File Number S360-30
Order Number GY28-6617-6

Program Logic

OS BDAM Logic

Release 21

Program Number 360S—DM—509

This book describes the internal logic of the basic direct access method (BDAM). It is intended as a reference book for programming support representatives and system programmers maintaining BDAM routines and for customer programmers modifying BDAM routines.

A general understanding of data management is prerequisite knowledge for understanding the information in this book. See *Data Management Services Guide,* GC26–3746, for background information on data management.

Seventh Edition (February 1972)

This book is a major revision of, and makes obsolete, GY28–6617–5. It corresponds to Release 21 of the operating system and to all subsequent releases unless otherwise indicated in new editions or technical newsletters. Changes may be made to the information in this manual at any time; before using this manual in connection with the operation of IBM systems, consult the latest *SRL Newsletter,* GN20–0360, for the editions that are applicable and current.

Forms for readers' comments are provided at the back of this book. If the forms have been removed, address comments to IBM Corporation, Programming Publications, Department D78, Monterey and Cottle Roads, San Jose, California 95114.

For copies of this or any other IBM publication, see your IBM representative or call your local IBM Branch Office.

ABOUT THIS BOOK

This book describes the internal logic of the basic direct access method (BDAM). It is intended as a reference book for programming support representatives and system programmers maintaining BDAM routines and for customer programmers modifying BDAM routines.

The major parts of this book and the information contained in them are as follows:

- The first part, the Introduction, explains what BDAM does and how it relates to the rest of the operating system.
- The second part, Program Description, describes what each BDAM module does, the intermodule relationships, and the flow of control between BDAM modules and other parts of the operating system.
- The last four parts of the book contain flowcharts and reference material that will be useful in analyzing BDAM listings and storage dumps.

A general understanding of data management is prerequisite knowledge for understanding the information in this book. See *Data Management Services Guide,* GC26–3746, for background information on data management.

The following publications are referred to in this book:

OS DADSM Logic, GY28–6607, which describes allocation of extents to a data set

OS System Control Blocks, GC28–6628, which describes the control blocks used by BDAM

OS Data Management Macro Instructions, GC26–3793, which contains information about many of the macro instructions referred to in this book

OS I/O Supervisor Logic, GY28–6616, which describes the appendage vector table

OS Messages and Codes, GC28–6631, which explains the messages issued by BDAM modules

OS Open/Close/EOV Logic, GY28–6609, which describes the open and close routines of data management

OS SAM Logic, GY28–6604, which describes the BPAM routines used in the address conversion process

OS Storage Estimates, GC28–6551, which contains information about BDAM modules that must be loaded into the link pack area as a unit

OS Supervisor Services and Macro Instructions, GC28–6646, which explains the ENQ and DEQ macro instructions

OS MVT Supervisor Logic, GY28–6659, which describes the exit effector routine of the task supervisor

IBM System/360 Component Descriptions — 2841 and Associated DASD, GA26–5988, which explains the coding of channel programs

CONTENTS

FIGURES

SUMMARY OF CHANGES FOR RELEASE 21

Module Eliminated

The start–I/O appendage module, IGG019KS, has been eliminated. Start–I/O logic is now in modules IGG019KL and IGG019LE, the dynamic buffering modules.

Material Added

- Hexadecimal displacements for control blocks and lists (in the "Data Areas" section of the manual)
- Messages and codes issued by BDAM modules (in Appendix E)
- A glossary (at the back of the manual)
- Figure 5, which will help you get to the description of a particular channel program generating module and then to the illustration of the channel program that that module builds

Miscellaneous Changes

- Much of the material in this manual has been rewritten for clarification.
- Minor technical changes have been made.

ACRONYMS USED IN THIS BOOK

ASI	asynchronous interrupt
AVT	appendage vector table
BCB	buffer control block
BDAM	basic direct access method
BPAM	basic partitioned access method
BSAM	basic sequential access method
CCW	channel command word
DADSM	direct–access device space management
DCB	data control block
DEB	data event block
DECB	data event control block
DSCB	data set control block
I/O	input/output
IOB	input/output block
IOS	input/output supervisor
IRB	interrupt request block
RPS	rotational position sensing
SVC	supervisor call
TCB	task control block
UCB	unit control block

INTRODUCTION

The basic direct access method (BDAM) is a group of routines that retrieves data from and stores data into data sets that are directly organized and reside on direct–access devices.

BDAM is a part of the operating system control program. BDAM routines are grouped into modules that are placed in the supervisor call (SVC) library at system generation. This library is part of the system residence library, which resides on direct–access storage. When BDAM is to be used by a processing program, the modules that will be used in the particular application are loaded from the system residence volume into main storage.

Relationship of BDAM to the Operating System

When a data set to be processed using BDAM is opened, the Open routine of data management (discussed in *OS Open/Close/EOV Logic)* gets control and calls the BDAM module that will begin opening the data set.

When an input/output operation terminates, the processing program is interrupted. The I/O supervisor then gives control to a BDAM module that will schedule the remaining processing required for the I/O request. When all processing for the request has been completed, the supervisor returns control to the processing program.

When a data set is closed, the Close routine of data management gets control and calls the BDAM module that closes the data set.

Overview of BDAM

For purposes of discussion, BDAM is divided into three sections, based on the type of operations it performs:

- Opening a data set
- Processing an I/O request
- Closing a data set

The major operations done in each of these sections are discussed in the following paragraphs. Figure 37 at the back of the book shows at what point in a BDAM application each of these operations is done and which BDAM modules are involved in a given operation.

Opening a Data Set

The modules involved in opening a BDAM data set are called the BDAM open executor modules. The open executor modules:

- Determine which BDAM modules will be used in a particular application and load them into main storage
- Build control blocks and lists for subsequent use by BDAM

Processing an I/O Request

The following operations may be performed when an I/O request is processed:

- Controlling the processing
- Converting addresses
- Building channel programs
- Allocating buffers dynamically
- Changing extents
- Scheduling remaining processing
- Maintaining exclusive control
- Checking for request completion

Controlling the Processing

The module that controls the processing of a BDAM I/O request is called the foundation module. The foundation module:

- Completes the preparations necessary before an I/O request can be processed
- Passes control to other modules involved in processing an I/O request
- Completes processing of a request after an I/O operation has terminated

Converting Addresses

The modules involved in converting addresses are called the address conversion modules. BDAM converts block addresses whenever relative addressing is specified in a processing program.

The address of a block may be converted:

- From a relative address to an actual address (necessary because the channel program, when it searches for a block, does so using the actual address of that block), or
- From an actual address to a relative address (done when the processing program requests feedback).

The BDAM address conversion modules initiate the conversion process.

When a relative address is being converted to an actual address, a BPAM convert–to–actual routine does the conversion from TTR to MBBCCHHR. When an actual address is being converted to a relative address, a BPAM convert–to–relative routine does the initial conversion from MBBCCHHR to TTR. However, if the relative block addressing scheme is being used, the BDAM address conversion module then converts TTR to a relative block number.

Building Channel Programs

The modules that build channel programs are called channel program generating modules. A channel program is built for every BDAM I/O request. All but one of the channel program generating modules build channel programs that search storage volumes either for data to be brought into main storage or for space on which to place data that is being transferred from main storage. The other channel program generating module builds a channel program that verifies data written on auxiliary storage devices. This module is used only if the write–validity–check option was specified in the processing program.

Allocating Buffers Dynamically

The modules that obtain a buffer, assign it to a request, and then release it are called the dynamic buffering modules. These modules, which are used in processing an I/O request only if the dynamic buffering option is in effect, can be entered from the I/O supervisor or the FREEDBUF SVC. The dynamic buffering modules have two entry points: start I/O and free dynamic buffer.

If entered at the start–I/O entry point, the dynamic buffer module assigns a buffer to the request.

If entered at the free dynamic buffer entry point, the dynamic buffer module either releases the buffer used by an I/O request that has been satisfied or extends the unscheduled list.

Changing Extents

When the extended search option was specified in the processing program and the channel program must switch from one extent to another while reading, writing, or adding a block, the I/O supervisor gives control to a module called the end–of–extent appendage module.

This module determines the address of the next extent (if one is available) to be searched and whether the search limit is reached in that extent.

Scheduling Further Processing

When a channel program ends, the I/O supervisor gives control to a module called the channel end/abnormal end appendage module. This module schedules further processing for an I/O request.

Maintaining Exclusive Control

A module called the exclusive control module is used in processing an I/O request if the exclusive control option is specified in the processing program. This module protects the data portion of a block. It ensures that if more than one request is made for use of a block, only the first request can use it while it is under exclusive control. The exclusive control module maintains a read–exclusive list, which contains the addresses of all blocks presently being used that require exclusive control.

Checking for Request Completion

A module called the check module is used in processing an I/O request if the processing program uses the CHECK macro instruction to ensure that a given I/O request has been satisfied. This module determines if, and waits if necessary until, a request is posted as complete. If errors have been detected during the I/O operation, the check module gives control to a user's error routine.

Closing a Data Set

One BDAM module, called the close executor module, is involved in closing a BDAM data set. The close executor module:

- Releases to the system all main storage obtained by BDAM
- Restores the fields of the DCB that have been changed by BDAM routines

PROGRAM DESCRIPTION

This section describes each of the major operations performed by BDAM. The modules used in each operation, the conditions under which they are used, and the intermodule relationships are discussed. Figure 37 at the back of the book is a composite of Figures 1, 3, 4, 9, and 11, which appear in this section. Figure 37 will give you a general perspective of BDAM and its relation to a processing program and to the operating system.

Open Executor Modules (IGG0193A, 3C, 3E, 3F, 3G)

BDAM has five open executor modules. Either three or four of these modules are given control during the opening of a BDAM data set. These modules:

- Determine which BDAM modules will be used in a particular application and load them into main storage
- Build control blocks and lists for subsequent use by BDAM

IGG0193A

When the OPEN macro instruction is encountered in a processing program that specifies BDAM, control is given to the open routine of data management (see Figure 1). The open routine gives control to BDAM module IGG0193A.

From information in the DCB and the DSCB for the data set, IGG0193A determines the amount of main storage needed to build the actual extents in the DEB. IGG0193A then gets main storage for the DEB and the appendage vector table, which immediately precedes the DEB in main storage. (See *OS I/O Supervisor Logic* for more information about this table.)

IGG0193A places control information in the fields of the DEB. It also provides space (in the DEBSUBID fields of the DEB) for the subroutine identifications of BDAM modules that are required as a result of specifications given in the DCBMACRF, DCBOPTCD, and DCBRECFM fields of the DCB. As each current DEB is being constructed, it is attached to the appropriate TCB.

Note: A user label track, if allocated, is not included in the extents reflected in the DEB.

IGG0193A also gets main storage for and initializes the read–exclusive list.

Then IGG0193A checks the where–to–go table, which is created by the data management Open routine, to determine whether it includes other DCBs that require the use of this module. If there are none, IGG0193A gives control to module IGG0193C. Otherwise, this module is reentered and does the required processing.

IGG0193C

Module IGG0193A gives control to module IGG0193C. IGG0193C's major activity, which it shares with module IGG0193G, is to load into main storage the BDAM modules that will be used by the processing program.

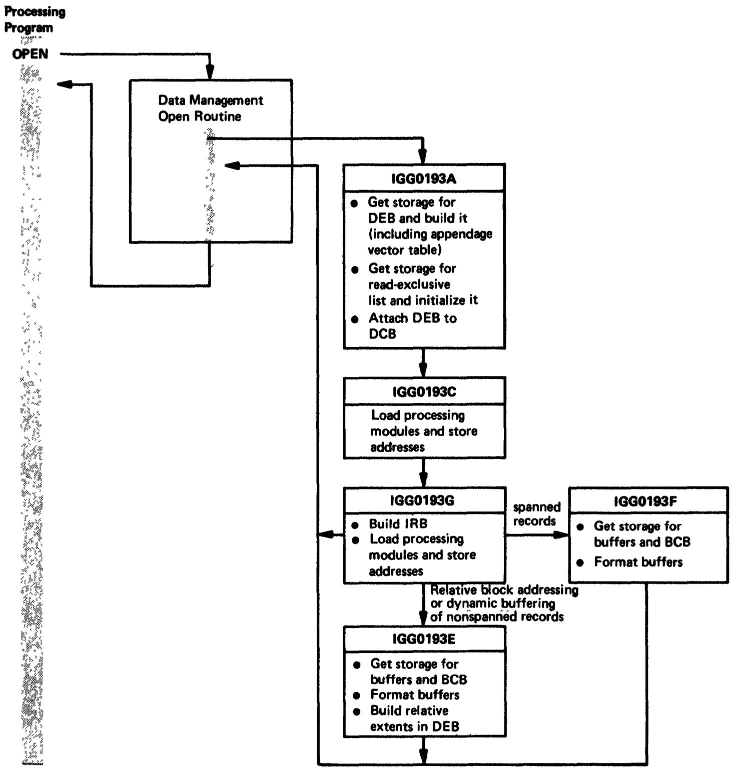

Processing Program
OPEN
Data Management Open Routine
IGG0193A
• Get storage for DEB and build it (including appendage vector table)
• Get storage for read-exclusive list and initialize it
• Attach DEB to DCB
IGG0193C
Load processing modules and store addresses
IGG0193G
• Build IRB
• Load processing modules and store addresses
spanned records
IGG0193F
• Get storage for buffers and BCB
• Format buffers
Relative block addressing or dynamic buffering of nonspanned records
IGG0193E
• Get storage for buffers and BCB
• Format buffers
• Build relative extents in DEB

Figure 1. When the processing program issues the OPEN macro instruction, the data management Open routine gets control and, in conjunction with the **BDAM open executor modules,** opens the data set.

There is a routine within the module for:

- Determining which foundation module will be used and loading it. IGG0193C puts the addresses of certain optional BDAM modules in the foundation module (see Figure 2).
- Determining whether an address conversion module is required and, if so, loading it. (An address conversion module is required only if an addressing scheme other than actual addressing is specified in the processing program.)
- Determining which channel program generating module(s) will be used and loading it.
- Determining if any optional modules will be used and, if so, loading them.

If main storage contains any BDAM modules used when a data set was previously opened, IGG0193C obtains their main-storage addresses from the supervisor, and they are not reloaded.

Module	Module, Control Block, or List in Which Address is Placed
Channel End/Abnormal End Appendage (IGG019KU)	Appendage Vector Table[1]
Check (IGG019LI)	DCB
Dynamic Buffering (IGG019KL,LE)[2]	Appendage Vector Table[1]
End–of–Extent Appendage (IGG019LC)	Appendage Vector Table[1]
Exclusive Control (IGC0005C)[3]	DCB
Foundation (IGG019KA,KJ)	DCB
ID (IGG019KK)	Foundation Module (IGG019KA,KJ)
Key (IGG019KI)	Foundation Module (IGG019KA,KJ)
Key Extended Search (IGG019KW)	Key Module (IGG019KI)
Preformat (IGG019KO)	DCB
Preformat Extended Search (IGG019LA)	Preformat Module (IGG019KO)
Relative Block Feedback (IGG019KG,KH)	DCB
Relative Track Conversion (IGG019KC)	Foundation Module (IGG019KA,KJ)
Self–format (IGG019KM,KN)	DCB
Self–format Extended Search (IGG019KY)	Self–format Module (IGG019KM,KN)
Write Verify (IGG019KQ)	Foundation Module (IGG019KA,KJ)

[1] The address of the appendage vector table is in the DEB

[2] The start–I/O entry point is in the first field of the appendage vector table; the free dynamic buffer entry point is at the start–I/O entry point +8.

[3] The address of the read–exclusive list is in the DCB

Figure 2. **BDAM module addresses are stored in the DCB, the AVT, or other BDAM modules** by BDAM open executor modules.

Notes:

- If BDAM modules are included in the link pack area, they are accessible to all jobs that request them (since BDAM modules are reenterable). All BDAM modules that branch to each other by means of addresses placed within them by open executor modules must be included as a unit in the link pack area. (See the publication *OS Storage Estimates* for further information on this subject.)
- If a given main task loads a BDAM module into a region assigned to that task, only that task or its attached subtask has access to the module. If another main task requires the same module, it must load the module into its own region. The *OS MVT Supervisor Logic* manual contains more detailed information concerning these situations.

As modules are loaded into main storage, IGG0193C puts their module identifications in the DEB. (After processing of a data set has been completed, the close routine uses this information for releasing storage areas.) Space for the subroutine identifications was allotted by IGG0193A. Then IGG0193C checks the where–to–go table, which was created by the data management open routine, to determine whether it includes other DCBs that require the use of this module. If there are none, control is given to module IGG0193G. Otherwise, the module is reentered and does the required processing.

IGG0193G

Module IGG0193C gives control to module IGG0193G. IGG0193G continues the processing begun by IGG0193C by loading any optional BDAM modules specified in the following DCB fields: DCBMACRF, DCBRECFM, DCBOPTCD, and DCBBFTK.

It then stores in the DEB the subroutine identifications of the loaded modules and the number of modules that were loaded. If main storage contains any BDAM modules used when a data set was previously opened, IGG0193G obtains their main–storage addresses from the supervisor, and they are not reloaded.

There is also a routine within IGG0193G for:

- Determining which appendage modules will be used and loading them into main storage.
- Initializing some of the fields of the DCB.
- Branching to a supervisor routine to build the IRB. *Note*: The IRB flags specify execution in problem program state using the protection key of the problem program task that opens the data set.

After the preceding operations are done, IGG0193G checks the where–to–go table, which is created by the data management Open routine, to determine if any more DCBs associated with the current OPEN macro instructions require the use of this module. If so, this module is reentered and does the required processing. If not, control passes to:

- IGG0193E, if relative block addressing or dynamic buffering of nonspanned records is specified in the DCB,
- IGG0193F, if spanned records are specified in the DCB, or
- The Open routine of data management

IGG0193E

Module IGG0193G gives control to IGG0193E if relative block addressing or dynamic buffering of nonspanned records is specified in the DCB.

If dynamic buffering is specified in the DCBOPTCD field of the DCB, IGG0193E uses the buffer information in the DCB macro instruction to obtain the main storage necessary for the buffers and a buffer control block (BCB). IGG0193E then divides the buffer area in main storage into the requested number of buffers and puts the address of the first available buffer in the BCB. Then it chains the buffers together by putting the address of the next buffer into the first word of each buffer.

If relative block addressing is specified in the DCBOPTCD field of the DCB, IGG0193E builds the relative extents in the DEB (see the DEB in the "Data Areas" section of this manual for the format of the relative extent areas). The relative extents contain information that, during BDAM processing, is used to convert relative block addresses to actual addresses. Address conversion is necessary because data set blocks on a direct–access device can only be accessed using actual addresses. One relative extent is built for each actual extent in the DEB.

If relative block addressing and track overflow are indicated in the DCBOPTCD and DCBRECFM fields of the DCB respectively, module IGG0193E builds modified relative extents in the DEB (see the DEB in the "Data Areas" section of this manual for the format of the relative extents). In addition, IGG0193E builds two fields between the last actual extent and the first relative extent in the DEB. These fields are referred to as the overflow section of the DEB and contain information about the size of a period, as discussed in Appendix A.

Before passing control to another routine, module IGG0193E determines if any more DCBs require the use of this module. If so, this module is reentered and does the required processing. If not, the data set is open as far as BDAM is concerned, and control is given either to an open executor module for another data set or to the open routine of data management.

IGG0193F

IGG0193G passes control to IGG0193F if spanned record format is specified in the DCB macro instruction.

If dynamic buffering is not specified in the DCB, IGG0193F gets main storage for a pool of segment work areas (illustrated under "Buffer Pools" in the "Data Areas" section of this book) and a BCB.

If dynamic buffering is specified in the DCBOPTCD field of the DCB, IGG0193F uses the buffer information in the DCB macro instruction to obtain the main storage necessary for the buffers and a BCB. IGG0193F then divides the buffer area in main storage into the requested number of buffers and puts the address of the first available buffer in the BCB. Then it chains the buffers together by putting the address of the next buffer into the first word of each buffer.

Before passing control to another routine, module IGG0193F determines if any more DCBs require the use of this module. If so, this module is reentered and does the required processing. If not, the data set is open as far as BDAM is concerned, and control is passed either to an open executor module for another data set or to the Open routine of data management.

Foundation Modules (IGG019KA, KJ)

BDAM has two foundation modules. Module IGG019KJ is used as the foundation module when spanned records are specified in a processing program; IGG019KA is used for nonspanned records. Each of the foundation modules has two routines: a base routine and an asynchronous interrupt (ASI) routine.

Base Routine

Control passes to the base routine of the foundation module when the READ or WRITE macro instruction issued by the processing program is executed (see Figure 3).

The base routine:

- Establishes the validity of options specified in the READ or WRITE macro instruction.
- Combines options specified in the type field of the READ or WRITE macro instruction with options specified in the DCBOPTCD field of the DCB. The result of this combination is placed in the DECB. When the IOB is built, this information is transferred to it.
- Determines the amount of main storage needed for the IOB. Since the channel program is one part of the IOB whose size varies, the size of the channel program to be constructed for the request must be determined. The base routine determines the size by checking the type field of the DECB.

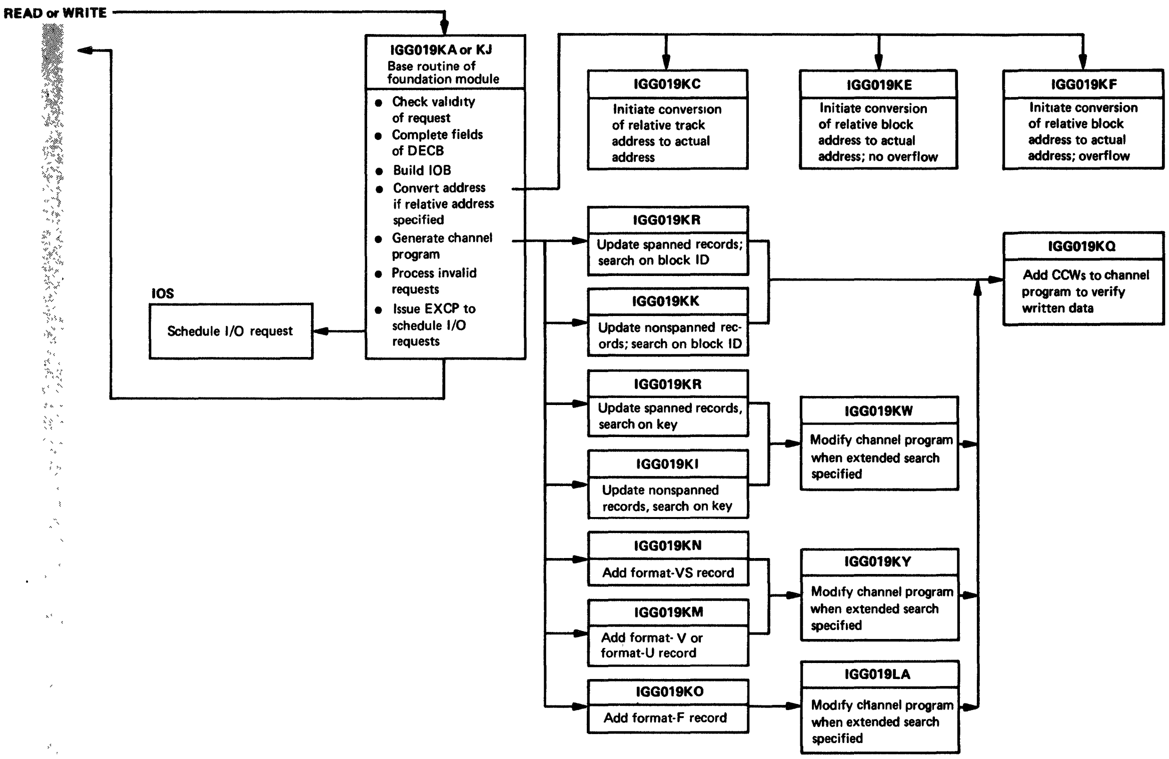

READ or WRITE
IGG019KA or KJ
Base routine of foundation module
● Check validity of request
● Complete fields of DECB
● Build IOB
● Convert address if relative address specified
● Generate channel program
● Process invalid requests
● Issue EXCP to schedule I/O requests
IOS
Schedule I/O request
IGG019KC
Initiate conversion of relative track address to actual address
IGG019KE
Initiate conversion of relative block address to actual address; no overflow
IGG019KF
Initiate conversion of relative block address to actual address; overflow
IGG019KR
Update spanned records; search on block ID
IGG019KK
Update nonspanned records; search on block ID
IGG019KQ
Add CCWs to channel program to verify written data
IGG019KR
Update spanned records, search on key
IGG019KI
Update nonspanned records, search on key
IGG019KW
Modify channel program when extended search specified
IGG019KN
Add format-VS record
IGG019KM
Add format-V or format-U record
IGG019KY
Modify channel program when extended search specified
IGG019KO
Add format-F record
IGG019LA
Modify channel program when extended search specified

Figure 3. When the processing program issues the READ or WRITE macro, the **base routine of the foundation module** gets control. The base routine, in performing its operations, gives control to the other BDAM modules shown.

