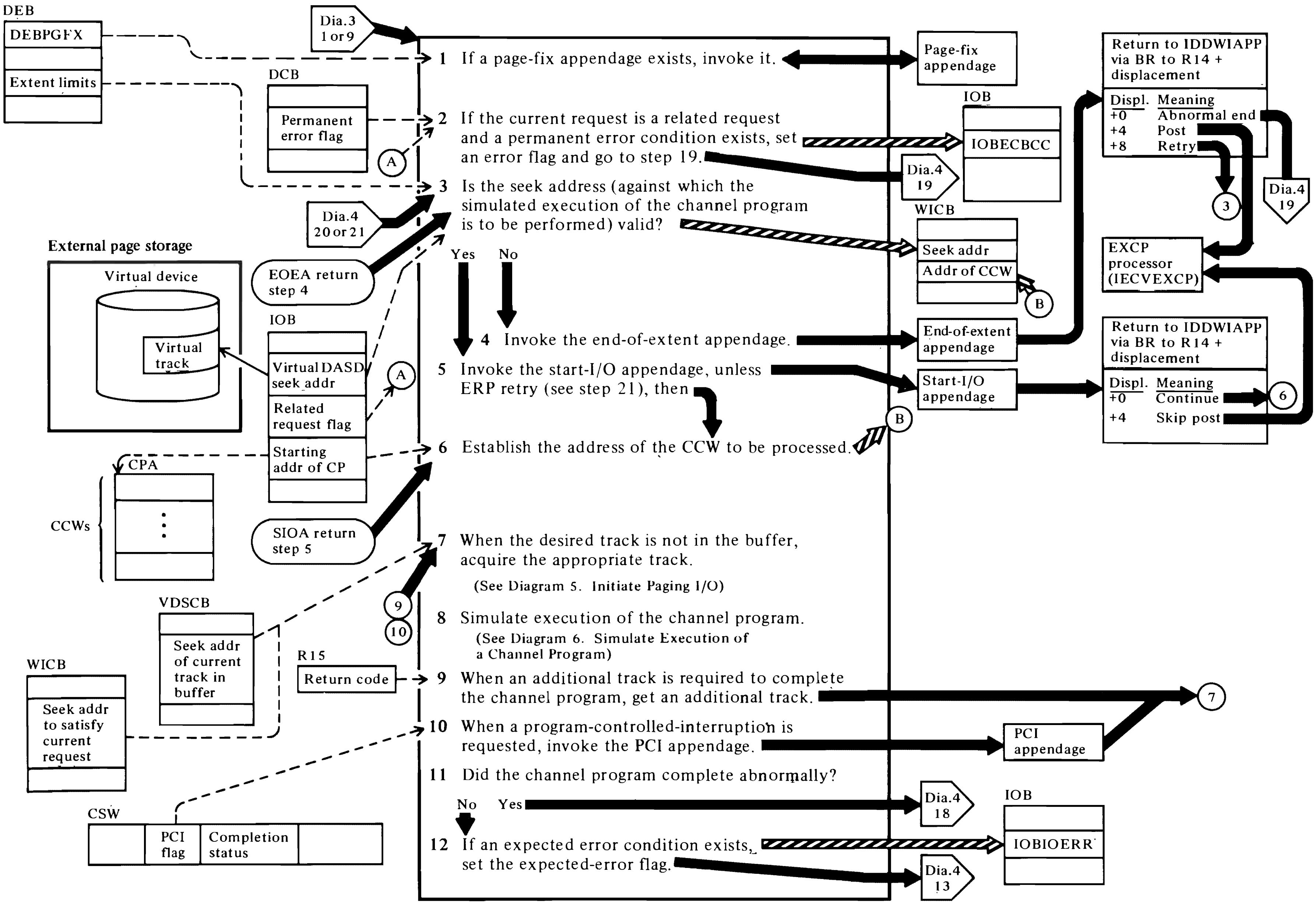


6. **IDAVBPO1:** Acquires storage for the VCBs in the LSQA and chains it from the DSPCT header.
7. **IDAVBPO1 calls IDAVBPR2 (CHEKMPPG), which in turn calls RSM, module IEAVAMSI:** If there are no records in the data set, do an assign-null (virtual allocate) to acquire a single page for DSPCT page-map entries. This operation identifies a virtual page as a VIO page. Otherwise (that is, when records exist in the data set for a restart situation), a read (assign) is done on all the DSPCT page-map pages. If an error occurs in assign or assign-null processing, IDAVBPO1 issues an ABEND macro instruction with a 0E1 (create processing) or 0E2 (restart processing) completion code.

Notes for Diagram 3 (Continued)

8. **IDAVBPO1:** Connect (assign-null) the buffer pages to the VIO user's address space by making a call to RSM, module IEAVAMSI, to perform an assign-null operation on the pages associated with the VIO buffer.
9. **IDAVBPO1 calls IDAVBPR2 (SETVCB):** Initialize read (assign), write and disconnect (move-out), and disconnect (move-out-null) VCBs for subsequent request processing. This processing occurs once during VIO open processing.

Diagram 4. Analyze the Request and Communicate with Appendage Routines



Notes for Diagram 4

When an EXCP request is issued against a VIO data set, the EXCP processor, module IECVEXCP, detects this condition and passes control to VIO, module IDDWIAPP. For the first EXCP request, IDDWIAPP initiates VIO open processing and initializes selected VIO control blocks (see Diagram 3). Then, for the first and each succeeding EXCP request, IDDWIAPP simulates I/O activity in response to the channel programs associated with the EXCP requests, and interfaces with appendages (as normally performed by the EXCP processor for non-VIO data sets). All returns from appendages that are supported by conventional, non-VIO, processing are supported by VIO. (See *OS/VS2 System Programming Library: Data Management* for information on appendage interfaces and processes.)

1. **IDDWIAPP:** Pass control to the user-supplied page-fix appendage. A dummy list of areas to be made nonpageable is passed to the appendage along with an indicator that signifies that there are no entries in the list. No page-fixing is performed even if the appendage routine modifies the count of pages to be fixed.
2. **IDDWIAPP.**
3. **IDDWIAPP.**
4. **IDDWIAPP:** When an invalid seek address exists in the IOB, call the user-supplied end-of-extent appendage.

   The end-of-extent appendage indicates the action that it wishes to be taken by returning to IDDWIAPP with a branch to register 14 plus a displacement. The displacements and their meanings are as follows:

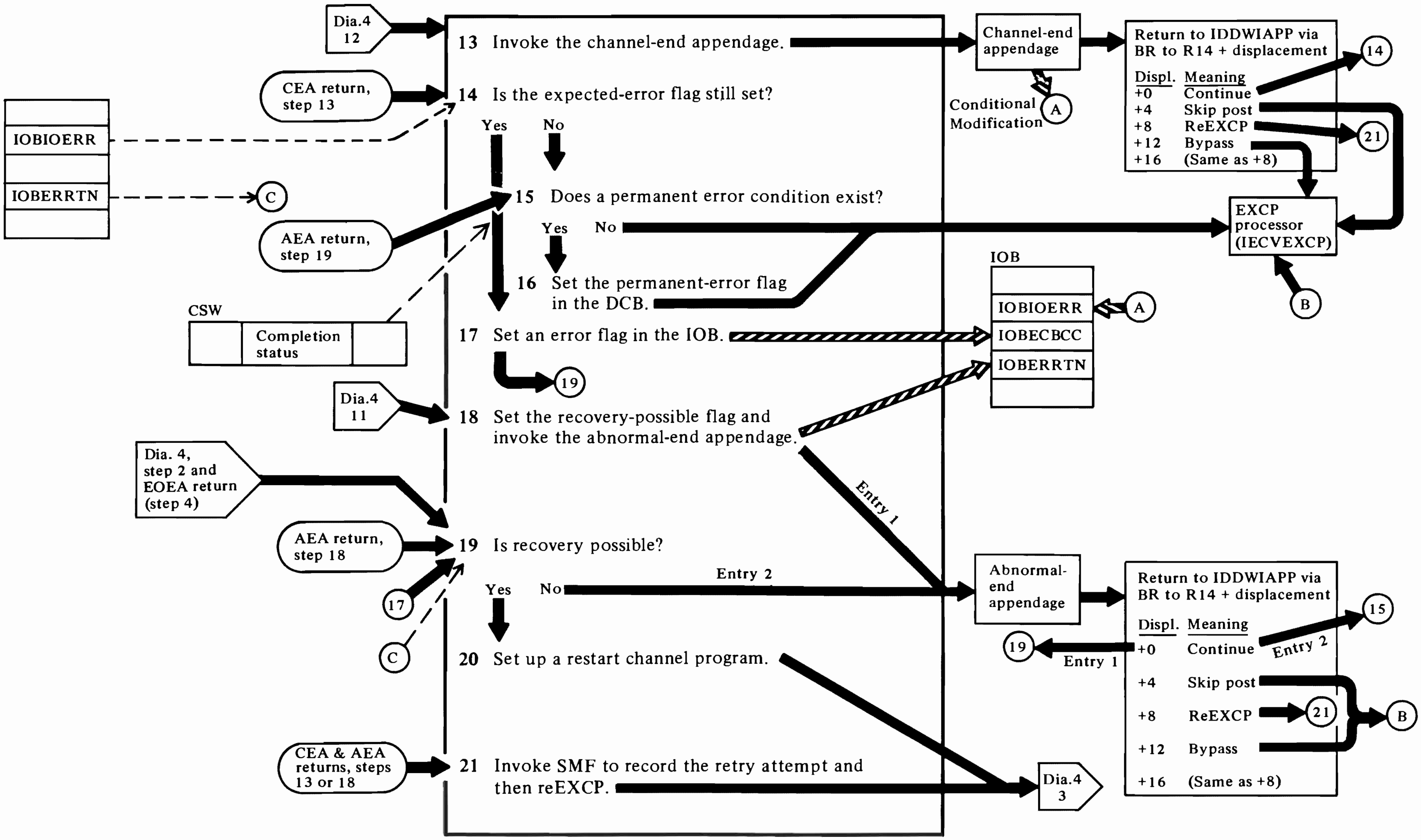
Disp.	Meaning and Action by IDDWIAPP
0	Abnormal end—Set an extent-violation completion code (X'42') in the IOB, set the recovery-not-possible flag, and pass control to the abnormal-end appendage (step 19).
+4	Post event—Set a complete-without-error indicator (X'7F') in the IOB and return to EXCP processor to post the ECB with the completion code in the IOB and to free the RQE.
+8	Retry—Return to EXCP processor to test the validity of a new seek address that was established by the end-of-extent appendage.

5. **IDDWIAPP:** Pass control to a SIO (start I/O) appendage unless error-recovery processing (see step 21) is being performed.

   Two returns from the SIO appendage are supported—a return via a branch to register 14 with a displacement of either +4 or 0. If the displacement factor is +4, VIO returns control to the EXCP processor with a return code directing the EXCP processor to free the current RQE and not to post its ECB as complete. Otherwise (that is, when the displacement factor is 0), processing continues at step 6.
6. **IDDWIAPP.**
7. **IDDWIAPP calls IDDWITRM, and IDDWITRM calls IDAVBPP1 via the VREADWR macro:** Compare the track address (MBBCCHHR) of the desired track against the address of the track which is currently in the buffer. If the addresses do not match, clear the buffer (that is, perform a move-out or move-out-null) and then perform a virtual read (assign). If the VIO buffer is empty (that is, if a write operation was previously performed on the pages in the buffer), perform a virtual read (assign).
8. **IDAVBPP1 returns to IDDWITRM, which then calls IDDWICPI.**
9. **IDDWICPI returns to IDDWITRM:** If IDDWICPI sets a return code that indicates that either a multitrack operation, a seek operation, or an operation involving track-overflow records could not be completed on the existing track, acquire a new track in order to complete the operation.

   If IDDWICPI detects the PCI flag in the CCW it is processing, it sets the PCI flag in the CSW associated with the channel program (see note 10).
10. **IDDWITRM returns to IDDWIAPP:** IDDWIAPP examines the CSW to determine whether IDDWICPI has set the PCI flag and invokes the PCI appendage if the flag is set. When the PCI appendage returns control, IDDWIAPP links to the SMF routine, module IEASMFEX, to record the PCI interruption and then returns control to IDDWITRM to complete the channel program.
11. **IDDWIAPP:** Examine status indicators in the CSW. Abnormal error conditions are unit check, program check, protection check, and channel-control check.
12. **IDDWIAPP:** Examine status indicators in the CSW. Expected error conditions are incorrect length and unit exception.

Diagram 4. Analyze the Request and Communicate with Appendage Routines (Continued)



Notes for Diagram 4 (Continued)

13. **IDDWIAPP invokes the channel-end appendage.** (See step 18's description of abnormal-end appendage returns. The returns supported for the channel-end appendage are the same as those for the abnormal-end appendage.

    The actions taken by IDDWIAPP in response to the returns are also the same—except for a return with a 0 displacement. For a 0 displacement, IDDWIAPP checks the expected-error flag. If it is not set, IDDWIAPP tests the ECB to determine its wait status. If it is being waited upon, the ECB is not posted and control is returned to the EXCP processor with a +4 displacement. If the ECB is not being waited upon, it is posted and control is returned to the EXCP processor with a 0 displacement. Otherwise (that is, when the expected-error flag is set), IDDWIAPP continues processing at step 17.)

14. **IDDWIAPP:** Examine the IOBIOERR flag set in step 12. If the channel-end appendage has reset the flag, return to the EXCP processor, IECVEXCP, either in step 15 or 16.

15. **IDDWIAPP:** If a permanent error condition doesn't exist, test the ECB to determine its wait status. If the ECB is being waited upon, return control to the EXCP processor with a 0 displacement. If it is not being waited upon, post the ECB and return control to the EXCP processor with a +4 displacement.

16. **IDDWIAPP.**

17. **IDDWIAPP.**

18. **IDDWIAPP:** The abnormal-end appendage performs its processing and selects one of the four options, which are indicated by a return to IDDWIAPP via a branch to register 14 plus a displacement. The displacements and their meanings are as follows:

Disp.	Meaning and Action by IDDWIAPP
0	Post event—Following "entry 1" to AEA: Test the error-recovery-possible flag. If it is set, continue at step 20; otherwise, reinvoke the abnormal-end appendage (entry 2). Following "entry 2" to AEA: If the ECB is not being waited upon, post the ECB with the completion code in the IOB and branch to register 14 plus 0. If the ECB is being waited upon, don't post it and branch to register 14 plus 4. The EXCP processor frees the RQE.
4	Skip posting—The ECB is not posted and control is returned to EXCP processor via a branch to register 14 plus 4. The EXCP processor frees the RQE.
8	Initiate new channel program—Clear error flags in the IOB and branch to SMF to record the request. Then reinitiate processing at step 3.
12	Bypass—The ECB is not posted and control is returned to EXCP processor via a branch to register 14 plus 8. The EXCP processor does not free the RQE because it is assumed that the RQE will be used subsequently to schedule an asynchronous exit routine.
16	(Same as displacement "8.")

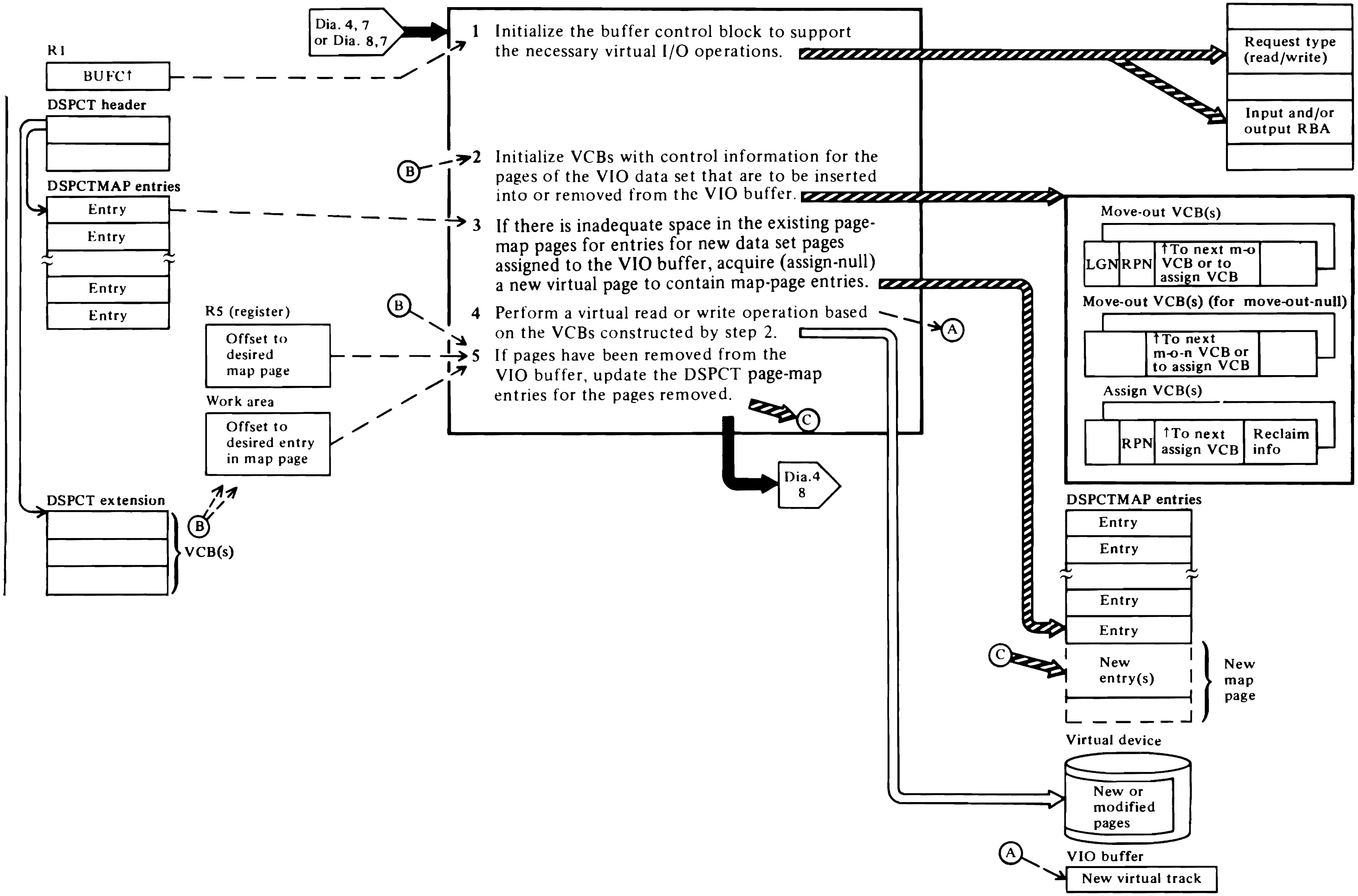
19. **IDDWIAPP:** Examine the error-routine flag (IOBERRTN) to determine whether the error was corrected. If the error condition still exists, set a flag that indicates that the error recovery procedure (ERP) was unable to correct the error and reinvoke the abnormal-end appendage ("entry 2"). Otherwise, continue recovery processing.

20. IDDWIAPP: See the table below to adjust CCW and track pointers.

21. IDDWIAPP calls IEASMFEX.

Correctable Error Condition	Response
File mask violation by seek operation	Position to the next track and restart processing with the next sequential CCW, unless the next CCW is a TIC; then restart processing with the CCW pointed to by the TIC command.
File mask violation by a multitrack search or read	Position to the next track and restart the search or read.
Overflow incomplete	Construct a 2-CCW prefix in the current CCW: RD | WR, TIC (next sequential CCW)—unless the next sequential CCW is a TIC; then TIC to the CCW specified by the TIC. and restart on the next sequential CCW, where byte count in the RD | WR CCW=CSW byte count (unless CSWCNT=0, then byte count=1 command address=(address of failing CCW)+(length of failing CCW)-(residual count from CSW). (If CSWCNT=0, command address points to a dummy field of zeros.

Diagram 5. Initiate Paging I/O



Notes for Diagram 5

1. **IDDWITRM:** If the VIO buffer has been modified (note: IDDWICPI sets an indicator in the buffer's header when it adds to or modifies the contents of the buffer; IDDWITRM sets an indicator when it initializes a new track), to initialize the BUFC (buffer control block) to cause the pages in the buffer to be written to external page storage. Then set—in the BUFC—the read-required flag and the RBA of the desired virtual track. This causes paging I/O, consisting of successive write operations followed by successive read operations, to be performed on the buffer.

   If the VIO buffer is empty or if the buffer does not contain new or modified data, set—in the BUFC— a read-required flag and the RBA of the desired virtual track. Calculate the input RBA using the seek address in the WICB, the virtual buffer size, and the characteristics of the virtual device.

2. **IDDWITRM calls IDAVBPP1 via VREADWR macro:** When a write (move-out) operation is specified because the contents of the current track have been modified (BUFCMW=ON), initialize the move-out VCBs in the DSPCT header with the RPNs of the pages in the VIO buffer up to and including the page that contains the last byte of data. (The displacement to the end of data in the VIO buffer is maintained by IDDWICPI in the VIO buffer's header (VTDATEND field).) If the prior request was a read request and the number of pages read exceeds the number currently in the buffer, base the number to be written on the number previously read to account for update and deletion processing.

   If a read (assign) operation is specified (BUFCRRD=ON) and the prior paging operation was a read but the contents of the buffer were not subsequently modified by simulated I/O activity, initialize move-out VCBs for move-out-null and pass the VCBs to RSM to disconnect the pages from the VIO buffer. Otherwise (that is, when the buffer contents were modified), initialize move-out VCBs to cause the pages in the buffer to be written to auxiliary storage. Then initialize assign VCBs for the pages to be read, and chain the assign VCBs to the move-out VCBs.

   If a read (assign) operation is specified (BUFCRRD=ON) and the prior operation was not a read, initialize assign VCBs and pass control to RSM to connect VIO data set pages to the VIO buffer. If the pages are "reclaimable" (see the "Introduction" for a discussion of reclaiming page frames), set real storage numbers that are associated with the page frames previously containing the pages in the assign VCBs.

   Note: If a read (assign) request is directed at a page of data that does not exist in the data set, IDAVBPP1 sets an indicator in the BUFC. When control is returned to IDDWITRM (either directly or after processing a move-out or move-out-null request) IDDWITRM initializes the new track with a home address (HA) and a record 0 (R0) based on values in the seek address and sets a flag that indicates that the track has been modified. This formatting process causes a page fault.