- Obtains an IOB from a pool of available IOBs (IOBs no longer in use because the I/O requests they were associated with have already been satisfied), if the pool contains one that is large enough. Otherwise, the base routine issues a GETMAIN for the main storage needed for the IOB. In either case, the base routine then builds the IOB.

- Determines the type of block addressing specified in the processing program. If actual addressing was specified, address conversion is not done. If relative addressing was specified, control is given to one of the BDAM modules that converts relative addresses to actual addresses. The address conversion module returns control to the base routine.

- Determines, by checking fields in the IOB and DCB, the type of channel program to be built. Control is then given to a channel program generating module to build the channel program required for the request. If the extended search option has been specified in the processing program, the channel program will reflect the search limit in the DCBLIMCT field of the DCB. When the channel program has been built, the base routine again receives control.

- Establishes and stores the address where the channel program should end. After execution of the channel program, the I/O supervisor compares the address where the channel program should have ended with the address where it actually ended. If the two do not match, the I/O supervisor concludes that the channel program terminated abnormally.

- Processes any invalid requests. An invalid request is based on the difference between the parameters specified in the DECB and the parameters specified in the DCB. Invalid requests can be detected in the processing done by the base routine itself or in the address conversion (IGG019KE, IGG019KC, IGG019KF) or channel program generating (IGG019KO) modules to which the base routine passes control.

When the base routine detects an invalid request, it releases the IOB associated with the request, indicates in the DECB that an invalid request has been detected, posts the ECB to indicate processing of that I/O request is complete, and returns control to the processing program.

Asynchronous Interrupt Routine

Before a request is considered completed, certain processing operations must be done by the foundation module. When the processing program is interrupted by the completion of an input/output operation, the I/O supervisor gives control to the BDAM channel end/abnormal end appendage module (see Figure 4). The ASI routine is scheduled by the supervisor after the BDAM channel end/abnormal end appendage module, IGG019KU, has indicated the need for the ASI routine.

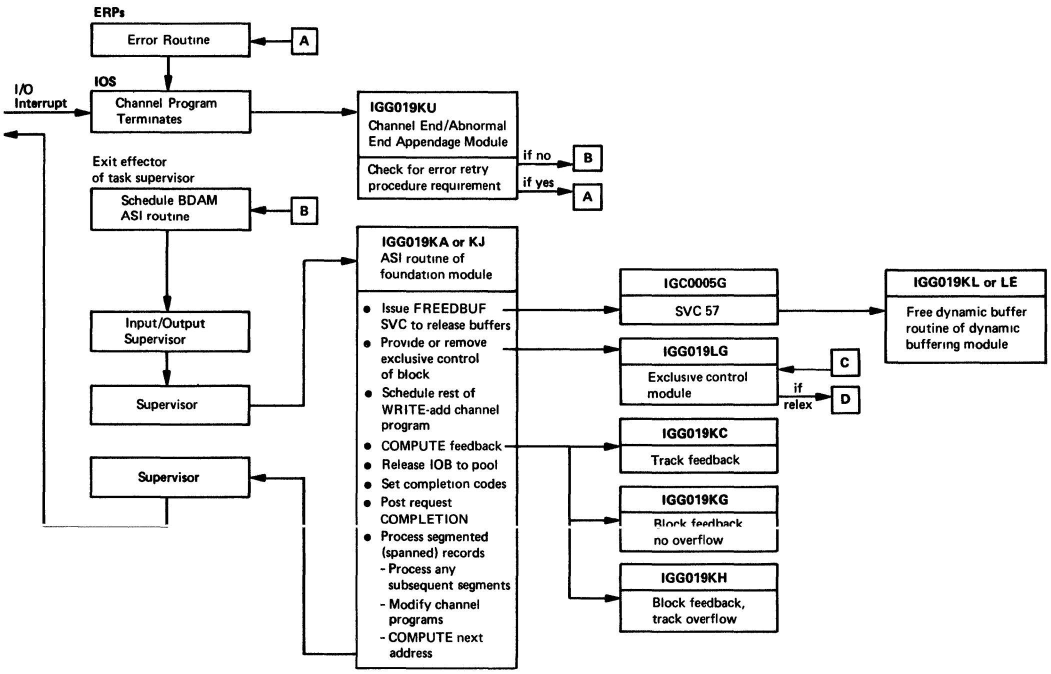

ERPs
Error Routine
A
IOS
I/O Interrupt
Channel Program Terminates
Exit effector of task supervisor
Schedule BDAM ASI routine
B
Input/Output Supervisor
Supervisor
Supervisor
IGG019KU
Channel End/Abnormal End Appendage Module
Check for error retry procedure requirement
if no
B
if yes
A
IGG019KA or KJ
ASI routine of foundation module
Issue FREEDBUF SVC to release buffers
Provide or remove exclusive control of block
Schedule rest of WRITE-add channel program
COMPUTE feedback
Release IOB to pool
Set completion codes
Post request COMPLETION
Process segmented (spanned) records
- Process any subsequent segments
- Modify channel programs
- COMPUTE next address
IGC0005G
SVC 57
IGG019KL or LE
Free dynamic buffer routine of dynamic buffering module
IGG019LG
Exclusive control module
C
if relex
D
IGG019KC
Track feedback
IGG019KG
Block feedback no overflow
IGG019KH
Block feedback, track overflow

Figure 4. The **ASI routine in the foundation module** is scheduled by module IGG019KU after the channel program ends. The ASI routine, in performing its operations, gives control to the other BDAM modules shown.

The ASI routine and those routines branched to by the ASI routine operate under the control of a data set's IRB. Because there is only one IRB for each DCB, it would be impossible for another task using the same DCB to update the same IOB queue or read–exclusive list at the same time. Since the IRB is associated with the task that opened the data set, all processing done on that data set by the ASI routine and routines branched to by the ASI routine is executed under the opener's TCB. Likewise, all system enqueuing by the exclusive control module reflects the opener's TCB. This is important to note when multiple tasks are sharing the same DCB.

The two foundation modules operate differently in the ASI routine and are therefore discussed separately. Each of the ASI routines contains a subroutine for processing requests whose completion is abnormal. A discussion of this subroutine appears after the separate discussions of the ASI routine.

ASI Routine for Spanned Records (Foundation Module IGG019KJ)

The operations performed by the ASI routine when spanned records have been specified are:

- To determine the cause of interruptions.
- To act on this determination either by initiating a restart of a channel program or by branching to error subroutines or other processing modules (such as address conversion and feedback).
- To move the segment just read on input to the user record area; to form the next segment on output and move it to the segment work area to be written.
- To modify the channel program, changing the length to account for the absence of keys on all but the first segment of a segmented record, eliminating searching for subsequent segments (because their location can be determined from the previous location of previous segments), changing the channel commands to search for successive segments by block identification rather than by key, and changing the sequence of CCWs for the next address option so they return the address of the next block rather than the next segment.
- To reinitiate the channel program when subsequent record segments are to be processed or when next address feedback is requested.
- To call the exclusive control module to acquire or release exclusive control of a block.
- To release the request's IOB to the pool of IOBs. After the IOB has been released, the request is posted as complete by the post routine, and control is returned to the supervisor.

ASI Routine for Nonspanned Records (Foundation Module IGG019KA)

The operations performed by the ASI routine when spanned records are not being processed are:

- To determine the cause of interruptions.
- To initiate a restart of a channel program if necessary or branch either to the error subroutine or other processing modules (such as self–format, or address conversion for either relative block or relative track feedback).

- If the request currently being processed is a request to write a block and if the dynamic buffering option has been specified, to issue the FREEDBUF macro instruction to free the buffer that has been allocated for the READ request that corresponds to this WRITE request. (A WRITE request issued without a previous READ request does not allow dynamic buffering, as the buffer is initially assigned for READ requests only.)
- If either the exclusive control option has been specified or a request to add new blocks of variable or undefined length has been specified, to call the exclusive control module to place blocks in or remove them from a queue.
- To release the request's IOB, making it available for reuse. (This operation is performed by the check module if a CHECK macro instruction is encountered and the DCB macro instruction specifies that the CHECK macro is being used.) After the IOB is released, the request is posted as complete by the post routine, and control is returned to the supervisor.

For each request, only those operations that are applicable are performed.

Routine for Abnormal Completion of a Request (Both Foundation Modules)

There are several situations in which abnormal completion of a request occurs in BDAM.

One situation results from device errors. Included in this category are any errors associated with input/output devices and control units or end–of–data–set conditions (which are received by the channel end/abnormal end appendage module as unit exception conditions). An error condition also results when, in the case of a request to add a fixed–length block to an existing data set, a dummy record cannot be found. An indication of device error (or other unit check condition) is established when the I/O supervisor enters the abnormal end routine of the BDAM channel end/abnormal end appendage module, IGG019KU.

Abnormal completion can also occur when a request is made to write a new block whose length is either variable or undefined. If it is determined that there is no available space in which to add the new block, a BDAM routine sets an indicator to inform the processing program so that appropriate action may be taken.

When abnormal completion occurs, this routine releases the related IOB to the IOB pool associated with the DCB, posts an indication of the type of error and an indication of the completion of the request, and returns control to the supervisor. Note that the IOB is released to the IOB pool by the check module if applicable.

Address Conversion Modules (IGG019KC, KE, KF, KG, KH)

The address conversion modules are used only if the processing program specifies relative addressing. These modules provide input for and pass control to BPAM routines that either convert relative addresses to actual addresses or convert actual addresses to relative addresses. Conversion to an actual address is necessary because the channel program searches for a block using its actual address. Conversion to a relative address is done when the processing program specified the feedback option.

If the purpose of address conversion is to get an actual address, one of the following modules is used:

- IGG019KC — processing program specifies relative track address

  Note: This module also used for relative track feedback.
- IGG019KE — processing program specifies relative block number, no track overflow
- IGG019KF — processing program specifies relative block number and track overflow

If the purpose of address conversion is to get feedback, one of the following modules is used:

- IGG019KG — processing program specifies relative block addressing, no track overflow
- IGG019KH — processing program specifies relative block addressing and track overflow
- IGG019KC — processing program specifies relative track addressing

  Note: This module also used for conversion of a relative address to an actual address.

IGG019KC (Relative Track Conversion Module)

Either the base routine or the ASI routine in the foundation module gives control to the relative track conversion module (see Figures 3 and 4). IGG019KC is entered from the base routine if the purpose is to initiate conversion of a relative track address to an actual address. Entry is from the ASI routine if the purpose is to initiate conversion of an actual address to a relative track address. The latter conversion is done if the processing program specifies the feedback option.

Entry From Base Routine

Actual conversion of track addresses is done by a BPAM conversion routine (referred to as the convert–to–actual routine). (For more information on this routine, refer to *OS SAM Logic.)* When the BDAM conversion module gives control to the BPAM routine, it supplies:

- The address at which the actual address is to be placed for use by the channel program (the address of the IOBSEEK field of the IOB)
- The relative track number that the user has indicated in the block address parameter of the request macro instruction
- The address of the DEB (supplied so that the BPAM routine can refer to the actual extents in the DEB)

From the DEB extents, which contain cylinder and track information for the various sections of the data set, the BPAM routine obtains the actual starting address of each extent and the number of tracks in each extent. From this information, the BPAM conversion routine derives the actual address of the block for which the user specified a relative track address. This address is then placed in the IOBSEEK field of the IOB, and the BPAM routine returns control to the BDAM relative track conversion module, which returns control to the base routine of the foundation module.

If the extended search option has been specified, the foregoing procedure is used to determine the actual address of the next track after the last track that will be searched. The starting track address and the number of tracks to be searched (as specified in the LIMCT parameter of the DCB macro instruction) enable the search limit to be computed. This limit is placed in the IOBUPLIM field of the IOB.

Entry From ASI Routine

The ASI routine in the foundation module gets control after a channel program ends. If the processing program specified relative track feedback, the ASI routine gives control to the BDAM relative track conversion module, which gives control to a resident BPAM conversion routine (referred to as the convert–to–relative routine). (For more information on the BPAM routine, refer to the publication *OS SAM Logic.)* The BDAM module gives the BPAM routine both the address where the actual block address can be found (the address of the IOBSEEK field in the IOB) and the address of the DEB. The BPAM routine converts the actual address to a relative track address, places the relative track address in a parameter register, and returns control to the BDAM relative track conversion module. The relative track conversion module stores the relative track address in the BLKREF field and returns control to the ASI routine.

IGG019KE (Relative Block Conversion Module — No Track Overflow)

The base routine of the foundation module gives control to module IGG019KE when the processing program specified a relative block number but did not specify track overflow (see Figure 3).

IGG019KE converts a relative block address to a relative track address. Appendix B shows the calculations done in this conversion process.

After calculating the relative track address, IGG019KE gives control to the BPAM convert–to–actual routine. This routine converts relative track addresses to actual addresses. (For more information on the BPAM routine, refer to *OS SAM Logic.)* When the BDAM conversion module gives control to the BPAM routine, it supplies:

- The address at which the actual address is to be placed for use by the channel program (the address of the IOBSEEK field of the IOB)
- The relative track number that the user has indicated in the block address parameter of the request macro instruction
- The address of the DEB (supplied so that the BPAM routine can refer to the actual extents in the DEB)

The BPAM conversion routine converts the relative track address to an actual address and places the converted address in the IOBSEEK field of the IOB. Then it returns control to the BDAM relative block conversion module, which gives control back to the foundation module.

If the extended search option has been specified, the foregoing procedure is used to determine the actual address of the next track after the last track that will be searched. The starting track address and the number of tracks to be searched (as specified in the LIMCT parameter of the DCB macro instruction) enable the search limit to be computed. This limit is placed in the IOBUPLIM field of the IOB.

IGG019KF (Relative Block Conversion Module — Track Overflow)

The base routine of the foundation module gives control to module IGG019KF when the processing program specified a relative block number and track overflow (see Figure 3).

IGG019KF converts a relative block address to a relative track address when track overflow is specified. Appendix C shows the calculations done in this conversion process.

After calculating the relative track address, IGG019KF gives control to the BPAM convert–to–actual routine and processing proceeds as described for the nonoverflow case (module IGG019KE).

IGG019KG, KH (Feedback Modules for Relative Block Addressing)

After the channel program ends, the ASI routine of the foundation module gives control to one of the feedback modules if the processing program specified feedback and relative block addressing (see Figure 4). Module IGG019KH gets control if track overflow was specified; otherwise, module IGG019KG gets control.

The feedback modules initiate conversion of an actual address to a relative block address. They give the actual address of the block whose address is to be converted to the BPAM convert–to–relative routine. The BPAM routine converts this address to a relative track address and returns the relative track address to the feedback module. (For more information on the BPAM routine, refer to *OS SAM Logic.)* Using information contained in the actual and relative extents and, if track overflow was specified, the overflow section of the DEB, the feedback module then converts the relative track address to a relative block address and places the relative block address in the BLKREF field.

The method by which the feedback modules convert a relative track address to a relative block address is basically a reversal of the technique used to convert a relative block address to a relative track address (the latter conversion is shown in Appendixes B and C).

The feedback modules return control to the ASI routine of the foundation module.

Channel Program Generating Modules (IGG019KI, KK, KR, KW, KO, KM, KN, LA, KY, KQ)

A channel program is constructed for every BDAM READ or WRITE request. BDAM channel programs can read or write blocks using either the key or block identification field of a block as a search argument.

The channel program generating modules receive control from the base routine in the foundation module. The base routine determines which channel program generating module should get control by checking the IOBDTYPE field in the IOB, which contains data from the parameters specified in the DCB and request macro instruction. The base routine determines whether the data set is on a device with the rotational position sensing (RPS) feature by checking the UCBTYP field in the UCB. If the device does have the RPS feature, foundation module IGG019KA places a X'FF' in the first byte of the IOBDCPND field of the IOB or IGG019KJ sets a X'23' in the same field. The channel program generating module checks this byte to determine whether the channel program should include RPS channel command words.

When the channel program generating module gets control, it constructs a channel program in the IOB associated with the READ or WRITE request and then returns control to the base routine in the foundation module.

BDAM constructs three types of channel programs: update, format, and verification. Figure 5 shows what these channel programs do, which modules build them, and which extended search modules can modify them.

Update Channel Programs (IGG019KI, KK, KR, KW)

The update channel programs read or write data for purposes other than adding a new block to an existing data set. Figure 5 shows which modules build a particular update channel program. Six channel programs can be built for reading or writing an existing block. Each channel program can take one of two forms depending on whether it is built in response to a READ or WRITE request.

Type of channel program	**What channel program does**	**Record format**	**Search argument**	**Module that builds basic channel program**	**Extended search module[1]**	**Channel program number[2]**	
						No extended search	**Extended search**
UPDATE	Reads or writes a block for purposes other than adding a new block to an existing data set	Non–spanned	Key	IGG019KI	IGG019KW	2	5
		Non–spanned	Block ID	IGG019KK	Not applicable	1	

		VS	Key	IGG019KR	IGG019KW	12(read)
						13(write)
							8(read)
							9(write)

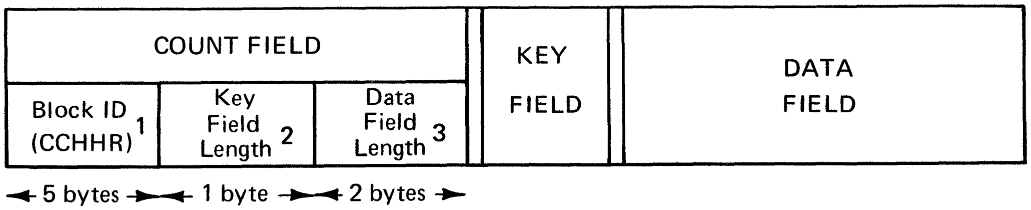
		VS	Block ID	IGG019KR	Not applicable	10(read)	11(write)
FORMAT	Adds a new block to an existing data set	F		IGG019KO	IGG019LA	3	6
		V,U		IGG019KM	IGG019KY	4	
		VS		IGG019KN	IGG019KY	7	
VERIFICATION[3]	Checks accuracy of information written by a channel program associated with a WRITE request			IGG019KQ			

[1] The extended search module modifies the basic channel program when the extended search option is specified in the processing program

[2] The channel program number refers to the number of the channel program in Appendix D.

[3] The verification module adds channel command words to the basic channel program whenever the write–validity–check option is in effect.

Figure 5. **BDAM channel program generating modules** build update, format, and verification channel programs.



[1] CCHHR gives the physical position of the block on the device.

[2] The key field length may be from 0 to 255 bytes.

[3] The data field length may be from 0 to 32,760 bytes.

Figure 6. A **block on a direct–access storage device** has count, key, and data fields.

Figure 6 shows the fields of a block as it appears on a direct–access storage device. The search argument, which in part determines which module will generate a channel program, is either the key field or the block identification in the count field.

If the extended search option has been specified, modules IGG019KI and IGG019KR pass control to the extended search module, IGG019KW. IGG019KW modifies the channel program so that it will search additional tracks or blocks beyond those specified in the READ or WRITE macro instruction. The extended search module returns control to module IGG019KI or IGG019KR.

If the write–validity–check option has been specified for a WRITE request, control is given to the write–verify module, IGG019KQ, after the channel program is built. This module adds to the channel program, channel command words that will verify the accuracy of the data written by the channel program. The write–verify module returns control to the foundation module.

Figure 15 summarizes the flow of control between BDAM modules for updating an existing block.

Format Channel Programs

The format channel programs, which are sometimes called write–add channel programs, write a new block from main storage onto a direct–access device. Format channel programs that write fixed–length records are called preformat channel programs; those that write variable–length, variable–length spanned, or undefined–length records are called self–format channel programs. Figure 5 shows which modules build a particular format channel program.

Preformat Channel Programs (IGG019KO, LA)

Preformat channel programs are built by module IGG019KO when a new fixed–length block is to be added to a data set. In order to add fixed–length blocks, the user must have initially formatted his data set blocks on the direct–access device using the BSAM write routine for creating a data set with direct organization. As blocks were placed on the direct–access device, dummy records will have been provided if new records were later to be added to the data set.

The channel program built by module IGG019KO, when executed, operates as follows:

- It searches the specified track for a dummy record, starting with the first block on the track.
- If a dummy record is not found, the no–space–found bit is set in the DECSDECB field of the DECB during the posting of the ECB.
- If a dummy record is found, the first byte of the data field, which contains the relative block number of the dummy record on the track (the R in MBBCCHHR), is read into the IOBSEEK field of the IOB (which now contains MBBCCHHR). The beginning of the dummy record can now be located by searching for the block identification in the count field (which contains CCHHR). Once the dummy record is found, the new record is written in its place.

If the extended search option has been specified, the channel program built by IGG019KO is modified by module IGG019LA, in which case the operation is as that outlined above except that the channel program searches for a dummy record on as many tracks as are specified in the IOBUPLIM field of the IOB. Module IGG019LA is given control by, and returns control to, module IGG019KO.

If the write–validity–check option was specified in the processing program, the channel program is enlarged by module IGG019KQ to include the channel command words that verify written data. IGG019KQ then gives control to the foundation module.

Self–Format Channel Programs

A self–format channel program is built when a new variable–length, undefined–length, or variable–length spanned record is to be added to an existing data set. Module IGG019KM builds the channel program when the record is of variable or undefined length; module IGG019KN builds it when the record is variable–length spanned. When the extended search option is specified, module IGG019KY modifies the channel program that has been built. Module IGG019KQ adds channel command words to the basic channel program when the write–validity–check option is specified.

As with fixed-length records, the BSAM write routine for creating a data set with direct organization must have been used initially to format the data set on the direct–access device. When the blocks are initially put on the direct–access device, a capacity record (block 0) is placed on each track. The capacity record contains both the block identification of the last block on the track and the number of usable bytes (bytes available for adding new blocks) remaining on the track. Figure 7 illustrates the data field of the capacity record.

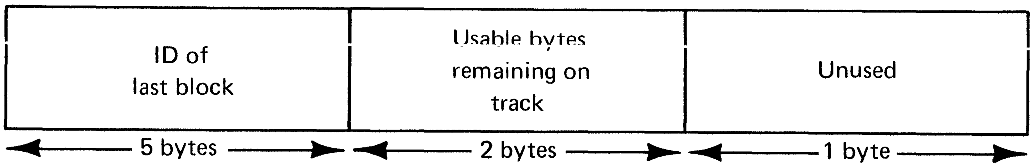


Figure 7. The **data field of a capacity record** contains information that allows BDAM to determine whether space in which to add a record is available on a particular track.

IGG019KM, IGG019KY (Nonspanned Records): Foundation module IGG019KA gives control to module IGG019KM to build the channel program for format–U and –V records.

To build the channel program, IGG019KM moves constants, representing elements of channel command words, into assigned positions of the request's IOB to form channel command words.

The channel program, when built, has two sections:

- The first reads the capacity record into main storage.
- The second writes the new block and then updates the capacity record to reflect inclusion of the new block on the track.

When the first section of the channel program is built, the last channel command word does not include a command chaining flag. This permits the I/O supervisor to give control to the channel end/abnormal end appendage module after the capacity record has been read. When the appendage module gets control, it branches to a supervisor routine to schedule the ASI routine.