3. **IDAVBPP1:** When there is insufficient unused space in the current DSPCT page-map page to contain new DSPCT page-map entries and if a read operation is being performed on a new track, a track that was not previously written to auxiliary storage, perform the following processing to acquire a DSPCT page-map page:

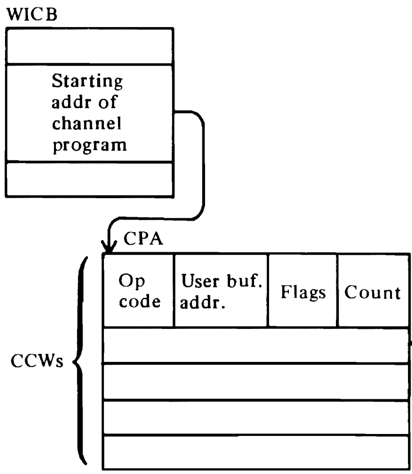
   IDAVBPP1 calls IDAVBPR2 (CHEKMPPG): Issue a GETMAIN macro instruction to acquire a virtual page frame for the DSPCT page-map page. Then call RSM, module IEAVAMSI, to do an assign-null operation against the new page in order to define the page as a VIO page.

4. **IDAVBPP1 calls IEAVAMSI:** RSM inserts or removes VIO pages into or from the VIO buffer by manipulating page table entries and page frame table entries which correspond to the pages associated with the VCBs initialized by step 2.

   IDAVBPP1: If the return code from RSM indicates that an error condition exists, IDAVBPP1 issues an ABEND macro with an X'0E3' completion code. Otherwise, IDAVBPP1 calls IEAVPSIB to free the page containing the VCBs.

5. **IDAVBPP1:** For a move-out or move-out-null request, save the RSNs (real storage numbers) of the page frames containing the pages that were in the VIO buffer. The RSNs are saved in the appropriate DSPCT page-map entries to enable the page frames to be reclaimed by subsequent read (assign) requests.

Diagram 6. Simulate Execution of a Channel Program



Dia.4 8

1 If the CCW format or the operation code is invalid, set an error completion status and return to the caller.

2 Simulate CCW execution.

- For read or write operations, see Diagram 7.
- For a search operation, see Diagram 8.
- For a sense or seek operation, see Diagram 9.
- For NOP, RESTORE, or SET SECTOR commands, continue processing at step 5.

3 If a program-controlled-interrupt (PCI) flag is detected in a CCW by step 2 processing, return to the caller.

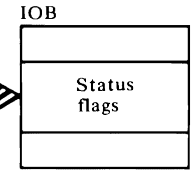
4 If an error condition is encountered in the simulated execution of a CCW, set an abnormal completion status and return to the caller.

5 If an additional CCW exists in the channel program, repeat steps 1–5.

6 If all CCWs in the channel program have been processed and no errors have been encountered, set a normal completion status and return to the caller.

Dia.4 8

Dia.4 8



Notes for Diagram 6

IDDWICPI simulates the execution of channel programs directed to a VIO data set. It does this by using CPU instructions to simulate the channel and device actions taken for a particular CCW string.

When IDDWICPI receives control, all necessary control information has been established, and the VIO buffer is positioned, by a virtual read (assign) operation, to a track containing pages that satisfy the seek address associated with the current channel program.

If the desired virtual pages are in real storage because of a page-in caused by a prior request, the pages are reclaimed when IDDWICPI attempts to access the data in the buffer. If the required pages are not in real storage, they are subsequently paged into real storage as a result of a page fault when IDDWICPI attempts to access the data in the buffer.

IDDWICPI simulates execution of the channel program by moving records between the VIO buffer and the area in the user's buffer that is pointed to by the CCW being processed.

Note: All notes pertaining to Diagram 6 involve module IDDWICPI.

1. The format of a CCW must satisfy the following requirements:
   - Doubleword alignment.
   - Bits 37-39 are off (0).
   - Nonzero count field, except for TICs.
   - Nonzero address field.
2. Analyze the CCW's operation code to determine which CCW interpretation routine is to be entered:

Bit Settings	Type of Operation
0000 0100	Sense
xxxx 1000	Transfer in channel (TIC)
00 **01	Write
11 **01	Search
**** **10	Read
**** **11	Control

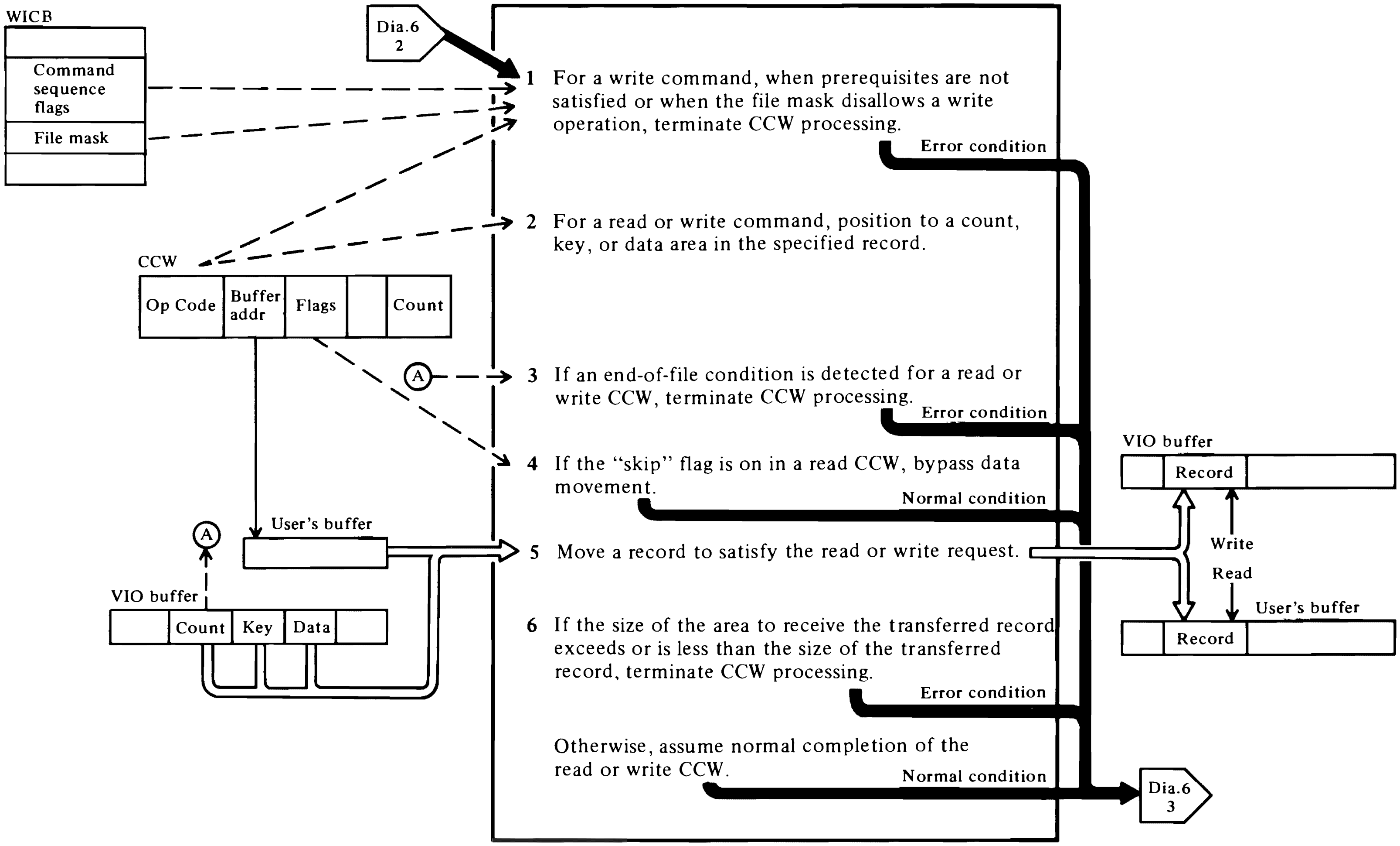
* Further define type of operation by providing count, key, and data information.

x Ignored.

3. CPIOPEND (label) is the return point in IDDWICPI mainline code that is branched to by the segments of code that relate to the various CCW operations reflected by step 2.

   If a PCI flag is detected in any CCW associated with the channel program, invoke the PCI appendage before continuing execution of the channel program.
4. Check the IOBCSW, IOBSENS0, and IOBSENS1 fields for error indicators that may have been set during simulated execution of the CCW. (See the "Diagnostic Aids" section for a description of error indicators in the IOB and for the labels of code seqments in IDDWICPI that detect the error conditions and set the error indicators.)
5. If an error has not occurred and the command-chaining bit in the current CCW (bit 33) is set, continue processing with the next CCW.
6. If there are no errors and the last operation is completed, set channel-end and device-end indicators in the CSW.

Diagram 7. Simulate a Read or Write



Notes for Diagram 7

In a non-VIO environment, a read or write CCW causes data (count-key-and-data, key-and-data, or data) to be copied to or from a particular area of a record on a real device to or from a user's buffer.

In a VIO environment, VIO data sets are maintained on virtual devices in external page storage, and I/O associated with read and write CCWs is simulated using a VIO buffer. VIO simulates execution of a read or write CCW by moving records between the VIO buffer and the user's buffer.

In either case (VIO or non-VIO), the data manipulation is governed by the direction of the move (read or write), the length specified in the CCW and the count field of the record (which determine how much data is moved), the file mask (which determines whether the movement can take place), and the CCW flags (which indicate whether data will be transferred and whether length errors will be accepted).

Note: All notes pertaining to Diagram 7 involve module IDDWICPI.

1. Only write CCWs have special requirements concerning the types of commands that must precede them. See the reference manual for the virtual device being simulated in paging space for a description of write command prerequisites. Note that the virtual device type is indicated in the WICB.

2. The operation code determines which areas are to be moved—either the count, key, or data area of the current record or a combination of the count, key, and data areas. Start positioning at the first of the desired areas.

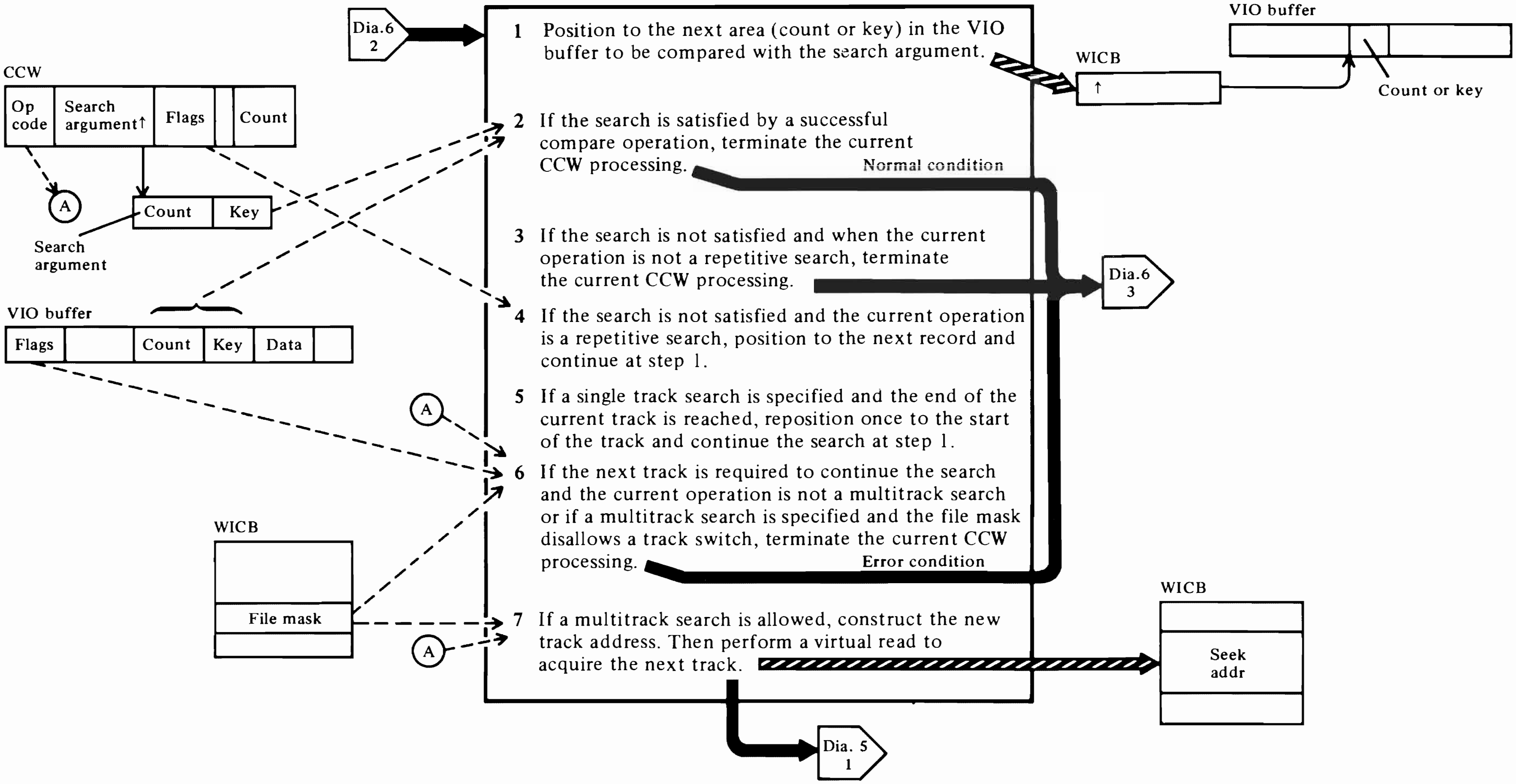
Operation Code	Desired Record Area
***1 00**	Count
***0 01**	Data
***0 10**	Key
***0 11**	Key, data
***1 01**	R0
***1 10**	HA
***1 11**	Count, key, data

*specifies the type of operation.

3. For RCKD, RKD, RD, WD, or WKD commands, when the data length of a record is zero, set a unit-exception indicator in the CSW. (Note that a record with a data length of zero is used as an end-of-file mark by all VIO users.)

4. The CCW's "skip" bit (bit 35) is used to avoid unwanted data. When the bit is on, do not perform data transfer; however, for read requests, adjust positioning as though data had been passed. For write requests, ignore the "skip" bit.

Diagram 8. Simulate a Search



Notes for Diagram 8

A search command compares a part of a record (count or key) with an area that is pointed to by a CCW. When the compare condition (EQ, HI, or HI | EQ) is satisfied, skip the next sequential CCW; otherwise, process it. (Key and data searches of the extended-file-scan feature are not supported by VIO.)

Search commands are usually found in TIC loops where the compare is repeated until the desired record is found; for example,

Command	Explanation
SID TIC*-8	Find a particular count field.
RD CNT SK EQ TIC*-16	Find a record with a given key and read the count of that record.

The logic of search processing reflected by Diagram 8 is also illustrated in the table below:

Conditions							
Search Satisfied	T	F	F	F	F	F	F
Multitrack Search		T	T			F	F
End-of-Track Condition		T	T		F	T	T
Repetitive Search		T	T	F	T	T	T
File Mask Permissive		T	F				
Index Point Passed (Latch=on | off)						T	F
Actions							
Position to Next Track		X					
Position to Next Record					X	X	
Reposition to Start of Track							X
Search again		X			X	X	X
Search Completed (normal)	X			X			
Search Completed (error)			X				

Note: All notes pertaining to Diagram 8 involve module IDDWICPI.

2. When the compare is satisfied, skip the next CCW.
3. A repetitive search is a group of CCWs that are handled as though they were one CCW. The following groups are handled as one CCW:

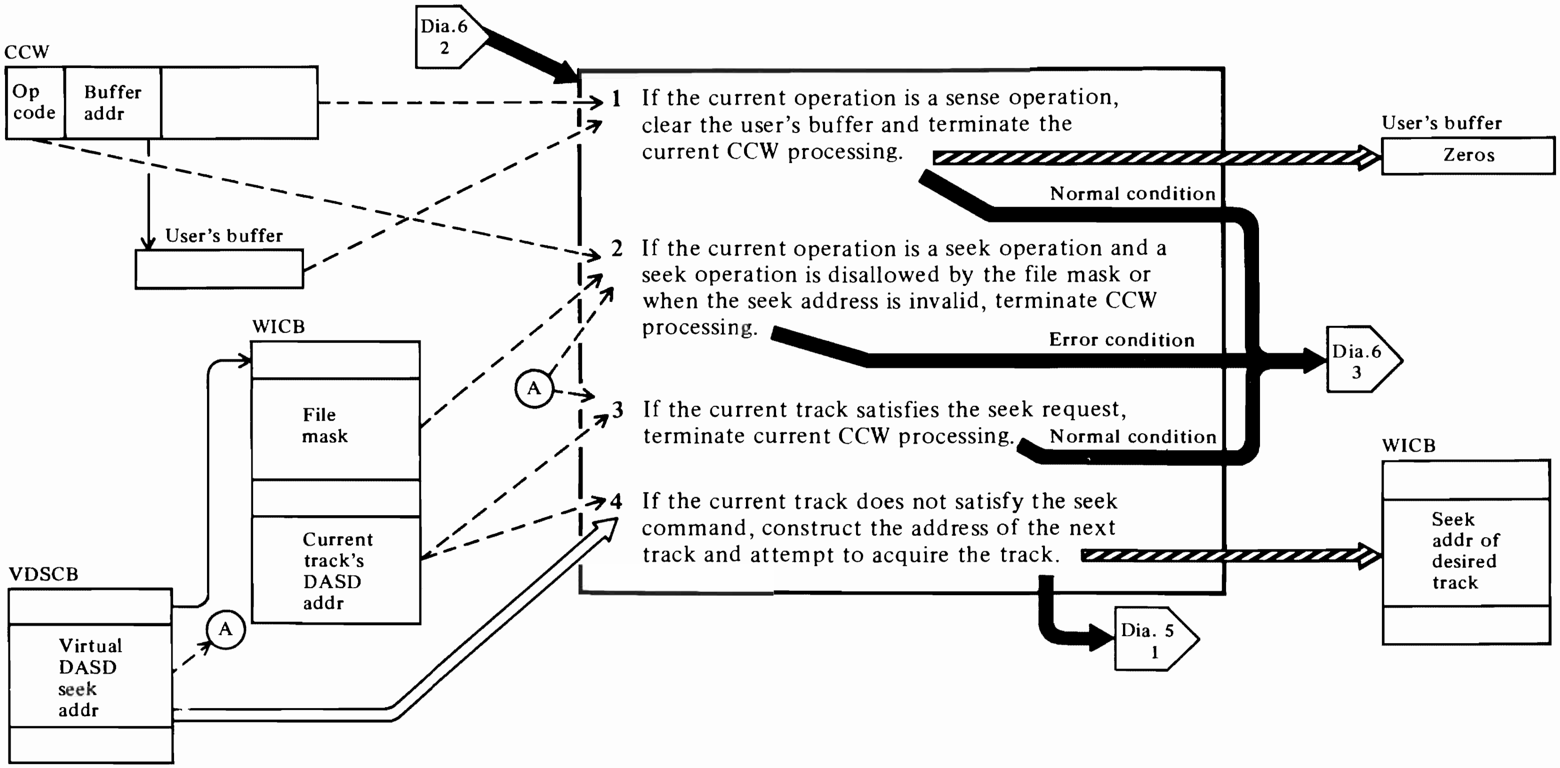
   SID
   TIC*-8

   SK
   TIC*-8

   RD CNT (single track)
   SK (single track)
   TIC*-16

   When a group of CCWs is handled as one CCW, the result to the user is transparent; however, it results in a more efficient operation since the CCW loop doesn't result in the overhead of repeatedly validating the same CCWs.
5. If a single-track search is specified and there are no more records on the current track, resume the search by repositioning to the start of the current track unless repositioning has already been performed once for this operation. If the end of a track is reached by a multitrack search operation, position to the next track if the file mask allows multitrack searches.
6. If a CCW's multiple-track- search indicator (bit 0) is off and if the end of a track has been reached a second time, set a no-record-found indicator.

Diagram 9. Simulate a Seek or Sense



Notes for Diagram 9

Sense Commands

The sense command is normally used after a unit check by the I/O supervisor's error-recovery procedures (ERPs) to check the status of a device for error-recovery operations. Since hardware errors cannot occur on a virtual device, when a sense command is issued against a virtual device, zeros are returned as sense-status information.

Seek Commands

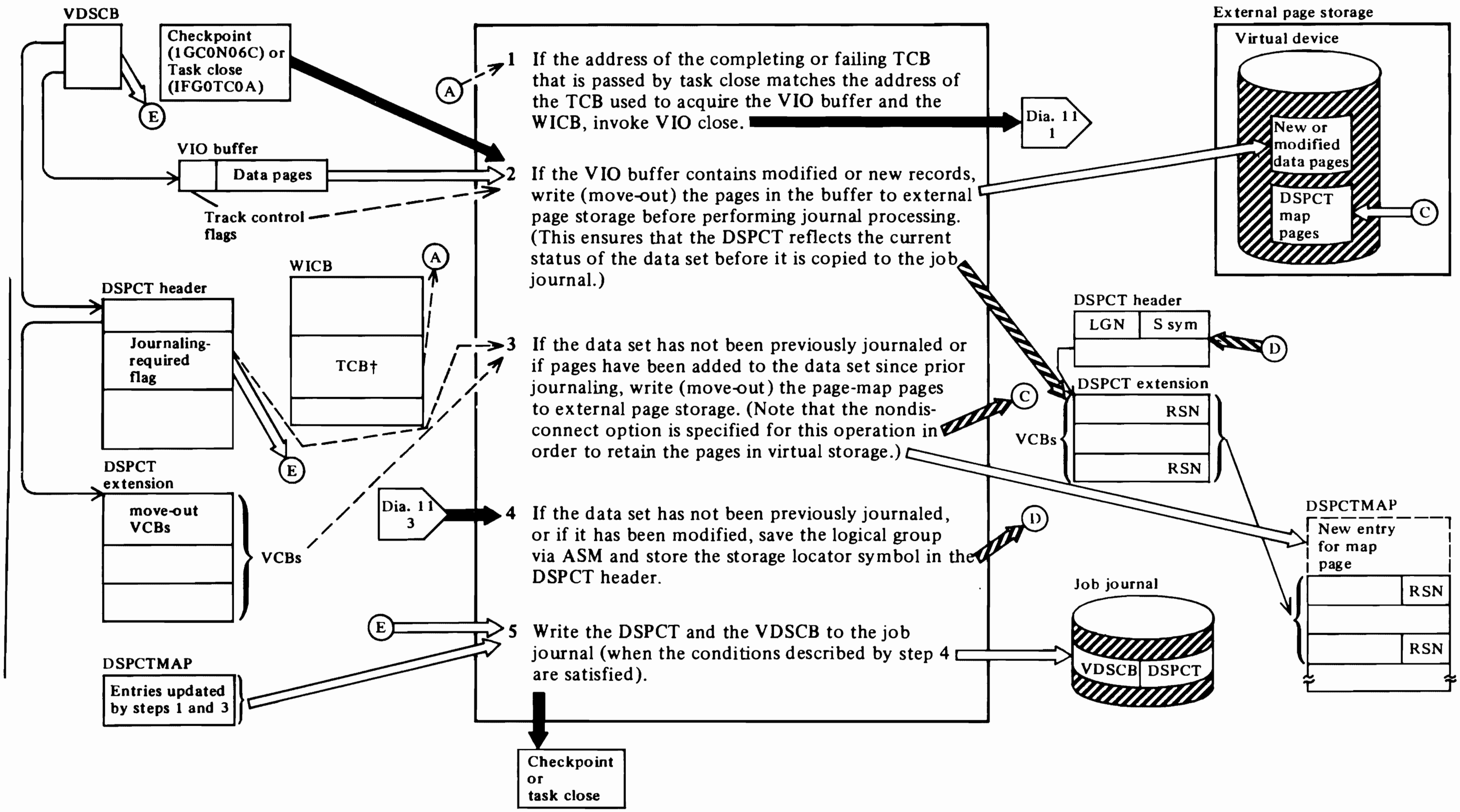
The seek command is used to acquire a new track for a channel program. The address of the BBCCHH of the desired track is contained in the seek CCW. If the desired track is not the track that is currently in the VIO buffer, IDDWICPI returns control to IDDWITRM in order to obtain the desired track. After the desired track has been connected (assigned) to the virtual buffer, control is returned to IDDWICPI to process the command following the seek command.

Note: All notes pertaining to Diagram 9 involve module IDDWICPI.

1. See comments above under "Sense Commands."
2. Test the file mask in the DEB to determine whether a seek command is allowed. When a seek command is disallowed by the file mask for the VIO data set's extent, set a file-mask violation indicator in the IOB to reflect the status of the channel program.

   When the seek address is invalid, terminate the channel program with a channel program error.
3. If the desired track is the track currently contained by the VIO buffer, continue processing with the simulated execution of the next CCW.
4. **IDDWICPI returns to IDDWITRM:** IDDWITRM causes pages in a new track to be connected (assigned) to the VIO buffer, and then returns control to IDDWICPI in order to resume simulated execution of the channel program.

Diagram 10. Save a VIO Data Set for a Restart



1. If the address of the completing or failing TCB that is passed by task close matches the address of the TCB used to acquire the VIO buffer and the WICB, invoke VIO close.
2. If the VIO buffer contains modified or new records, write (move-out) the pages in the buffer to external page storage before performing journal processing. (This ensures that the DSPCT reflects the current status of the data set before it is copied to the job journal.)
3. If the data set has not been previously journaled or if pages have been added to the data set since prior journaling, write (move-out) the page-map pages to external page storage. (Note that the nondisconnect option is specified for this operation in order to retain the pages in virtual storage.)
4. If the data set has not been previously journaled, or if it has been modified, save the logical group via ASM and store the storage locator symbol in the DSPCT header.
5. Write the DSPCT and the VDSCB to the job journal (when the conditions described by step 4 are satisfied).

Notes for Diagram 10

During task-close processing, or when a checkpoint routine encounters a UCB for a virtual device, the VIO journaling (checkpoint) function is invoked by a call to IDDWIJRN via the WIJOURN macro to do the following:

- to ensure that all VIO data sets in external page storage for the current job step are current (that all pending output is completed).
- to save the status of logical groups by causing ASM to save a map of the data set, the ASPCT (address space page correspondence table), in the SYS1.STGINDEX data set. (This information is retrievable at restart via the storage locator symbol passed back by ASM and stored by VIO in the journaled copy of the DSPCT.)
- to write VIO data set control blocks (VDSCBs and DSPCTs) to the job journal.

The journaled records are used by a VIO restart routine to regain access to a VIO data set. The data set status is saved only if it has never been done before, or if data has been added or modified.

When IDDWIJRN is entered, it determines whether journaling can be done. If the JSCB (job step control block) indicates that errors exist in the job journal (for example, an I/O error occurred when writing records to the job journal) or that the job journal has not been activated for the current job, VIO journaling is not performed and control is returned to the caller, the checkpoint or task close routine. Journaling is also not performed when the ASCB indicates that the current job is a time sharing job or when in step termination, the data set is to be deleted.

To determine which data sets are to be journaled, IDDWIJRN examines each UCB associated with the current job step to determine whether the UCB identifies a virtual device. The UCBs are located by TIOT entries which are pointed to by DSABs (data set association blocks). Journal processing is performed on each virtual UCB, and the processing continues until all UCBs representing virtual devices have been journaled.

1. This condition occurs only in a situation involving program fetch. When a STEPLIB DD statement specifies a VIO data set, the data set is opened and closed under the initiator's TCB even though I/O involving the data set is performed by program fetch under a job step TCB. If—when the job step is completed—task close invoked the VIO journaling function (module IDDWIJRN), VIO close processing must be invoked to free VIO data areas and to zero-out the VIO-buffer pointer in the VDSCB. (If this close processing was not performed, when the initiator's close processing occurs, an attempt would be made to write the nonexistent VIO buffer pointed to by the VDSCB.)

2. **IDDWIJRN calls IDDWITRM:** If a status flag in the VIO buffer indicates that the buffer contains new or modified records, IDDWITRM issues a VREADWR macro to write (move-out) the buffer contents. (For a description of processing associated with the VREADWR macro, see the notes for Figure 3 in the "Program Organization" section of this manual.)

3. **IDDWIJRN:** If DSPCT page-map pages exist in storage and if the data set was modified, then the page-map pages are written to auxiliary storage.

   IDDWIJRN calls RSM, module IEAVAMSI: RSM writes DSPCT page-map pages to the VIO data set in auxiliary storage. With the nondisconnect option, the page table entries governing the real storage page frames that contain the DSPCT are not modified; the page frames remain in real storage associated with the VIO user's address space identifier (ASID).

Notes for Diagram 10 (Continued)

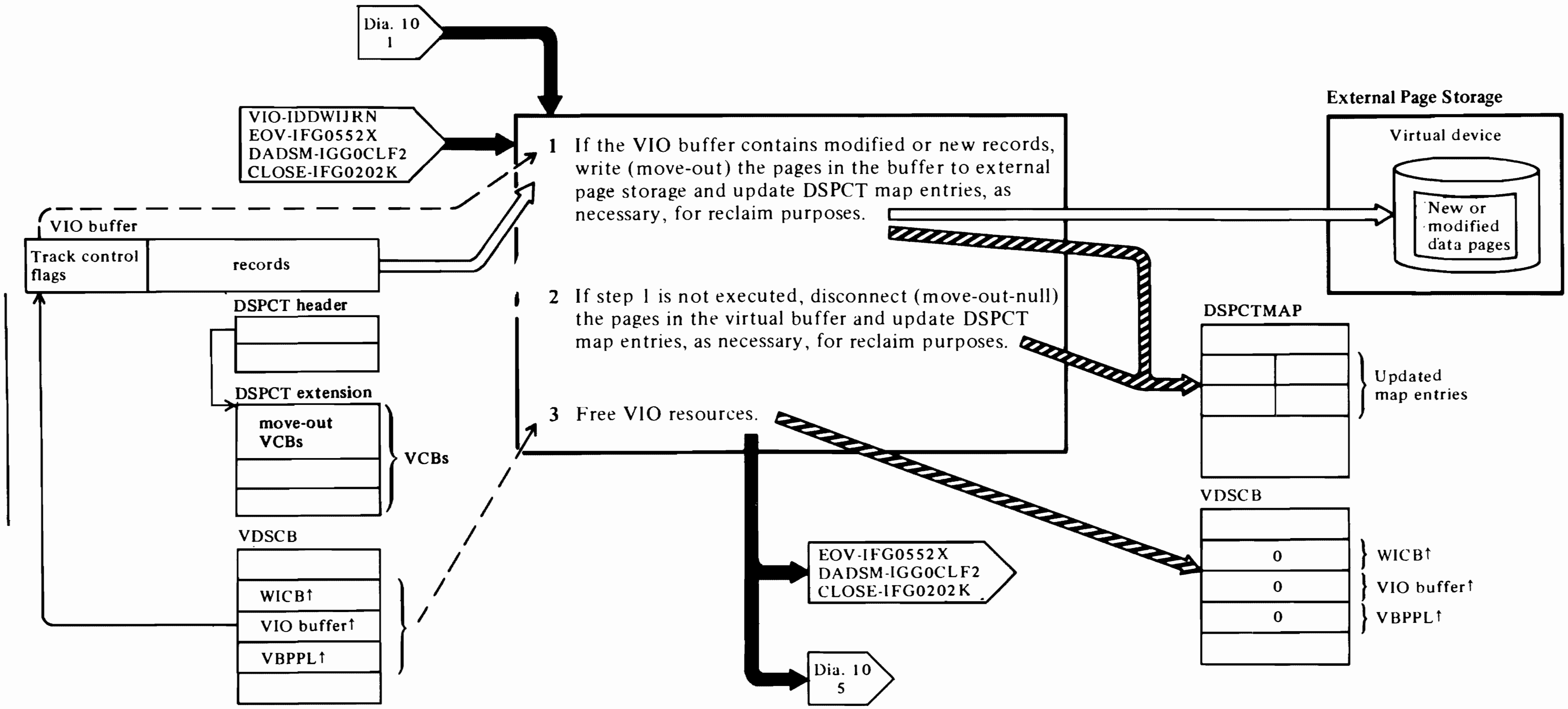
4. **IDDWIJRN calls ASM, module ILRINT00:** If the data set has not been journaled or if pages have been modified or added since the data set was last journaled, control is passed to ASM so that it can save its control records that reflect the current status of the data set. ASM returns a storage locator symbol ("S" symbol) which is used in recovering the data set if either a system failure or job step failure occurs. If ASM's attempt to create a storage locator symbol is unsuccessful or if ASM returns a symbol that differs from the symbol it returned previously for the same data set, IDDWIJRN issues an ABEND macro with a 0E7 completion code. The "S" symbol is saved in the DSPCT header.

6. **IDDWIJRN calls to IEFXB500 (scheduler's journal-write module):** Write the DSPCT header (if it exists) and VDSCB to the job journal.

Notes: A 0E7 ABEND results in control being passed to the ESTAE routine which records the error on the SYS1.LOGREC and SYS1.DUMP data sets. Control is then passed to the mainline module which gives a nonzero return code if called by the checkpoint module. The checkpoint module then issues a message to the operator and terminates the checkpoint. A zero return code is always passed to task close.

During restart processing after all blocks are merged into the SWA, the VDSCB and DSPCT are written to the job journal by the scheduler.

Diagram 11. Perform VIO Close Processing



Notes for Diagram 11

Common close invokes VIO close processing, to allow VIO to free its resources, whenever it closes the last active DCB for a VDSCB. (Note that the VDSCB and DSPCT are retained between job steps in the event that another DCB is opened against the VDSCB. The DSPCT is released when the data set is scratched. The VDSCB is released when the current job is terminated.)

For a description of the conditions that exist when EOV or DADSM invoke VIO close processing, see the notes to Figure 9 in the "Program Organization" section.

1. **IDDWICLS:** Call IDDWITRM before freeing the VIO buffer and perform the following processing:

   IDDWITRM: Determine whether the VIO buffer has been updated, that is, determine whether it contains new or modified records. If so, set write-operation parameters in the buffer control block and call IDAVBPP1 via the VREADWR macro. IDAVBPP1 performs the following processing:

   Sets control information in the VCBs that correspond to the pages to be written to external page storage.

   Calls RSM, module IEAVAMSI, to cause the pages in the VIO buffer to be paged out.

   Moves the RSNs in the VCBs into their corresponding DSPCT page-map entries for possible use in reclaiming the page frames to satisfy a subsequent read (assign) request.

2. **IDDWICLS calls IDAVBPC1 via VCLOSE macro:** Determine whether close processing has been invoked directly following a restart. If so, return control to IDDWICLS.

   If there is data in the window from a prior assign operation, then disconnect the pages from the window via the move-out-null function. The following processing is performed to disconnect the pages:

   IDAVBPC1 calls RSM, module IEAVAMSI: By modifying page table entries, disconnect the pages assigned to the VIO buffer from the VIO user's address space.

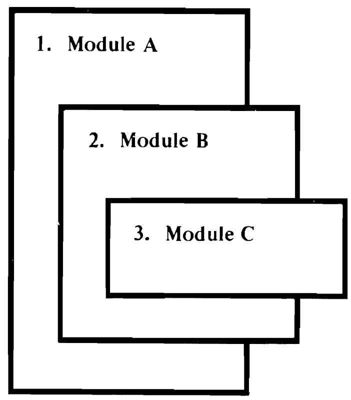
3. **IDDWICLS:** Free the VIO buffer, WICB, and VBPPL by issuing FREEMAIN macro instructions.

PROGRAM ORGANIZATION

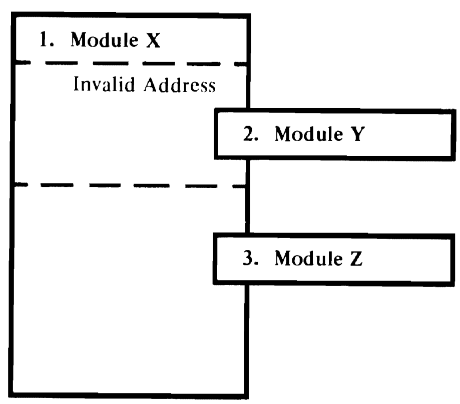
The compendiums (or hierarchial tables) used in this section do not usually show you all of the exits and entrances to a given module. A compendium depicts the flow of control, among modules, that relates to a particular function.

How to Read Compendiums

Each block in a compendium is associated with a module name. The blocks are nested (indented) to show the sequence of calls. For example, in the following diagram, module "A" at some point in its processing calls module "B," and module "B" at some point in its processing calls module "C." Unless otherwise indicated by an exit indicator, the called modules always return control to the calling modules in the sequence in which they are called.



In many instances, a call to a module is conditional. In these cases, the condition that must be met is shown in the block that represents the calling module. Dotted lines delineate the calls that are affected by a given condition. This is illustrated in the following example:



In this example, module "Y" is called when an invalid address is detected, and module "Z" is always called. In either case, module "Y" and "Z" always return control to module "X."

Each module is numbered to key it to the extended descriptions on the page that faces the diagram. The extended descriptions provide details about the conditions that exist when a call is made and about the general processing that is performed by the modules.

For a description of ABEND completion codes shown in the figures, see the "Diagnostic Aids" section.

Module-Calls Directory (Forward and Backward Reference)

The "module-calls directory" allows you to relate each VIO module and external procedure to the modules that it is called by and to the modules that it calls.

Module Name	Title	Calling Modules	Modules Called*
IDAVBPC1	VBP Close Module (entered via call to VCLOSE)	IDDWICLS	IDAVBPR2 (MONWINDO)
IDAVBPJ2	VBP Restart Module (activate/journal-merge function)	IEFXB601 (scheduler)	ACTIVATE LG (ILRINT00)
IDAVBPO1	VBP Open Module (entered via call to VOPEN)	IDDWIAPP	IDAVBPR2 (CHEKMPPG), IEAVAMSI (CVTVSI), IDAVBPR2 (SETVCB), ASSIGN LG (ILRINT00)
IDAVBPP1	VBP Record Management Module (entered via call to VREADWR)	IDDWITRM	IDAVBPR2 (CHEKMPPG), IEAVAMSI (CVTVSI)
IDAVBPR1	VBP Functional Recovery	Recovery/Termination Manager	SDUMP (SVC branch entry)
IDAVBPR2	VBP Common Routines Internal Procedures:	—	
	CHEKMPPG	IDAVBPO1, IDAVBPP1	IEAVAMSI (CVTVSI)
	MONWINDO	IDAVBPC1	IEAVAMSI (CVTVSI)
	SETVCB	IDAVBPO1	—
IDAVBPS1	VBP Scratch (entered via call to VSCRATCH)	IGG0002I (DADSM scratch)	RELEASE LG (ILRINT00)
IDDWIAPP	EXCP Intercept Appendage Simulator	IECVEXCP	Appendages, IDAVBPO1 (entered via call to VOPEN), IEASMFEX, IDDWITRM
IDDWICLS	EXCP Intercept Close Function (entered via call to WICLOSE)	IFG0552X (EOV), IGG0CLF2 (DADSM), IFG0202K (Close), IDDWIJRN	IDDWITRM, IDAVBPC1 (entered via call to VCLOSE)

*The name of the procedure called is enclosed in parentheses following the module name.

Module Name	Title	Calling Modules	Modules Called*
IDDWICPI	Channel Program Interpreter	IDDWITRM	—
IDDWIFRR	Functional Recovery Routines	Recovery/Termination Manager	SDUMP (SVC and branch entry)
IDDWIJRN	EXCP Intercept Journaling	IFG0TC0A (Task Close), IGC0N06C (Checkpoint)	IDDWITRM, IEFXB500, IDDWICLS, IEAVAMSI (CVTVSI), IEFXB500, SAVELG (ILRINT00)
IDDWIMRG	EXCP Intercept Merge Function	IEFXB601 (Scheduler)	—
IDDWITRM	EXCP Intercept Track Manager	IDDWIAPP, IDDWICLS, IDDWIJRN	IDDWICPI, IDAVBPP1 (entered via call to VREADWR)
Non-VIO modules that either invoke or are invoked by VIO:			
IEASMFEX	SMF Routine	IDDWIAPP	—
IEAVAMSI	RSM VIO Services Interface	IDAVBP01, IDAVBPP1, IDAVBPR2, IDDWIJRN	—
IECVEXCP	EXCP Processor (IOS)	—	IDDWIAPP
IEFXB500	Scheduler	IDDWIJRN	—
ILRINT00	Auxiliary Storage Manager (ASM)	6,8,10,11,12	3
IEFXB601	Scheduler	—	IDAVBPJ2, IDDWIMRG
IFG0TC0A	Task Close	—	IDDWIJRN
IFG0202K	Common Close	—	IDDWICLS
IFG0552X	End-of-Volume	—	IDDWICLS
IGC0N06C	Checkpoint	—	IDDWIJRN
IGC0002I	DADSM Scratch	—	IDAVBPS1
IGG0CLF2	DADSM	—	IDDWICLS
ILRINT00	Auxiliary Storage Manager (ASM)	IDDWIJRN IDAVBPJ2 IDAVBPO1 IDAVBPS1	

*The name of the procedure called is enclosed in parentheses following the module name.

Program Organization Compendiums

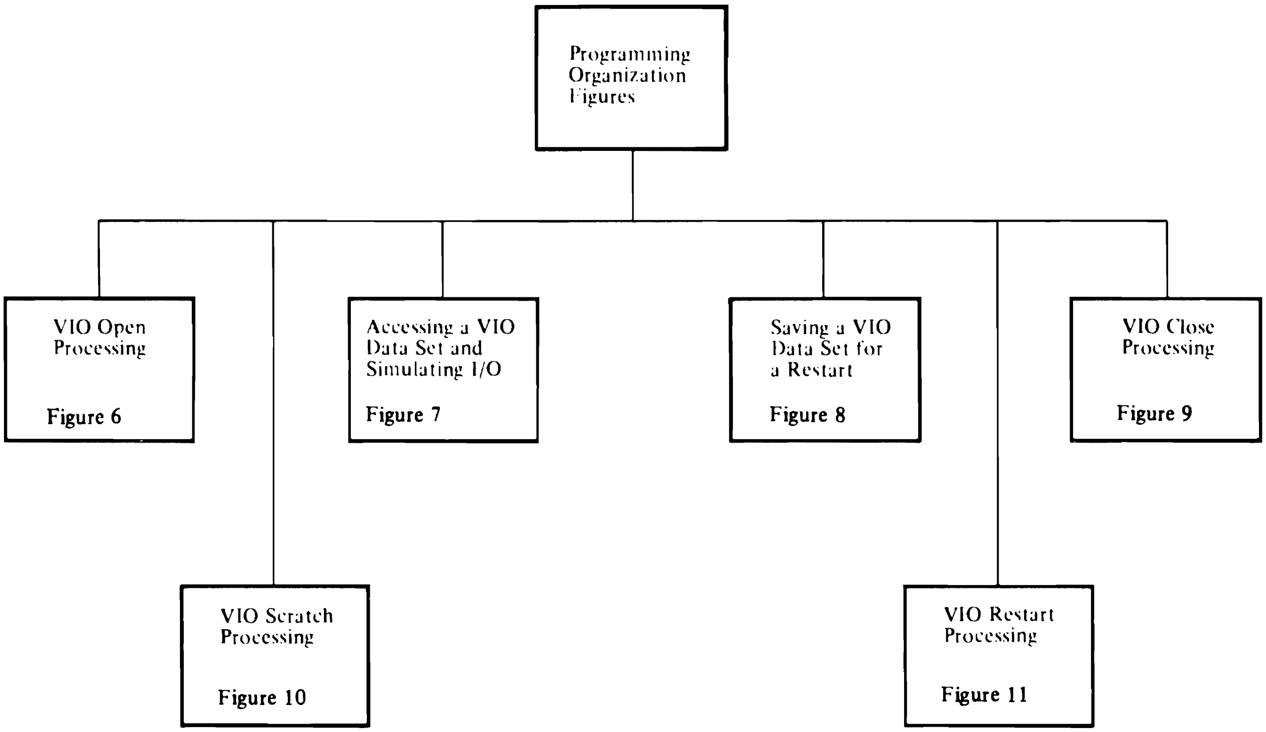


Figure 5. Table of Contents for Program-Organization Figures

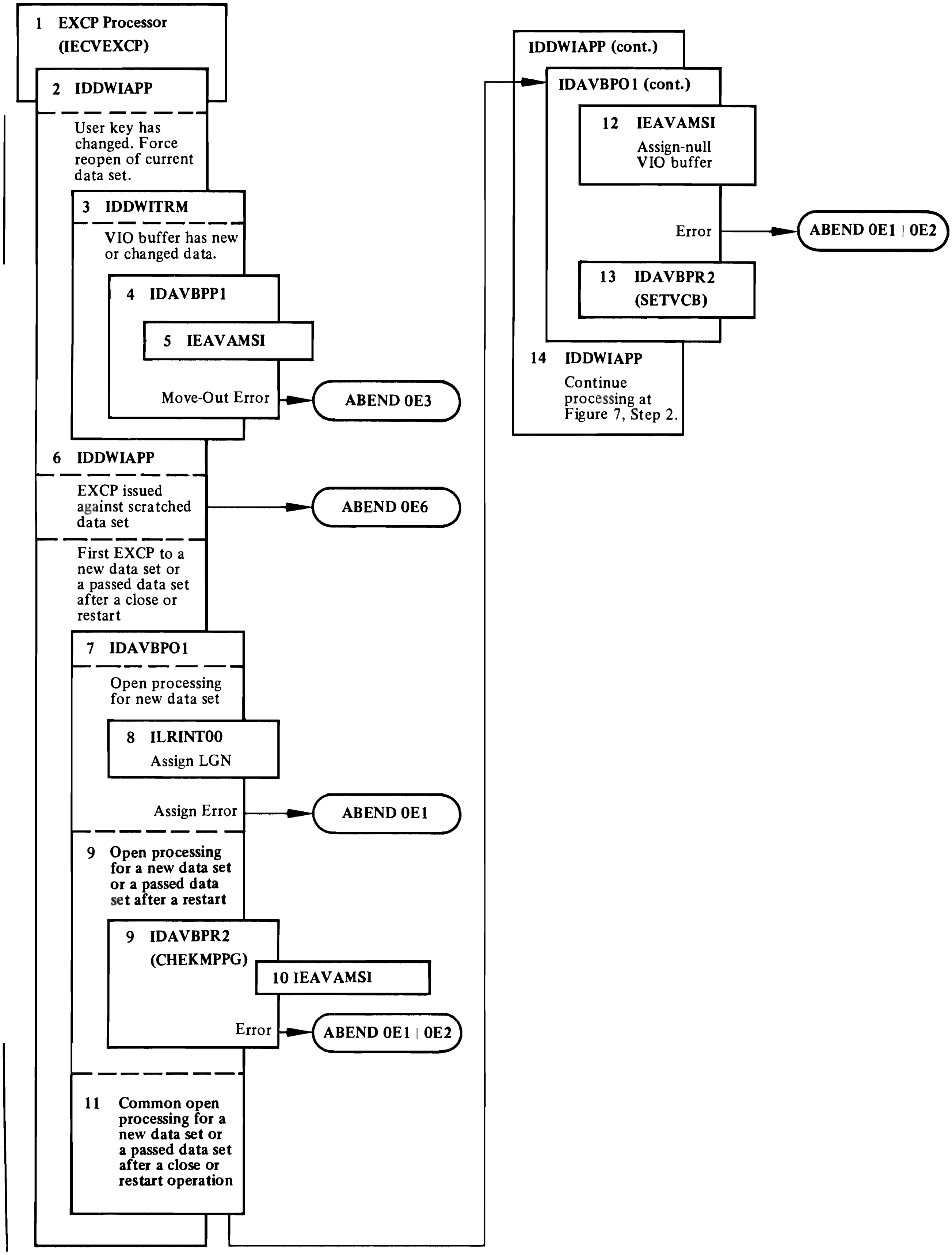


Figure 6. VIO Open Processing

Notes for Figure 6

1. The EXCP processor builds RQEs. The RQE consolidates control information that is provided by the VIO user and needed by VIO to complete its processing. (See *OS/VS2 I/O Supervisor Logic* for details about the EXCP processor.)