In the following description, numbers in parentheses refer to the numbers in Figure 8. The ASI routine gives control to the exclusive control module (2). The exclusive control module gets control so that if more than one request is made to add a block to a particular track, only one of the requests can examine and update the capacity record at a time. If the capacity record has already been placed in the read–exclusive list as the result of a previous request, the exclusive control module places the IOB for the capacity record on the unposted queue (4). Otherwise, it places the capacity record for that track in the read–exclusive list and then in an intertask queue by using the ENQ macro instruction (3). (The operations performed by the exclusive control module are described more fully in the section "Exclusive Control Module (IGG019LG).") In either case, control is then given to the supervisor.

After the channel program reads the capacity record again in connection with a given WRITE–add request, the supervisor gives control to the ASI routine (5). The ASI routine then gives control to module IGG019KM so that the information in the capacity record can be tested (6) to determine if the block to be written will fit on the track.

If the new block will fit on the track, the capacity record is updated. The channel program is then modified to reflect the correct search argument, and an EXCP macro instruction is issued to write the new block and the updated capacity record. Module IGG019KM then gives control to the exclusive control module (7). If the new block will not fit on the track, control is given to the exclusive control module immediately.

The exclusive control module determines whether the unposted queue has an IOB waiting for the capacity record that was just updated. If it does not, the capacity record is no longer needed for the performance of the current task, and the exclusive control module issues a DEQ macro instruction to release the record to other tasks that may require it. Control is then given to module IGG019KM (8). If the unposted queue does contain an IOB waiting for this capacity record, the unposted queue is updated and both the address of the IOB and program control are given to self–format module IGG019KM. No system or list dequeuing takes place.

The self–format module determines whether the block was placed on the track (that is, whether the track had room for the block).

If it was and if the unposted queue contained another IOB for the same capacity record (9), the value in the IOBDBYTR field of the IOB, which contains the number of remaining bytes on the track, is moved from the IOB of the current request to the IOB of the next request for this capacity record. In this situation, the capacity record is still retained as a result of the ENQ macro instruction issued for the current task. Therefore, the self–format module can immediately begin processing this next request at the point of determining whether the block will fit on the track (6).

If the block was placed on the track and if the unposted queue did not contain any more IOBs for this capacity record, module IGG019KM gives control to the supervisor (10).

If the block was not placed on the track because of space limitations and if the extended search option has not been specified, an indication that no space is available is placed in the IOB (11). The ASI routine then gets control to post the request as complete and to place a no–space–found indication in the DECB (12). The self–format module then determines if the unposted queue contained another IOB for the same capacity record (13) and either returns control to the supervisor or moves the value in the IOBDBYTR field and continues as described in the preceding paragraph.

If the block was not placed on the track because of space limitations but the extended search option has been specified, control is given to the self–format extended search module IGG019KY. This module updates the current track address by 1 and proceeds as follows:

A. If the updated track address is equal to the search limit indicated in the IOBUPLIM field of the IOB, the no–space–found indication is set in the DECB for this request (14). Control then returns to the self–format module and processing continues as if the extended search option had not been specified.

B. If the updated track address is not in the current extent, control is given to the BDAM end–of–extent module, IGG019LC. This module determines if more extents are available for searching and proceeds as follows:

   1. If there are more extents available and if the upper limit of the search has not been reached, the address of the first track of the next extent is given to the self–format module. Since access to this new track may be required in the performance of other tasks, it is necessary to give control to the exclusive control module at this point (2). The read–exclusive list is checked for the occurrence of the new track address and processing continues as previously described (in the beginning of this section) for the first capacity record.

   2. If there are no more available extents or the search limit has been reached, the procedure is as described for condition (A).

C. If the updated track address is within the current extent and the search limit has not been reached, the processing of condition (B1) is continued at the point where control is given to the exclusive control module.

Where applicable, the procedures described in the preceding paragraphs are repeated as many times as necessary until either track space on which to write the block is found or the search limit is reached.

If the unposted queue contained another IOB for the same capacity record, processing of that IOB then continues from point D in Figure 8.

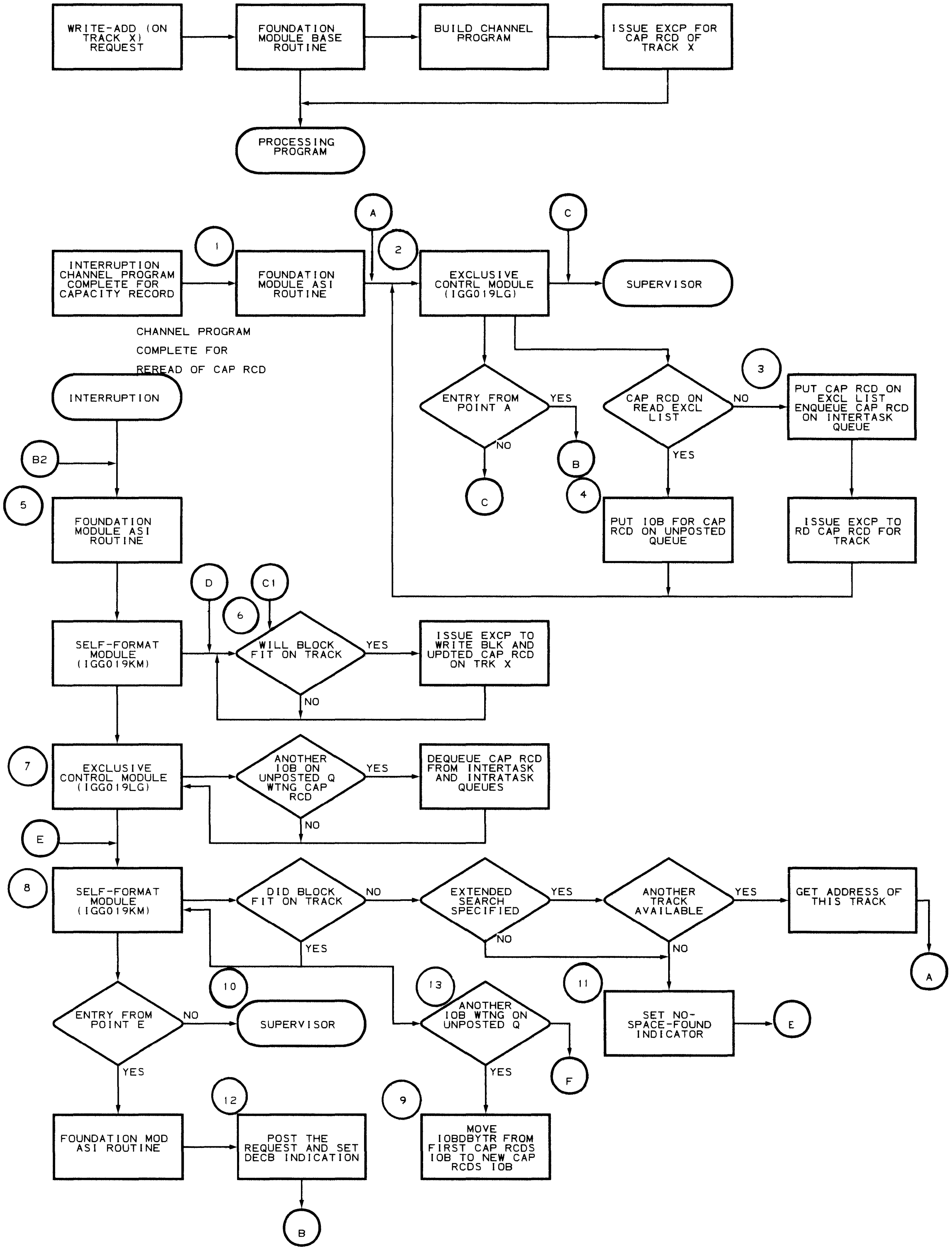

WRITE-ADD (ON TRACK X) REQUEST
FOUNDATION MODULE BASE ROUTINE
BUILD CHANNEL PROGRAM
ISSUE EXCP FOR CAP RCD OF TRACK X
PROCESSING PROGRAM
A
C
1
2
INTERRUPTION CHANNEL PROGRAM COMPLETE FOR CAPACITY RECORD
FOUNDATION MODULE ASI ROUTINE
EXCLUSIVE CONTRL MODULE (IGG019LG)
SUPERVISOR
CHANNEL PROGRAM COMPLETE FOR REREAD OF CAP RCD
3
INTERRUPTION
ENTRY FROM POINT A
YES
NO
CAP RCD ON READ EXCL LIST
NO
PUT CAP RCD ON EXCL LIST ENQUEUE CAP RCD ON INTERTASK QUEUE
B2
B
YES
5
C
4
FOUNDATION MODULE ASI ROUTINE
PUT IOB FOR CAP RCD ON UNPOSTED QUEUE
ISSUE EXCP TO RD CAP RCD FOR TRACK
D
C1
6
SELF-FORMAT MODULE (IGG019KM)
WILL BLOCK FIT ON TRACK
YES
ISSUE EXCP TO WRITE BLK AND UPDTED CAP RCD ON TRK X
NO
7
EXCLUSIVE CONTROL MODULE (IGG019LG)
ANOTHER IOB ON UNPOSTED Q WTNG CAP RCD
YES
DEQUEUE CAP RCD FROM INTERTASK AND INTRATASK QUEUES
NO
E
8
SELF-FORMAT MODULE (IGG019KM)
DID BLOCK FIT ON TRACK
NO
EXTENDED SEARCH SPECIFIED
YES
ANOTHER TRACK AVAILABLE
YES
GET ADDRESS OF THIS TRACK
NO
YES
NO
A
10
13
11
ENTRY FROM POINT E
NO
SUPERVISOR
ANOTHER IOB WTNG ON UNPOSTED Q
SET NO-SPACE-FOUND INDICATOR
E
F
YES
YES
12
9
FOUNDATION MOD ASI ROUTINE
POST THE REQUEST AND SET DECB INDICATION
MOVE IOBDBYTR FROM FIRST CAP RCDS IOB TO NEW CAP RCDS IOB
B

Figure 8. Module relationships for WRITE–add requests in a multitask environment

IGG019KN, IGG019KY (Spanned Records): Foundation module IGG019KJ gives control to module IGG019KN to build the channel program for variable–length spanned records.

To build the channel program, IGG019KN moves constants, representing elements of channel command words, into assigned positions of the request's IOB to form channel command words.

The channel program, when built, has two sections:

- The first reads the capacity record into main storage.
- The second writes the new record segment(s) and then updates the capacity record(s) to reflect inclusion of the new block on the track(s).

Add Logic Section. The ASI routine in foundation module IGG019KJ gives control to this section of IGG019KN after a capacity record has been read and placed on an intertask queue to prevent interference. The add logic section adds the track balance from the capacity record just read to any previous balance accumulated for this request to add a record. Thus, it maintains a cumulative record of the total space available for the request. After updating the total, it checks whether the space is large enough to contain the proposed addition.

If the total space is large enough to accommodate the record, the first (or only) segment of the record to be added is formed and moved to the segment work area. An EXCP is issued to write the segment. Any subsequent segments will be formed and written by the ASI routine in foundation module IGG019KJ.

If the total space is not large enough to accommodate the record, the self–format extended search module, IGG019KY, is called to obtain the address of the next track. IGG019KY may return, in addition to the address of the next track, an indication that the search limit (specified in the LIMCT parameter of the DCB macro instruction) has been reached. If it does, the no–space–found bit is set in the IOB and processing continues.

Whenever control is returned from the extended search module, the no–space–found bit in the IOB is checked. If it is on and if the track obtained by the extended search module contains a data record, the requestor's DECB is posted with a no–space–found indication and control is immediately returned to the processing program.

If the return from the extended search module is normal, IGG019KN checks the next track address to be sure it is on the same volume as the other tracks that will contain parts of the new record. (A record is not permitted to span volumes.) If the next track is on the same volume, the exclusive control module is called to enqueue on the new track and issue an EXCP to read its capacity record. IGG019KN then issues an SVC3 (exit). When the channel program finishes reading the capacity record, the ASI routine gets control, the new space is added to the available space, and processing continues as above.

If the new track is on a new volume, all previous tracks are released from exclusive control, the space available for a record to be added is set to 0, and the search for space is started again on the new volume. However, the tracks searched on the old volume are still counted in determining if the search limit has been reached. If the no–space–found bit in the IOB is on when a volume switch is detected, the request is posted with no space found indicated in the DECB for the request.

DEQ Section. The DEQ section of module IGG019KN is entered from the add logic section:

- When the add logic section determines that it must release a track, or
- When the add logic section is about to issue an exit but finds that a DEQ loop indication (explained below) has been set in the current IOB during a previous pass through the DEQ routine for a different IOB.

The DEQ section calls the exclusive control module (IGG019LG) to dequeue one track at a time. If the return from IGG019LG indicates that there is an IOB waiting for this track, the waiting IOB is made the current IOB and the DEQ section returns to the beginning of the add logic section after setting the DEQ loop indication.

The DEQ loop indication operates as follows: If a particular WRITE–add request is finished with a track, control goes to the DEQ section to dequeue the track(s) just used. If, however, another IOB is waiting for the track, the track is not taken off the exclusive control list, but the DEQ loop bit in the IOB is set and control is given to the request represented by the waiting IOB. When this second IOB can be processed no further (because, for example, it is waiting for I/O or for another track), the WRITE-add routine would normally issue an SVC3 (exit). However, it inspects the DEQ loop bit and, if it is on, gives control to the DEQ section at the point where the original IOB lost control.

IGG019KQ (Write–Verify Module)

If the processing program specifies the write–validity–check option in the DCB macro for the data set, the write–verify module, IGG019KQ, is used to generate additional channel command words to verify information that has been written by a channel program. These channel command words are added to the existing channel program.

The write–verify module gets control from the module that generates the channel program for the request macro instruction. As data blocks are written, the control unit develops a check code for each field of the block. This code is based on the information that is written in the fields of the block. As each field is written, the check code developed for it is appended to it. Verification is accomplished by reading back the block to be checked to permit the control unit to recompute the check codes. The control unit then compares the check code written on the track with the check code it just recomputed. If the two codes are not equal, a data check indication is set by the channel. The skip flag in the last channel command word of the verification program is set to 1 (on) so that the data that is read back is not placed into main storage.

The write–verify module returns control to the base routine of the foundation module.

BDAM Appendage Modules (IGG019KL, LE, KU, LC)

BDAM has the following appendage modules:

- Dynamic buffering module (IGG019KL,LE)
- Channel end/abnormal end appendage module (IGG019KU)
- End–of–extent appendage module (IGG019LC)

The routines in these appendage modules can receive control from the I/O supervisor, from other BDAM modules, or from a routine associated with a macro instruction.

IGG019KL, LE (Dynamic Buffering Modules)

The I/O supervisor gives control to one of the dynamic buffering modules when a request for which the dynamic buffering option is specified is ready to be executed.

IGG019KL handles buffering for spanned records, IGG019LE for nonspanned records. The modules operate the same way except for differences that result from different buffer formats. The buffer for spanned records has a segment work area that the buffer for nonspanned records doesn't have. This work area is used by segment–processing routines in the module to assemble the parts of a record for input operations or to segment a record for output operations. The address of the segment area in the segment work area is the address specified in the channel program and accessed by the channel. The address of the segment work area is in the word preceding the IOB. See the "Data Areas" section of this manual for a detailed description of buffers and buffer control blocks.

The dynamic buffer modules have start I/O and free dynamic buffer entry points. The following discussion is organized by entry point to the dynamic buffering modules.

Start–I/O Appendage Entry Point

The I/O supervisor gives control to the start–I/O routine in the dynamic buffering module when a request to read data is ready to be executed (see Figure 9).

The start–I/O routine checks the IOBTYPE field of the IOB to determine whether a buffer should be assigned to the request. If not, the start–I/O routine returns control to IOS so the request can be executed. If a buffer should be assigned, the start I/O routine determines whether a buffer has already been assigned.

If a buffer has already been assigned, this routine gives control to the I/O supervisor, which initiates execution of the channel program.

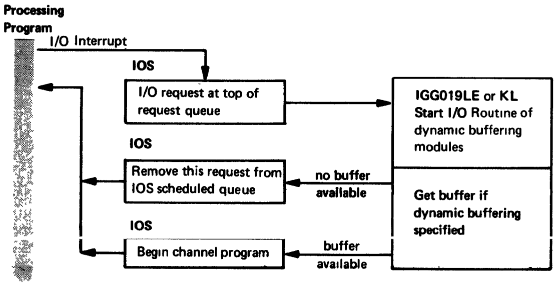

Processing Program
I/O Interrupt
IOS
I/O request at top of request queue
IGG019LE or KL Start I/O Routine of dynamic buffering modules
IOS
Remove this request from IOS scheduled queue
no buffer available
Get buffer if dynamic buffering specified
IOS
Begin channel program
buffer available

Figure 9. If dynamic buffering is in effect, the I/O supervisor gives control to the **dynamic buffering module** before executing the request.

If a buffer has not been assigned, this routine determines whether the buffer pool contains an available buffer. If one is available, it is assigned to the request and the buffer pool is updated to reflect removal of the buffer. Then the address of the buffer is placed in the channel program associated with the request. The channel program is now complete and ready for execution. Control is given to the I/O supervisor, which initiates execution of the channel program.

If a buffer is not available in the buffer pool, the address of the IOB associated with the request is placed in the IOB buffer queue, which is a queue of requests waiting for buffer assignment (the address of this queue is in the BCBFRQT field of the buffer control block). As buffers subsequently become available, they are allocated to the requests in the IOB buffer queue. As each request is added to the queue, it becomes the last request. When a buffer becomes available, it is allocated to the request currently at the top of the queue and that request is then removed from the queue. After putting a request on the IOB buffer queue, the start–I/O routine returns control to the I/O supervisor.

Free Dynamic Buffer Entry Point

When the ASI routine in the foundation module issues the FREEDBUF SVC, control passes to module IGC0005G. IGC0005G gives control to the dynamic buffering module at the free dynamic buffer entry point to release a buffer.

When a request that specifies dynamic buffering has been satisfied (either successfully or unsuccessfully), the buffer assigned to the request may be made available to other requests. This can happen in one of two ways:

- A buffer assigned to a READ request for a block not being updated can be released only by the FREEDBUF macro instruction issued by the processing program.
- A buffer assigned to a READ request for a block that is being updated is released when a corresponding WRITE request that specifies dynamic buffering is satisfied. Control is given to the ASI routine in the foundation module when the WRITE request is satisfied, and the ASI routine issues the FREEDBUF macro instruction to release the buffer.

The expansion of the FREEDBUF macro instruction includes a supervisor call instruction (SVC 57). The supervisor call routine gives control to the dynamic buffering module at the free dynamic buffer entry point.

Because the buffer control block and the IOB buffer queue are updated during the processing required for both freeing and obtaining a buffer, the free dynamic buffer routine must operate with I/O interruptions prevented.

The free dynamic buffer routine releases the buffer used by the request that has just completed. If it finds an IOB waiting in the IOB buffer queue for a buffer, this routine assigns to the IOB the buffer it just freed. It updates the buffer queue by moving each request up one position in the queue.

After updating the buffer queue, the free dynamic buffer routine puts the address of the buffer assigned to the request in the channel program associated with the request. The channel program is now complete, and the free dynamic buffer routine issues an EXCP request to execute the channel program.

If there are no IOBs waiting in the buffer queue for a buffer, the buffer just released is placed in the list of available buffers. The buffer control block is then updated to include the added buffer.

The free dynamic buffer routine then returns the system to enabled state and gives control to the supervisor, which in turn gives control to the processing program.

IGG019KU (Channel End/Abnormal End Appendage Module)

The I/O supervisor gives control to module IGG019KU when a channel program terminates (either normally or abnormally).

There are two routines in this module: one for channel end and one for abnormal end. The discussion that follows describes the conditions that cause each of the routines to be entered and the operations performed by each of the routines in response to those conditions.

Channel End Routine

This routine is entered if:

- A channel program terminates normally. In this case, the channel end routine schedules the ASI routine in the foundation module and then gives control to the I/O supervisor. To schedule the ASI routine, the channel end routine branches to the exit effector routine of the task supervisor. (For information on the exit effector routine, refer to *OS MVT Supervisor Logic.)* The exit effector routine then schedules the ASI routine and returns control to the channel end/abnormal end appendage module. When the ASI routine is executed, it will run under the TCB of the task that opened the data set.

- A channel program encounters a unit exception condition, which is interpreted by BDAM as an end–of–data–set condition. In this case, the channel end routine sets indicators in the IOB, requests scheduling of the ASI routine in the foundation module, and then gives control to the I/O supervisor. To schedule the ASI routine, the channel end routine branches to the exit effector routine of the task supervisor. (For information on the exit effector routine, refer to *OS MVT Supervisor Logic.)* The exit effector routine then schedules the ASI routine and returns control to the channel end/abnormal end appendage module. When the ASI routine is executed, it will run under the TCB of the task that opened the data set.

- The execution of a channel program reveals that the block length specified in the READ or WRITE macro instruction is not equal to the number of bytes read from or written to the device. In this case, the channel end routine determines the type of request and responds as follows:

  If the request was a READ request for a variable–length block, the length of the block being read is compared to the number of bytes of data actually read by the channel program. (The length of the block being read is specified in the first 2 bytes of the data field of the block read into the designated area. The number of bytes actually read is determined from a calculation involving the bytes–remaining field of the channel status word.) If the two lengths are equal, the incorrect–length indication in the IOBCSW field is set to 0 (off), and the ASI routine is scheduled by the exit effector routine. Control is then returned to the I/O supervisor. If the two lengths are not equal, the ASI routine is not scheduled, the incorrect–length indication in the IOB is left at 1 (on), and the channel end routine gives control to the I/O supervisor.

  If the request was a READ request for format–U records, the incorrect–length indication in the IOBCSW field is set to 0 (off), and the ASI routine is scheduled by the exit effector routine. Control is then returned to the I/O supervisor.

If the request was a WRITE request or a READ request for format–F records, the ASI routine is not scheduled, the incorrect–length indication is left at 1 (on), and the channel end routine gives control to the I/O supervisor.

Abnormal End Routine

This routine is entered when either a device error or a permanent error (one from which the system cannot recover) has occurred. If a device error occurs, the I/O supervisor receives control and uses a standard IBM error–recovery procedure. If the error condition remains after this procedure, the error is classed as a permanent error.

For permanent errors, the abnormal end routine sets an indicator in the IOB, schedules the ASI routine in the foundation module, and returns program control to the I/O supervisor.

IGG019LC (End–of–Extent Appendage Module)

The BDAM end–of–extent appendage module is entered if the extended search option is specified in the DCB macro instruction. This module can receive control from the I/O supervisor or the self–format extended search module.

Operations Performed When Control Received From I/O Supervisor

The I/O supervisor gives control to the end–of–extent appendage module when a channel program reaches the end of a data set extent while searching for a block to be read or written, or a dummy record in which to write a new preformat block.

The end–of–extent appendage module establishes the address of the next extent to be searched. Note that if the search has begun at some point other than the beginning of the first extent (as reflected in the first actual extent in the DEB), the address of the extent may, at some point in the search, become that of the first extent.

The next operation performed by this module depends on whether the search limit is reached in the next extent to be searched.

If the search limit is not in the next extent, the end–of–extent module either:

- Returns control to the I/O supervisor to restart the channel program using the new extent, or
- If the new extent refers to another device, gives control to the BDAM ASI routine in the foundation module to reschedule the channel program using a search address in the new extent.

If the search limit is in this new extent but the new search address is not equal to the search limit, the channel program will be rescheduled by either the I/O supervisor or the ASI routine as before.

If the search limit is in this extent and the new search address equals the search limit, the search has ended unsuccessfully. An indicator is then set in the IOBDSTAT field of the IOB to show that either no space was found or no block was found, and control is given to the I/O supervisor, which, in turn, gives control to the abnormal end routine in the BDAM channel end/abnormal end appendage module.

Operations Performed When Control Received From Self–Format Extended Search Module

The self–format extended search module (IGG019KY) gives control to the end–of–extent appendage module if, in the process of establishing search addresses, it reaches the end of an extent.

The end–of–extent appendage module determines the availability of other extents to be searched, establishes a new search address, and determines whether the search limit has been reached. If the search limit has not been reached, the end–of–extent appendage module uses the search address related to the new extent and reschedules the channel program (to read in the capacity record of the next track). Then it gives control back to the self–format module.