   Then, if the EXCP processor determines via a VIO indicator in the UCB that the EXCP request is directed against a VIO data set, it passes control to IDDWIAPP.

2. IDDWIAPP ensures that the key of the new request matches the current user's key and that VIO open processing has been performed.

   If either of these conditions is not satisfied, IDDWIAPP performs the following processing:

   - When a key conflict exists, IDDWIAPP calls IDDWITRM to ensure that the current pages in the VIO buffer are written to the VIO data set in external page storage if the pages contain new or modified records. (See the introductory notes to Diagram 3 in "Method of Operation" for a description of the situation in which a key conflict can exist.)

     Upon return from IDDWITRM, IDDWIAPP frees the WICB and the VIO buffer and sets an indicator to invoke VIO open processing.

   - If VIO open processing has not been previously performed for the new request string, IDDWIAPP acquires a VBPPL, WICB, and VIO buffer. (Note that when VIO open processing is invoked as a result of a key change, only the WICB and the VIO buffer are reacquired; the VBPPL is reused if it was previously acquired.)

   In either case, IDDWIAPP then initializes the WICB with the characteristics of the virtual device and calls IDAVBPO1 via the VOPEN macro (see step 9).

3. When a key conflict exists, IDDWIAPP calls IDDWITRM. IDDWITRM determines whether the pages in the VIO buffer contain new or modified records, and if so, calls IDAVBPP1 via the VREADWR macro to write the pages in the VIO buffer to the VIO data set in external page storage. Otherwise, if the VIO buffer contents have not been modified, IDDWITRM simply returns to IDDWIAPP.

4. IDAVBPP1 initializes write (move-out) VCBs to be used by RSM in writing the pages in the buffer to the VIO data set in auxiliary storage.

   IDAVBPP1 calls RSM, module IEAVAMSI, to cause the pages in the buffer to be written to auxiliary storage. If the write operation is successful, IDAVBPP1 updates the RSNs (real storage numbers) in the DSPCT page-map entries for the pages written to auxiliary storage so that their real storage page frames may be reclaimed by a subsequent read (assign) operation.

6. IDDWITRM returns to IDDWIAPP, and before continuing VIO open processing, IDDWIAPP frees the WICB and the VIO buffer by issuing FREEMAIN macro instructions and then reacquires them by issuing GETMAIN macro instructions. A VBPPL is acquired for a new or a passed data set.

   IDDWIAPP invokes further VIO open processing by calling IDAVBPO1 via the VOPEN macro.

7-8. For a new data set, IDAVBPO1 acquires a DSPCT header, initializes it, and then issues a call to ASM, module ILRINT00, to assign an LGN (logical group number) for the data set. IDAVBPO1 stores the LGN, which is returned by ASM, in the DSPCT header.

9-10. For new data sets and for passed data sets after a restart, IDAVBPO1 calls IDAVBPR2 (CHEKMPPG). If the data set is new, CHEKMPPG connects (assign-null function) a DSPCT page-map page to the VIO user's address space. If the data set is being reactivated (accessed after a restart), CHEKMPPG (assign function) reads the page-map pages, which were written to auxiliary storage by a prior journaling operation, into the address space associated with DSPCT page-map pages.

11-12. For all VIO data sets, IDAVBPO1 calls IEAVAMSI to connect (assign-null) the buffer pages to the VIO user's address space.

13. IDAVBPO1 calls IDAVBPR2 (SETVCB) to initialize the read (assign), write and disconnect (move-out), and disconnect (move-out-null) VCBs for subsequent request processing.

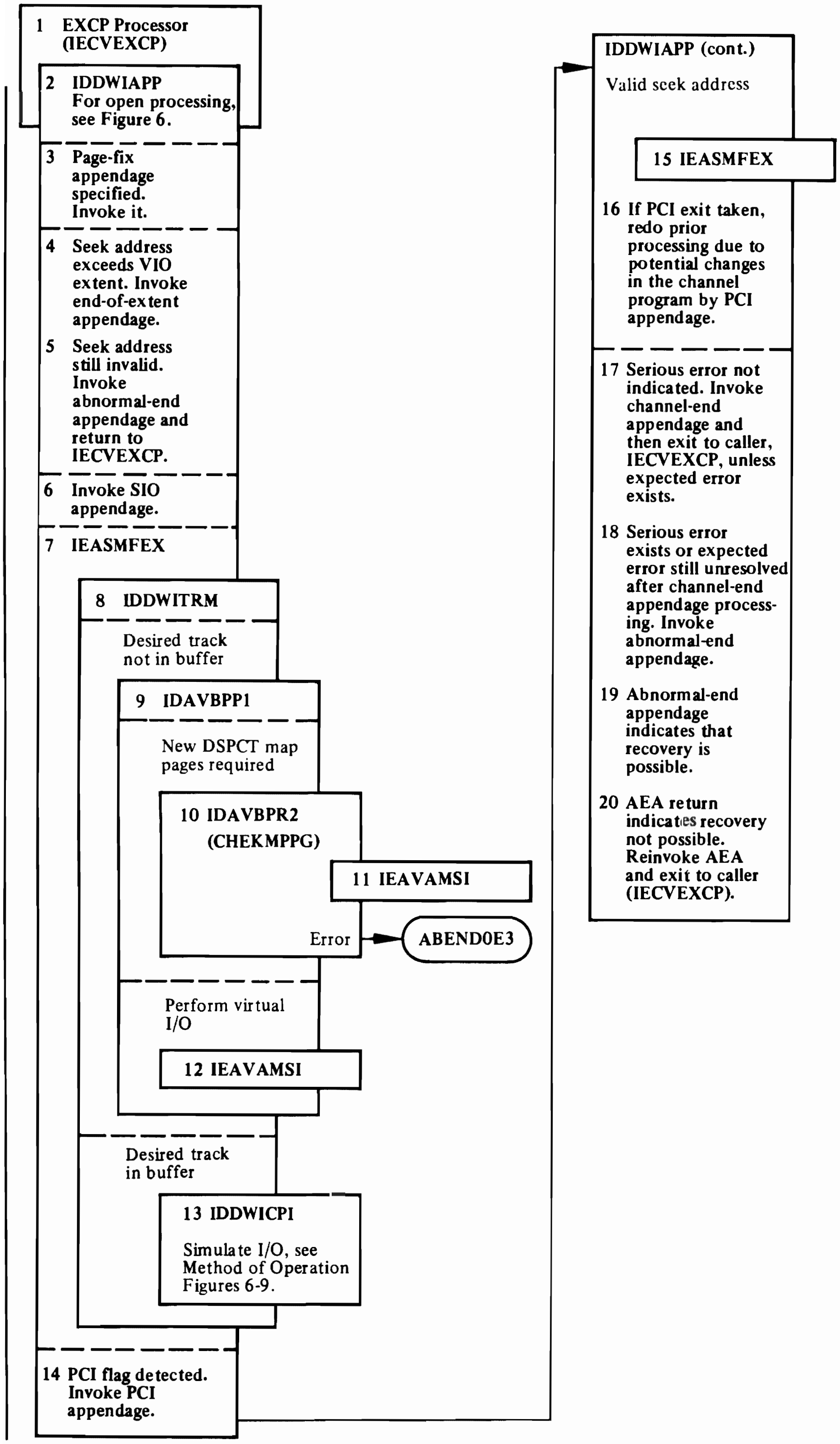


Figure 7. Accessing a VIO Data Set and Simulating I/O

Notes for Figure 7

1. The EXCP processor builds a request queue element (RQE). The RQE consolidates control information that is provided by the VIO user and needed by VIO to complete its processing. (See *OS/VS2 I/O Supervisor Logic* for details about the EXCP processor.)

   After building the RQE, the EXCP processor passes control to VIO when it determines that processing is directed to a VIO data set on a virtual device.

2. IDDWIAPP controls VIO open processing (see Figure 6) and controls the simulated execution of a channel program in conjunction with user-supplied page-fix, end-of-extent, start-I/O, channel-end, abnormal-end, and program-controlled-interrupt appendages (see *OS/VS2 System Programming Library: Data Management* for a description of appendage interfaces and functions).

   For details on the flow of control following returns from the appendages, see Diagram 4 in "Method of Operation."

   Conditions under which IDDWIAPP invokes user-supplied appendages or other VIO modules are as follows:

   3. Invoke the page-fix appendage once per EXCP request if the user provides the appendage. Pass an empty list of pages; no page-fixing is performed.

   4. Invoke the end-of-extent appendage when the device address associated with the EXCP request exceeds the address range of the VIO data set's extent.

   5. Invoke the abnormal-end appendage if an abnormal error condition exists. Abnormal error conditions are unit check, program check, protection check, and channel control check.

   6. Invoke the start-I/O appendage once per EXCP unless an attempt is made to re-execute a channel program following the correction of an error condition. If the current processing represents an effort to resume processing of an EXCP request after a PCI exit has been taken, the SIO appendage is not invoked.

   7. IDDWIAPP links to IEASMFEX (MF/1 EXCP count routine) to record PCI and reEXCP requests.

   8. Call IDDWITRM to determine whether the desired track currently resides in the VIO buffer. IDDWITRM calls IDAVBPP1 to manipulate, if necessary, the track in the VIO buffer to meet the address requirements of the request. IDDWITRM then invokes IDDWICPI to interpret and simulate execution of the channel program.

9-11. If a new DSPCT page-map page is required to contain entries for new data set pages, IDAVBPP1 initializes a VCB and then calls IDAVBPR2 (CHEKMPPG) to acquire (assign-null) a new page of virtual storage to contain DSPCT page-map entries.

10-11. IDAVBPR2 (CHEKMPPG) issues a GETMAIN macro instruction to acquire a DSPCT page-map page and then calls IEAVAMSI to connect (assign-null) the page to the VIO user's address space.

12. IDAVBPP1 initializes VCBs that control the paging I/O to be performed on the pages in the VIO buffer. Then, IDAVBPP1 calls IEAVAMSI to perform the necessary write (move-out), or disconnect (move-out-null), and read (assign) operations.

13. IDDWICPI interprets the channel program associated with an EXCP request and simulates execution of the channel program by moving and comparing data between the VIO buffer and the VIO user's buffer(s).

14. IDDWIAPP calls the PCI appendage if IDDWICPI (see step 19) encountered a PCI flag in a CCW it is processing and set a corresponding flag in the CSW associated with the channel program.

15. If the PCI appendage was invoked, IDDWIAPP calls IEASMFEX to record the event and returns to step 19 to process the next CCW.

16. IDDWIAPP reinvokes IDDWITRM (go to step 8).

17. IDDWIAPP calls the channel-end appendage unless a serious error condition is indicated in the CSW. Serious errors are unit check, program check, protection check, or channel check.

    Expected errors indicated in the CSW are unit exception and incorrect length.

18. IDDWIAPP calls abnormal-end appendage when a serious or expected error condition exists (see note 23).

19. When the return from the abnormal-end appendage indicates that recovery is possible, IDDWIAPP attempts to reexecute the CCW, starting with the test to ensure validity of the seek address (go to step 4).

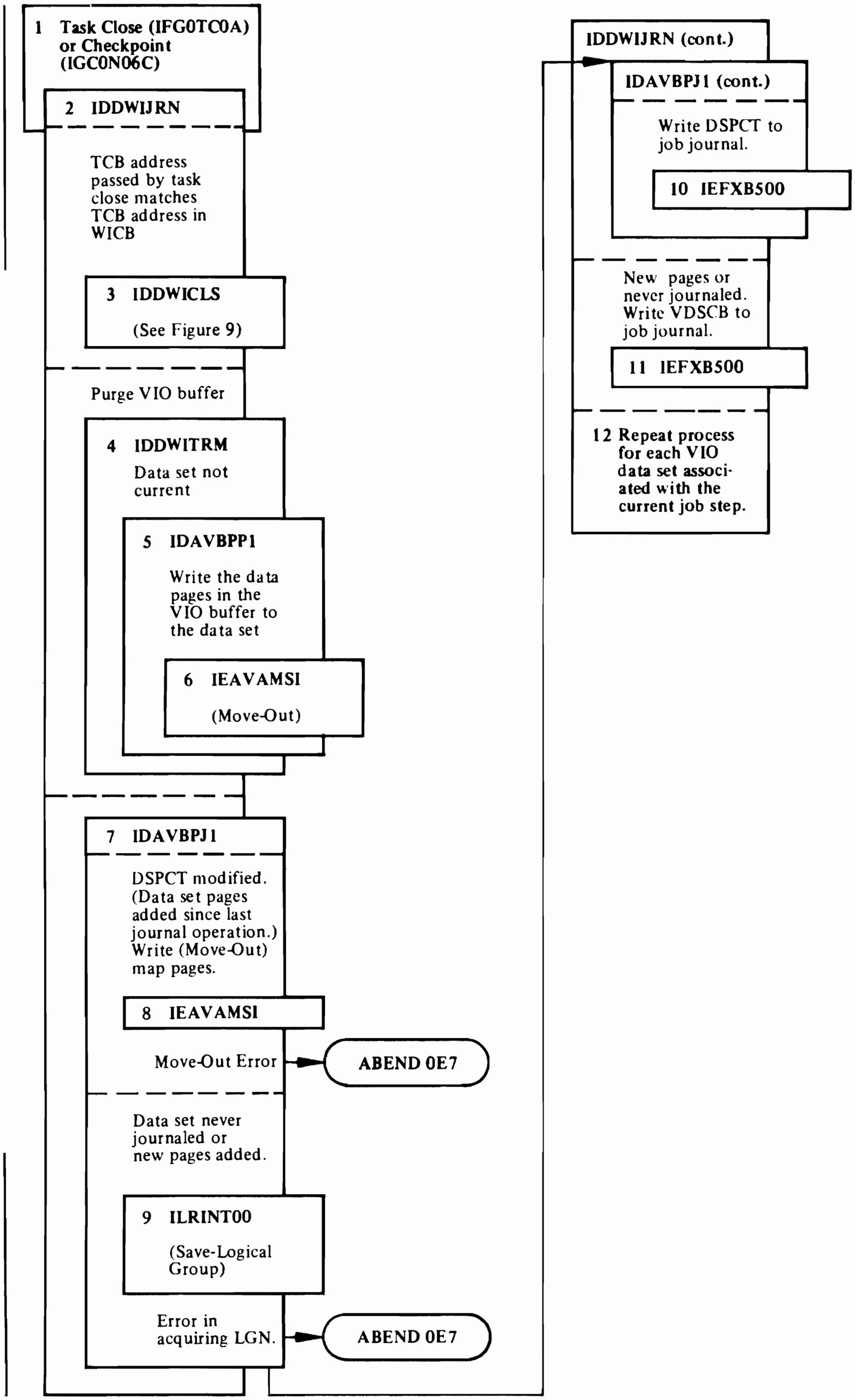


Figure 8. Saving a VIO Data Set for a Restart

Notes for Figure 8

1. During task-close processing (abnormal or normal job step termination) and when the checkpoint routine encounters a UCB for a virtual device, VIO journal processing is invoked. Note that for abnormal termination due to a task or system failure, task close subsequently invokes VIO abnormal termination processing (see Figure 11). VIO journal processing ensures that all VIO data sets associated with the current job step are current, saves the logical groups via a call to ASM, and then writes the DSPCT(s) and VDSCB(s) for the VIO data set(s) to the job journal for recovery purposes if they have not been previously journaled or if they have been altered since the last journaling.

2. IDDWIJRN calls IDDWICLS (see Figure 9) if the TCB address passed by task close matches the TCB address in the WICB. If this condition does not exist, IDDWIJRN calls IDDWITRM when the VDSCB contains a pointer to the VIO buffer. IDDWITRM ensures that all output to the VIO data set is completed and that the DSPCT page-map is current before journaling is performed.

3. This condition occurs only in a situation involving program fetch. When a STEPLIB DD statement specifies a VIO data set, the data set is opened and closed under the initiator's TCB even though I/O involving the data set is performed by program fetch under a job step TCB. If—when the job step is completed—task close invokes the VIO journaling function (module IDDWIJRN), VIO close processing must be invoked to free VIO data areas and to zero-out the VIO-buffer pointer in the VDSCB. (If this close processing was not performed, when the initiator's close processing occurs, an attempt would be made to write the nonexistent VIO buffer pointed to by the VDSCB.)

4. IDDWITRM tests the status of the pages in the VIO buffer to determine whether they contain new or modified data. When the buffer contains new or modified data, IDDWITRM issues a VREADWR macro to write (move-out) the pages in the buffer to the VIO data set in external page storage. Otherwise, IDDWITRM returns control to IDDWIJRN.

5-6. IDAVBPP1 initializes move-out VCBs for use by RSM, module IEAVAMSI.

IDAVBPP1 calls IEAVAMSI to cause the pages to be written to the VIO data set in external page storage.

7-8. If any DSPCT page-map entries have been created or updated since the last journaling operation or if journaling has not been performed, IDDWIJRN calls IEAVAMSI to write (move-out) the DSPCT page-map to auxiliary storage.

9. If the data set has not been journaled or if pages have been added or modified to the data set, IDDWIJRN calls ASM, module ILRINT00, to write ASM's control records to the SYS1.STGINDEX data set.

IDDWIJRN stores the storage-locator symbol returned by ASM in the DSPCT header.

10. If the data set has not been previously journaled or if new pages have been added since a prior journaling, IDDWIJRN calls IEFXB500 to write the DSPCT header to the job journal for recovery purposes.

If a write to the job journal is unsuccessful, a return code is passed to the checkpoint routine IGC0N06C if originally called by checkpoint. Otherwise, job journaling is bypassed if it was originally called by task close.

11. **When the conditions stated for step 10 exist, IDDWIJRN calls IEFXB500 to write the VDSCB to the job journal.**

Note: All abnormal terminations are intercepted in the ESTAE routine, IDDESTJR (IDDWIJRN), and, after recording on the SYS1.LOGREC and SYS1.DUMP data sets, control is returned to the mainline module (IDDWIJRN) to pass a nonzero return code if the invoker is checkpoint. Task close is always passed a zero return code.

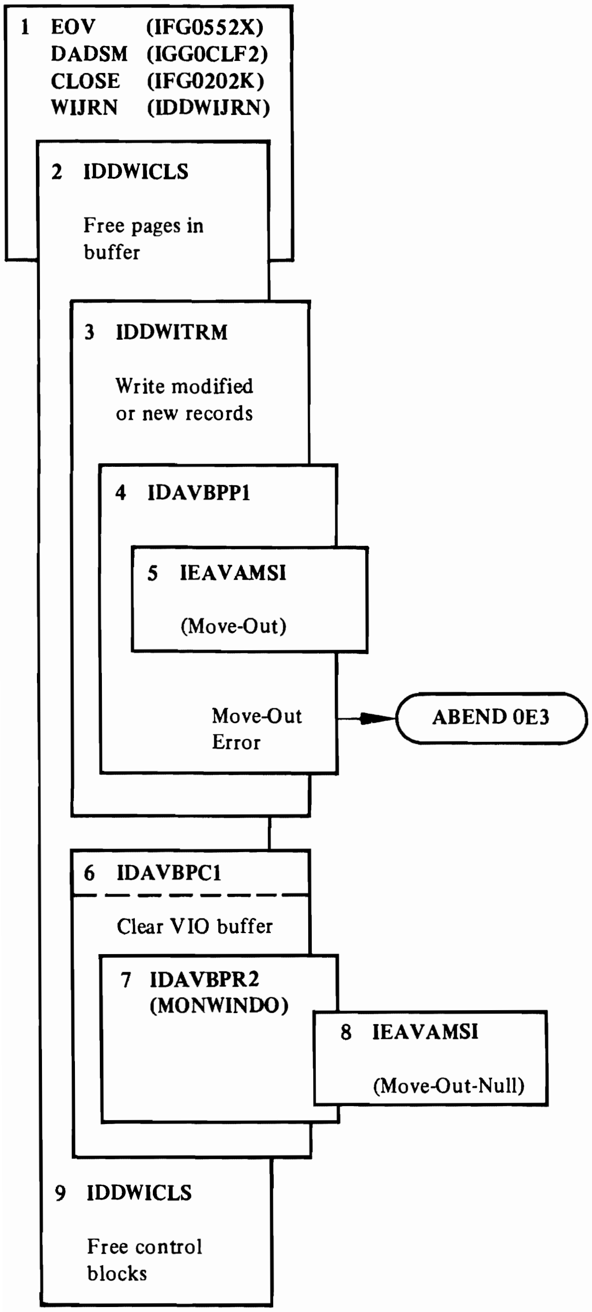


Figure 9. VIO Close Processing

Notes for Figure 9

1. The following modules invoke VIO close processing with a WICLOSE macro.

   IDDWIJRN In a journaling situation, if the TCB address passed by task close matches the TCB address in the WICB, IDDWIJRN invokes IDDWICLS. (See the notes to Figure 8 for an extended description of the conditions under which this situation occurs.)

   IFG0202K If common close determines that the UCB associated with the DCB that it is closing is for a virtual device and if the data management count in the UCB is zero, close invokes VIO close processing to free VIO resources.

   IGG0CLF2 If DADSM formats PDS (partitioned data set) directories for a BPAM data set, EXCPs are issued and processed by VIO. When PDS formatting is completed, DADSM invokes VIO close processing to free VIO resources.

   IFG0552X If EOV encounters concatenated VIO data sets with like attributes and if the data management count in the UCB is zero, EOV invokes VIO close processing to free resources associated with the current data set. If the VIO data sets have unlike attributes, IFG0552X gives control to common close, module IFG0202K. Close then calls VIO close when the data management count in the UCB is zero.

2. IDDWICLS calls IDDWITRM to ensure that new or modified data in the VIO buffer is written to the VIO data set.

   IDDWICLS calls IDAVBPC1 to disconnect the pages in the VIO buffer from the VIO user's address space.

   IDDWICLS frees the VIO buffer, WICB, and VBPPL by issuing FREEMAIN macro instructions.

3. IDDWITRM determines whether pages in the current VIO buffer contain new or modified data, and if so, calls IDAVBPP1 via the VREADWR macro to cause the buffer to be written to auxiliary storage and then disconnected from the VIO user's address space.

4. VCBs are used by IDAVBPP1 to pass control information to RSM. Using the parameters that IDAVBPP1 establishes in the VCBs, RSM causes the contents of the VIO buffer to be written to the VIO data set in auxiliary storage.

5. IDAVBPP1 calls RSM, module IEAVAMSI, to cause the pages in the VIO buffer to be written to external page storage.

6-8. IDAVBPC1 determines whether the page table entries associated with the pages in the VIO buffer need to be cleared. If so, it calls IDAVBPR2 (MONWINDO), which then calls RSM, module IEAVAMSI, to disconnect (move-out-null) the pages in the VIO buffer from the VIO user's address space.

9. IDAVBPC1 returns to IDDWICLS, and IDDWICLS issues FREEMAIN macro instructions to free the VIO buffer, WICB, and VBPPL.

Note: The functional recovery modules are entered because of errors which occurred while VIO close was executing. The incident is then recorded on the SYS1.LOGREC and SYS1.DUMP data sets and control is returned to the system recovery manager for further recovery attempts.

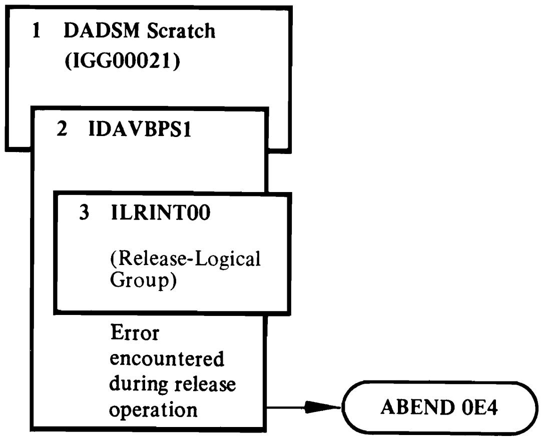


Figure 10. VIO Scratch Processing

Notes for Figure 10

1. When DADSM scratch processing is directed against a VIO data set, IGC0002I invokes VIO scratch processing to free auxiliary storage and control blocks associated with the VIO data set.

2. IDAVBPS1 calls ASM, module ILRINT00, to release the VIO data set's auxiliary storage. If the data set has been journaled, a storage-locator symbol (or "S" symbol) is passed to ASM. Otherwise, an LGN (logical group number) is passed.

   When ASM returns an error code to IDAVBPS1, IDAVBPS1 issues an ABEND macro instruction with an X'0E4' completion code.

   IDAVBPS1 releases the DSPCT extension, the DSPCT page-map pages, and the DSPCT header by issuing FREEMAIN macro instructions.

   The VDSCB is not released by scratch processing; it is modified to indicate that the data set has been scratched and that the DSPCT has been released.

Note: The 0E4 ABEND is intercepted in the FRR routine VBPS1FRR (IDAVBPR1) and, after the information is recorded on the SYS1.LOGREC and SYS1.DUMP data sets, control is returned to IDAVBPS1. At the end of IDAVBPS1 processing, a nonzero return code is passed to DADSM.

The information on this page has been deleted by OS/VS2 MVS Supervisor Performance #2.

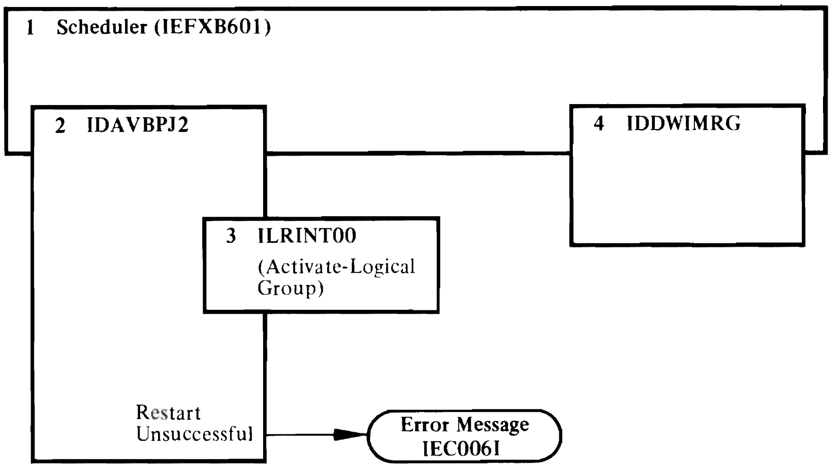


Figure 12. VIO Restart Processing

Notes for Figure 12

1. When a journaled VIO data set is being reopened in a restart situation, the scheduler's journal-write routine passes control to IDAVBPJ2 and IDDWIMRG to reconstruct the VDSCB and DSPCT from copies in the job journal.

2. IDAVBPJ2 issues a GETMAIN macro instruction to acquire space for a DSPCT header and then copies the journaled DSPCT into the area acquired. (Note: When several journaled DSPCTs exist for a single VIO data set as a result of several checkpoints having been taken, successive calls to IDAVBPJ2 do not result in another GETMAIN macro instruction being issued; the most current DSPCT journal entry is copied into the previously acquired area, overlaying a less current DSPCT.)

3. IDAVBPJ2 calls ASM, module ILRINT00, to reactivate the data set being restarted. ASM reconstructs its control information pertaining to the data set. If ASM is unable to reactivate the data set, it returns an error code, and IDAVBPJ2 issues an error message (IEC006I) and passes an error return code to IEFXB601; otherwise, IDAVBPJ2 places in the DSPCT the activated data set's storage-locator symbol that ASM returns.

   Note: The DSPCT is also built for a scratched data set. When ASM returns a code of 08 (data set not found), and control returns to the scheduler, an indicator is left in the DSPCT that is checked by module IDDWIAPP. An 0E6 ABEND is then issued to indicate that an EXCP has been attempted on a scratched data set.

4. IDDWIMRG issues a GETMAIN macro instruction to acquire space for a VDSCB and then copies the journaled copy into the area acquired. Because the WICB, VIO buffer, and VBPPL were previously freed by task close, IDDWIMRG clears the fields in the VDSCB that point to them. (Note: When several journaled VDSCBs exist for a single VIO data set as a result of several checkpoints having been taken, successive calls to IDDWIMRG do not result in another GETMAIN macro instruction being issued; the most current VDSCB journal entry is copied into the previously acquired area, overlaying a less current VDSCB.)

   Note: Additional special restart processing is performed by IDAVBPO1 after the first EXCP is issued against the restarted data set (see Figure 6).

DIRECTORY

This section contains a cross-reference list that connects module and external procedure names to sections that refer to them within this manual. (For details on the flow of control among the modules and procedures, see the "Module-Calls Directory" in the "Program Organization" section.)

This section also contains a list of macro instructions and the names of VIO modules that issue them.

Module	Function	Figure Number	Diagram Number
VBP (Virtual Block Processor) Modules			
IDAVBPC1	VBP Close	9	11
IDAVBPJ2	VBP Restart (journal merge)	12	
IDAVBPO1	VBP Open	6	3
IDAVBPP1	VBP Read/Write	6,7,8,9	4,5
IDAVBPR1	VBP Recovery		
IDAVBPR2	VBP Common Routines	6,7,9	3,5
IDAVBPS1	VBP Scratch	10	
EIP (EXCP Intercept Processor) Modules			
IDDWIAPP	EIP Appendage Interface	6,7	3,4
IDDWICLS	EIP Close	9	11
IDDWICPI	EIP Channel Program Interpretation and I/O Simulation	7	4,6,7,8,9
IDDWIFRR	EIP Recovery		
IDDWIJRN	EIP Checkpoint (journal write)	8	10
IDDWIMRG	EIP Restart (journal merge)	12	
IDDWITRM	EIP Track (buffer) Management	6,7,8,9	3,4,5,10,11
External Procedures (Residing in Module IDAVBPR2)			
CHEKMPPG	Map-page-assign	6,7	3,5
MONWINDO	Clear VIO Buffer	8,9	10
SETVCB	Initialize VCBs	6	3
Non-VIO Modules Invoked by VIO			
IEASMFEX	MF/1 EXCP Count Processing	7	4
IEAVAMSI	Paging I/O (RSM)	6,7,8,9	3,5,10,11
IEFXB500	Create Journal Entry (scheduler's journal-write routine)	8	11
ILRINT00	Auxiliary Storage Manager (ASM)	6,8,10,11,12	3

Module	Function	Figure Number	Diagram Number
Non-VIO Modules that Invoke VIO Routines			
IECVEXCP	EXCP Processor	6,7	2,3,4
IEFXB601	Restart (scheduler)	12	
IFG0TC0A	Task Close	8,11	10
IFG0202K	Common Close	9	11
IFG0552X	EOV	9	11
IGC0002I	Scratch (DADSM)	10	
IGG0CLF2	Create PDS (DADSM)	9	11
IGC0N06C	Checkpoint	8	10

Macro Where-Issued Report

Control Block Mapping Macros:

Macro Instruction	Description and Issuing Module
ACA	Maps the ASM control area. Issued by: IDDWIJRN, IDAVBPJ2, IDAVBPO1, IDAVBPS1
CVT	Maps the communications vector table. Issued by: IDAVBPC1, IDAVBPJ2, IDAVBPO1, IDAVBPP1, IDAVBPR1, IDAVBPR2, IDAVBPS1, IDDWIAPP, IDDWICLS, IDDWIFRR, IDDWIJRN, IDDWIMRG, IDDWITRM
IDABUFC	Maps the buffer control block that is appended to the VBPPL. Issued by: IDAVBPP1, IDAVBPR1, IDDWIAPP, IDDWITRM
IDAVBPH	Maps the DSPCT header. Issued by: IDAVBPC1, IDAVBPJ2, IDAVBPO1, IDAVBPP1, IDAVBPR1, IDAVBPR2, IDAVBPS1, IDDWIAPP, IDDWICLS, IDDWIJRN, scheduler journal write module
IDAVBP1	Maps entry points of VBP modules. Issued by: IDAVBPP1, IDDWIAPP, IDDWICLS, IDDWIJRN, IDDWITRM, IGC0002I
IDAVBPM	Maps a DSPCT page-map entry. Issued by: IDDWIJRN, IDAVBPO1, IDAVBPP1, IDAVBPR2
IDAVOP1	Maps the VOPEN parameter list that is appended to the VBPPL. Issued by: IDAVBPO1, IDAVBPR1, IDDWIAPP
IDDTRACK	Maps the VIO buffer. Issued by: IDDWIAPP, IDDWICPI, IDDWITRM, IDDWIFRR
IDDVBPPL	Maps the VBP parameter list. Issued by: IDDWIAPP, IDDWICLS, IDDWICPI, IDDWITRM, IDDWIFRR
IDDVDSCB	Maps the VIO data set control block. Issued by: IDDWIAPP, IDDWICLS, IDDWICPI, IDDWIFRR, IDDWIJRN, IDDWIMRG, IDDWITRM
IDDWICB	Maps the EIP (EXCP intercept processor) control block. Issued by: IDDWIAPP, IDDWICLS, IDDWICPI, IDDWIFRR, IDDWIJRN, IDDWIMRG, scheduler journal write module, close, EOV, DADSM, checkpoint
IECDRQE	Maps the request queue element. Issued by: IDDWIAPP, IDDWIFRR, IDDWITRM
IEFTIOT1	Maps the task I/O table. Issued by: IDDWIJRN
IEFUCBOB	Maps the unit control block. Issued by: IDDWIAPP
IEFZB507	Maps the direct-journal-write parameter list. Issued by: IDDWIJRN
IEZDEB	Maps the data extent block. Issued by: IDDWIAPP, IDDWIFRR

Macro Instruction	Description and Issuing Module
IEZJSCB	Maps the input/output block. Issued by: IDAVBPJ2, IDAVBPO1, IDAVBPR2, IDAVBPS1, IDDWIFRR, IDDWIJRN IDDWIMRG
IHAASCB	Maps the address space control block. Issued by: IDAVBPJ2, IDAVBPO1, IDAVBPR1, IDAVBPR2, IDAVBPS1. IDDWIFRR, IDDWIJRN
IHADSAB	Maps the data set association block. Issued by: IDDWIJRN
IHAFRRS	Maps the functional recovery routine stack. Issued by: IDAVBPC1, IDAVBPO1, IDAVBPP1, IDAVBPR1, IDAVBPS1, IDDWIAPP, IDDWICLS, IDDWIJRN
IHAPSA	Maps the prefix save area in the fixed-real storage nucleus. Issued by: IDAVBPC1, IDAVBPO1, IDAVBPP1, IDAVBPR1, IDAVBPS1, IDDWIAPP, IDDWICLS, IDDWIJRN, IDAVBPJ2
IHAQDB	Maps the queue descriptor block. Issued by: IDDWIJRN
IHARMPL	Maps the resource manager parameter list. Issued by: IDAVBPR1
IHASDWA	Maps the STAE diagnostic work area. Issued by: IDAVBPR1, IDDWIFRR
IHAVCB	Maps the VIO control block. Issued by: IDAVBPO1, IDAVBPP1, IDAVBPR2, IDDWIJRN
IKJTCB	Maps the task control block. Issued by: IDAVBPJ2, IDAVBPO1, IDAVBPR2, IDAVBPS1, IDDWIAPP, IDDWICLS, IDDWIFRR, IDDWIJRN IDDWIMRG
ILRACA	Maps the ASM control area (expands to ACA macro). Issued by: IDAVBPJ2, IDAVBPO1. IDAVBPS1, IDDWIJRN
ILRASMVT	Maps the ASM vector table. Issued by: IDAVBPJ2, IDAVBPO1, IDAVBPS1, IDDWIJRN

Action Macros:

Macro Instruction	Issuing Module
ABEND	IDAVBPO1, IDAVBPP1, IDAVBPR2, IDAVBPS1, IDDWIAPP, IDDWIJRN
ESTAE	IDAVBPS2, IDDWIJRN
LINK	IDDWIJRN
SDUMP (SVC and branch entry)	IDAVBPR1, IDDWIFRR
SETFRR	IDAVBPC1, IDAVBPO1, IDAVBPP1, IDAVBPR1, IDAVBPS1, IDDWIAPP, IDDWICLS
SETLOCK	IDDWIJRN, IDAVBPJ2
SGIDA400	System generation link-edit macro
VCLOSE	IDDWICLS
VREADWR	IDDWITRM
VOPEN	IDDWIAPP
VSCRATCH	IGC0002I
WICLOSE	IFG0552X, IGG0CLF2, IFG0202K, IDDWIJRN
WIJOURN	IGC0N06C, IFG0TC0A

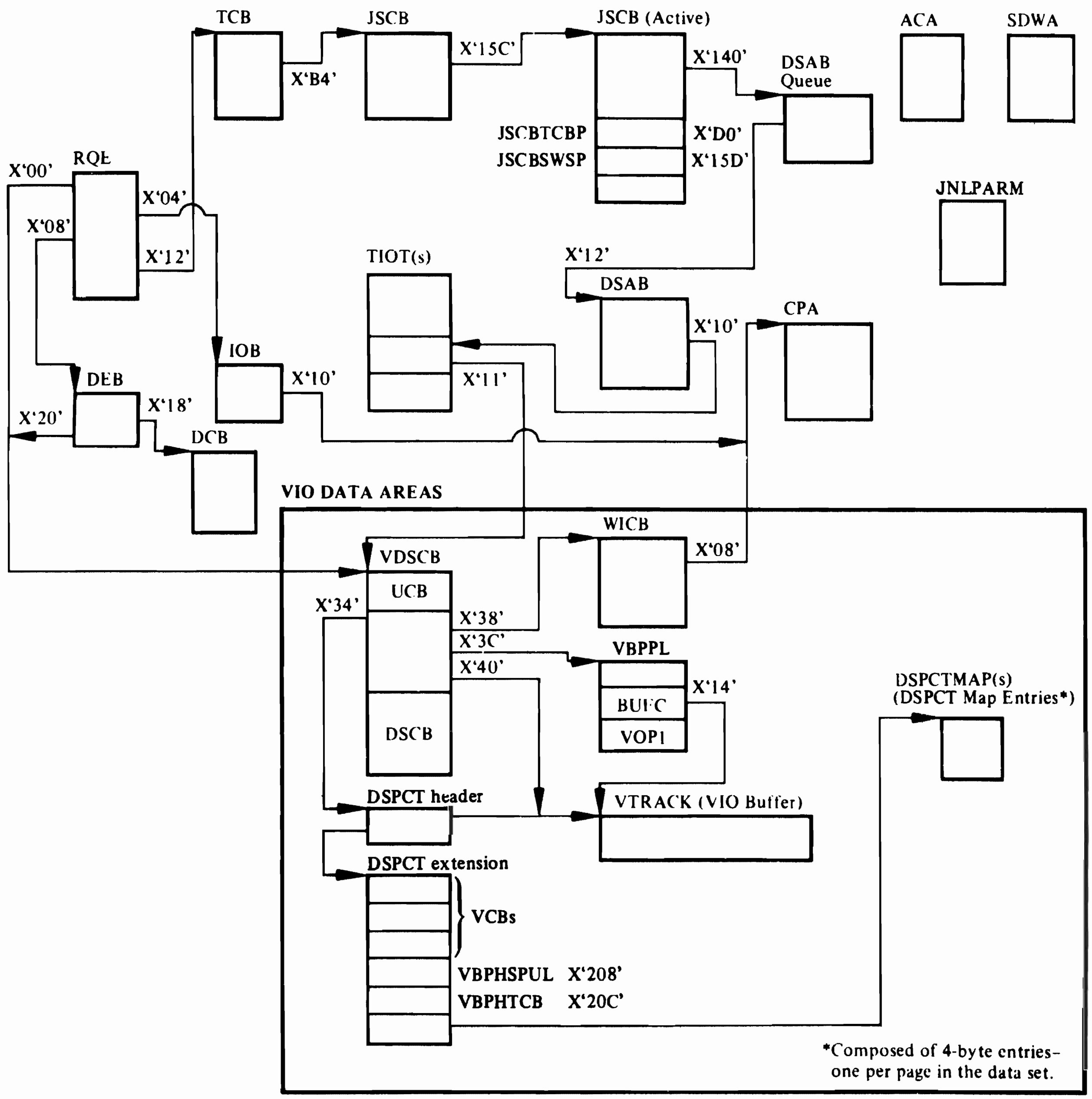
DATA AREAS

This section describes each VIO data area and the purpose of all data fields and flag bytes within each data area. "Where-Modified Reports" list data-area field names and the VIO module(s) that at some time modify the contents of the respective fields.

"Data Areas" also includes a graphic representation of the addressing relationships between non-VIO data areas (that relate to a VIO environment) and between VIO data areas.

VIO Control Block Relationships

Figure 13 shows the addressing relationships of VIO data areas and non-VIO data areas. The offset (X'nn') to a data-area address field is placed on the line that points to the data area associated with the address.



ACRONYMS

ACA	ASM Control Area	JSCB	Job Step Control Block
ASCB	Address Space Control Block	RQE	Request Queue Element
BUFC	Buffer Control Block	SDWA	STAE Diagnostic Work Area
CPA	Channel Program Area	TCB	Task Control Block
DCB	Data Control Block	TIOT	Task I/O Table
DEB	Data Extent Block	UCB	Unit Control Block
DSAB	Data Set Association Block	VBPPL	VBP Parameter List
DSCB	Data Set Control Block	VCB	VIO Control Block
DSPCT	Data Set Page Correspondence Table	VDSCB	VIO DSCB and UCB
		VOP1	VOPEN Parameter List
DSPCTMAP	DSPCT Page-Map Entry	VTRACK	Virtual Track (VIO Buffer)
IOB	Input/Output Block	WICB	EIP Control Block
JNLPARM	Journal Write Parameter List		

Figure 13. VIO Control Block Structure and Related System Control Blocks

Where-Modified Report for Non-VIO Data Areas

This report lists the VIO modules that modify fields within non-VIO data areas.