If the search limit has been reached, an indication that no space has been found is placed in the request's IOB. When the request is posted, this indication is placed in the DECSDECB field of the DECB.

Exclusive Control Module (IGG019LG)

The ASI routine in the foundation module gives control to the exclusive control module if the processing program specifies the exclusive control option.

IGG019LG handles the block queuing and dequeuing that is required with the exclusive control option. In addition, IGG019LG is used to place records in a queue when the processing program issues a request to add to a data set a new block of either variable–length or undefined–length records.

With exclusive control in effect for a block, the block may not be updated (or otherwise acted upon) by processing associated with other requests until exclusive control for that block has been released. If the MACRF operand of a DCB macro instruction for BDAM contains the exclusive control specification, the following BDAM macro instructions require the exclusive control module:

- READ (with an exclusive control specification)
- WRITE (with an exclusive control specification)
- RELEX

Until the exclusive control module is first given control, the read–exclusive list (see the "Data Areas" section of this manual) consists of an 80–byte segment of main storage obtained by open executor module IGG0193A. This segment contains space for nine entries, each entry consisting of the UCB pointer and the actual address of a block for which exclusive control is required. When more than nine entries are needed on the read–exclusive list, additional main storage is obtained in increments of 80 bytes. Each 80 bytes can contain nine entries. The address of the first segment is contained in the DCBXARG field of the DCB, and each succeeding segment is chained to the one preceding it. The read–exclusive list is an intratask list of actual addresses of blocks (capacity records and data blocks) requested for the performance of the current task.

There are two situations in which a block is to be read under exclusive control:

- A self–format WRITE–add request is encountered. (See the section "Self–Format Channel Programs.")
- A READ macro instruction that requests exclusive control is encountered.

When either of these situations occurs, module IGG019LG determines if the actual address of the appropriate block is on the read–exclusive list. The appropriate block is the track capacity record in the case of a WRITE–add request; in the case of a READ–exclusive request, it is the block to be read.

If the block address is in the list, the IOB for the request is placed in a queue called the unposted queue. This is an intratask queue of IOBs representing requests for blocks whose addresses are currently in the read–exclusive list and are associated with the current task. The DCB contains the addresses of the first and last IOBs in this queue, and each intermediate entry is chained to the one preceding it. Control is then given to the routine from which module IGG019LG was entered.

If the block address is not on the read–exclusive list, this is the only request for the record for any task sharing this DCB. The IOB contains the block address. The UCB pointer and the CCHHR bytes of the actual address of the block are put in the read–exclusive list. Since the same block may be required in the performance of another concurrent task using a different DCB, it is necessary to provide protection against unwanted changes to the block. Therefore, the exclusive control module causes the block to be placed on an intertask queue by issuing an ENQ macro instruction for the block. Before an entry can be removed from this queue, a DEQ macro instruction must be issued for the entry. Note that the block must be dequeued by a routine associated with, or processing under, the TCB of the task that opened the data set. Since a block on this intertask queue cannot be used in the performance of one task until it is disassociated from another task (if the tasks are not sharing the same DCB), a waiting period of indeterminate length may result. Module IGG019LG then issues an EXCP macro instruction for the rereading of the block that has just been enqueued. Control is then given to the ASI routine which, in turn, gives control to the supervisor.

The *OS Supervisor Services and Macro Instructions* publication further explains the use of the ENQ and DEQ macro instructions.

In searching the read–exclusive list for either an address equal to the address of the block to be read or, having found that the block address is not on the list, a place on the list in which to place the block address, it may be necessary to scan through several segments of the read–exclusive list. A new segment is obtained if the second part of the search does not locate a place in which to put the block address.

Releasing Blocks Under Exclusive Control

For the task that opened the DCB, blocks that have been read under exclusive control may be released from exclusive control either by use of a WRITE macro instruction that specifies the exclusive control feature or by use of a RELEX macro instruction. The RELEX macro instruction is used for blocks that have not been updated or modified (that is, whose data fields remain unchanged). Only the task that opened the DCB can use RELEX.

Release by Writing

When a request (called a WRITE–exclusive request) to write a block that has been previously read under exclusive control is executed, the read–exclusive list is scanned to locate the block's address. When the address is found, the unposted queue is searched for other requests (associated with the current task) that may have been issued for the same block.

If a WRITE–exclusive request is given to release a block from exclusive control and the block had not been read under exclusive control, the WRITE request is invalid. Module IGG019LG sets an exception code in the IOBDSTAT field of the IOB so the user may identify the error. Control is then given to the ASI routine to free the IOB and post the request as complete.

If the unposted queue does not contain IOBs for other intratask requests for the block, module IGG019LG clears the block's address from the read–exclusive list. This permits another entry to be made in the list at that space. To free the block for another task, the exclusive control module then issues a DEQ macro instruction for the block. Module IGG019LG then returns control to the supervisor.

If the search of the unposted queue indicates the presence of other requests for the block being written, it is necessary that these other requests be provided with the current version of the data portion of the block. Therefore, before the current WRITE–exclusive request is posted as complete, the exclusive control module moves the data portion of the current block into the input data area of each request for that block in the queue. Control then passes to the ASI routine so that the first of these requests may be posted as complete and its IOB made available. The ASI routine then gives control back to the exclusive control module and processing continues as if the unposted queue did not contain any READ–exclusive requests for the block.

Release by RELEX

When a RELEX macro instruction is given to release a block that was read under exclusive control, it is assumed that the data portion of the block has not been changed. Therefore, data is not moved into input areas of other requests for that block. The procedures performed by the exclusive control module are otherwise similar to those performed in the case of a WRITE request for blocks read exclusively.

The RELEX module, IGC0005C, receives control when a RELEX macro instruction is issued by the processing program. After initialization, determination of the type of addressing that has been specified, and conversion (if necessary) of a block address to an actual address, module IGC0005C gives control (by issuing the SYNCH macro) to the exclusive control module. The exclusive control module searches the read–exclusive list for the block specified for release from exclusive control. If it can't find the block, it indicates an error condition; the programmer has requested the release of a block that was not under exclusive control. The exclusive control module sets an error code in register 15 and gives control to the RELEX module. The RELEX module gives control back to the processing program.

Because RELEX is an SVC (and therefore runs under the TCB of the task that makes the request), it can be issued only by the task that opened the data set. (The exclusive control option must be specified in a WRITE macro instruction to dequeue all other tasks.) The exclusive control module then gains control from the ASI routine running under the TCB of the task that opened the data set.

Figure 10 summarizes the main operations performed by the exclusive control module as they have been described in the preceding paragraphs.

Macro Instruction That Requests Action	Block Address Already on Read–Exclusive List	Other IOBs for Same Block on Unposted Queue	Action Taken
READ (Exclusive)	Yes	—	Place request's IOB on unposted queue. Go to supervisor.
READ (Exclusive)	No	—	Add block's address to read–exclusive list. Enqueue block on intertask queue. Schedule the block for reading. Go to supervisor.
WRITE (Exclusive)	No	—	Take error exit to ASI routine.
WRITE (Exclusive)	Yes	Yes	Remove READ request from queue. Move data into all READ request areas. Go to ASI routine to post first READ request on queue and free its IOB. Go to ASI routine to post WRITE request and free its IOB.
WRITE (Exclusive)	Yes	No	Go to ASI routine to post WRITE request and free the IOB. Remove entry from read–exclusive list and from intertask queue. Go to supervisor.
RELEX	No	—	Return to RELEX routine with error code.
RELEX	Yes	Yes	Remove READ request from queue. Go to ASI routine to post READ request and free the IOB. Return to RELEX routine.
RELEX	Yes	No	Remove block from read–exclusive list and from intertask queue. Return to RELEX routine.

Figure 10. The **operations performed by the exclusive control module** are determined by the conditions in the first three columns.

Check Module (IGG019LI)

The CHECK or WAIT macro instruction is coded in the processing program to ensure that a given READ or WRITE request is completed before a certain point in the processing program. The BDAM check module, IGG019LI, is used when the CHECK macro instruction has been specified and the DCB macro instruction for the data set includes the check specification. The address of a user's synchronous error recovery

(SYNAD) routine should be given in the DCB macro instruction that contains the check specification.

When the check module receives control, it establishes a wait condition if the request with which it is associated has not been posted as complete. If the request is complete at this point or is completed while the processing program is in the wait state, and if no errors have been indicated in the DECB, the IOB for the request is released to the IOB pool, and control is given to the processing program. (When the DCB macro instruction includes the check specification, the CHECK macro instruction must be used to effect the wait condition; if the WAIT macro instruction is used, the IOB for the request is not released.)

After a request is posted as complete and if error indications have been placed in the DECB, the check routine identifies both the type of request and the types of errors listed. Error types are placed in a register and control is given to the user's SYNAD routine if there is one. The manual *OS Data Management Macro Instructions* indicates the contents of registers when the BDAM check module gives control to a SYNAD routine. The absence of a SYNAD routine causes BDAM to terminate the processing program by issuing an SVC 55 (control passes to end of volume) to request abnormal termination.

Close Executor Module (IGG0203A)

The BDAM close executor module, IGG0203A, is given control during the closing of a DCB that specifies BDAM (see Figure 11). When the CLOSE macro instruction is issued by a processing program, the expansion of the macro instruction causes program control to be given to the data management close routine. This routine passes control to the BDAM close executor module.

The main purpose of the BDAM close executor module is to release to the system all BDAM–acquired storage areas that have been associated with the DCB to which the CLOSE macro instruction refers. This is done in from two to four steps, depending on the type of requests used in the application.

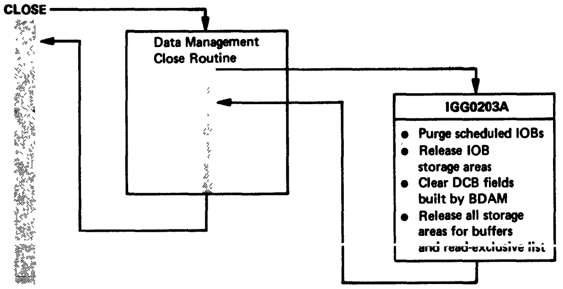

CLOSE
Data Management Close Routine
IGG0203A
● Purge scheduled IOBs
● Release IOB storage areas
● Clear DCB fields built by BDAM
● Release all storage areas for buffers and read-exclusive list

Figure 11. When the processing program issues the CLOSE macro instruction, the data management Close routine gets control and, in conjunction with the **BDAM close executor module,** closes the data set.

The first step consists of removing from the I/O supervisor's queue of scheduled requests any requests that have been scheduled but whose channel programs have not yet completed. The purge routine of the I/O supervisor accomplishes this removal. The BDAM close executor program then releases the main–storage area assigned to the IOBs for these requests. These requests are chained together, beginning with an address placed in the DEB by the purge routine.

The second step is the releasing of main–storage areas assigned to available IOBs or the list of IOBs. The IOB list includes the IOBs that are in either the I/O supervisor scheduled queue or a buffer queue. Therefore, only storage for IOBs not currently being used is released at this time.

The third step is the releasing of storage that has been allotted to any IOBs remaining in the unposted queue. These IOBs were placed on the queue by the exclusive control module. As a fourth step, the storage allotted to IOBs in the IOB buffer queue is released. These are the IOBs for requests waiting for buffer assignment. The main–storage areas that have been obtained for the BCB, for buffers to be assigned dynamically, and for segment work areas are also released.

In addition to releasing the storage areas assigned to IOBs, the BDAM close executor module clears from the DCB all fields that BDAM has built for, and that specifically refer to, the current use of the DCB to which the CLOSE macro instruction refers.

BDAM FLOWCHARTS

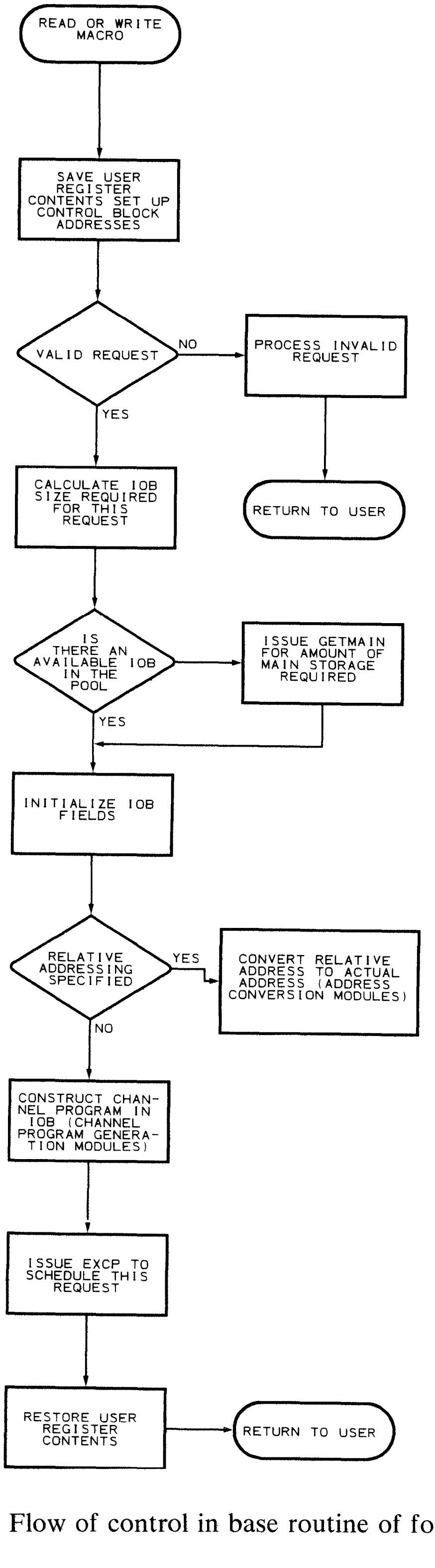

READ OR WRITE MACRO
SAVE USER REGISTER CONTENTS SET UP CONTROL BLOCK ADDRESSES
VALID REQUEST
NO
PROCESS INVALID REQUEST
RETURN TO USER
YES
CALCULATE IOB SIZE REQUIRED FOR THIS REQUEST
IS THERE AN AVAILABLE IOB IN THE POOL
ISSUE GETMAIN FOR AMOUNT OF MAIN STORAGE REQUIRED
YES
INITIALIZE IOB FIELDS
RELATIVE ADDRESSING SPECIFIED
YES
CONVERT RELATIVE ADDRESS TO ACTUAL ADDRESS (ADDRESS CONVERSION MODULES)
NO
CONSTRUCT CHANNEL PROGRAM IN IOB (CHANNEL PROGRAM GENERATION MODULES)
ISSUE EXCP TO SCHEDULE THIS REQUEST
RESTORE USER REGISTER CONTENTS
RETURN TO USER

Figure 12. Flow of control in base routine of foundation module

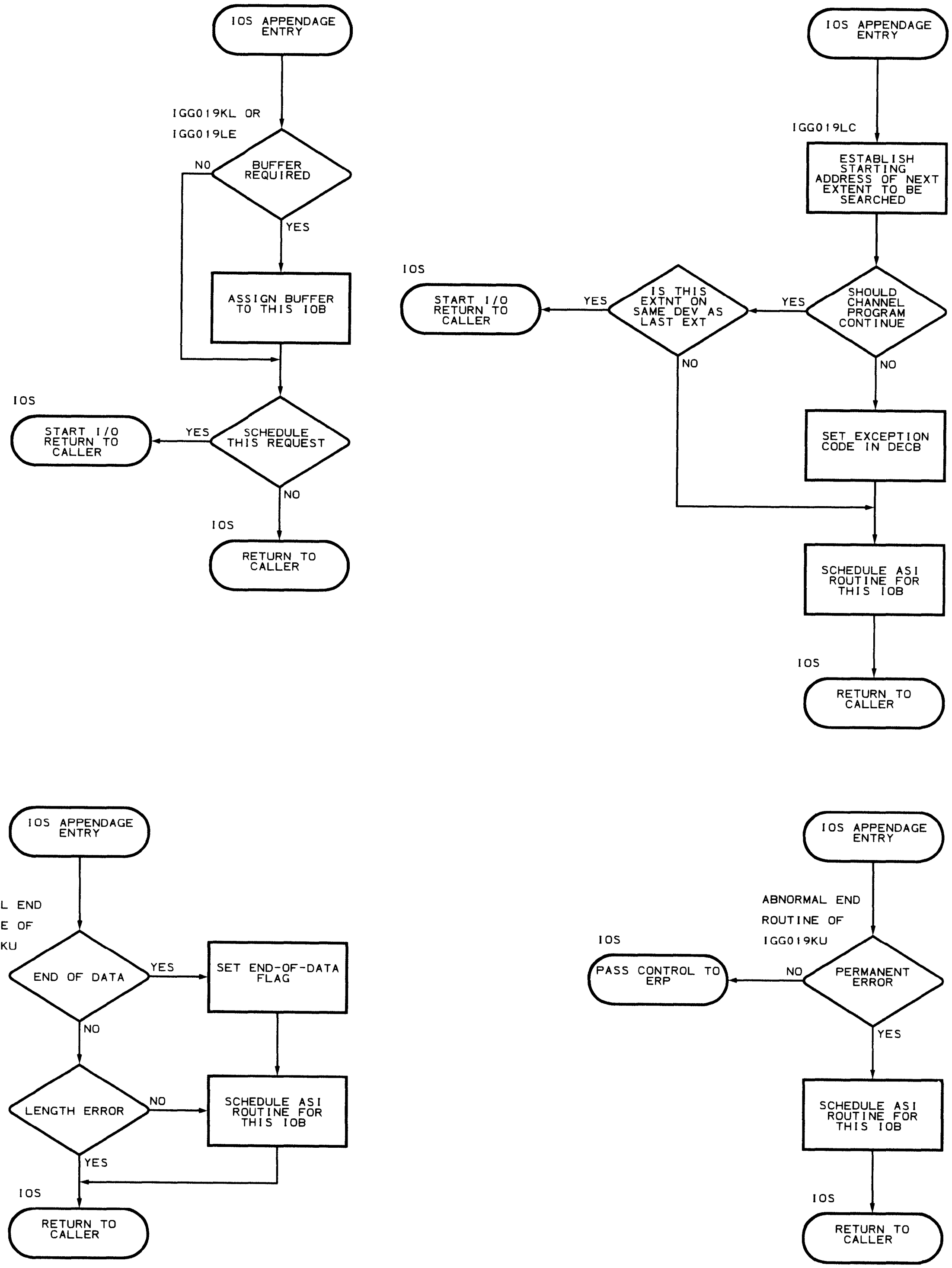

IOS APPENDAGE ENTRY
IGG019KL OR IGG019LE
BUFFER REQUIRED
NO
YES
ASSIGN BUFFER TO THIS IOB
IOS
START I/O RETURN TO CALLER
YES
SCHEDULE THIS REQUEST
NO
IOS
RETURN TO CALLER
IOS APPENDAGE ENTRY
IGG019LC
ESTABLISH STARTING ADDRESS OF NEXT EXTENT TO BE SEARCHED
IOS
START I/O RETURN TO CALLER
YES
IS THIS EXTNT ON SAME DEV AS LAST EXT
YES
SHOULD CHANNEL PROGRAM CONTINUE
NO
NO
SET EXCEPTION CODE IN DECB
SCHEDULE ASI ROUTINE FOR THIS IOB
IOS
RETURN TO CALLER
IOS APPENDAGE ENTRY
CHANNEL END ROUTINE OF IGG019KU
END OF DATA
YES
SET END-OF-DATA FLAG
NO
LENGTH ERROR
NO
SCHEDULE ASI ROUTINE FOR THIS IOB
YES
IOS
RETURN TO CALLER
IOS APPENDAGE ENTRY
ABNORMAL END ROUTINE OF IGG019KU
IOS
PASS CONTROL TO ERP
NO
PERMANENT ERROR
YES
SCHEDULE ASI ROUTINE FOR THIS IOB
IOS
RETURN TO CALLER

Figure 13. Flow of control in IOS appendages

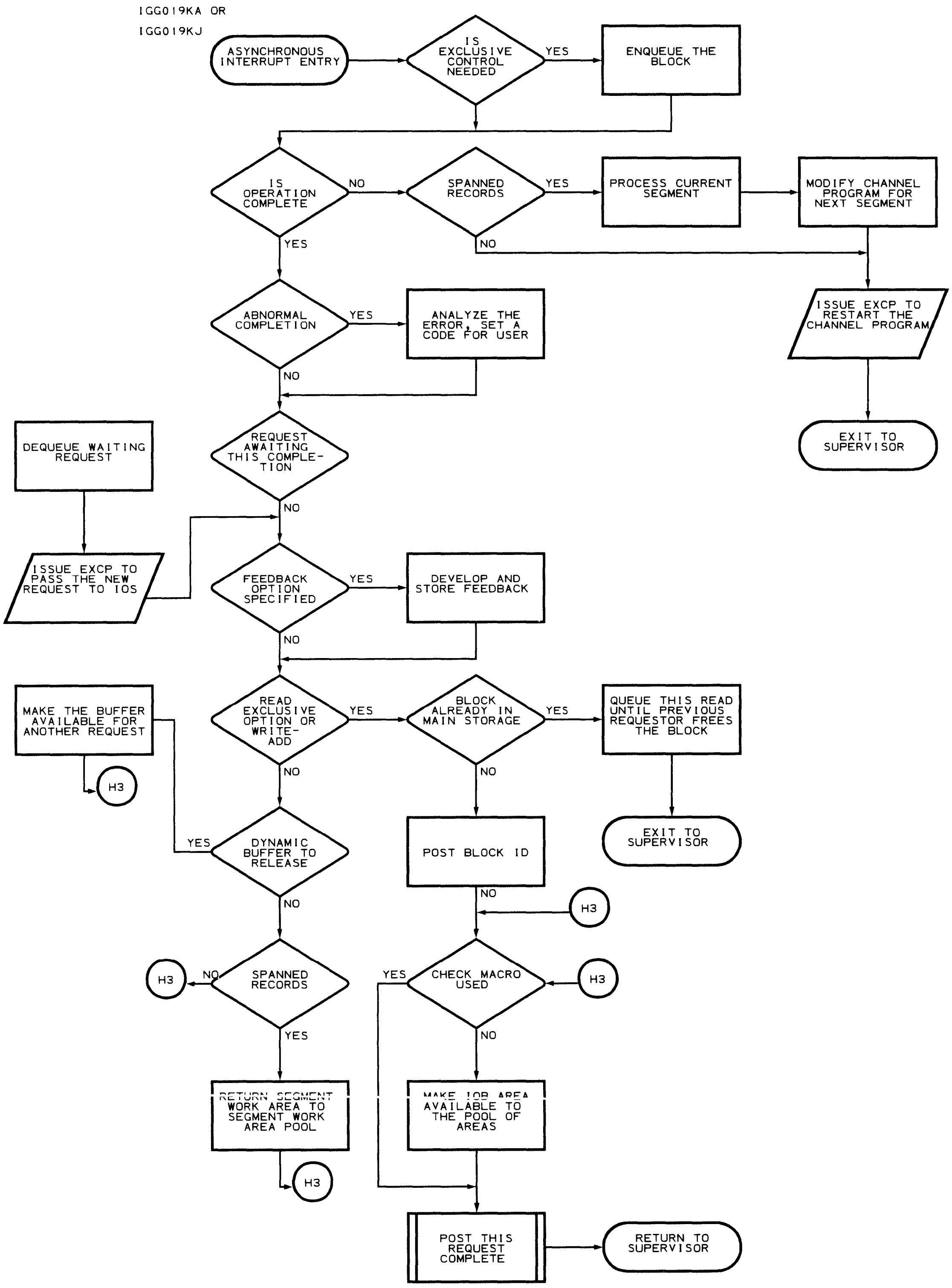

IGG019KA OR
IGG019KJ
ASYNCHRONOUS INTERRUPT ENTRY
IS EXCLUSIVE CONTROL NEEDED
YES
ENQUEUE THE BLOCK
IS OPERATION COMPLETE
NO
SPANNED RECORDS
YES
PROCESS CURRENT SEGMENT
MODIFY CHANNEL PROGRAM FOR NEXT SEGMENT
YES
NO
ABNORMAL COMPLETION
YES
ANALYZE THE ERROR, SET A CODE FOR USER
ISSUE EXCP TO RESTART THE CHANNEL PROGRAM
NO
EXIT TO SUPERVISOR
DEQUEUE WAITING REQUEST
REQUEST AWAITING THIS COMPLETION
NO
ISSUE EXCP TO PASS THE NEW REQUEST TO IOS
FEEDBACK OPTION SPECIFIED
YES
DEVELOP AND STORE FEEDBACK
NO
MAKE THE BUFFER AVAILABLE FOR ANOTHER REQUEST
READ EXCLUSIVE OPTION OR WRITE-ADD
YES
BLOCK ALREADY IN MAIN STORAGE
YES
QUEUE THIS READ UNTIL PREVIOUS REQUESTOR FREES THE BLOCK
H3
NO
NO
YES
DYNAMIC BUFFER TO RELEASE
POST BLOCK ID
EXIT TO SUPERVISOR
NO
NO
H3
H3
NO
SPANNED RECORDS
YES
CHECK MACRO USED
H3
YES
NO
RETURN SEGMENT WORK AREA TO SEGMENT WORK AREA POOL
MAKE IOB AREA AVAILABLE TO THE POOL OF AREAS
H3
POST THIS REQUEST COMPLETE
RETURN TO SUPERVISOR