ASM Control Area (ACA)

Symbol	Module(s)
ACAASID	IDAVBPJ2, IDAVBPO1, IDDWIJRN
ACAFLG1	IDAVBPO1, IDAVBPS1, IDAVBPJ2, IDDWIJRN
ACALGN	IDAVBPS1, IDDWIJRN
ACAMAXPN	IDAVBPO1
ACASYM	IDAVBPJ2, IDAVBPS1

I/O Block (IOB)

Symbol	Module(s)
IOBCSW	IDDWICPI
IOBCSWBC	IDDWICPI
IOBCSWCC	IDDWICPI
IOBCSWCE	IDDWICPI
IOBCSWDE	IDDWICPI
IOBCSWIL	IDDWICPI
IOBCSWPG	IDDWICPI
IOBCSWPI	IDDWICPI
IOBCSWSM	IDDWICPI
IOBCSWSO	IDDWICPI
IOBCSWS1	IDDWICPI
IOBCSWUC	IDDWICPI
IOBCSWUE	IDDWICPI
IOBECBCC	IDDWIAPP
IOBERRTN	IDDWIAPP
IOBIOERR	IDDWIAPP
IOBSIOCC	IDDWICPI

Request Queue Element (RQE)

Symbol	Module(s)
RQENRQE	IDDWIAPP
RQESRB	IDDWIAPP

STAE Diagnostic Work Area (SDWA)

Symbol	Module(s)
SDWACMPC	IDAVBPR1
SDWAEBC	IDAVBPR1, IDDWIFRR
SDWAFLLK	IDAVBPR1
SDWAFREE	IDAVBPR1
SDWAFRM1	IDAVBPR1, IDDWIFRR
SDWAFRM2	IDAVBPR1, IDDWIFRR
SDWAFRM3	IDAVBPR1, IDDWIFRR
SDWAFRM4	IDAVBPR1
SDWALSQA	IDDWIFRR
SDWAREQ	IDAVBPR1, IDDWIFRR
SDWASR03	IDAVBPR1
SDWASR06	IDAVBPR1
SDWASR08	IDDWIFRR
SDWASR09	IDAVBPR1

Symbol	Module(s)
SDWASR10	IDAVBPR1, IDDWIFRR
SDWASR11	IDAVBPR1
SDWASR12	IDAVBPR1
SDWASR13	IDAVBPR1, IDDWIFRR
SDWASWA	IDAVBPR1, IDDWIFRR
SDWATO1	IDAVBPR1, IDDWIFRR
SDWATO2	IDAVBPR1, IDDWIFRR
SDWATO3	IDAVBPR1, IDDWIFRR
SDWATO4	IDAVBPR1
SDWAURAL	IDAVBPR1, IDDWIFRR

Where-Modified Report for VIO Data Areas

This report lists the VIO modules that modify fields within VIO data areas.

DSPCT (Data Set Page Correspondence Table)

Symbol	Module(s)
VBPECB	IDAVBPO1, IDAVBPP1, IDAVBPR2
VBPHBUFC	IDAVBPP1
VBPHCLO	IDAVBPC1
VBPHDSG	IDAVBPJ2, IDAVBPO1
VBPHDSSZ	IDAVBPO1
VBPHFRAR	IDAVBPC1, IDAVBPO1, IDAVBPP1, IDAVBPS1
VBPHJOU	IDDWIJRN
VBPHRST	IDAVBPO1, IDDWIJRN
VBPHJRN	IDAVBPP1, IDDWIJRN
VBPHJRNP	IDDWIJRN
VBPHLEN	IDAVBPO1
VBPHLGN	IDAVBPJ2, IDAVBPO1
VBPHMMP	IDAVBPO1
VBPHNMP	IDAVBP01, IDAVBPR2
VBPHOPE	IDAVBPO1
VBPHOPT	IDAVBPO1
VBPHPADD	IDDWIJRN
VBPHPAS	IDAVBPO1, IDAVBPP1, IDAVBPR2
VBPHPGLD	IDAVBPO1
VBPHPNP	IDAVBPO1, IDAVBPP1, IDAVBPR2
VBPHPRL	IDAVBPO1, IDAVBPP1, IDAVBPR2
VBPHRBN	IDAVBPJ2, IDAVBPO1, IDDWIJRN
VBPHRW	IDAVBPP1
VBPHSAVO	IDAVBPCI, IDAVBPO1, IDAVBPP1, IDAVBPS1
VBPHSPUL	IDAVBPO1, IDAVBPJ2
VBPHSTA	IDAVBPO1
VBPHSYM	IDDWIJRN
VBPHSYM1	IDAVBPO1
VBPHSYM2	IDAVBPO1
VBPHTCB	IDAVBPO1, IDAVBPJ2
VBPHVOP	IDAVBPO1
VBPHWAD	IDAVBPC1, IDAVBPO1
VBPHWSZ	IDAVBPO1

DSPCT Page-Map Entry

Symbol	Module(s)
VBPMRLPG	IDAVBPP1
VBPMRSN	IDAVBPP1, IDAVBPR2

VBPPL (VBP Parameter List)—Includes BUFC and VOPEN Parameter List

Symbol	Module(s)
VBPOPPRM	IDDWIAPP
VBPODSSZ	IDDWIAPP
VBPRWPRM	IDDWIAPP
BUFCBAD	IDDWIAPP
BUFCDDDD	IDDWITRM
BUFCDSPC	IDDWIAPP
BUFCMW	IDDWITRM
BUFCNLAS	IDAVBPP1
BUFCORBA	IDDWITRM
BUFCRRD	IDDWITRM
BUFCWLEN	IDDWITRM
VBPOHPTR	IDAVBPO1, IDDWIAPP
VBPOLEN	IDDWIAPP
VBPOWIA	IDDWIAPP

VCB (VIO Control Block)

Symbol	Module(s)
VCBCPFLG	IDAVBPO1, IDAVBPR2
VCBINSTG	IDAVBPR2
VCBLINK	IDAVBPO1, IDAVBPP1, IDAVBPR2, IDDWIJRN
VCBLOAD	IDAVBPR2
VCBOPFLG	IDAVBPO1, IDAVBPR2
VCBRSN	IDAVBPO1 , IDAVBPP1
VCBVSA	IDAVBPO1, IDAVBPR2, IDDWIJRN

VDSCB (VIO Data Set Control Block)

Symbol	Module(s)
VDSDSPCT	IDDWIAPP
VDSRBN	IDDWIJRN, IDDWIMRG
VDSVBPPL	IDDWIAPP, IDDWICLS
VDSWICB	IDDWIAPP, IDDWICLS
VDSWINDW	IDDWIAPP, IDDWICLS, IDDWIMRG
VDSWINSI	IDDWIAPP

VTRACK (VIO Buffer)

Symbol	Module(s)
VTDATEND	IDDWICPI
VTOVFL	IDDWICPI
VTRACK	IDDWITRM
VTRKBAL	IDDWICPI
VTST	IDDWICPI
VTUPDATE	IDDWICPI

WICB (EXCP Interrupt Control Block)

Symbol	Module(s)	Symbol	Module(s)
WICAFRO	IDDWICPI	WICLWCKD	IDDWICPI
WICAUDIT	IDDWIAPP,	WICNDXPT	IDDWICPI
	IDDWITRM	WICOFLG1	IDDWICPI
WICAVXFR	IDDWICPI,	WICOFLG2	IDDWICPI
	IDDWIFRR	WICOPINT	IDDWICPI
WICB	IDDWIAPP	WICOVFIP	IDDWICPI
WICBFKEY	IDDWICPI	WICOVFOP	IDDWICPI
WICCAM	IDDWICPI	WICPCIRS	IDDWICPI
WICCCWER	IDDWICPI	WICPRT	IDDWIAPP
WICCPINT	IDDWICPI	WICRCSKT	IDDWICPI
WICCSWSV	IDDWICPI	WICRELTP	IDDWICPI
WICCUROP	IDDWICPI	WICREQXF	IDDWICPI,
WICDATAX	IDDWICPI		IDDWIFRR
WICDATND	IDDWICPI	WICRESTA	IDDWICPI
WICDC	IDDWICPI	WICRGSV0	IDDWICPI
WICDEVTP	IDDWIAPP	WICRGSV1	IDDWICPI
WICDL	IDDWICPI	WICRSTRT	IDDWICPI
WICERASE	IDDWICPI	WICSCHEQ	IDDWICPI
WICERDSP	IDDWICPI,	WICSCHTC	IDDWICPI
	IDDWIFRR	WICSECT	IDDWIAPP
WICERPOV	IDDWIAPP	WICSEKRS	IDDWICPI
WICERPSV	IDDWIAPP	WICSENSE	IDDWICPI
WICERPSW	IDDWICPI	WICSEQFG	IDDWICPI
WICERPWR	IDDWIAPP,	WICSID	IDDWICPI
	IDDWICPI	WICSKACC	IDDWICPI
WICERROR	IDDWICPI,	WICSKAHH	IDDWICPI
	IDDWIFRR	WICSTART	IDDWIAPP
WICFMASK	IDDWIAPP	WICTCB	IDDWIAPP
WICFMTW	IDDWICPI	WICTICFG	IDDWICPI
WICFSIDE	IDDWICPI	WICTOL	IDDWIAPP
WICFSKE	IDDWICPI	WICTRKRS	IDDWICPI
WICIGAP	IDDWIAPP	WICTRMSK	IDDWICPI
WICIGP	IDDWIAPP	WICWCKD	IDDWICPI
WICINTRP	IDDWICPI	WICWORK	IDDWICPI
WICKL	IDDWICPI	WICWRTIP	IDDWICPI
WICKYGP	IDDWIAPP	WICWSCKD	IDDWICPI
WICLGAP	IDDWIAPP	WICXFCOM	IDDWICPI
WICLGP	IDDWIAPP	WICXOFLG	IDDWICPI
WICLSTOP	IDDWICPI		

ACA-ASM (Auxiliary Storage Management) Control Area

Mapping Macro: ILRACA

Size: 24 bytes

Created by: IDAVBPO1, IDAVBPJ2, IDAVBPS1, IDDWIJRN

Purpose: Parameter block between VIO and ASM

Released by: IDAVBPS1 (as part of DSPCT header), IDAVBPJ2

Field Description and Format:

Offset	Bytes and Bit Pattern	Field Name	Description
0(0)	1	ACAOP	Operation flag field Transfer page=X'04' Assign LGN=X'08' Release LGN=X'0C' Save LG/LGN=X'10' Activate LG=X'14'
1(1)	1	ACAFLG1	Flag field
	1...	ACAFMEM	Virtual memory logical group. If off VIO logical group
	..1.	ACAFFIX	ASPCT must be fixed. If off ASPCT may be paged Note: bit always off for VIO
	.... 1...	ACAFSYM	Storage locator symbol being released. If off logical group being released
2(2)	2	ACAASID	ASID of memory associated with logical group
4(4)	4	ACAESV4	Reserved
8(8)	8	ACALGN	Logical group number
8(8)	4	ACALGID	Logical group identifier
12(C)	4	ACARPN	Relative page number
16(10)	8	ACASYM	Locator symbol of group
16(10)	8	ACATOLP	Target LPID associated with target page
16(10)	4	ACATOLGI	Logical group ID
16(10)	4	ACAMAXPN	Largest relative page number to be allowed for the group
20(14)	4	ACATORPN	Relative page number

| DSPCT Header

Mapping Macro: IDAVBPH

| **Size:** 216 bytes (plus a variable number of 4-byte fields pointing to DSPCT page-map pages).

Created by: IDAVBPO1 (open) or IDAVBPJ2 (restart). Located in the scheduler work area (SWA), subpool 236 or 237, protection key 1. The subpool number and TCB for GETMAIN are determined by the JSCBSWSP and JSCBTCBP fields in the active JSCB.

Purpose: The DSPCT contains the following:

- Data set identifiers.
- Indicators reflecting the type of VIO operation being performed.
- VIO control blocks (VCBs) that are used to control the paging operations performed on the VIO buffer by RSM.
- Pointers to the DSPCT page-map pages, to the buffer control block (BUFC) used in performing actual paging I/O on the VIO buffer, and to the VOPEN parameter list (VOP1).
- Subpool number and TCB used by GETMAIN for DSPCT storage.

Referenced by: IDAVBPC1, IDDWIJRN, IDAVBPJ2, IDAVBPO1, IDAVBPP1, IDAVBPR1, IDAVBPR2, IDAVBPS1, IDDWIJRN.

Released by: IDAVBPS1.

Field Description and Format:

Offset	Bytes and Bit Pattern	Field Name	Description
0(0)	4	VBPHNAME	Block identifier-DSPC.
4(4)	4	VBPHLEN	Length of DSPCT header.
8(8)	8	VBPHLGN	Logical group number (LGN) assigned to VIO data set by ASM.
8(8)	4	VBPHLGID	First word of the LGN is a logical group identifier (LGID) that is used by ASM to access page tables that connect page identifiers to auxiliary storage slots.
16(10)	8	VBPHSYM	Storage-locator symbol ("S" symbol) assigned by ASM during a save operation. Passed to ASM and used to regain access to a data set during a restart or for scratching a saved data set.
24(18)	2	VBPHPNP	Number of pages (minus 1) containing data that were read into the VIO buffer by a prior operation. Note that an assign operation is done on only those pages of the virtual track that were written to auxiliary storage previously. Subsequent move-out operations are done only on those pages—not the entire track—unless module IDDWICPI has moved data into pages that have not been previously written.
26(1A)	2	VBPHDSG	Set at VIO open to the LGID value provided by ASM and stored in VBPHLGID. Referred to as "DSPAGEID generator" because when combined with a relative page number (RPN) value it enables RSM to access a specific data

Offset	Bytes and Bit Pattern	Field Name	Description
			set page in real storage. (If LGID is greater than 64K, VBPHDSG is set to 0 and reclaim is not attempted.)
28(1C)	4	VBPHDSSZ	Size in pages of the VIO data set. The calculation is based on the number of tracks in the VIO extent times the track size rounded-up to a page boundary divided by the page size (4096 bytes) plus the number of pages required for the DSPCT page-map. (1023 data set pages require one map page.)
32(20)	1	VBPHOPT	Processing-option flags:
	..1.	VBPHRST	Indicates that the DSPCT is a copy from the job journal and is not initialized by IDAVBPO1.
	...1	VBPHPAS	Indicates that the prior operation was an assign (read) request.
	.... 1...	VBPHDEL	The logical group has been released. This was detected during restart processing.
	.... .1..	VBPHPADD	Indicates that data set pages exist in auxiliary storage.
	.... ..1.	VBPHJRN	Indicates that pages were added to or modified in the data set (1) since the data set was created if the VBPHJOU flag is off or (2) since the last time it was journaled if the VBPHJOU flag is on.
	1	VBPHJOU	Indicates that the DSPCT has been journaled and the logical group saved.
33(21)	1	VBPHSTA	Routine-in-control indicators and data set status flags:
	1...	VBPHOPE	VIO open processing is in progress. Indicates that IDAVPBO1 is in control.
	.1..	VBPHRW	The VIO buffer is being filled (assign), emptied (move-out), or cleared (move-out-null). Indicates that IDAVBPP1 (VREADWR processing) is in control.
	...1	VBPHSCR	The VIO data set is being scratched. Indicates that IDAVBPS1 is in control.
	.... 1...	VBPHCLO	VIO close processing is in progress. Indicates that IDAVBPC1 is in control.
34(22)	2	VBPHNMP	Current number of DSPCT page-map pages.
36(24)	2	VBPHMMP	Maximum number of DSPCT page-map pages based on the maximum size of the data set with one 4-byte DSPCT page-map entry per data set page.
38(26)	2	VBPHWSZ	Size in pages of the VIO buffer. The size of a VIO buffer is based on the size of a virtual track rounded-up to a page boundary.
40(28)	4	VBPHWAD	Address of the VIO buffer. The VIO buffer resides in subpool 229.

Offset	Bytes and Bit Pattern	Field Name	Description
44(2C)	4	VBPHPRL	Relative page number (RPN) of the first page in the current VIO buffer that was placed there by an assign (read) request. This value is used when initializing VCBs to move-out (write) or move-out-null (disconnect) the pages in the VIO buffer.
48(30)	4	VBPHRBN	Relative block number (RBN) of a journaled DSPCT header. A unique identifier used to differentiate DSPCT headers of different VIO data sets for a given job.
52(34)	4	VBPHBUFC	Address of the buffer control block (BUFC) used to control actual I/O.
52(34)	4	VBPHVOP	Address of the VIO open parameter list (VOP1).
56(38)	4	VBPHFRAR	Address of the recovery work area established by the SETFRR macro instruction.
60(3C)	4	VBPHVCB	Address of the DSPCT extension.
64(40)	72	VBPHSAVE	VBP's first register save area.
136(88)	72	SECSAVE	VBP's second register save area.
208(D0)	1	VBPHSPUL	SWA subpool number used by GETMAIN/FREEMAIN for DSPCT header and DSPCT page-map pages.
209(D1)	3	VBPHJRNP	Pointer to module IEFXB500 which is passed from IDDWIJRN to IDDWIJRN.

212(D4)	4	VBPHTCB	TCB under which GETMAIN/FREEMAIN is done for DSPCT header and DSPCT page-map pages.
216(D8)	Multiple 4-byte entries	VBPHMAD	Addresses of DSPCT page-map pages (variable number: one address entry for each DSPCT page-map page).

Note: The first 48 bytes of the DSPCT (through field VBPHRBN) are written to the job journal.

DSPCT Extension

Mapping Macro: IDAVBPH pointed to by DSPCT header.

Size: 36-byte fixed section plus a variable section (48 times the window size in pages).

Created by: IDAVBPO1 in LSQA (subpool 254).

Purpose: Used to map VCBs for move-out and assign requests.

Referenced by: IDAVBPO1, IDAVBPP1, IDAVBPR2.

Released by: IDAVBPS1 or IDAVBPS2.

Field Description and Format:

Offset	Bytes and Bit Pattern	Field Name	Description
0(0)	36	VBPVFXD	Fixed section.
0(0)	4	VBPVNAME	Block ID-VCB
4(4)	4	VBPVLEN	Length of block.
8(8)	24	VBPVMAP	Used as a VCB for move-out of DSPCT page-map pages.
8(8)	4	VCBSAVAD	Address of first VCB to be passed to RSM.
32(20)	4	VBPXAPTR	Address of first assign VCB.
36(24)		VBPVMOVE	Move-out VCBs. The number of VCBs here is equal to the window size in pages. Each VCB is 24 bytes long.
		VBPVASIN	Assign VCBs. The number of VCBs here is equal to the window size in pages. Each VCB is 24 bytes long.

DSPCTMAP-DSPCT Page-Map Entry

Mapping Macro: IDAVBPM

Size: 4 bytes each. One page-map entry is created for each VIO data set page and for each page of page-map entries. Assigned in multiples of 1024 (that is, 4096-byte pages).

Created by: DSPCT page-map pages are acquired by IDAVBPR2 and individual entries are generated by IDAVBPO1, IDAVBPP1, and IDAVBPR2. Acquired in SWA, subpool 236 or 237, protection key 1. The subpool number and TCB for this GETMAIN or FREEMAIN is determined by fields VBPHSPUL and VBPHTCB in the DSPCT header.

Purpose: When a page in the VIO buffer is written to auxiliary storage, the real storage number of its page frame is placed in its DSPCT page-map entry.

When an assign (read) request is made, the RSN in the DSPCT page-map entry is passed to the paging supervisor. If the page-frame-table entry (PFTE) for the RSN is still valid, the page frame is reclaimed, and no actual I/O is performed.

Referenced by: IDAVBPO1, IDAVBPP1, IDAVBPR2.

Released by: IDAVBPS1.

Field Description and Format:

Offset	Bytes and Bit Pattern	Field Name	Description
0(0)	1	VBPMFLG	Flag byte:
	1...	VBPMRLPG	Indicates that the VIO data set page associated with this DSPCT page-map entry has been written to auxiliary storage.
1(1)	1		Not used.
2(2)	2	VBPMRSN	Real storage number (RSN) of the real storage page frame that last contained the page associated with this DSPCT page-map entry.

The information on this page has been deleted by OS/VS2 MVS Supervisor Performance #2.

JNLPARM—Journal Write Parameter List

Mapping Macro: IEFZB507

Size: variable. 8 bytes for the header plus 16 bytes for each entry.

Created by: IDDWIJRN. Located in subpool 230, key 0.

Purpose: Parameter list passed from VIO journaling routine to scheduler journal write routine. List contains one entry for each VDSCB and each DSPCT associated with each VIO data set to be journaled.

Referenced by: IEFXB500.

Released by: IDDWIJRN.

Offset	Bytes and Bit Pattern	Field Name	Description
0(0)	1	JNLPCALL	Caller Flags
	...1	JNLDRCT	Direct Write to Journal
1(1)	1	JNLPRTCD	Return code field
	1...	JNLERR	Journal error return code
	.1..	JNLABSNT	No journal return code
2(2)	2		Reserved
4(4)	4	JNLPPTRX	Pointer to first entry
0(0)	4	JNLBLKAD	Address of block to be journaled
4(4)	1	JNLPID	I.D. (defines type of block as DSPCT or VDSCB)
5(5)	3	JNLPRLNG	Block length
8(8)	4	JNLRBN	RBN or zero
12(C)	4	JNLNBLK	Pointer to the next block

VBPPL—VBP Parameter List

Mapping Macros: IDDVBPPL, IDABUFC, IDAVOP1

Size:

- IDDVBPPL, 184 bytes.
- IDABUFC (VREADWR parameter list), 40 bytes.
- IDAVOP1 (VOPEN parameter list), 24 bytes.

Created by: IDDWIAPP. Located in subpool 230, key 1.

Purpose: EIP (EXCP intercept processor) uses the VBPPL to pass control information to VBP (virtual block processor). As a VOPEN parameter list, a pointer to the VOP1 (VOPEN interface block) is passed to VBP. As a VREADWR parameter list, VOP1 is overlaid with the BUFC (buffer control block), and a pointer to the BUFC is passed to VBP.

Referenced by:

- IDDVBPPL—IDDWIAPP, IDDWICLS, IDDWICPI, IDDWITRM.
- IDABUFC—IDAVBPP1, IDAVBPR1, IDDWIAPP, IDDWITRM.
- IDAVOP1—IDAVBPO1, IDAVBPR1, IDDWIAPP.

Released by: IDDWICLS.

Field Description and Format:

Offset	Bytes and Bit Pattern	Field Name	Description
0(0)	72	VBPPLSAV	First register save area.
72(48)	72	VBPPLSV2	Second register save area.
144(90)	40	VBPRWPRM	VREADWR parameter list (IDABUFC).
144(90)	24	VBPOPPRM	VOPEN parameter list (IDAVOP1).