Figure 14. Flow of control in ASI routine of foundation module

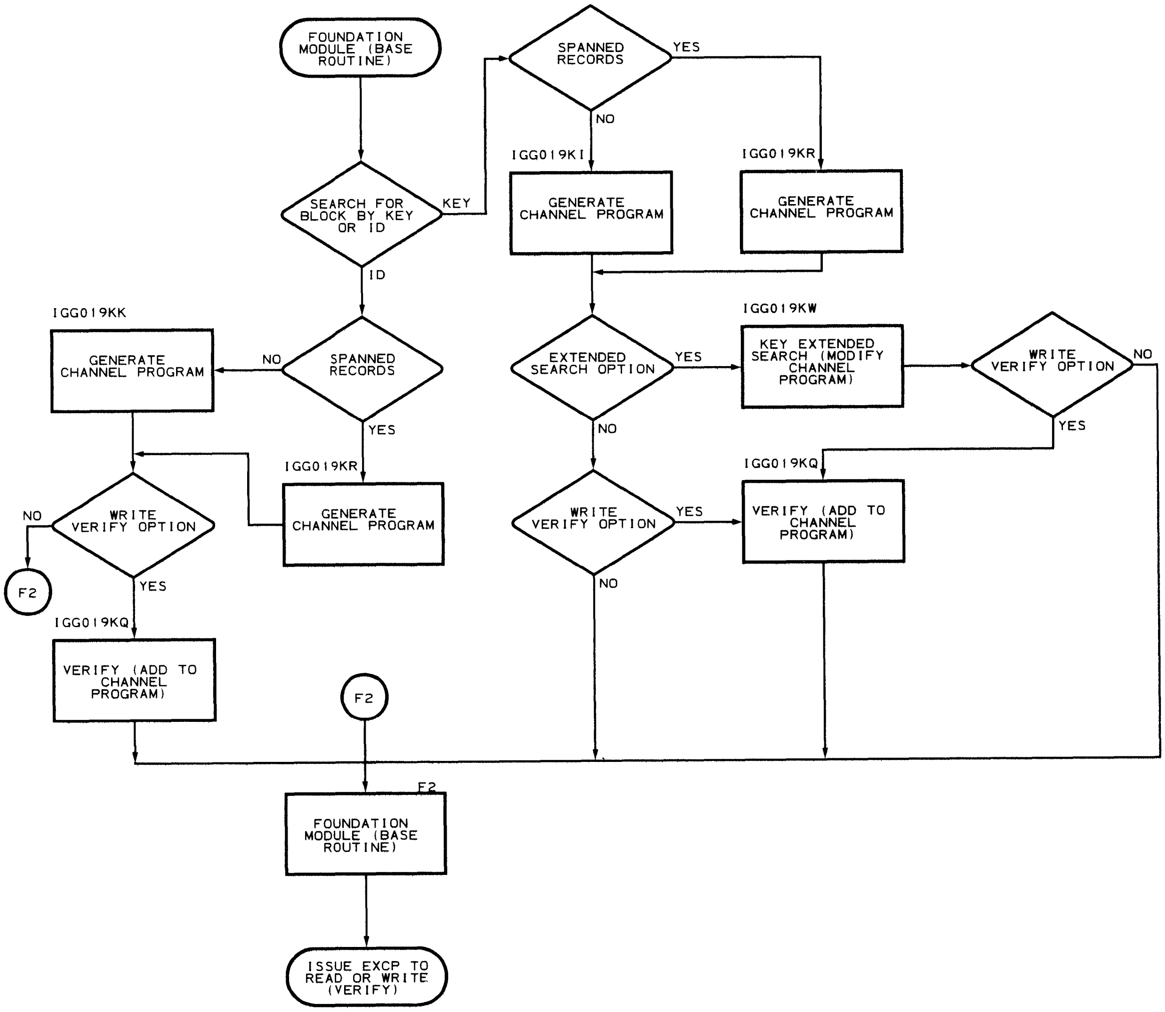


Figure 15. Flow of control for block updating

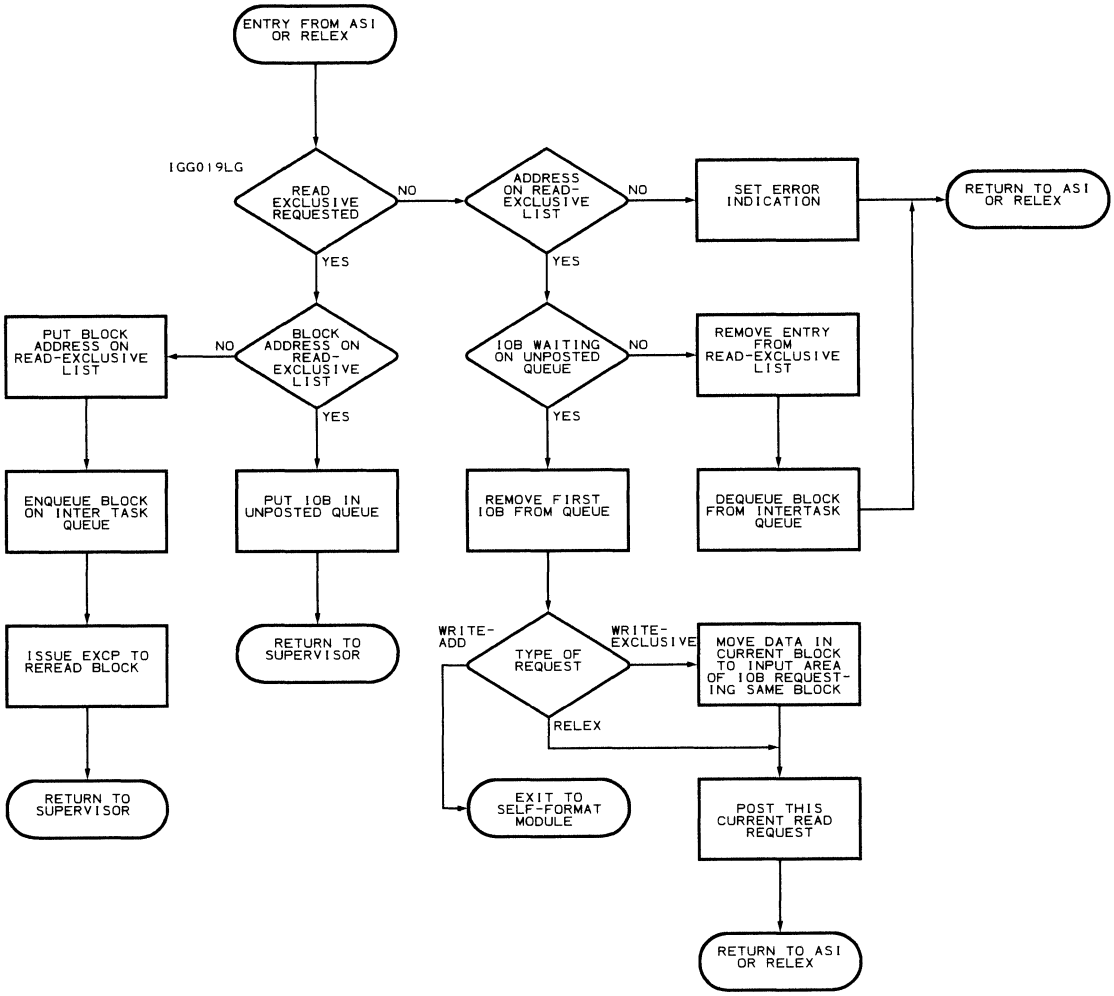

ENTRY FROM ASI OR RELEX
IGG019LG
READ EXCLUSIVE REQUESTED
NO
YES
ADDRESS ON READ-EXCLUSIVE LIST
NO
YES
SET ERROR INDICATION
RETURN TO ASI OR RELEX
BLOCK ADDRESS ON READ-EXCLUSIVE LIST
NO
YES
PUT BLOCK ADDRESS ON READ-EXCLUSIVE LIST
IOB WAITING ON UNPOSTED QUEUE
NO
YES
REMOVE ENTRY FROM READ-EXCLUSIVE LIST
ENQUEUE BLOCK ON INTER TASK QUEUE
PUT IOB IN UNPOSTED QUEUE
REMOVE FIRST IOB FROM QUEUE
DEQUEUE BLOCK FROM INTERTASK QUEUE
ISSUE EXCP TO REREAD BLOCK
RETURN TO SUPERVISOR
WRITE-ADD
TYPE OF REQUEST
WRITE-EXCLUSIVE
MOVE DATA IN CURRENT BLOCK TO INPUT AREA OF IOB REQUESTING SAME BLOCK
RELEX
RETURN TO SUPERVISOR
EXIT TO SELF-FORMAT MODULE
POST THIS CURRENT READ REQUEST
RETURN TO ASI OR RELEX

Figure 16. Exclusive control in a multitasking system

MODULE DIRECTORY

In this directory, each BDAM module is listed by its microfiche name. The following information is listed opposite the microfiche name:

- The record format, where applicable, since it frequently determines which module is used to perform a given operation
- An indication of the operation performed by each module
- The number of the page on which the description of the module begins

Microfiche Name	Record Formats	Function/Description	Page on which Module Description Begins
IGC0005C		Release Exclusive Control SVC	31
IGC0005G	F,V,U,VS	FREEDBUF SVC	26
IGG019KA	F,V,U	Foundation Module	9
IGG019KC		Relative Track (TTR) Address Conversion	14
IGG019KE		Relative Block Conversion Without Track Overflow	15
IGG019KF		Relative Block Conversion With Track Overflow	16
IGG019KG		Relative Block Feedback Without Track Overflow	16
IGG019KH		Relative Block Feedback With Track Overflow	16
IGG019KI	F,V,U	Update By Key	17
IGG019KJ	VS	Foundation Module	9
IGG019KK	F,V,U	Update By Block ID	17
IGG019KL	VS	Dynamic Buffering	25
IGG019KM	V,U	WRITE–add, Self–Format	20
IGG019KN	VS	WRITE–add, Self–Format, Spanned Records	23
IGG019KO	F	WRITE–Add, Preformat	18
IGG019KQ		Write Verification	24
IGG019KR	VS	Update Spanned Records	17
IGG019KU		Channel End/Abnormal End Appendage	27
IGG019KW		Extended Search for Update	17
IGG019KY	V,U	Extended Search for Self–format	20,23
IGG019LA	F	Extended Search for Preformat	18
IGG019LC		End–of–Extent Appendage	28
IGG019LE	F,V,U	Dynamic Buffering	25
IGG019LG		Exclusive Control	29
IGG019LI		Check Module	32
IGG0193A		Open Executor	5
IGG0193C		Open Executor	5
IGG0193E		Open Executor	8
IGG0193F		Open Executor	9
IGG0193G		Open Executor	7
IGG0203A		Close Executor	33

DATA AREAS

BCB (Buffer Control Block)

A BCB is built if the dynamic buffering option has been specified or if spanned records are being processed. Module IGG0193E or IGG0193F obtains a continuous area of main storage for both the BCB and the required number of buffers. The BCB is initialized by module IGG0193E or IGG0193F but subsequent entries are placed in it by the dynamic buffering module. The main–storage area for both the BCB and the buffers is released by the BDAM close executor module.

A BCB is also used to control the pool of segment work areas for processing spanned records. The format of this block is shown in Figure 17. The format of the BCB for dynamic buffering is shown in Figure 18.

0(0) Flag [1]	1(1) Address of Next Available Buffer
4(4) Number of Buffers	6(6) Buffer Length

[1] X'FF' = Buffer being used
X'00' = Buffer available

Figure 17. BCB for spanned records without dynamic buffering

0(0)	BCBFRQT
4(4)	BCBFRQB
8(8)	BCBNABFR
12(C)	BCBTBRS

Figure 18. BCB for dynamic buffering

Field	Field Size (in bytes)	Field Contents and Comments
BCBFRQT	4	Address of the first IOB waiting to be assigned a buffer from the buffer queue. The dynamic buffering module inserts this address.
BCBFRQB	4	Address of the last IOB waiting to be assigned a buffer from the buffer queue. The dynamic buffering module inserts this address.
BCBNABFR	4	Address of the next buffer available for assignment to an IOB. Initially, module IGG0193E or IGG0193F inserts this address. Subsequent addresses are inserted by the dynamic buffering module.
BCBTBRS	4	The total size, in bytes, of the buffer pool and the BCB. Module IGG0193E or IGG0193F inserts this value.

Figure 19. Fields, field size, and field contents of the BCB for dynamic buffering

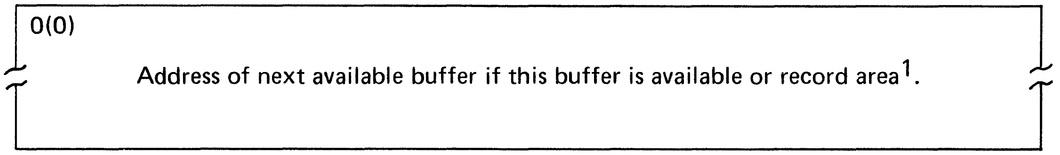
Buffer Pools

A buffer pool is a continuous area of main storage divided up into buffers. BDAM uses three kinds of buffer pools:

- A pool of segment work areas for processing spanned records without dynamic buffering
- A pool of segment work areas and record areas for processing spanned records with dynamic buffering
- A pool of record areas for dynamic buffering of nonspanned records

A buffer pool follows its buffer control block in main storage. The dynamic buffering pools use the buffer control block described in Figure 18, while the segment work area pool uses the buffer control block described in Figure 17.

Figure 20 describes a section of the buffer pool used for dynamic buffering of nonspanned records, Figure 21 describes a section of the buffer pool used for dynamic buffering of spanned records, and Figure 22 describes a section of the buffer pool used for processing spanned records without dynamic buffering.



0(0)

Address of next available buffer if this buffer is available or record area[1].

[1] This field is 4 bytes long if it contains the address of the next available buffer. If it is the record area, its length is DCBBLKSI + DCBKEYLE, with the length, if necessary, rounded to the next multiple of 8.

Figure 20. Buffer in buffer pool: dynamic buffering of nonspanned records

Offset	Field
0(0)	Address of Next Available Buffer
4(4)	Address of Record Area
8(8)	Offset[1]
10(A)	Segment Currently Being Processed [2]
0(0)	Address of this Buffer. Used to free the buffer after a WRITE request.
4(4)	Record Area (as the user sees it)[3]. Segments read into the segment area will be read here.

[1] When this offset is added to the address of the record area (above), the sum is the address of the next segment to be processed.

[2] Length of segment = smaller of: track size and DEBBLKSI + DCBKEYLE.
The end of this field is padded with up to 7 bytes, if necessary, for alignment of the next field.

[3] Length of record area = DCBBLKSI + DCBKEYLE.
The end of this field is padded, if necessary, to round the buffer length to a multiple of 8.

Note: The first three fields (bytes 0 to 10 in decimal) compose the segment work area.

Figure 21. Buffer in buffer pool: dynamic buffering of spanned records

Offset	Field
0(0)	Flag Field[1]
1(1)	Address of Next Available Buffer
4(4)	Address of Record Area
8(8)	Offset[2]
10(A)	Segment Currently Being Processed[3]

[1] X'FF' = buffer in use
X'00' = buffer available

[2] When this offset is added to the address of the record area (above), the sum is the address of the next segment to be processed.

[3] Length of segment = smaller of: track size and DCBBLKSI + DCBKEYLE

Note: The first four fields of the buffer (bytes 0 to 10 in decimal) compose the segment work area.

Figure 22. Buffer in buffer pool: simple buffering of spanned records

DCB (Data Control Block)

The DCB contains information about the current use of a data set. Figure 23 shows the fields of the DCB especially important in BDAM applications. A more complete description of the DCB is in *OS System Control Blocks.*

16(10)	DCBKEYLE	17(11)	DCBREL
		49(31)	DCBREAD or DCBWRITE
52(34)	DCBOPTCD	53(35)	DCBCHECK
56(38)	DCBSYNAD		
60(3C)	Reserved	62(3E)	DCBBLKSI
72(48)	DCBIOBUQ		
76(4C)	DCBUQND		
		81(51)	DCBLIMCT
		85(55)	DCBXARG
88(58)	DCBMVXNO	89(59)	DCBDRDX
92(5C)	DCBDFOR		
96(60)	DCBDFBK		
100(64)	DCBDYNB		

Figure 23. Fields of the DCB for BDAM

Field	Field Size (in bytes)	Field Contents and Comments
DCBKEYLE	1	Length of key field for each block in the data set.
DCBREL	3	Number of relative tracks or blocks in the data set. This number is placed in the DCBREL field by a BDAM open executor module (IGG0193A for tracks, IGG0193E for blocks), and it can be used by the processing program in converting a relative address.
DCBREAD or DCBWRITE	3	Address of the BDAM foundation module IGG019KA.
	1	Indication of options specified for the data set. Bits of the DCBOPTCD field and their interpretations for BDAM are as follows (when the bit is set to 1, the interpretation is in effect; when set to 0, the interpretation is not in effect): Bit 0: Write–validity–check option has been specified. Bit 1: Reserved for future use. Bit 2: Extended search has been specified. Bit 3: Feedback has been specified. Bit 4: Actual addressing has been specified. Bit 5: Dynamic buffering has been specified. (This bit is set by BDAM.) Bit 6: Reserved for future use. Bit 7: Relative block addressing has been specified. *Notes:* If neither actual addressing nor relative block addressing has been specified (that is, if bits 4 and 7 are both 0), relative track addressing is specified.
DCBCHECK	3	Address of the check module, IGG019LI.
DCBSYNAD	3	Address of user's SYNAD routine.
DCBBLKSI	2	Maximum size of a record block in the data set.
	4	Reserved for future use.
	4	Reserved for future use.
DCBIOBUQ	4	Address of the first IOB in the unposted queue.
DCBUQND	4	Address of the last IOB in the unposted queue.
DCBLIMCT	3	Number of tracks (for relative track addressing) or number of blocks (for relative block addressing) to be searched when extended search option is specified.
	1	Reserved for future use.
DCBXARG	3	Address of the read–exclusive list.
DCBMVXNO	1	Total number of extents in a multivolume data set.
DCBDRDX	3	Address of the exclusive control module, IGG019LG.
DCBDFOR	4	Address of the format channel program generating module required for the block format indicated in the DCB macro instruction. This address is placed in DCBDFOR by the BDAM open executor IGG0193C.
DCBDFBK	4	Address of the feedback module, IGG019KG (if relative block feedback was specified).
DCBDYNB	4	If dynamic buffering was not specified and BFTEK=R, address of the segment work area; otherwise, unused.

Figure 24. Fields, field sizes, and field contents of the DCB for BDAM

The initial value of each of the fields containing an address in Figure 23 is 00000001. When the data set is opened, the value of each of these fields that corresponds to a required BDAM module is changed to the main–storage address of the module; the values of address fields corresponding to modules that are not required remain at 00000001, and the values of the DCBIOBUQ and DCBUQND fields are set to 00000000.

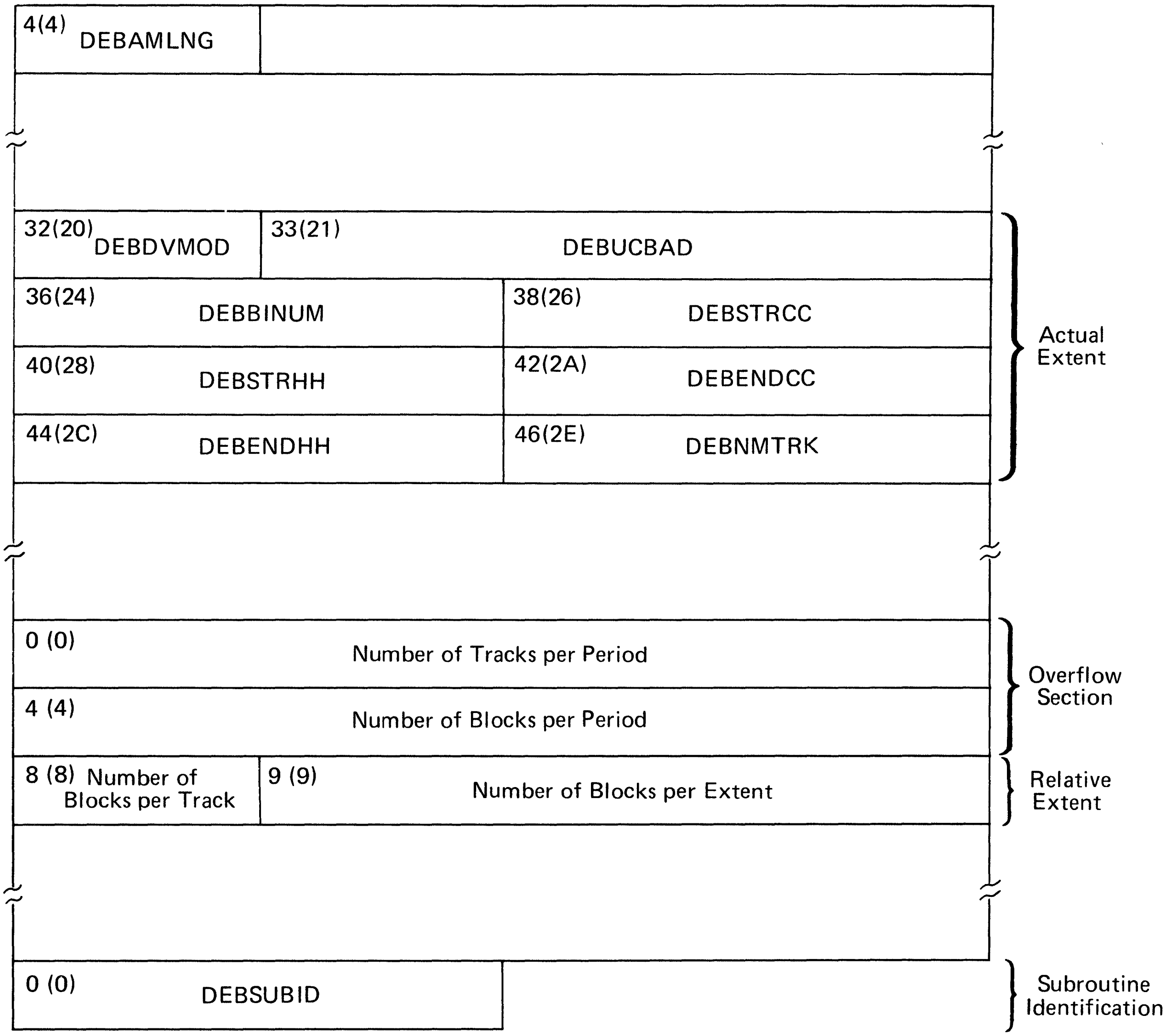


Figure 25. Fields of the DEB for BDAM

DEB (Data Extent Block)

The DEB, which is built by module IGG0193A, contains information about the physical characteristics of a data set.

The DEB fields of special importance in BDAM applications are shown in Figure 25.

The relative extents in the DEB are built only when relative block addressing has been specified. There is one relative extent for each actual extent in the DEB.

If track overflow has not been specified, each relative extent consists of a 1–byte field that contains the number of blocks on a track (for the device used) and a 3–byte field that contains the number of blocks in the extent. The latter value is obtained by multiplying the number of tracks in the extent (given as the value in the last 2 bytes of the associated actual extent) by the number of blocks on a track.

If track overflow has been specified, each relative extent consists of only a 3–byte blocks–per–extent field. The byte preceding each blocks–per–extent field is unused. In addition, two 1–word fields constituting an overflow section are inserted between the last actual extent and the first relative extent. The values in these fields are based on the size of the period that is calculated by module IGG0193E.

Field	Field Size (in bytes)	Field Contents and Comments
DEBAMLNG	1	Number of words of main storage used to contain the relative extents.
DEBDVMOD	1	Device modifier: file mask.
DEBUCBAD	3	Address of UCB associated with this extent.
DEBBINUM	2	Bin number.
DEBSTRCC	2	Cylinder address for the start of an extent.
DEBSTRHH	2	Read/write track address for the start of an extent.
DEBENDCC	2	Cylinder address for the end of an extent.
DEBENDHH	2	Read/write track address for the end of an extent.
DEBNMTRK	2	Number of tracks in the actual extent corresponding to this relative extent.
DEBSUBID	2 repeated for each routine name	Each access method subroutine, appendage subroutine, and IRB routine has a unique 8–byte name. The low–order 2 bytes of each subroutine name are put in this field if the subroutine is loaded into main storage by an open executor module.

Figure 26. Fields, field sizes, and field contents of the DEB for BDAM

DECB (Data Event Control Block)

The DECB, which results from the expansion of either a READ or a WRITE macro instruction, contains information about the I/O operation requested by the macro instruction. Figure 27 contains the fields of the DECB used by BDAM.

0(0)	DECSDECB		
4(4)	DECTYPE	6(6)	DECLNGTH
8(8)	DECDCBAD		
12(C)	DECAREA		
16(10)	DECIOBPT		
20(14)	DECKYADR		
24(18)	DECRECPT		
28(1C)	DECNA		

Figure 27. Fields of the DECB for BDAM

IOB (Input/Output Block)

The IOB contains information required by the I/O supervisor to perform an input/output operation. In BDAM, the IOB is built by the base routine in the foundation module (IGG019KA or IGG019KJ).

The fields of the IOB are constructed as a processing program is executed. The storage area used for the IOB is obtained either from a pool of available IOBs for which storage has been previously obtained or by use of the GETMAIN routine. If the area is taken from a pool of IOBs, that area is made unavailable to the pool until the request associated with the IOB is completed.

When a request is completed, the IOB is either replaced in the pool or assigned to the pool for the first time, depending on how it was obtained. If the IOB was obtained from the pool, the availability byte in the IOB is set to 0 indicating the IOB has been returned to its former position in the pool. If the IOB was obtained by the GETMAIN routine, it is placed in the pool (placement is by size), the next IOB pointers are updated as necessary, and the availability byte is set to 0.

All storage areas assigned to IOBs are released to the system by the FREEMAIN routine when the data set is closed.

Note: If the first use of an IOB storage area occurs with an invalid request, the area is returned to the system by the FREEMAIN routine rather than being placed in the pool when the request is completed.

Various BDAM routines use fields in the IOB as temporary work areas until such time as these fields are filled in with the information described in Figure 30.

Field	Field Size (in bytes)	Field Contents and Comments	
DECSDECB	4	Standard event control block (ECB). (Refer to the IOBDSTAT field of the IOB.)	
DECTYPE	2	Type of request operation. The contents of this field are described in the discussion of the IOBDTYPE field.	
DECLNGTH	2	Length of data portion of the block being processed.	
DECDCBAD	4	Address of the DCB to which a request is related.	
DECAREA	4	Address of area into which the data portion of a block is to be written or from which it is to be read.	
DECIOBPT	4	Address of the IOB associated with this DECB.	
DECKYADR	4	The contents of this field vary depending on the type of request the DECB refers to.	
		Type of Request	*DECKYADR Contents*
		Write by ID	Address of the key to be written.
		Write–Add	Address of the key to be written.
		Read by ID	Address of the area into which the key is to be read.
		Read by Key	Address of the key to be used as a search argument.
		Write by Key	Address of the key to be used as a search argument.
		Write–Add (Format-F)	Key to be written to replace the dummy key. (Searching is done on the hexadecimal 'FF' that is in the first byte of the key field of the dummy record.)
DECRECPT	4	Address of the BLKREF field.	
DECNA	4	Address of the next address feedback field. This field is present only if "R" is coded in the READ macro.	

Figure 28. Fields, field sizes, and field contents of the DECB for BDAM

There are four main sections to the IOB as used by BDAM (see Figure 29).

The first part of the IOB, an 8–byte prefix, contains information applicable only to spanned records. It is present only if BFTEK=R and RECFM=VS.

The second part is a standard 40–byte section and is described in the publication *OS System Control Blocks*. BDAM refers to this part, for example, to determine the status of a completed channel program and to locate addresses of storage areas to be used as work areas.

The third part of the IOB is a 40–byte section that contains information needed by BDAM to process a request. The 11 fields in this part are described in Figure 30.

The fourth part contains the channel program constructed for the input or output request. The channel command words are placed in this part of the IOB as they are formed.

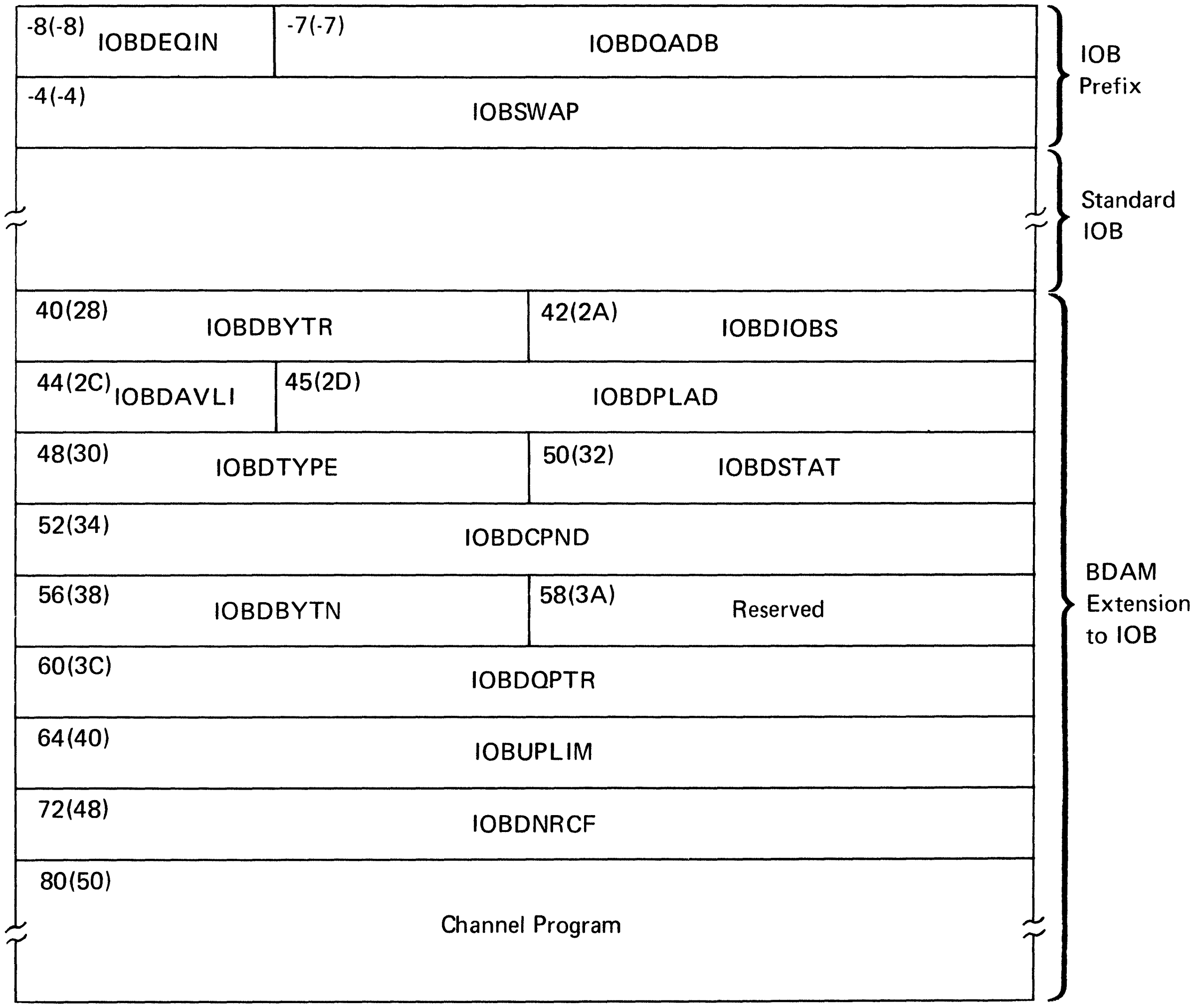


Figure 29. Fields of the IOB for BDAM

Field	Field Size (in bytes)	Field Contents and Comments
IOBDEQIN	1	Bit 0: Indicates that a track containing spanned records is being dequeued. Bits 1–7: Reserved for future use.
IOBDQADB	3	Address of the IOB waiting to dequeue the tracks occupied by a spanned record.
IOBSWAP	4	Address of the segment work area.

Figure 30 (Part 1 of 4). Fields, field sizes, and field contents of the IOB for BDAM

Field	Field Size (in bytes)	Field Contents and Comments
IOBDBYTR	2	Number of unused bytes remaining on a track on which a new variable– or undefined–length block is to be written. This value is initially placed in IOBDBYTR when the channel program reads the capacity record. Subsequent updating of the IOBDBYTR field is done by the self–format module, IGG019KM. The channel program later updates the capacity record before the I/O operation is completed.
IOBDIOBS	2	Overall size of the IOB, in bytes. The base routine of the foundation module places this value in IOBDIOBS after the main–storage area for an IOB has been obtained.
IOBDAVLI	1	Indication of the availability of the IOB. When the IOB is taken from the pool of available IOBs, the value of IOBDAVLI is set to hexadecimal FF to indicate that the IOB is being used (is unavailable). This is done by the base routine of the foundation module. When an I/O operation is posted as complete, and an IOB is either returned to or placed in the pool of available IOBs, the value of IOBDAVLI is set to 0. Depending on the cause of the completion of the I/O operation, the 0 value is set by either the asynchronous interrupt routine or the base routine of the foundation module.
IOBDPLAD	3	Address of the next IOB area in the pool of IOBs attached to the current DCB. If there are no more IOBs in the pool, the value of this field is 0. Each IOB in the pool became a member of the pool after the first use of the IOB. When a new IOB is added to the pool, the IOBDPLAD field of the preceding IOB is updated.
IOBDTYPE	2	Indication of the type of request and the options specified in the DECB related to the request. The contents of the DECTYPE field (see DECB) are placed in the IOBDTYPE field when the IOB is initialized. Significant bits of the IOBDTYPE field and their interpretations for BDAM are as follows (when the bit is set to 1, the interpretation is in effect; when set to 0, the interpretation is not in effect): *First Byte:* Bit 0: Verification of written block has been specified. Bit 1: Track overflow (that is, overflow blocks are being used). (Refer to the publication *OS SAM Logic.)* Bit 2: Extended search has been specified. Bit 3: Feedback of block address has been specified. Bit 4: Actual block addressing is being used. Bit 5: Dynamic buffering is being used. Bit 6: Exclusive control is being used. Bit 7: Relative block addressing is being used. *Note:* If bits 4 and 7 are both 0, relative track addressing is being used. *Second Byte:* Bit 0: S has been specified in the key address operand of the READ or WRITE macro instruction. For dynamic buffering, a buffer is to be allocated for a READ request, and the key part of a buffer is to be freed after a WRITE request. Bit 1: S has been specified in the length operand of the READ or WRITE macro instruction. When variable–length blocks are being written, the setting of this bit is ignored (that is, not tested) since the block length is given in the first 2 bytes, of the data field. Bit 2: Next address feedback can be the address of either a data record or a capacity record, whichever occurred first. This bit can be set to 1 only when bit 3 (below) is set to 1.

Figure 30 (Part 2 of 4). Fields, field sizes, and field contents of the IOB for BDAM

Field	Field Size (in bytes)	Field Contents and Comments
		Bit 3: R is suffixed to the type of the READ or WRITE macro instruction, meaning that next address feedback has been requested. Bit 4: A READ request. (0 indicates a WRITE request.) Bit 5: The search argument is the block key. (0 indicates the search argument is the block ID.) Bit 6: Indicates a WRITE request to add a new block. Bit 7: A RELEX macro instruction has been issued.
IOBDSTAT	2	Indication of status of the request. Significant bits of the IOBDSTAT field and their interpretations for BDAM are as follows (when the bit is set to 1, the interpretation is in effect; when set to 0, the interpretation is not in effect): *First Byte:* Bit 0: The request has completed abnormally. See second byte for details. Bit 1: On extended search, the ASI routine in the foundation module is to issue the EXCP macro instruction after the end–of–extent appendage module has determined that the next extent is on a new volume. The end–of–extent appendage module cannot issue an EXCP macro instruction in this case. Bit 2: Reserved for future use. Bit 3: On extended search, indicates to relative block conversion routine that the second pass of a two–pass conversion routine has been completed. *Note:* The first pass of the routine converts the starting address for a track search, and the second pass converts the address for the search limit. The bit is set to 1 when the second pass begins. This bit is also set by the self–format module after the module has calculated the number of bytes required to write a block on a track. Then, if additional tracks must be examined for space, the calculation of bytes required is bypassed. Bit 4: The read–exclusive request related to this IOB has been placed on an intertask queue by the exclusive control module. Bit 5: A buffer has been assigned to this IOB. Bit 6: A given block (to be written) can fit on the track associated with the capacity record that has just been read into storage. Module IGG019KM sets this indicator. Bit 7: Unused. *Second Byte:* The bits in this byte, when on, indicate abnormal completion of a request. When the request is posted as complete, these indications of abnormal completion are placed in byte 1 (the second byte) of the DECSDECB field of the DECB. Bit 0: The requested block was not found on the indicated track. Bit 1: The length of the block was incorrect. Refer to "IGG019KU (Channel End/Abnormal End Appendage Module)" for more information about incorrect length. Bit 2: No space in which to write a new block was found. Bit 3: Request is invalid. Bit 4: A READ operation (either to bring data into main storage or to verify written data) has resulted in a data check error that has not been corrected by the standard IOS error retry procedure. (Refer to "IGG019KQ (Write–Verify Module)" for more detailed information about verification of data.)

Figure 30 (Part 3 of 4). Fields, field sizes, and field contents of the IOB for BDAM

Field	Field Size (in bytes)	Field Contents and Comments
		Bit 5: The request has been completed but the block the user has requested to be read or written is an end–of–data set record (indicated as having a data field length of 0). Refer to the section "IGG019KU (Channel End/Abnormal End Appendage Module)" for more information about the end–of–data set conditions.
		Bit 6: An error that cannot be attributed to any of the other causes indicated by bits in this byte has occurred.
		Bit 7: No match has been found on the read–exclusive list.
IOBDCPND	4	The main–storage address of the expected end of the channel program if the program goes to a normal completion. This address is placed in the IOBDCPND field by the base routine of the foundation module. At the completion of a request, the I/O supervisor places an address in the channel status word. This address is equal to the address of the last channel command word executed plus 8 bytes. A normal completion is indicated if the two addresses are equal and there have been no error indications.
IOBDBYTN	4	The required number of bytes to contain a new block. This value is calculated by the self–format module after control has been given to it by the ASI routine in the foundation module.
IOBDQPTR	4	Either the address of the next IOB in the unposted queue of IOBs (address determined by the exclusive control module, IGG019LG) or the address of the next IOB in the IOB buffer queue (address determined by dynamic buffering module).
IOBUPLIM	8	Address at which to begin the search for the start of a track on which the indicated block is contained or is to be written. On extended search, the address of the first track following the last track to be searched.
IOBDNRCF	8	The count field developed by the self–format module when a new block of variable–length or undefined–length records is to be added to a track.

Figure 30 (Part 4 of 4). Fields, field sizes, and field contents of the IOB for BDAM

Read–Exclusive List

The read–exclusive list is an area of main storage containing the UCB address and actual address of blocks requiring exclusive control. The list is composed of one or more 80–byte segments. Each segment has space for nine 8–byte entries, each entry identifying a block for which exclusive control is required. IGG0193A obtains storage for the first 80–byte segment; if additional segments are required, the exclusive control module obtains storage for them. The close executor module releases the storage obtained for the read–exclusive list segments.

Figure 31 indicates the contents of the fields of a typical segment of the read–exclusive list. Figure 32 indicates the contents of an entry in a segment.

0(0)	Address of This Segment
4(4)	Pointer to Next Segment if One Exists
8(8)	First Entry in Segment
16(10)	Space for Seven More Entries
72(48)	Last Entry in Segment

Figure 31. Segment of the read–exclusive list

0(0) Address of the UCB for the block	2(2) Address (CCHHR) of the block being placed on the read-exclusive list. For WRITE-add requests for variable- or undefined-length blocks, the address of the track capacity record (R0).

Figure 32. Entry in the read–exclusive list

APPENDIXES

Appendix A: Periods of an Extent

BDAM data sets are created by the basic sequential access method (BSAM). When BSAM places blocks on a direct–access device, it is possible that a block may start on one track and finish on a following track within the same extent. Such a block is called an overflow block. At least 1 byte of the data portion of a block must fit on a track in order for the block to overflow to the next track. For purposes of calculating the size of a period, the track on which a block begins is considered to contain the block. When a track is reached on which not enough space is left to contain at least 1 byte of the data portion of a new block to be written, the end of a period has been reached.

Thus, a period is a group of tracks containing a group of blocks such that the first track does not begin with an overflow block and the last track does not contain a block that overflows to another track. Example 1 illustrates the concept of a period.

Example 1: In this example, assume the following:

- A given data set is on a device that permits 3625 bytes per track to be allocated to data blocks, excluding a track capacity record (R0).
- The block length for the data set is 844 bytes, divided as follows:

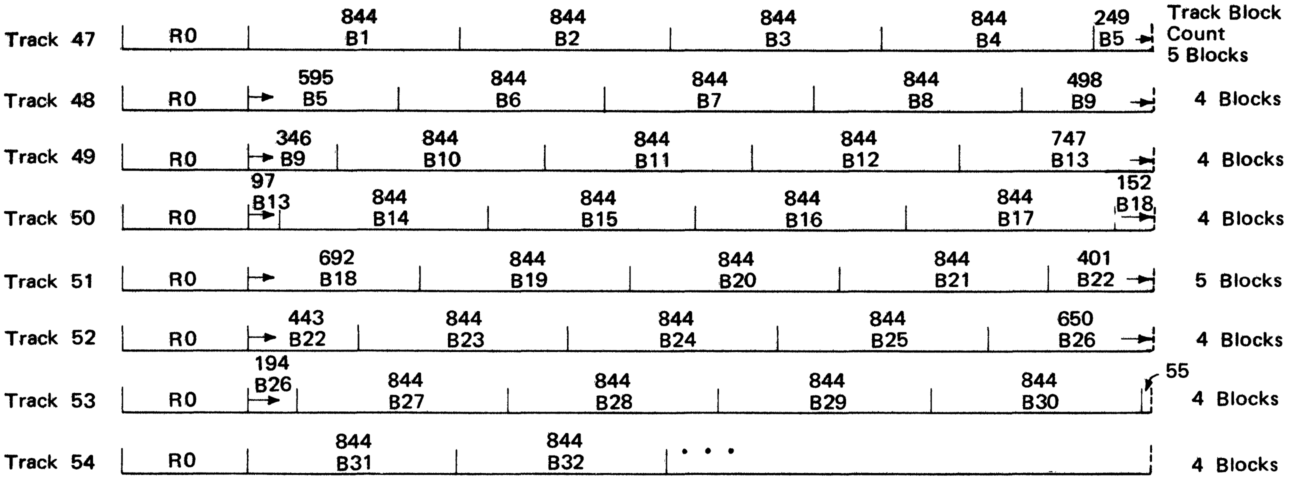
  14 bytes for address marker plus count area
  100 bytes for the block key portion
  730 bytes for the block data portion
- There are no interrecord gaps on the tracks. (This assumption simplifies the calculations in the example.)
- Part of the given data set occupies an extent consisting of consecutive tracks beginning with track 47 on this device.

From the second assumption, it is determined that at least the first 115 bytes (count + key + 1 data byte) of a block must fit on a track for track overflow to occur. Figure 33 illustrates the manner in which the data blocks would appear on the tracks of the extent.

The identifications B1, B2, ..., B32 represent the first 32 blocks placed in this extent. The numbers above the blocks represent the number of bytes of the block appearing on a track. Arrows at the end of a track and at the beginning of a track indicate where track overflow occurs.

As shown in Figure 33, track overflow occurs from track 47 to track 48 (block B5), from track 48 to track 49 (block B9), and so on until the end of track 53. Since track overflow (in this example) requires at least the first 115 bytes of a block to appear on a track, and only 55 bytes remain on track 53 after block B30 has been placed there, block B31 cannot overflow from track 53 to track 54. Therefore, tracks 47–53 constitute a period, and a new period begins with block B31 on track 54. For purposes of calculating relative block addresses (see Examples 2 and 3 in Appendixes B and C), the number of blocks on each track is given in Figure 33 as the "Track Block Count."

Module IGG0193E computes, based on block characteristics (key length and data length) and device characteristics, the size of the period for the data set. It then computes both the number of blocks in a period and the number of tracks in a period and places the computed values in the two fields of the overflow section of the DEB. These fields are each 1 word long, and they occur only once for a given data set. The values placed in these fields are constant for a given data set.



Track Block Count
Track 47 R0 844 B1 844 B2 844 B3 844 B4 249 B5 5 Blocks
Track 48 R0 595 B5 844 B6 844 B7 844 B8 498 B9 4 Blocks
Track 49 R0 346 B9 844 B10 844 B11 844 B12 747 B13 4 Blocks
Track 50 R0 97 B13 844 B14 844 B15 844 B16 844 B17 152 B18 4 Blocks
Track 51 R0 692 B18 844 B19 844 B20 844 B21 401 B22 5 Blocks
Track 52 R0 443 B22 844 B23 844 B24 844 B25 650 B26 4 Blocks
Track 53 R0 194 B26 844 B27 844 B28 844 B29 844 B30 55 4 Blocks
Track 54 R0 844 B31 844 B32 • • • 4 Blocks

Figure 33. Track overflow

Because the allocation of actual extents to members of a data set is performed by space management routines (refer to the publication *OS DADSM Logic),* while the period is a concept used by BDAM, the boundaries of extents and periods may differ. However, the end of an extent terminates the last period in the extent. In this case, the last period may be complete or it may be only partially complete. In either case, the start of a new extent coincides with the start of a new period.

Appendix B: Calculations Done in Module IGG019KE to Get a Relative Track Address

To convert a relative block address to a relative track address, module IGG019KE does the following calculations: Starting with the first extent in the data set, IGG019KE subtracts the number of blocks in that and each successive extent from the BLKREF field. (The BLKREF field, whose address is in the DECRECPT field of the DECB, contains the relative block number that was specified in the block address field of the READ macro instruction.) As the number of blocks in each extent is subtracted, the number of tracks in each subtracted extent is accumulated. Thus, the value in the BLKREF field decreases while the number of accumulated tracks increases. This process continues until an extent is reached in which the number of blocks, if subtracted from the number that remains in the BLKREF field, would result in a negative value. This extent is called the terminal extent.

When the terminal extent is reached, the number of remaining blocks in the BLKREF field is divided by the blocks per track field in the DEB. The quotient in this division is added to the cumulative total of tracks. This cumulative total represents the track number, relative to the beginning of the data set, on which the block whose address is to be converted resides. The remainder in this division is the number of blocks on the terminal extent that must be counted to get to the block whose address is to be converted.