VREADWR Parameters Set in the BUFC:*

Offset	Bytes and Bit Pattern	Field Name	Description
146(92)	1	BUFCIOFL	Control flags:
	1...	BUFCMW	Indicates that the VIO buffer is to be written to auxiliary storage.
	..1.	BUFCRRD	Indicates that pages are to be read into the VIO buffer from the VIO data set in auxiliary storage.
147(93)	1	BUFCFLG2	Status flags:
	.1..	BUFCNLAS	Indicates that the current channel program command is attempting to access a nonexistent track.
152(98)	4	BUFCDDDD	RBA of the first page to be read into the VIO buffer.
156(9C)	4	BUFCORBA	RBA of the first page in the VIO buffer to be written to auxiliary storage.
164(A4)	4	BUFCBAD	Address of the VIO buffer.

*Note that only the fields used by VIO are listed here—the complete BUFC is not shown.

Offset	Bytes and Bit Pattern	Field Name	Description
| 172(AC)	2	BUFCWLEN	Length of data in VIO buffer in bytes (that is, the displacement from the beginning of the VIO buffer to the end of the last byte of data in the buffer).
176(B0)	4	BUFCDSPC	Address of DSPCT.
VOPEN Parameters Set in the VOP1:			
144(90)	4	VPBOHPTR	Address of DSPCT.
148(94)	4	VBPOWIA	Address of VIO buffer.
152(98)	4	VBPOLEN	Length in pages of the VIO buffer.
156(9C)	4	VBPODSSZ	Maximum data set size in bytes.
160(A0)	4	VBPOFLG	Option flags: not used.

VCB—VIO Control Block

Mapping Macro: IHAVCB

Size: 24 bytes.

Created by: IDAVBPO1. Acquired in LSQA (subpool 254) as part of an extension of the DSPCT header.

Purpose: Contains information that is passed to RSM to control paging.

Referenced by: IDAVBPO1, IDAVBPP1, IDAVBPR2, IDDWIJRN.

Released by: The system, when the job step terminates.

Field Description and Format:

Offset	Bytes and Bit Pattern	Field Name	Description
0(0)	4	VCBLINK	Address of next VCB to be processed or 0, if last.
4(4)	4	VCBVSA	Virtual storage address of the page to be read (assign function) or written (move-out function) or disconnected (move-out-null function).
8(8)	8	VCBLPID	Logical page identifier.
16(10)	1	VCBOPFLG	Operation flags:
	1...	VCBMVOUT	Write (move-out) requested.
	..1.	VCBASIGN	Read (assign) requested.
	.... 1...	VCBNDISC	Nondisconnect option specified with a write (move-out) request. Only used when writing DSPCT page-map pages.
17(11)	1	VCBCPFLG	Completion flags:
	1...	VCBNOVAC	Assign error. The page identified by the VCBVSA field was previously read and was not disconnected by a move-out or move-out-null request before the current read (assign) operation.
	.1..	VCBINVSA	Invalid virtual storage address. A GETMAIN macro instruction was not issued for the page identified by the VCBVSA field.
	..1.	VCBELPID	The page identified by the VCBVSA field belongs to an LPID other than the LPID in the VCBLPID field.
	...1	VCBNOAUX	The page identified by the VCBVSA field will not be written to the data set because it was not brought into real storage by the prior read request or because data in the page has not been modified.
	.... 1...	VCBEFIX	Write (move-out) error. the page identified by the VCBVSA field is fixed in real storage and cannot be written to the data set in auxiliary storage.
18(12)	2	VCBRSN	RSN (real storage number) of the page frame that last contained the page associated with the VCBVSA.
20(14)	4	VCBDSPID	DSPID (data set page id) that is used to reclaim the page associated with the VCB.

VDSCB—Virtual Data Set Control Block

Mapping Macro: IDDVDSCB

Size: 205 bytes.

Created by: DADSM allocate, module IGG0325I (and, if during restart, module IDDWIMRG) in the scheduler work area (SWA) with a protection key of 1 (scheduler key). The subpool used is determined by the JSCBSWSP field in the active JSCB.

Purpose: The virtual data set control block contains a format-1 DSCB describing the VIO data set and a UCB for the virtual device identified by the UNIT keyword parameter in the user's DD statement. It also contains pointers to various VIO control blocks.

Referenced by: IDDWIAPP, IDDWICLS, IDDWICPI, IDDWIFRR, IDDWIJRN, IDDWIMRG, IDDWITRM.

Released by: Scheduler at job termination (along with other data areas contained in the SWA).

Field Description and Format:

Offset	Bytes and Bit Pattern	Field Name	Description
0(0)	45	VDSUCB	UCB for the virtual device being simulated in external page storage.
0(0)	1		VIO UCB identifier.
4(4)	2		X'3FFF' channel and unit address.
13(D)	3		"VIO"—UCBNAME.
28(1C)	6		X'404040404040' volume serial number.
45(2D)	7	VDSSEEKA	DASD seek address of the current track in the VIO buffer.
52(34)	4	VDSDSPCT	Address of DSPCT header.
56(38)	4	VDSWICB	Address of WICB.
60(3C)	4	VDSVBPPL	Address of VBPPL.
64(40)	4	VDSWINDW	Address of VIO buffer.
68(44)	2	VDSTRKSI	Maximum track size, in bytes, of the virtual device.
70(46)	2	VDSWINSI	Size, in pages, of the VIO buffer. Equal to the track size of the virtual device rounded-up to a page boundary.
72(48)	2	VDSABSTT	Absolute track number of the first track of the extent for the VIO data set. Set to X'FFFF', by module IGC0002I, when the data set is scratched.
74(4A)	2	VDSNMTRK	Number of tracks in the extent for the VIO data set.
76(4C)	4	VDSRBN	Relative block number (RBN). A unique identifier for a data set's VDSCB created by the scheduler's journal-write routine, module IEFXB500, when a data set is journaled for the first time. Used by the scheduler to reaccess a particular data set's VDSCB at restart time.
80(50)	125	VDSDSCB	Format-1 DSCB.

VTRACK—VIO Buffer

Mapping Macro: IDDTRACK

Size: Varies according to the track size (rounded-up to a page boundary) of the virtual DASD associated with the unit name provided by the user issuing the EXCP request.

Created by: IDDWIAPP. Located in the LSQA, subpool 229, in the protection key of the user issuing the EXCP request.

Purpose: Used to simulate I/O operations associated with the EXCP request. Records are moved between the VIO buffer and either the user or the access method buffer. When necessary, actual paging I/O is performed; that is, pages in the VIO buffer are paged to and from the VIO data set in external page storage.

Referenced by: IDDWITRM, IDDWICPI.

Released by: IDDWICLS and IDDWIAPP via a FREEMAIN macro instruction.

Field Description and Format:

Offset	Bytes and Bit Pattern	Field Name	Description
0(0)	4	VTDATEND	Displacement from the beginning of the track to the end of the last byte of data on the track.
4(4)	4	VTRKBAL	Current track balance.
8(8)	1	VTST	Track status flags:
	1...	VTINIT	Indicates that the track has been initialized.
	.1..	VTOVFL	Indicate that the last record in the track is an overflow record.
	..1.	VTUPDATE	Indicates that the track contains new or modified records.
9(9)	5	VTHA	Home address.
9(9)	1	VTHAFLAG	Flag byte in home address.
10(A)	4	VTHACCHH	Address portion of home address.
14(E)	16	VTR0	R0 (record zero).
14(E)	8	VTR0CNT	R0 count.
22(16)	8	VTR0DATA	R0 data.
30(1E)	1	VTR1	This field marks the beginning of the first record in the track.

WICB—EXCP Intercept Control Block

Mapping Macro: IDDWICB

Size: 124 bytes.

Created by: IDDWIAPP. Located in subpool 230 with the protection key of the user that issued the EXCP request.

Purpose: Contains status and control information that is used in interpreting and simulating execution of the channel program associated with the EXCP request.

Referenced by: IDDWIAPP, IDDWICLS, IDDWICPI, IDDWIFRR, IDDWIJRN, IDDWIMRG.

Released by: IDDWIAPP and IDDWICLS.

Field Description and Format:

Offset	Bytes and Bit Pattern	Field Name	Description
0(0)	8	WICSEEKA	Actual device address (DASD seek address) of the track to be placed in the VIO buffer.
0(0)	1	WICSKAM	Extent number. This number is used to access a 16-byte extent description segment in the DEB.
1(1)	6	WICSKBCH	BBCCHH. Cell number (BB), cylinder number (CC), and head number (HH) of a track.
1(1)	2	WICSKABB	Two bytes of zeros as cell number (BB).
3(3)	4	WICSKACH	Cylinder head number (CCHH).
3(3)	2	WICSKACC	Cylinder number (CC).
5(5)	2	WICSKAHH	Head number (HH).
7(7)	1	WICSKAR	Relative block number of a track, or physical record (R).
8(8)	4	WICSTART	Starting address of channel program.
12(C)	3	WICAUDIT	Contains EBCDIC characters denoting the EIP module in control: 'APP', 'CPI', 'TRM'.
			Used in module IDDWIFRR's recovery processing.
15(F)	1	WICPRT	Protection key of the current user and of the WICB and VIO buffer.
16(10)	16	WICRSTCP	Channel program used by an error recovery procedure (ERP) to restart processing following the identification of a recoverable error condition.
16(10)	8	WICCCW1	First restart CCW.
24(18)	8	WICCCW2	Second restart CCW.
32(20)	1	WICERPSW	ERP control flags:
	1...	WICERPOV	Indicates an ERP restart following an overflow-incomplete condition.
	.1..	WICERPWR	Indicates an ERP restart of a write CCW with prerequisite checking having been previously satisfied.
33(21)	3	WICERPSV	Address of the CCW that was restarted.

Offset	Bytes and Bit Pattern	Field Name	Description
36(24)	4	WICCAM	Address of the count field of the current record.
40(28)	2	WICRELTP	Position in current record: home address, count, key, or data.
42(2A)	2	WICKL	Key length of the current record.
44(2C)	4	WICDL	Data length of the current record.
48(30)	4	WICAVXFR	Number of record bytes available for a read or unformatted write operation.
52(34)	4	WICREQXF	Number of bytes moved for a CCW operation.
56(38)	1	WICOFLG1	Current operation-code flags:
	1...	WICWCKD	Write-count-key-data (WCKD).
	.1..	WICWSCKD	Write-special-count-key-data (WSCKD).
	..1.	WICERASE	Erase.
	...1	WICFMTW	Format-write operation. Set for WCKD.
	.... 1...	WICOVFOP	Indicates that a RD, RKD, RCKD, WD, or WKD operation qualifies for overflow processing.
	.... .1..	WICSCHTC	Indicates that the current operation sequence is a SID, TIC *-8, SID.
	.... ..1.	WICSID	Indicates that the current operation is a SID, TIC *8, SID.
57(39)	1	WICOFLG2	Additional operation-code flags:
	1...	WICXFCOM	Indicates that data transfer is completed or not desired.
	.1..	WICDATAX	Indicates that data transfer has been initiated.
	..1.	WICINTRP	Indicates that a condition exists that warrants termination of channel-program-interpretation processing by module IDDWICPI.
	...1	WICERROR	Indicates that an error was detected after the current operation was initiated.
	.... 1...	WICSCHEQ	Indicates that the current operation is either a SID or SK.
	.... .1..	WICDC	Indicates that data chaining has been requested.
	.... ..1.	WICWRTIP	Indicates that a write operation is in progress.
58(3A)	1	WICSEQFG	Sequence-checking flags:
	1...	WICLWCKD	Indicates that the last operation was a write-count-key-data (WCKD).
	.1..	WICFSIDE	Indicates that a search-ID-equal operation has been satisfied.
	..1.	WICFSKE	Indicates that a search-key-equal operation has been satisfied.
	..1.	WICTICFG	Indicates that a TIC command is being processed.
59(3B)	1	WICXOFLG	Channel program flags:
	1...	WICNDXPT	Indicates that the index point has been passed.
	.1..	WICRCSKT	Indicates that the current operation sequence is: RC, SK, TIC *-16.
60(3C)	4	WICWORK	General work area.

Offset	Bytes and Bit Pattern	Field Name	Description
64(40)	2	WICERDSP	Displacement in the error table for the CCW in error. The error table is used to determine the particular sense information that is set in the IOB.
66(42)	1	WICLSTOP	Operation code of the last CCW.
67(43)	1	WICCUROP	Operation code of the current CCW.
68(44)	1	WICRESTA	Restart flags:
	1...	WICPCIRS	PCI restart.
	.1..	WICOVFIP	Indicates that current processing is due to the restart of an overflow-incomplete operation.
	..1.	WICSEKRS	Seek routine invoked the track manager, module IDDWITRM.
	...1	WICTRKRS	Indicates that processing has been interrupted to acquire a new track.
	.... 1...	WICTRMSK	Invalid seek address detected by the track manager, module IDDWITRM.
	1	WICRSTRT	Indicates that a restart condition exists. The specific type of restart is indicated by the other bit settings in this byte.
69(45)	1	WICFMASK	File mask.
70(46)	16	WICDVTAB	Device-characteristics-table information for the device being simulated in paging space:
70(46)	2	WICMAXCC	Maximum cylinder number.
72(48)	2	WICMAXHH	Maximum head number.
74(4A)	2	WICTRKCP	Track capacity.
76(4C)	2	WICIGAP (WICIGP)	Normal interrecord gap overhead.
78 (4E)	2	WICLGAP (WICLGP)	Interrecord gap overhead for last record.
80(50)	2	WICKEYGP (WICKYGP)	Interrecord gap overhead for keyed record processing.
82(52)	2	WICTOL	Tolerance.
84(54)	1	WICSECT	Highest permissable sector value. If zero, rotational position sensing (RPS) is not supported.
85(55)	1	WICDEVTP	Device type.
86(56)	2	WICSENSE	Save area for sense information.
88(58)	4	WICEXPTR	Address of DEB extent.
92(5C)	28	WICREGSV	Register save area.
116(74)	4	WICDATND	Address of the end of the current track.
120(78)	4	WICTCB	TCB used by GETMAIN when storage is acquired for WICB, VBPPL, and VTRACK.

DIAGNOSTIC AIDS

This section contains the following information:

- ABEND completion codes issued by VIO are listed with their reason codes. The modules that set the ABEND codes are identified, and explanations of the error conditions associated with the ABEND and reason codes are provided. (See "Error Completion Codes Issued by VIO.")
- Processing performed by VIO functional recovery routines for individual VIO modules is described. Diagnostic information that is recorded by these routines in system data areas is also described. (See "Processing Performed by VIO's Recovery Routines.")
- Tables (Figures 14 and 15) are provided that relate channel program error conditions to the labels of VIO code segments that detect the conditions. (See "Locating Code Segments that Detect Channel Program Errors.")
- A path is shown that can be followed in a system dump to isolate a VIO module that detects an error condition. (See "Debugging VIO Errors.")
- The single VIO error message that can be issued by the VIO component is described. (See "Messages Issued by VIO.")

Error Completion Codes Issued by VIO

The following completion codes are issued by VIO modules. (For an extended description of the meanings of the codes and of the appropriate programmer responses, refer to *OS/VS Message Library: VS2 System Codes.* Reason codes are contained in register 15.

ABEND Code	Reason Code	Explanation
0E1		Issued by IDAVBPO1. Indicates that a VIO data set could not be successfully opened.
	2XX	An assign-null (virtual allocate) operation performed by RSM to acquire a DSPCT page-map page or the VIO buffer resulted in an error.
	3XX	ASM processing is unable to assign a logical group number (LGN) to the VIO data set.
0E2		Issued by IDAVBPO1. Indicates that an error occurred while processing the first EXCP request after a restart.
	2XX	An assign-null (virtual allocate) operation on the VIO buffer or an assign (read) operation on the DSPCT page-map pages performed by RSM resulted in an error.

ABEND Code	Reason Code	Explanation
0E3		Issued by IDAVBPP1. Indicates that an error occurred when RSM attempted to perform paging operations against the VIO buffer.
	004	An invalid RBA was passed to VBP. Probable cause of error is a DEB with extent descriptions which do not fall within the allocated extents as described in the virtual data set control block (VDSCB). Can also be caused if an EXCP is issued to read the format-1 DSCB on the volume.
	2XX	RSM is unable to move-out (write) or assign (read) pages in the VIO buffer to or from the VIO data set.
0E4		Issued by IDAVBPS1. Indicates that an error occurred while ASM attempted to release (or scratch) the resources associated with a VIO data set—release-logical-group function.
	3XX	ASM was unable to RELEASE the logical group for this record.
0E5		Issued by IDAVBPR1. Indicates that VIO close processing performed against the VIO buffer resulted in an error.
	2XX	RSM is unable to move-out-null (disconnect) the pages in the VIO buffer from the VIO user's address space because of an error condition.
0E6		Issued by IDDWIAPP. Indicates that restart processing is attempting to access a scratched data set.
	018	See the explanation for the 0E6 ABEND code.
	008	Scheduler module IEFXB500 returned an unsuccessful completion code for a write request for the job journal.
0E7		Issued by IDDWIJRN. Indicates that an error occurred while journaling the DSPCT header at either step termination or at a checkpoint.
	2XX	RSM is unable to move-out (write and disconnect) the DSPCT page-map page(s).
	3XX	An error occurred when ASM attempted to save the status of a logical group—ASM's save-logical-group function.

Processing Performed by VIO's Recovery Routines

VIO modules IDDWIFRR and IDAVBPR1 contain the recovery procedures that perform recovery processing and cause diagnostic information to be retained if the VIO user's address space is accessible. The diagnostic information consists of volatile data areas and/or system work areas—the scheduler work area (SWA) and the local system queue area (LSQA)—that contain other VIO data areas and job-related system control blocks, respectively.

General Notes on Recovery Processing

VIO source modules use the SETFRR and ESTAE macro instructions to establish addressability to recovery routines that relate to the individual source modules. When invoked by R/TM (recovery/termination manager), the recovery routines use SDUMP and SETRP macro instructions to cause selected control blocks and/or system work areas to be written to system data sets. Further information about the effects of these macro instructions and about the situations in which they are employed is provided under the headings that follow.

Establishing Addressability to Recovery Procedures

The functional recovery environment for VIO processing is set by individual VIO modules by issuing SETFRR or ESTAE macros. The macro expansions set the address of recovery routines for particular modules in a last-in/first-out (LIFO) list that is used by R/TM (recovery/termination manager) during error-recovery processing. The macro—ESTAE or SETFRR—employed to establish addressability to VIO FRRs (functional recovery routines) is selected based on the following considerations:

- If the local lock is held by the module setting the recovery environment and if the recovery routine wants the local lock to be held when either it or the retry routine is processing, the SETFRR macro is used. (Note that routines established by the SETFRR macro receive control with the local lock being held.)
- If the local lock is not held by the module setting the recovery environment or if retry processing can operate without the local lock being held, the ESTAE macro is used. (Note that if an ESTAE recovery routine is to be entered, the local lock (if held) is released before entry.)

Methods of Recording Diagnostic Information

Functional and ESTAE recovery procedures can record the status of their virtual storage environments in the following ways:

- By issuing an SDUMP macro, which causes the contents of the 4K-byte SDUMP buffer in the SQA (when the buffer is available) and the specified system work areas and user region to be written in a SYS1.DUMP data set. (There are ten SYS1.DUMP data sets, SYS1.DUMP00-09.) A branch-entry form of the SDUMP macro is used when the local lock is held; otherwise, an SVC form is used.
- By issuing a SETRP macro, specifying RECORD=YES, which directs R/TM to write the SDWA (STAE diagnostic work area) to the SYS1.LOGREC data set.
- By setting flags in the SDWA that cause R/TM to write specified system work areas and VIO data areas to a user-specified dump data set (SYSABEND or SYSUDUMP) when percolation through recovery routines is completed. However, the VIO data areas specified by a given VIO recovery procedure may not be written because subsequent recovery routines may overlay the data area addresses with other addresses—the data areas corresponding to the final four addresses listed in the SDWA are written out when percolation is completed.

  Note: The data areas are not written if the indicator in the SDWA (SDWAREQ flag) is turned off by a succeeding FRR and remains so until percolation is completed. Also, the system work areas (SWA and LSQA) are not dumped if succeeding routines alter the indicators set by a recovery routine in the SDWA (SDWASWA and SDWALSQA flags). Since the dump is not taken by branch entry SDUMP until the local lock is released, the header of the dump may be overlayed by other data.

Analyzing Diagnostic Information Recorded by Recovery Routines

The information that is recorded on SYS1.LOGREC and SYS1.DUMP can be retrieved by the IFCEREP0 and AMDPRDMP service aids, respectively.

Analyzing SYS1.DUMP Data Sets

To get a dump of a SYS1.DUMP data set, use the AMDPRDMP service aid as described in *OS/VS2 System Programming Library: Service Aids.* The SDUMP macro allows VIO recovery routines to dynamically take limited dumps containing volatile workareas before the areas are swapped-out or modified. SDUMP macros are issued to cause dumps to be taken to record the status of the current environment whether retry or percolation is performed.

Analyzing SYS1.LOGREC Data Set

Information written by recovery routines to the SYS1.LOGREC data set is used primarily to monitor incidents—both when retry is attempted and when percolation to the next recovery routine takes place. To get a dump of the SYS1.LOGREC data set, use the IFCEREP0 service aid as described in *OS/VS2 System Programming Library: SYS1.LOGREC Error Recording.* IFCEREP0 will format 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. IFCEREP0 will put the variable area—the last 108 bytes—of each SDWA in an alphanumeric or hexadecimal format, whichever is specified. The VIO recovery modules, IDDWIFRR and IDAVBPR1, require the data in the variable area to be in alphanumeric format so that the following messages can be constructed.

Module IDAVBPR1 constructs a diagnostic message of the following format:

`jobname,VIO,modname,addresses`

where:

`jobname` An eight-byte job name from the TIOT.

`modname` The VIO module in control or last in control when the error was detected.

`addresses` This area contains a list of addresses of the pertinent control blocks which existed at the time of the error. The list supplied depends on the module in control at the time of error. The format and list for each module are as follows:

Module	Format and List
IDAVBPC1	A(DSPC)=a
IDAVBPO1	A(DSPC)=a,A(VOP1)=a
IDAVBPP1	A(DSPC)=a,A(BUFC)=a
IDAVBPS1	A(DSPC)=a

where:

`a` is the address of the control block.

`DSPC` locates the DSPCT header.

`VOP1` locates the VOPEN parameter list passed to module IDAVBPO1.

`BUFC` locates the VREADWR parameter list passed to module IDAVBPP1.