The relative track address, in the form TTR, is the result of these calculations. TT is the number of the track on which the block whose address is to be converted resides, and R is the number of that block.

Example 2 illustrates conversion of a relative block number to a relative track address.

Example 2: Assume a data set is contained in four extents identified as I, II, III, and IV. Let extent I contain 10 tracks with 80 data blocks; extent II contain 14 tracks with 112 data blocks; extent III contain 8 tracks with 64 data blocks; and extent IV contain 12 tracks with 96 data blocks. (This assumes that the data set is on a device permitting 8 data blocks to be placed on one track.) The information needed from the DEB for this data set is summarized in Figure 34.

Without Track Overflow

DEB Field	Extent I	Extent II	Extent III	Extent IV
Blocks per Track	8	8	8	8
Tracks per Extent	10	14	8	12
Blocks per Extent	$B_1=80$	$B_2=112$	$B_3=64$	B=96

Figure 34. DEB information needed to calculate relative track addresses

If the BLKREF field contains 284, the calculations to find the relative track address are as follows:

BLKREF value $- B_1 = R_1$ (remainder)

$284 - 80 = 204$ The 80 blocks from extent I are on 10 tracks.

$R_1 - B_2 = R_2$

$204 - 112 = 92$ The 112 blocks from extent II are on 14 tracks.

$R_2 - B_3 = R_3$

$92 - 64 = 28$ The 64 blocks from extent III are on 8 tracks.

$R_3 - B = R$

$28 - 96 < 0$

Since R is less than 0, the last extent (IV) cannot be subtracted. Extent IV becomes the terminal extent. The previous remaining value ($R_3 = 28$) is divided by the blocks per track value (8) to give a quotient of 3 and a remainder of 4. The 3 represents the number of tracks of the terminal extent that must be added to the sum of the underlined numbers of tracks from extents I, II, and III. The 4 represents the number of data blocks that must be counted from the beginning of the terminal track. Thus, the relative track address (TTR) of the block in this example is 35 tracks (the TT value) and 4 blocks (the R value) from the beginning of the data set.

Appendix C: Calculations Done in Module IGG019KF to Get a Relative Track Address

To convert a relative block address to a relative track address, module IGG019KF does the following calculations:

Starting with the first extent in the data set, IGG019KF subtracts the number of blocks in that and each successive extent from the BLKREF field. (The BLKREF field, whose address is in the DECRECPT field of the DECB, contains the relative block number that was specified in the block address field of the READ macro instruction.) As the number of blocks in each extent is subtracted, the number of tracks in each subtracted extent is accumulated. This process continues until an extent is reached in which the number of blocks, if subtracted from the number that remains in the BLKREF field, would result in a negative value. This extent is called the terminal extent.

The number of blocks in each period of the terminal extent is then subtracted from the BLKREF field. As the number of blocks in each period are subtracted, the number of tracks on which these blocks reside is added to the cumulative total of tracks. This process continues until a period is reached in which the number of blocks, if subtracted from the number that remains in the BLKREF field, would result in a negative value. This period is called the terminal period.

The number of blocks in each track of the terminal period is then subtracted from the BLKREF field. As the number of blocks in each track of the terminal period is subtracted, the numbers of tracks on which these blocks reside is added to the cumulative total of tracks. This process continues until a track is reached in which the number of blocks, if subtracted from the BLKREF field, would result in a negative value. This track is the terminal track.

The accumulated tracks and the number remaining in the BLKREF field at this point become the relative track address, in the form TTR. TT is the track on which the block whose address is to be converted resides, and R is the actual number of the block on that track.

Example 3 illustrates conversion of a relative block number, when track overflow has been specified, to a relative track address.

Example 3: Assume a data set is contained in three extents identified as I, II, and III. Let extent I contain 20 tracks with 114 data blocks; extent II contain 10 tracks with 57 data blocks; and extent III contain 27 tracks with 153 data blocks. Further, assume that open executor module IGG0193E has established that each period contains 3 tracks with a total of 17 blocks and that the blocks are placed on the tracks so that the first 2 tracks contain 6 blocks each and the third track contains 5.

The information needed from the DEB for this data set is summarized in Figure 35.

With Track Overflow

DEB Field	Extent I	Extent II	Extent III
Tracks per Extent	20	10	27
Tracks per Period	3	3	3
Blocks per Period	17	17	17
Blocks per Extent	$B_1=114$	$B_2=57$	$B_3=153$

Figure 35. DEB information needed to calculate relative track addresses

If the BLKREF field contains 217, the calculations to find the relative track address are as follows:

BLKREF value $- B_1 = R_1$ (remainder)

$217 - 114 = 103$ The 114 blocks (from extent I) are on 20 tracks

$R_1 - B_2 = R_2$

$103 - 57 = 46$ The 57 blocks (from extent II) are on 10 tracks

$R_2 - B_3 = R_3$

$46 - 153 < 0$

Since R_3 is less than 0, the last extent (III) cannot be subtracted. Extent III becomes the terminal extent. Now the periods in the terminal extent are considered.

Let the periods of extent III be designated as IIIa, IIIb, IIIc, etc. The calculations proceed as follows:

R_2 − IIIa = R_3

$46 - 17 = 29$ The 17 blocks (from period IIIa) are on 3 tracks.

R_3 − IIIb = R

$29 - 17 = 12$ The 17 blocks (from period IIIb) are on 3 tracks.

R − IIIc = R

$12 - 17 < 0$

Since R is less than 0, all of the blocks in the period (IIIc) cannot be subtracted. Period IIIc becomes the terminal period. Now the blocks on the tracks in the terminal period are subtracted. The 12 remaining blocks (the R value) are equivalent to 1 track (of 6 blocks) plus 6 blocks (on the terminal track).

The total number of tracks plus additional blocks thus is equal to the sum of the underlined numbers of tracks in the two full extents (I and II) and the two full periods (IIIa and IIIb) in extent III plus the 1 track and 6 blocks from period IIIc. This value is 37 tracks and 6 blocks, giving a TTR value that can be used by the BPAM convert–to–actual routine to obtain an actual address for the block.

Appendix D: Channel Programs for BDAM

The channel program for each request using BDAM is constructed by the appropriate channel program generating module and placed in the IOB for that request. A channel program consists of a group of channel command words (CCWs), each word having the following format:

Command Code (1 byte)	Address (3 bytes)	Flags (5 bits)	000 (3 bits)	(ignored) (1 byte)	Count (2 bytes)

Note: The last 4 bytes are ignored by a transfer–in–channel (TIC) command word.

The entry in the address field is one of the following:

- The main–storage address where data is to be placed or found; this is for a read or write command word.
- The location of the search argument; this is for a search command word.
- The address of the CCW to which a transfer is made; this is for a transfer–in–channel command word.

The entry (or entries) in the flags field have the following meanings:

CC	Command chaining
DC	Data chaining
SKIP	Skip the transferring of data
SILI	Suppress incorrect length indication

The entry in the count field represents either the number of bytes of data to be transferred or the number of bytes of data on which a search is to be made for comparison.

Note: For more detailed information on channel programs, see *IBM System/360 Component Descriptions — 2841 and Associated DASD.*

In the channel programs that follow, the purpose of each command word is given in the comment following the count field. The channel command words are identified by the number to the left of the command code.

If track overflow has been specified, the applicable form of the channel program will end with a CCW having NOP as the command code and ignoring the other fields. The preceding CCW will also have the command chaining (CC) flag bit set on.

1. *Channel Program for Reading or Writing by Block ID (Type DI)*

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1[1,2]	23	Set sector	Sector address 1	40	CC	1	Wait for record — sector value is precalculated for format–F record
2	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for specified block
3	08	TIC	CCW2	00			Repeat search if block not found
4[3]	0E 0D	Read Write Key and data	Contents of DECKYADR	80	DC	Key length	Read or write key portion of block
5	06 05	Read Write Data	Contents of DECAREA	00 40[4]	(CC)	Data length	Read or write data portion of block
6[2,4,5]	22	Read sector	Sector address 2	40	CC	1	Get sector value for validity–check routine
7[4,6]	1B	Seek cylinder head (CCHH)	IOBDNRCF	40	CC	6	Seek back to track where block begins
8[2,4]	23	Set sector	Sector address 2	40	CC	1	Wait for record to come into position again
9[4]	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for block just updated
10[4]	08	TIC	CCW9	00			Repeat search if block not found
11[4]	0E	Read key and data		30	SILI, SKIP	256	Read block just written to check for errors — write–validity–check

[1] CCW1 is present only for fixed–length records.
[2] This CCW is present only for devices with the rotational position sensing feature
[3] CCW4 is omitted if either the DCBKEYLE or DECKYADR is zero
[4] CCWs 6–11 are present and the command–chain bit in CCW4 is on only when the DCPOPTCD field of the DCB specifies the write–validity–check option
[5] CCW6 is omitted for fixed–length records
[6] CCW7 is present only if track overflow is specified

2. *Channel Program for Reading or Writing by Block Key (TYPE KD)*

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1	12	Read count	IOBDNRCF+2	60	CC, SILI	5	Read full address (CCHHR) for feedback
2	29	Search key equal	Contents of DECKYADR	60	CC, SILI	Key length	Search for block with specified key
3	08	TIC	CCW1	00			Repeat search if block not found
4	06 05	Read Write Data	Contents of DECAREA	00 40[2]	(CC)	Data length	Read or update the desired block
5[1,2]	22	Read sector	Sector address 2	40	CC	1	Get sector value for validity–check routine
6[2,3]	1B	Seek head (CCHH)	IOBDNRCF	40	CC	6	Seek back to track where block begins
7[1,2]	23	Set sector	Sector address 2	40	CC	1	Set to sector of last write
8[2]	29	Search key equal	Contents of DECKYADR	60	CC, SILI	Key length	Search for block just updated
9[2]	08	TIC	CCW6	00			Repeat search if block not found
10[2]	06	Read data		30	SILI, SKIP	256	Read block just written to check for errors — write–validity–check

[1] These CCWs are present only for devices with the rotational position sensing feature

[2] CCWs 5–10 are present and the command–chain bit in CCW4 is on only when the DCBOPTCD field of the DCB specifies the write–validity–check option.

[3] CCW6 is present only if track overflow is specified.

3. *Channel Program for Writing a New Block of Fixed-Length Records (Type DA)*

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1	12	Read count	IOBDNRCF+2	60	CC, SILI	5	Read full address (CCHHR) for feedback
2	29	Search key equal	Dummy key	60	CC, SILI	1	Search for a dummy record
3	08	TIC	CCW1	00			Repeat search if record not found
4	06	Read data	IOBDNRCF+6	60	CC, SILI	1	Read dummy record's position — "R" of CCHHR
5[1]	22	Read sector	Sector address 2	40	CC	1	Note sector value of dummy record
6[2]	1B	Seek head (CCHH)	IOBDNRCF	40	CC	6	Seek back to track where block begins
7[1]	23	Set sector	Sector address 2	40	CC	1	Wait for dummy record to come into position again
8	31	Search ID equal	IOBDNRCF+2	40	CC	5	Search for dummy record
9	08	TIC	CCW6	00			Repeat search if record not found
10[3]	0D	Write key and data	Contents of DECKYADR	80	DC	Key length	Write the new key
11	05	Write data	Contents of DECAREA	00 40[4]	(CC)	Data length	Write the new data
12[2,4]	1B	Seek head (CCHH)	IOBDNRCF	40	CC	6	Seek back to track where block begins
13[1]	23	Set sector	Sector address 2	40	CC	1	Wait for block just written to come into position again
14	31	Search ID equal	IOBDNRCF+2	40	CC	5	Search for block just written
15	08	TIC	CCW14	00			Repeat search if block not found
16	8E	Read key and data		30	SILI, SKIP	256	Check block just written for errors — write–validity–check

[1] These CCWs are present only for devices with the rotational position sensing feature.

[2] CCWs 6 and 12 are present only if track overflow is specified.

[3] CCW10 is omitted if either the DCBKEYLE or DECKYADR field is 0.

[4] CCWs 12–16 are present and the command–chain bit in CCW11 is on only when the DCBOPTCD field of the DCB specifies the write–validity–check option.

4. *Channel Program for Writing a New Block of Variable–Length or Undefined–Length Records (Type DA)*

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1[1]	23	Set sector	Sector = 0	40	CC	1	Set to beginning of track
2	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for track capacity record
3	08	TIC	CCW2	00			Repeat search if record not found
4	06	Read data	IOBDNRCF	20	SILI	7	Read capacity record into IOB
5[1,2]	23	Set sector	Sector = 0	40	CC	1	Wait for beginning of track to come into position again
6	31	Search ID equal	IOBUPLIM	40	CC	5	Search for track capacity record
7	08	TIC	CCW6	00			Repeat search if record not found
8	05	Write data	IOBSEEK+3	60	CC, SILI	7	Update capacity record
9	31	Search ID equal	CCW2	40	CC	5	Search for current last block on track
10	08	TIC	CCW9	00			Repeat search if block not found
11	1D	Write count	IOBDNRCF	80	DC	8	Write new record
12[3]	1D	Write key	Contents of DECKYADR	80	DC	Key length	
13	1D	Write data	Contents of DECAREA	00 40[4]	DC	Data length	
14[1,4]	22	Read sector	Sector address 2	40	CC	1	Note sector value of new record
15[1,4]	23	Set sector	Sector address 2	40	CC	1	Wait for new record to come into position again
16[4]	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for new record
17[4]	08	TIC	CCW16	00			Repeat search if record not found
18[4]	0E	Read key and data		70	CC, SILI, SKIP	256	Check new record for errors — write–validity
19[1,4]	23	Set sector	Sector = 0	40	CC	1	Wait for beginning of track to come into position again
20[4]	16	Read R0		00		16	Check capacity record for errors — write–validity–check

[1] These CCWs are present only for devices with the rotational position sensing feature

[2] This is actually two channel programs. The first program (CCWs 1–4) reads the capacity record. After the BDAM routines find a capacity record that indicates enough space is available for the add request, CCWs 1–4 are overlaid with data that includes the address of the current last block. The channel program that actually writes the new block consists of CCWs 5–20.

[3] CCW12 is omitted if either the DCBKEYLE OR DECKYADR field is 0.

[4] CCWs 14–20 are present and the command–chain bit in CCW13 is on only when the DCBOPTCD field of the DCB specifies the write–validity–check option.

5. *Channel Program for Reading or Writing by Block Key Using Extended Search (Type DK)*

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1[1]	23	Set sector	Sector=0	40	CC	1	Set to beginning of track
	03	NOP		60	CC, SILI	1	
2	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for capacity record
3	08	TIC	CCW2	00			Repeat search if record not found
4	B1	Search ID equal (multitrack)	IOBUPLIM+3	40	CC	5	Check for search limit
5	08	TIC	CCW7	00			Go to key search if still within search limit
6	03	NOP		20	SILI	1	End channel program if search limit reached
7	29	Search key equal	Contents of DECKYADR	60	CC, SILI	Key length	Search for given key
8	08	TIC	CCW4	00			Check limit again before continuing search
9[1]	22	Read sector	Sector address 2	40	CC	1	Note sector value to be used for read or write
	03	NOP		60	CC, SILI	1	
10[1,2]	23	Set sector	Sector=0	40	CC	1	Wait for beginning of track to begin feedback loop
	03	NOP		60	CC, SILI	1	
11	1A	Read home address		70	CC, SILI, SKIP	1	Note beginning of track
12[1]	23	Set sector	Sector address 2	40	CC	1	Wait for desired record to come into position
	03	NOP		60	CC, SILI	1	
13	12	Read count	IOBDNRCF+2	60	CC, SILI	5	Read CCHHR for feedback
14	29	Search key equal	Contents of DECKYADR	60	CC, SILI	Key length	Search for user–specified record
15	08	TIC	CCW13	00			Update CCHHR and repeat search if wrong record
16	05 00	Read Data Write	Contents of DECAREA	00 40[3]	(CC)	Data length	Read or update the block
17[3,4]	1B	Seek head (CCHH)	IOBDNRCF	40	CC	6	Seek back to track where block beings
18[1]	23	Set sector	Sector address 2	40	CC	1	Wait for desired record to come into position
	03	NOP		60	CC, SILI	1	
19	29	Search key equal	Contents of DECKYADR	60	CC, SILI	Key length	Search for block just updated
20	08	TIC	CCW19	00			Repeat search if block not found
21	06	Read data		30	SILI, SKIP	256	Check block for errors — write–validity–check

[1] These CCWs are set to NOP for devices without the rotational position sensing feature.

[2] CCWs 10–15 are present only if feedback is requested.

[3] CCWs 17–21 are present and the command–chain bit in CCW16 is on only when the DCBOPTCD field of the DCB specifies the write–validity–check option.

[4] CCW17 is present only if track overflow is specified.

6. Channel Program for Adding a New Fixed–Length Block Using Extended Search (Type DA)

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1[1]	23	Set sector	Sector = 0	40	CC	1	Wait for beginning of track to come into position
2	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for capacity record
3	08	TIC	CCW2	00			Repeat search if record not found
4	06	Read data	IOBDNRCF+2	60	CC, SILI	5	Read full address (CCHHR) for feedback
5	31	Search ID equal	IOBDNRCF+2	40	CC	5	Search for last data record on track
6	08	TIC	CCW13	00			If not last data block, check for dummy record
7	29	Search key equal	Dummy key address	60	CC, SILI	1	Search for dummy record
8	08	TIC	CCW10	00			If not dummy record, go to extended search routine
9	08	TIC	CCW15	00			If dummy record found, go to add routine
10	B1	Search ID equal (multitrack)	IOBUPLIM+3	40	CC	5	Check for search limit
11	08	TIC	CCW4	00			Read capacity record from next track if still within search limit
12	03	NOP		20	SILI	1	End channel program if search limit reached
13	29	Search key equal	Dummy head address	60	CC, SILI	1	Search for dummy record
14	08	TIC	CCW5	00			Check next record if dummy record not found
15	06	Read data	IOBDNRCF+6	60	CC, SILI	1	Read dummy record's position — "R" of CCHHR
16[1]	22	Read sector	Sector address 2	40	CC	1	Get sector value of dummy record
17[2]	1B	Seek head (CCHH)	IOBDNRCF	40	CC	6	Seek back to track where dummy record began
18[1]	23	Set sector	Sector address 2	40	CC	1	Wait for dummy record to come ino position again
19	31	Search ID equal	IOBDNRCF+2	40	CC	5	Search for dummy record
20	08	TIC	CCW19	00			Repeat search if record not found
21[4]	0D	Write key and data	Contents of DECKYADR	80	DC	Key length	Write new key
22	05	Write data	Contents of DECAREA	00 40[3]	(CC)	Data length	Write data portion of record
23[3,2]	1B	Seek head (CCHH)	IOBDNRCF	40	CC	6	Seek back to track that contains beginning of track just written
24[1]	23	Set sector	Sector address 2	40	CC	1	Wait for record just written to come into position again
25	31	Search ID equal	IOBDNRCF+2	40	CC	5	Search for record just written
26	08	TIC	CCW25	00			Repeat search if record not found
27	0E	Read key and data		30	SILI, SKIP	256	Check block just written for error — write–validity–check

[1] This CCW is present only for devices with the rotational position sensing feature.

[2] This CCW is present only if track overflow is specified.

[3] CCWs 23–27 are present and the command–chain bit in CCW22 is on only when the DCBOPTCD field of the DCB specifies the write–validity–check option.

[4] This CCW is omitted if either the DCBKEYLE or DECKYADR is 0.

7. *Steps of the Channel Program for Adding a New Variable–Length Spanned (Format–VS) Record (Type DA)*

A. Original channel program for reading capacity record and writing first segment

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1[1]	23	Set sector	Sector=0	40	CC	1	Wait for beginning of track to come into position
	03	NOP		60	CC, SILI	1	
2	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for capacity record
3	08	TIC	CCW2	00			Repeat search if record not found
4	06	Read data	IOBDNRCF	20	SILI	7	Read capacity record into IOB
5[1]	23	Set sector	Sector=0	40	CC	1	Wait for beginning of track to come into position again
	03	NOP		60	CC, SILI	1	
6	31	Search ID equal	CCW2[2]	40	CC	5	Search for capacity record
7	08	TIC	CCW6	00			Repeat search if record not found
8	05	Write data	IOBSEEK+3	60	CC, SILI	7	Update capacity record
9	31	Search ID equal	CCW1[2]	40	CC	5	Search for last block on track
10	08	TIC	CCW9	00			Repeat search if block not found
11	1D	Write count	IOBDNRCF	80	DC	8	Write count field of new record
12[3]	1D	Write key	Contents of DECKYADR	80	DC	Key length	Write key field
13	1D	Write data	Segment work area	00 40[4]	(CC)	Segment length	Write data field
14[4,5]	22	Read sector	Sector address 2	40	CC	1	Note sector value of record just written
15[4,5]	23	Set sector	Sector address 2	40	CC	1	Wait for new record to come into position again
16[4]	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for record just written
17[4]	08	TIC	CCW16	00			Repeat search if record not found
18[4]	0E	Read key and data		70	CC, SILI, SKIP	256	Check the new block for errors — write–validity–check
19[4]	16	Read R0		10	SKIP	16	Check the capacity record

[1] This CCW is set to NOP for devices without the rotational position sensing feature.

[2] This is really two channel programs. The first, CCWs 1–4, reads the capacity record. The second, CCW 5–19, updates the capacity record and writes the first segment of the new record.

[3] CCW12 is omitted if the DCBKEYLE or DECKYADR field is 0.

[4] CCWs 14–19 are present and the command–chain bit in CCW13 is on only when the DCBOPTCD field of the DCB specifies the write–validity–check option.

[5] This CCW is present only for devices with the rotational position sensing feature

B. Channel Program for Writing All Segments but the First (Subsequent Segments Have no Keys)

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1[1]	23	Set sector	Sector = 0	40	CC	1	Wait for beginning of track
	03	NOP		60	CC, SILI	1	
2	31	Search ID equal	CCW2 of Part A	40	CC	5	Search for capacity record
3	08	TIC	CCW2	00			Repeat search if record not found
4	05	Write data	IOBSEEK+3	60	CC, SILI	7	Update capacity record
5	1D	Write count, key,	IOBDNRCF	80	DC	8	Write new record (key length is specified as 0 in IOBDNRCF)
6	1D	And write data	Segment work area	00	(CC)		
7[1,2]	23	Set sector	Sector = 0	40	CC	1	Wait for beginning of track to come into position again
	03	NOP		60	CC, SILI	1	
8[2]	16	Read R0		50	CC, SKIP	16	Check capacity record for errors
9[2]	0E	Read key and data		30	SILI, SKIP	256	Check new record for errors — write–validity–check

[1] This CCW is set to NOP for devices without the rotational position sensing feature.

[2] CCWs 7–9 are present and the command–chain bit in CCW6 is on only when the DCBOPTCD field of the DCB specifies the write–validity–check option.

8. *Stages of the Channel Program to Read a Variable-Length Spanned Record by Block Key Using Extended Search*

A. Original channel program as developed by IGG019KR and IGG019KW

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1[1]	23	Set sector	Sector=0	40	CC	1	Wait for beginning of track
	03	NOP		60	CC, SILI	1	
2	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for capacity record
3	08	TIC	CCW2	00			Repeat search if record not found
4	B1	Search ID equal (multitrack)	IOBUPLIM+3	40	CC	5	Check for search limit
5	08	TIC	CCW7	00			Check record's key if not beyond search limit
6	03	NOP		20	SILI	1	End channel program if search limit reached
7	29	Search key equal	Contents of DECKYADR	60	CC, SILI	Key length	Search for specified key
8	08	TIC	CCW4	00			Repeat search if key not found
9[1]	22	Read sector	Sector address 2	40	CC	1	Note sector value of desired record
	03	NOP		60	CC, SILI	1	
10[1]	23	Set sector	Sector=0	40	CC	1	Go back to beginning of track for feedback loop
	03	NOP		60	CC, SILI	1	
11	1A	Read home address		70	CC,SILI, SKIP	1	Note beginning of track
12[1]	23	Set sector	Sector address 2	40	CC	1	Wait for desired record to come into position
	03	NOP		60	CC, SILI	1	
13	12	Read count	IOBDNRCF+2	60	CC, SILI	5	Read CCHHR for feedback
14	29	Search key equal	Contents of DECKYADR	60	CC, SILI	Key length	Search again for desired record
15	08	TIC	CCW13	00			Repeat search if record not found
16	06	Read data	Segment work area	00		Data length	Read the record

[1] This CCW is set to NOP for devices without the rotational position sensing feature.