Module IDDWIFRR constructs a diagnostic message of the following format:

`JOBNAME=jobname,VIO LMOD=IDDWI,function,addresses`

where:

`jobname` An eight-byte job name from the TIOT.

`function` A code which indicates which VIO module was in control or last in control when the error was detected. If the code is WIEXCP, then module IDDWIAPP, IDDWICPI, or IDDWITRM was in control. If the code is WICLOSE, then module IDDWICLS was in control. If the code is WIJOURN, then module IDDWLIRN was in control.

`addresses` This area contains a list of addresses of the pertinent control blocks which existed at the time of the error. The list supplied depends on the module in control at the time of error. The format and list for each module are as follows:

Module	Format and List
IDDWIAPP	A(IOB)=a,A(VDSCB)=a
IDDWICLS	A(VDSCB)=a
IDDWICPI	A(IOB)=a,A(VDSCB)=a
IDDWIJRN	A(TCB)=a,A(VDSCB)=a
IDDWITRM	A(IOB)=a,A(VDSCB)=a

where:

`a` is the address of the control block.

`IOB` locates the Input/Output Block.

`TCB` locates the Task Control Block.

`VDSCB` locates the Virtual Data Set Control Block.

The remaining topics in this section describe the output—the SDUMP buffer records, the system work areas, and the SDWA variable areas—of VIO recovery procedures for particular VIO modules.

Diagnostic Output for VIO Module IDAVBPC1

Module IDAVBPR1 is entered at its starting address to perform error recovery processing for module IDAVBPC1. IDAVBPC1 issues a SETFRR macro to establish the entry point.

Control Information and Messages Established in the SDWA

If the user's address space is accessible, the recovery routine sets the starting and ending addresses of the DSPCT header in the SDWA (SDWADPSL field).

A diagnostic message is then built in the variable recording area in the SDWA (SDWAVRA field). See "Analyzing the SYS1.LOGREC Data Set" for the format of this message as constructed by module IDAVBPR1.

Output to the SYS1.DUMP Data Set

If the address space is not accessible or if a machine check interruption has occurred, an SDUMP macro is not issued. If the SDUMP macro can be issued, the user region and the data area (DSPCT header) listed in the SDWA are written to SYS1.DUMP via the SDUMP macro (branch-entered form).

Output to the SYS1.LOGREC Data Set

Parameters are set in the SDWA via the SETRP macro to request R/TM to write the SDWA to SYS1.LOGREC and to percolate to the next FRR in its list.

Output to the User-Specified Dump Data Set

If the user's address space is accessible, indicators are set in the SDWA to cause R/TM to also write the SWA and the DSPCT header to the dump data set specified by the user's SYSABEND or SYSUDUMP DD statement.

Diagnostic Output for VIO Module IDAVBPO1

Module IDAVBPR1 is entered at its starting address to perform functional recovery processing for module IDAVBPO1.

Control Information and Messages Established in the SDWA

If the user's address space is accessible, the recovery routine sets the following information in the SDWA:

- The starting and ending addresses of the DSPCT header are set in the SDWAFRM1 and SDWAT01 fields.
- The starting and ending addresses of the open parameter list (VOP1) are set in the SDWAFRM2 and SDWATO2 fields.

A diagnostic message is then built in the variable recording area in the SDWA (SDWAVRA field). See "Analyzing the SYS1.LOGREC Data Set" for the format of this message as constructed by module IDAVBPR1.

Output to the SYS1.DUMP Data Set

If the address space is not accessible or if a machine check interruption has occurred, an SDUMP macro is not issued. If the SDUMP macro can be issued, the message established in the SDWAVRA field and the user region are written to the SYS1.DUMP data set via the SDUMP macro (branch-entered form).

Output to the SYS1.LOGREC Data Set

Parameters are set in the SDWA via the SETRP macro to request R/TM to write the SDWA to the SYS1.LOGREC data set and to percolate to the next FRR in its list.

Output to the User-Specified Dump Data Set

If the user's address space is accessible, indicators are set in the SDWA to cause R/TM to also write the SWA and the data areas listed in the SDWA to the user-specified dump data set when percolation is completed.

Diagnostic Output and Recovery Processing for VIO Module IDAVBPP1

Module IDAVBPR1 is entered at its starting address to perform error recovery processing for module IDAVBPP1. IDAVBPP1 issues a SETFRR macro to establish the entry point.

Control Information and Messages Established in the SDWA

If the user's address space is accessible, the recovery routine sets the starting and ending address of the DSPCT header and the BUFC in the SDWA (SDWADPSL field).

A diagnostic message is then built in the variable recording area in the SDWA (SDWAVRA field). See "Analyzing the SYS1.LOGREC Data Set" for the format of this message as constructed by module IDAVBPR1.

Output to the SYS1.DUMP Data Set

If the address space is not accessible or if a machine check interruption has occurred, an SDUMP macro is not issued. If the SDUMP macro can be issued, the user region and the message established earlier in the SDWAVRA field are written to SYS1.DUMP via the SDUMP macro (branch-entered form).

Output to the SYS1.LOGREC Data Set

Parameters are set in the SDWA via the SETRP macro to request R/TM to write the SDWA to SYS1.LOGREC and to percolate to the next FRR in its list.

Output to the User-Specified Dump Data Set

If the user's address space is accessible, flags are set in the SDWA to cause R/TM to write the data areas listed in the SDWADPSA field to the user-specified dump data set.

Diagnostic Output and Recovery Processing for VIO Module IDAVBPS1

VBPS1FRR is the entry point in module IDAVBPR1 for the error recovery routine for module IDAVBPS1. IDAVBPS1 issues a SETFRR macro to establish the entry point in a LIFO list of recovery routine addresses maintained by R/TM.

Control Information and Messages Established in the SDWA

If the user's address space is accessible, the recovery routine sets the following information in the SDWA:

The starting and ending addresses of the DSPCT header are set in the SDWAFRM1 and SDWATO1 fields.

A diagnostic message is then built in the variable recording area in the SDWA (SDWAVRA field). See "Analyzing the SYS1.LOGREC Data Set" for the format of this message as constructed by module IDAVBPR1.

Output to the SYS1.DUMP Data Set

If the address space is not accessible or if a machine check interruption has occurred, an SDUMP macro is not issued. If the SDUMP macro can be issued, the message established in the SDWAVRA field and the user region are written to the SYS1.DUMP data set via the SDUMP macro (branch-entered form).

Output to the SYS1.LOGREC Data Set

After issuing the SDUMP macro, an attempt is made to reenter module IDAVBPS1 to continue scratch processing. If R/TM has set flags in the SDWA that indicate that the caller's address space is accessible and that a retry is possible, the SETRP macro is issued specifying the RECORD=YES option and an address (FREEMAP, or FREEDSPC) for reentry into IDAVBPS1. Control is then returned to R/TM to write the SDWA to the SYS1.LOGREC data set and to pass control to IDAVBPS1 at the reentry point specified by the SETRP macro.

Otherwise (that is, when a continuation of scratch processing in module IDAVBPS1 is not feasible), a SETRP macro is issued with only the RECORD=YES option, and control is returned to R/TM to write the SDWA to the SYS1.LOGREC data set and to then percolate to the next FRR routine.

The information on this page has been deleted by OS/VS2 MVS Supervisor Performance #2.

Diagnostic Output for VIO Modules IDDWIAPP, IDDWICPI, and IDDWITRM

Module IDDWIFRR is entered at its starting address to perform error recovery processing for modules IDDWIAPP, IDDWICPI, and IDDWITRM. Modules IDDWIAPP, IDDWICPI, and IDDWITRM issue a SETFRR macro to establish the entry point.

Control Information and Messages Established in the SDWA

A diagnostic message is then built in the variable recording area in the SDWA (SDWAVRA field). See "Analyzing the SYS1.LOGREC Data Set" for the format of this message as constructed by module IDDWIFRR.

If the user's address space is accessible, the recovery routine also sets in the SDWA (SDWADPSL field) the starting and ending addresses of the current channel program, IOB, and DEB. Flags are also set in the SDWA to request a dump of these control blocks and the SWA and LSQA when percolation through the FRR routine is completed.

Output to the SYS1.DUMP Data Set

The messages established in the SDWAVRA field and the user region are written to the SYS1.DUMP data set via the SDUMP macro (branch-entry form).

Output to the SYS1.LOGREC Data Set

If recovery processing is for module IDDWICPI, special processing is performed under the following conditions:

- R/TM allows IDDWICPI processing to be resumed.
- Current processing does not represent error recovery following a prior attempt to resume IDDWICPI processing.
- Flags in the SDWA indicate that the error type is a protection check or addressing exception.

Under these circumstances, a SETRP macro is issued with the RECORD=YES option and with an address for reentry to module IDDWICPI. Control is returned to R/TM to write the SDWA to the SYS1.LOGREC data set and to pass control to IDDWICPI at the reentry point specified by the SETRP macro.

Note: When control is returned, IDDWICPI interprets the error condition as a channel program error, and VIO processing continues on that basis.

In all other cases (that is, processing is for module IDDWIAPP or IDDWITRM or that special processing conditions for module IDDWICPI do not exist), a SETRP macro is issued with only the RECORD=YES option, and control is returned to R/TM to write the SDWA to the SYS1.LOGREC data set and to then percolate to the next FRR.

Output to the User-Specified Dump Data Set

Unless overlaid with addresses supplied by subsequent FRRs, when percolation through FRRs and ESTAE routines is completed, the data areas whose address ranges are placed in the SDWA (as described earlier in this section under "Control Information and Messages Established in the SDWA") are dumped to the data set defined by the user's SYSABEND or SYSUDUMP DD statement. The SWA and LSQA are also dumped unless succeeding routines alter the flags set by this FRR in the SDWA (SDWASWA and SDWALSQA flags). The dump will not be taken if succeeding ESTAE routines suppress the dump by specifying DUMP=NO.

Diagnostic Output for VIO Module IDDWICLS

IDDFRRCL is the entry point in module IDDWIFRR for error recovery processing for module IDDWICLS. The entry point is established by IDDWICLS via a SETFRR macro.

Control Information and Messages Established in the SDWA

A diagnostic message is then built in the variable recording area in the SDWA (SDWAVRA field). See "Analyzing the SYS1.LOGREC Data Set" for the format of this message as constructed by module IDDWIFRR.

Output to the SYS1.DUMP Data Set

The messages established in the SDWAVRA field and the user region are written to the SYS1.DUMP data set via the SDUMP macro (branch-entry form).

Output to the SYS1.LOGREC Data Set

A SETRP macro specifying the RECORD=YES option is issued, and control is returned to R/TM to write the SDWA to the SYS1.LOGREC data set and to then percolate to the next FRR.

Output to the User-Specified Dump Data Set

The SWA and LSQA are written to the dump data set unless succeeding routines alter the flags set by this FRR in the SDWA (SDWASWA and SDWALSQA flags). The dump will also not be taken if succeeding FRRs or ESTAE routines suppress the dump by specifying DUMP=NO.

Diagnostic Output and Recovery Processing for VIO Module IDDWIJRN

IDDESTJR is the entry point in module IDDWIFRR for error recovery processing for module IDDWIJRN. The entry point is established by IDDWIJRN via an ESTAE macro. If R/TM does not pass an SDWA pointer when it passes control to IDDWIJRN's ESTAE recovery routine, the recovery routine immediately returns control to R/TM.

Control Information and Messages Established in the SDWA

A diagnostic message is then built in the variable recording area in the SDWA (SDWAVRA field). See "Analyzing the SYS1.LOGREC Data Set" for the format of this message as constructed by module IDDWIFRR.

If the user's address space is accessible, the recovery routine also sets in the SDWA (SDWADPSL field) the starting and ending addresses of the TCB and the JSCB. Flags are also set in the SDWA to request a dump of these control blocks and the SWA and LSQA when percolation through the FRRs and ESTAE routines is completed.

Output to the SYS1.DUMP Data Set

The messages established in the SDWAVRA field and the user region are written to the SYS1.DUMP data set via the SDUMP macro.

Output to the SYS1.LOGREC Data Set

If resumption of journal processing is not possible, the SETRP macro is issued with the RECORD=YES option, and control is returned to R/TM to write the SDWA to the SYS1.LOGREC data set and to then percolate to the next ESTAE routine.

Otherwise, if resumption of IDDWIJRN processing is possible, the SETRP macro is issued with only an address for reentry to module IDDWIJRN. Control is then returned to R/TM to exercise the option specified via the SETRP macro.

Output to the User-Specified Dump Data Set

Unless overlaid with addresses supplied by subsequent ESTAE routines, when percolation through ESTAE routines is completed, the data areas whose address ranges are placed in the SDWA (as described earlier in this section under "Control Information and Messages Established in the SDWA") are dumped to the data set defined by the user's SYSABEND or SYSUDUMP DD statement. The SWA and LSQA are also dumped unless succeeding routines alter the flags set by this ESTAE routine in the SDWA (SDWASWA and SDWALSQA flags). The dump will not be taken if succeeding ESTAE routines suppress the dump by specifying DUMP=NO.

Locating Code Segments that Detect Channel Program Errors

Channel program ending status is presented in the following IOB fields: IOBCSW (IOB+9), IOBSENS0 (IOB+2), and IOBSENS1 (IOB+3). The posting of these fields is consistent with real device processing and is transparent to the user of VIO. IOBSENS0 and IOBSENS1 are posted only for unit-check conditions.

The following error or unusual conditions are posted in the IOBCSW+4:

Error or Unusual Condition	Bit Settings
Program check	X'0020'
Unit exception	X'0100'
Incorrect length	X'0040'
Channel-control check	X'0004'
Unit check	X'0200'

Figure 14 provides a reference to labels of subroutines in module IDDWICPI that set error flags for all conditions except the unit-check condition. (Unit check is handled separately in Figure 15 due to the way unit-check and sense bytes are posted.)

Note that more than one subroutine can set a *program check* status indicator in the IOBCSW. To determine which subroutine detected the error condition, use the address of the failing CCW (IOBCSW+1) to examine the op-code of the failing CCW. The correlation of this information with the information in the "reason set" column should provide you with the label of the subroutine that detected the error.

CSW Status	Routine Where Set	Prior Label	Reason Set
Program Check	Op code breakdown	CPIIOSTC	First CCW not on doubleword boundary
	CCW format check	CPIPGMCK	TIC to CCW not on doubleword boundary
	CCW format check	CPIPGMCK	TIC to a TIC
	CCW format check	CPIPGMCK	Byte count zero*
	CCW format check	CPIPGMCK	Bits 37-39 of CCW nonzero*
Unit Exception	Read routine	CPIRD	Data length of zero for RD, RKD, RCKD, WD, or WCKD
Incorrect Length	Op code end routine	CPIEND40	CCW byte count differed from the actual byte count and the SILI bit was not set
Channel Control Check	Op code breakdown	CPIDCERR	A control, sense, or search CCW specified data chaining (VIO does not support data chained CCWs for these operation codes.)
Unit Check (See Figure 15)			

*TIC operations excluded

Figure 14. Routines that Detect CCW Errors (Except for Unit Check)

Figure 15 provides a reference to labels of routines that cause the unit-check and sense bytes to be posted in the IOB. In order to determine which routine caused the posting, do the following:

- Locate the WICERDSP field in the WICB (WICB + X'40'). If unit check has been set, this field is used to move the appropriate sense information into the IOB (see the footnote to Figure 15).
- Using the value in the WICERDSP field, locate the same value in the "WICERDSP" column in Figure 15. Information in columns aligned with the located value will indicate which routine in module IDDWICPI detected the error.
- When the same value is set by several subroutines, use the failing CCW's address (IOBCSW+1) to determine the op-code, and then correlate the op-code with information in the "routine where set" and "reason set" columns in the table to identify the code segment where the error was detected.

WICERDSP*	Routine Where Set	Prior Label	Reason Set	Sense Bytes—2314	Sense Bytes—RPS Devices	Names of Bits Set
2	Op code breakdown	CPIOPERR	Invalid op code	X'8000'	X'8000'	Command reject
	Op code breakdown	CPIIOSFM	Invalid file mask			
	Read routine	CPIRSTER	Read sector to non-RPS device			
	NOP. Restore set sector	CPISSTE2	Set sector to non-RPS device or value exceeds maximum			
4	Op code breakdown	CPIOPERR	Op code cannot be preceeded by set-file-mask op code	X'8010'	X'8000'	Command reject and/or invalid sequence
	Write routine	CPISEQ05	Write prerequisites not satisfied			
6	Op code breakdown	CPIIOSSK	Invalid initial seek	X'8001'	X'8000'	Command reject and/or seek check
	Seek routine	CPISEKE2	Invalid seek argument			
8	Write routine	CPIFMER	File mask prohibits a write	X'8004'	X'8000'	Command reject and/or overflow incomplete
10(A)	Write routine	CPIWRT40	Track overrun condition	X'0040'	X'0040'	Track overrun
	Write routine	CPIWRT50	Track overrun condition			
12(C)	Positioning routine	CPIPOS90	Single track with no record found	X'0008'	X'0008'	No record found
	Search HA routine	CPISHAER	Single track with no record found			
	Search ID routine	CPISIDER	Single track with no record found			
	Search key routine	CPIKEYER	Single track with no record found			
14(E)	Seek command routine	CPISEKE1	File mask prohibits a seek	X'0004'	X'0004'	File protect
16(10)	Track request routine	CPITRK20	File mask prohibits a head switch	X'0004'	X'0004'	File protect
18(12)	Track request routine	CPITRK20	File mask prohibits head switch for overflow-record processing	X'0005'	X'0005'	File protect and overflow incomplete
20(14)	Track request routine	CPITRK20	End of cylinder reached while trying to switch heads	X'0020'	X'0020'	End of cylinder
22(16)	Track request routine	CPITRK20	End of cylinder reached while trying to switch heads when processing an overflow record			

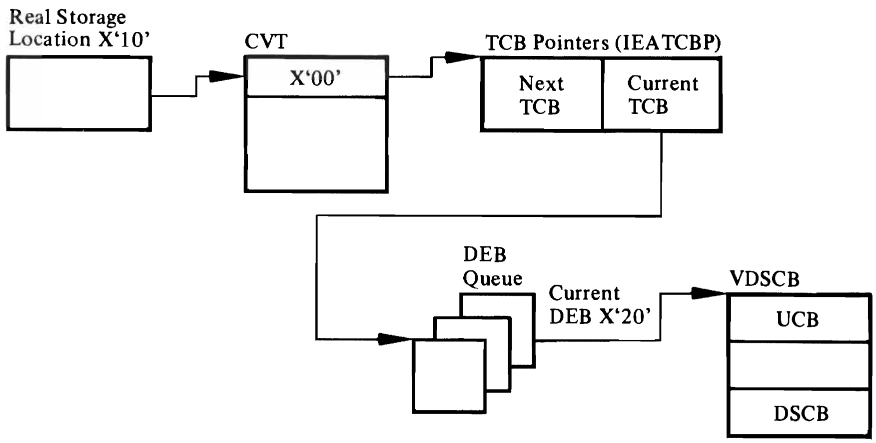
*Bit settings for the IOB sense bytes are acquired from the error table based on a displacement value established in the WICB (WICERDSP field) by the routine that detects an error.

Figure 15. Routines that Detect CCW Errors (for Unit Check)

Debugging VIO Errors

The following path can be followed in a system dump to isolate a VIO module that detects an error condition.

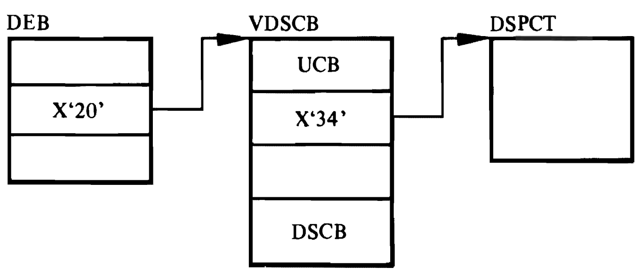
1. Locate the DEB by examining the following chain of pointers and control blocks:



   However, if it is an EXCP request, locate the EXCP debugging area from offset X'C0' in the TCB. See *OS/VS2 I/O Supervisor Logic* for the format of this area, which contains indications of where the error occurred, the PSW when R/TM was entered, the registers when R/TM was entered, and the RQE. The RQE has the addresses of the VDSCB, IOB, DEB, TCB, etc.

2. If the UCB address in the DEB is greater than 64K, then you know it is a VIO UCB because VIO UCBs have 3-byte addressability and do not reside in "low-core."

3. Then locate the DSPCT by examining the following chain of pointers and control blocks:



4. Using status information in the DSPCT header, you can determine which VBP module was in control if the error was detected during VBP's processing, and using addresses in the VDSCB, you can find other key VIO control blocks—the VIO buffer, WICB, and VBPPL. From the WICB, you can determine which EIP module was in control if the error was detected during EIP's processing.

Messages Issued by VIO

IEC006I—UNABLE TO ACTIVATE A VIO DATA SET DURING RESTART PROCESSING

Explanation: ASM (auxiliary storage management) was unable to reset its control blocks for a VIO data set to the data set's status at the time it was journaled. This message should be followed by message IEF086I.

System Action: Terminate restart processing.

Programmer Action: Rerun job. Also refer to message IEF086I.

Problem Determination: Refer to Table 1, items 1, 3, 4, and 29, in the *OS/VS Message Library: VS2 System Messages.* Refer to message IEF086I.

ASM Module that Detects the Error: ILRINT00.

VIO Module that Issues Message: Module IDAVBPJ2 examines the return code in register 15 that is returned by ASM, and if the return code is greater than 8, IDAVBPJ2 issues the message.

INDEX

For additional information about any subject listed in this index, refer to the publications that are listed under the same subject in *OS/VS2 Master Index of Logic*, GY28-0694.

A

B

C

D

E

R

S

T

U

V

W

X

SY26-3859-0

OS/VS2 VIO Logic - VS2.03.807 (File No. S370-30) Printed in U.S.A. SY26-3859-0

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

Reader's Comment Form

OS/VS2 VIO Logic
SY26-3859-0

Your comments about this publication will help us to improve it for you. Comment in the space below, giving specific page and paragraph references whenever possible. All comments become the property of IBM.

Please do not use this form to ask technical questions about IBM systems and programs or to request copies of publications. Rather, direct such questions or requests to your local IBM representative.

If you would like a reply, please provide your name and address (including ZIP code).

Fold on two lines, staple, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

SY26-3859-0

Fold and Staple

First Class Permit
Number 6090
San Jose, California

Business Reply Mail

No postage necessary if mailed in the U.S.A.

Postage will be paid by:

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and Staple

OS/VS2 VIO Logic - VS2.03.807 (File No. S370-30) Printed in U.S.A. SY26-3859-0

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)