B. Channel Program After Reading First Segment

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1[1]	23	Set sector	Sector = 0	40	CC	1	Wait for beginning of track
	03	NOP		60	CC, SILI	1	
2	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for second block on next track
3	08	TIC	CCW2	00			Repeat search if second block not found
4	06	Read data	Segment work area	00		Data length	Read the segment
5[2]	03	NOP		20	SILI	1	

[1] This CCW is set to NOP for devices without the rotational position sensing feature

[2] This CCW is present only when "next address" feedback is requested

9. *Stages of the Channel Program to Write a Variable-Length Spanned Record by Block Key Using Extended Search*

A. Original Channel Program as Developed by IGG019KR and IGG019KW

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1[1]	23	Set sector	Sector = 0	40	CC	1	Wait for beginning of track
	03	NOP		60	CC, SILI	1	
2	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for capacity record
3	08	TIC	CCW2	00			Repeat search if record not found
4	B1	Search ID equal (multitrack)	IOBUPLIM+3	40	CC	5	Check for search limit
5	08	TIC	CCW7	00			Check record's key if not beyond search limit
6	03	NOP		20	SILI	1	End channel program if search limit reached
7	29	Search key equal	Contents of DECKYADR	60	CC, SILI	Key length	Search for specified key
8	08	TIC	CCW4	00			Check limit again if key not found
9[1]	22	Read sector	Sector address 2	40	CC	1	Note sector value of desired record
	03	NOP		60	CC, SILI	1	
10[1]	23	Set sector	Sector = 0	40	CC	1	Wait for beginning of track to begin feedback loop
	03	NOP		60	CC, SILI	1	
11	1A	Read home address		70	CC, SILI, SKIP	1	Note beginning of track
12[1]	23	Set sector	Sector address 2	40	CC	1	Wait for desired record to come into position
	03	NOP		60	CC, SILI	1	
13	12	Read count	IOBDNRCF+2	60	CC, SILI	1	Read "R" of CCHHR for feedback
14	29	Search key equal	Contents of DECKYADR	60	CC, SILI	Key length	Search for desired segment
15	08	TIC	CCW13	00			Repeat search if segment not found
16	06	Read data	Segment work area	20	SILI	8	Read block and segment descriptor words
17[1,2]	23	Set sector	Sector address 1	40	CC	1	Wait for record to come into position again
	03	NOP		60	CC, SILI	1	
18[2]	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for record just written
19[2]	08	TIC	CCW18	00			Repeat search if record not found
20[2]	0E	Read key and data		30	SILI, SKIP	256	Check data just written

[1] This CCW is set to NOP for devices without the rotational position sensing feature.

[2] CCWs 17–20 are not used in the first pass through this channel program.

B. Channel Program After Locating and Reading the Count Field of the First Segment to Determine Its Length

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1[1]	23	Set sector	Sector address 1	40	CC	1	Wait for desired record
	03	NOP		60	CC, SILI	1	
2	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for record found during last pass
3	08	TIC	CCW2	00			Repeat search if record not found
4	B1	TIC	CCW16	00			If record is found, go to write it
5	08	TIC	CCW7	00			
6	03	NOP		20	SILI	1	
7	29	Search key equal	Contents of DECKYADR	60	CC, SILI	Key length	
8	08	TIC	CCW4	00			
9[1]	22	Read sector	Sector address	40	CC	1	
	03	NOP		60	CC, SILI	1	
10[1]	23	Set sector	Sector=0	40	CC	1	
	03	NOP		60	CC, SILI	1	
11	1A	Read home address		70	CC, SILI, SKIP	1	
12[1]	23	Set sector	Sector address	40	CC	1	
	03	NOP		60	CC, SILI	1	
13	12	Read count	Segment work area	60	CC, SILI	8	
14	29	Search key equal	Contents of DECKYADR	40	CC	Key length	
15	08	TIC	CCW13	00			
16	05	Write data	Segment work area	00 40[2]	(CC)	Segment length	Write the updated record
17[1,2]	23	Set sector	Sector address 1	40	CC	1	Wait for the record to come into position again
	03	NOP		60	CC, SILI	1	
18[2]	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for an equal CCHHR
19[2]	08	TIC	CCW18	00			Repeat search if not equal
20[2]	0E	Read key and data		30	SILI, SKIP	256	Check record for errors

[1] This CCW is set to NOP for devices without the rotational position sensing feature.

[2] CCWs 17–20 are present and the command–chain bit in CCW16 is on only when the DCBOPTCD field of the DCB specifies the write–validity–check option.

The channel program now consists entirely of CCWs 1–4 and CCWs 16–20, which will be used to write all the segments. The channel command words were modified by the ASI routine in IGG019KJ.

10. Stages of the Channel Program to Read a Variable-Length Spanned Record (Type DI)

A. Original Channel Program as Developed by IGG019KR

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1[1,2]	23	Set sector	Sector=0	40	CC	1	Wait for beginning of track
	03	NOP		60	CC, SILI	1	
2[2]	31	Search ID equal	IOBUPLIM+3	40	CC	5	Search for capacity record
3[2]	08	TIC	CCW2	00			Repeat search if record not found
4[2]	06	Read data	IOBDNRCF+2	06	CC, SILI	5	Read data portion of R0 to find next address
5	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for specified record
6	08	TIC	CCW5	00			Repeat search if record not found
7[3]	0E	Read key and data	Contents of DECKYADR	80	DC	Key length	Read the key
8	06	Read data	Segment work area	00		Data length	Read the data

[1] This CCW is set to NOP for devices without the rotational position sensing feature

[2] CCWs 1–4 are present only when "next address" feedback is specified

[3] CCW7 is omitted if either the DCBKEYLE or DECKYADR field is zero.

B. Channel Program After Reading the First Segment

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1[1,2]	23	Set sector	Sector=0	40	CC	1	Wait for beginning of track
	03	NOP		60	CC, SILI	1	
2[1]	31	Search ID equal	IOBUPLIM+3	40	CC	5	Search for capacity record
3[1]	08	TIC	CCW2	00			Repeat search if record not found
4[1]	06	Read data	IOBDNRCF+2	60	CC, SILI	5	Read capacity record for "next address" feedback
5	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for next segment
6	08	TIC	CCW5	00			Repeat search if segment not found
7	06	Read data	Segment work area	00		Data length	Read the data
8	06	Read data	Segment work area	00		Data length	(This CCW not used)

[1] CCWs 1–4 are included only if "next address" feedback is specified.

[2] This CCW is set to NOP for devices without the rotational position sensing feature.

CCW8 has replaced CCW7 since there are no keys for subsequent segments. The channel program now effectively ends with CCW7. The change was made by the ASI routine in IGG019KJ.

11. Stages of the Channel Program to Write a Variable-Length Spanned Record (Type DI)

A. Original channel program as developed by IGG019KR. This channel program reads the segment descriptor word to determine the segment's length.

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1	03	NOP		60	CC, SILI	1	(Will be changed in later pass)
2	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for specified ID
3	08	TIC	CCW2	00			Repeat search if ID not found
4	06	Read data	Segment work area	20	SILI	8	Read 8 bytes from the data field
5	05	Write data	Segment work area	00 40	(CC)	Data length	These CCWs not used in this pass
6	22	Read sector	Sector address 2	40	CC	1	
	03	NOP		60	CC, SILI	1	
7	23	Set sector	Sector address 2	40	CC	1	
	03	NOP		60	CC, SILI	1	
8	31	Search ID equal	IOBSEEK+3	40	CC	5	
9	08	TIC	CCW8	00			
10	0E	Read key and data		60	SILI, SKIP	256	

B. Channel Program After Reading the Block Descriptor Word of the First Segment to Determine Its Length

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1	03	NOP		60	CC, SILI	1	(Will be changed in later pass)
2	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for first segment
3	08	TIC	CCW2	00			Repeat search if segment not found
4[1]	0D	Write key and data	Contents of DECKYADR	80	DC	Key length	Update the key
5	05	Write data	Segment work area	00 40[2]	(CC)	Data length	Update the data
6[2,3]	22	Read sector	Sector address 2	40	CC	1	Note position of this segment
	03	NOP		60	CC, SILI	1	
7[2,3]	23	Set sector	Sector address 2	40	CC	1	Wait for segment to come into position again
	03	NOP		60	CC, SILI	1	
8[2]	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for segment
9[2]	08	TIC	CCW8	00			Repeat search if segment not found
10[2]	0E	Read key and data		30	SILI, SKIP	256	Check the updated record for errors — write–validity–check

[1] CCW4 is omitted if either the DCBKEYLE OR DECKYADR field is 0.

[2] CCWs 6–10 are present, and the command–chain bit in CCW5 is on only when the DCBOPTCD field of the DCB specifies the write–validity–check option.

[3] This CCW is set to NOP for devices without the rotational position sensing feature.

C. Channel Program After Writing the First Segment

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1[1]	23	Set sector	Sector=0	40	CC	1	Wait for beginning of track
	03	NOP		60	CC, SILI	1	
2	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for segment ID
3	08	TIC	CCW2	00			Repeat search if ID not found
4	08	TIC	CCW5	00			When found, go to update segment
5	05	Write data	Segment work area	00 40[2]	(CC)	Data length	Update the segment
6[1,2]	22	Read sector	Sector address 2	40	CC	1	Note its sector value
	03	NOP		60	CC, SILI	1	
7[1,2]	23	Set sector	Sector address 2	40	CC	1	Wait for segment to come into position again
	03	NOP		60	CC, SILI	1	
8[2]	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for updated segment
9[2]	08	TIC	CCW8	00			Repeat search if segment not found
10[2]	0E	Read key and data		30	SILI, SKIP	256	Check the record for errors — write–validity–check

[1] This CCW is set to NOP for devices without the rotational position sensing feature.

[2] CCWs 6–10 are present and the command–chain bit in CCW5 is on only when the DCBOPTCD field of the DCB specifies the write–validity–check option.

CCW4 (when present) has changed to TIC to CCW5. This was the CCW to write the key, but keys are not necessary for subsequent segments. This change was made by the ASI routine in IGG019KJ.

12. *Stages of the Channel Program to Read a Variable-Length Spanned Record by Block Key (Type DK)*

A. Original Channel Program as Developed by IGG019KR

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1	12	Read count	IOBDNRCF+2	60	CC, SILI	5	Read CCHHR for feedback
2	29	Search key equal	Contents of DECKYADR	60	CC, SILI	Key length	Search for given key
3	08	TIC	CCW1	00			Update feedback and search again if key not found
4	06	Read data	Segment work area	00		Data length	Read the segment
5[1,2]	23	Set sector	Sector = 0	40	CC	1	Wait for beginning of track
	03	NOP		60	CC, SILI	1	
6[1]	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for capacity record
7[1]	08	TIC	CCW6	00			Repeat search if record not found
8[1]	06	Read data	Segment work area	60	CC, SILI	5	Read capacity record for "address" feedback
9[1]	03	NOP		20	SILI	1	

[1] CCWs 5–9 are present only when "next address" feedback is specified.

[2] This CCW is set to NOP for devices without the rotational position sensing feature.

CCWs 5–9 are not used in this pass.

B. Channel Program After Reading the First Segment

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1[1]	23	Set sector	Sector = 0	40	CC	1	Wait for beginning of track
	03	NOP		60	CC, SILI	1	
2	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for next segment
3	08	TIC	CCW2	00			Repeat search if segment not found
4	06	Read data	Segment work area	00		Data length	Read the segment
5[1,2]	23	Set sector	Sector = 0	40	CC	1	Wait for beginning of track to come into position again
	03	NOP		60	CC, SILI	1	
6[2]	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for track capacity record
7[2]	08	TIC	CCW6	00			Repeat search if record not found
8[2]	06	Read data	Segment work area	60	CC, SILI	5	Read the capacity record
9[2]	03	NOP		20	SILI	1	

[1] This CCW is set to NOP for devices without the rotational position sensing feature.

[2] CCWs 5–9 are present only when "next address" feedback is specified.

CCWs 5–9 are not used in this pass.

CCWs 1–3 of Part A have been changed to search on the block ID for a subsequent segment. The change was made by the ASI routine in IGG019KJ.

C. Channel Program After Reading the Last Segment

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
5	23	Set sector	Sector = 0	40	CC	1	Wait for beginning of track
	03	NOP		60	CC, SILI	1	
6	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for capacity record
7	08	TIC	CCW2	00			Repeat search if record not found
8	06	Read data	Segment work area	60	CC, SILI	5	Read the next address
9	03	NOP		20	SILI	1	(This command code serves as switch for ASI routine)

[1] This CCW is set to NOP for devices without the rotational position sensing feature.

This channel program is contained in the channel programs shown in parts A and B, but it was not executed with them. It is executed independently, after the last segment has been read, in order to provide the disk address of the next record when "next address" feedback has been requested in the READ macro instruction.

13. Stages of the Channel Program to Write A Variable-Length Spanned Record (Type DK)

A. Original Channel Program as Developed by IGG019KR

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1	12	Read count	IOBDNRCF+2	60	CC, SILI	5	Read CCHHR for internal feedback
2	29	Search key equal	Contents of DECKYADR	60	CC, SILI	Key length	Search for specified key
3	08	TIC	CCW1	00			Update feedback and search again if key not found
4	06	Read data	Segment work area	60	CC, SILI	8	Read segment descriptor words
5[1]	22	Read sector	Sector address 2	40	CC	1	Note segment's sector value
	03	NOP		20	SILI	1	
6	31	Search ID equal	IOBSEEK+3	40	CC	5	These CCWs not used in this pass
7	08	TIC	CCW6	00			
8	0E	Read key and data		30	SILI, SKIP	256	

[1] This CCW is set to NOP for devices without the rotational position sensing feature.

B. Channel Program After Reading the Count Field of the First Segment to Determine Its Length

CCW	Command Code			Flags			
No.	Hex	Description	Address	Hex	Description	Count	Comments
1[1]	23	Set sector	Sector address 1	40	CC	1	Wait for positioning found on last pass
	03	NOP		60	CC, SILI	1	
2	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for segment
3	08	TIC	CCW2	00			Repeat search if segment not found
4	05	Write data	Segment work area	00 40[2]	(CC)	Segment length	Update segment when found
5[1,2]	23	Set sector	Sector address 1	40	CC	1	Wait for updated segment to come into position again
	03	NOP		60	CC, SILI	1	
6[2]	31	Search ID equal	IOBSEEK+3	40	CC	5	Search for updated segment
7[2]	08	TIC	CCW6	00			Repeat search if segment not found
8[2]	0E	Read key and data		30	SILI, SKIP	256	Check the record for errors

[1] This CCW is set to NOP for devices without the rotational position sensing feature.

[2] CCWs 5–8 are present and the command–chain bit in CCW4 is on only when the DCBOPTCD field of the DCB specifies the write– validity–check option.

CCWs 1–4 of part A have been changed to write the updated segment and validity check it when necessary. All segments will now be written by searching on a known ID. The changes to the channel program were made by the ASI routine in IGG019KJ.

Appendix E: Messages and Codes Issued by BDAM Modules

Figure 36 directs you to the BDAM module that issued a particular completion code and/or message. Refer to *OS Messages and Codes* for the meaning of a specific message or completion code.

Completion Code	Message Number	BDAM Module Issuing Message or Code
013	IEC141I	IGG0193E
020		IGG0193A
001	IEC020I	IGG019LI
026		IGG019LG

Figure 36. BDAM modules issuing messages and codes

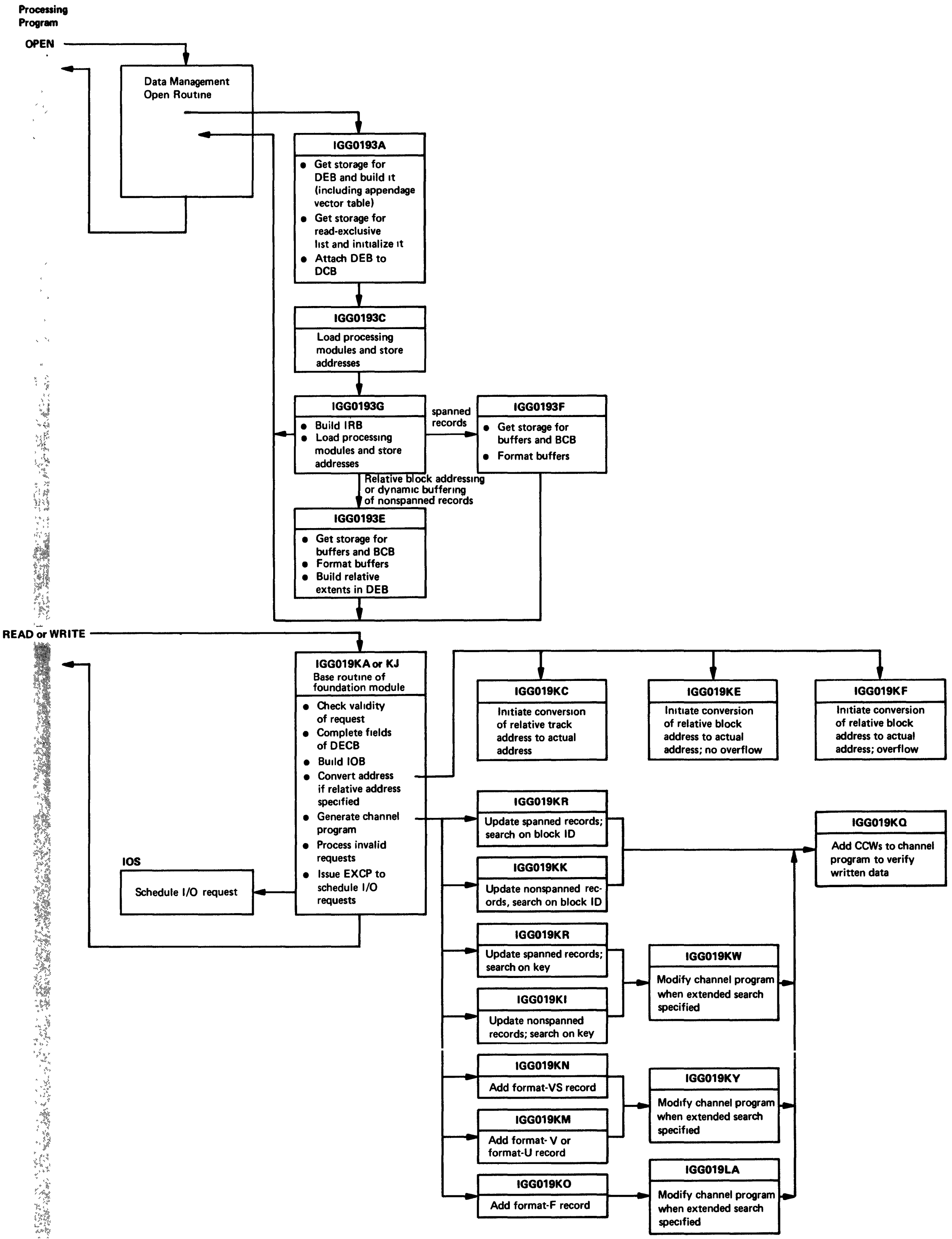
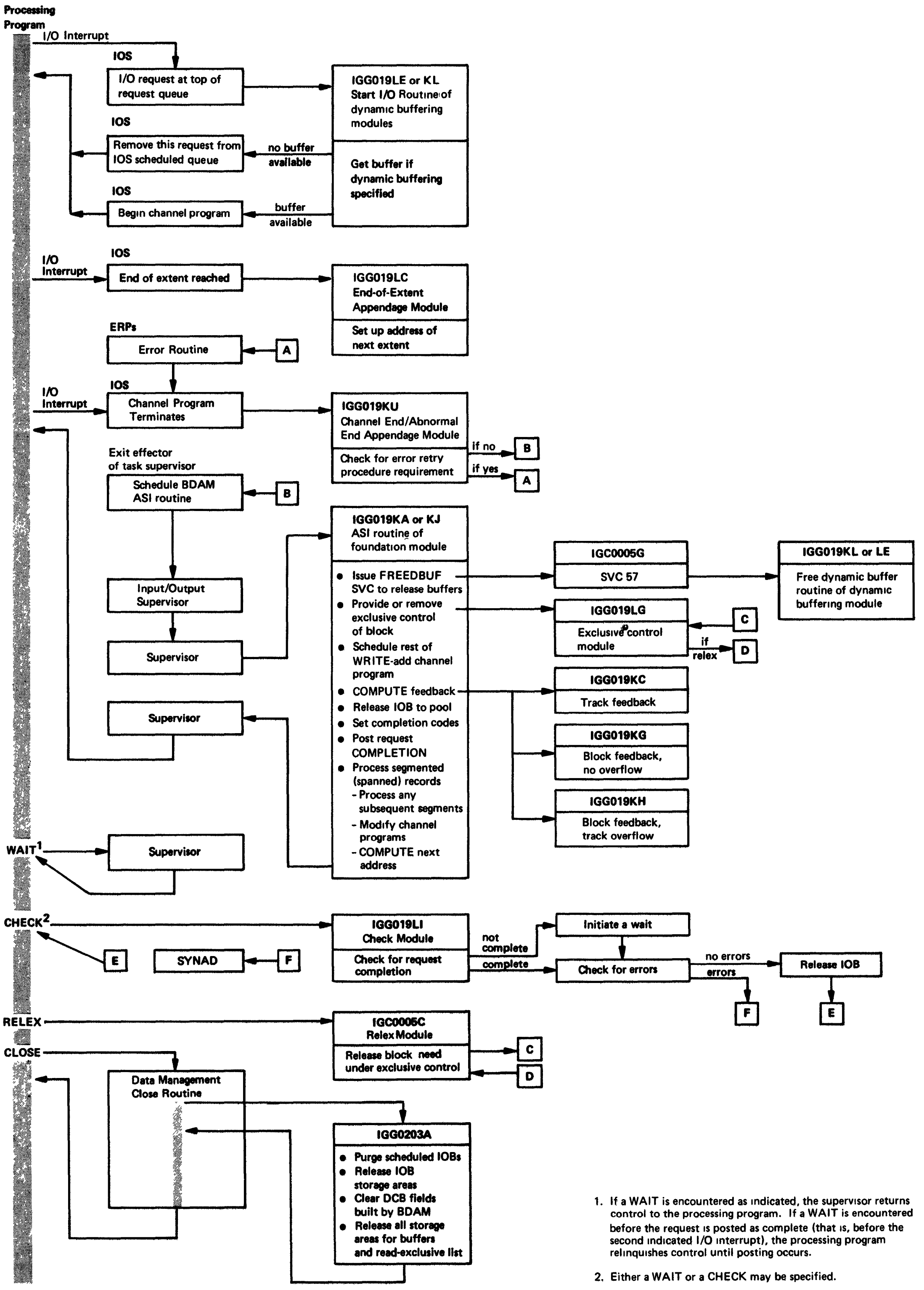

Processing Program
OPEN
Data Management Open Routine
IGG0193A
Get storage for DEB and build it (including appendage vector table)
Get storage for read-exclusive list and initialize it
Attach DEB to DCB
IGG0193C
Load processing modules and store addresses
IGG0193G
Build IRB
Load processing modules and store addresses
spanned records
IGG0193F
Get storage for buffers and BCB
Format buffers
Relative block addressing or dynamic buffering of nonspanned records
IGG0193E
Get storage for buffers and BCB
Format buffers
Build relative extents in DEB
READ or WRITE
IGG019KA or KJ
Base routine of foundation module
Check validity of request
Complete fields of DECB
Build IOB
Convert address if relative address specified
Generate channel program
Process invalid requests
Issue EXCP to schedule I/O requests
IOS
Schedule I/O request
IGG019KC
Initiate conversion of relative track address to actual address
IGG019KE
Initiate conversion of relative block address to actual address; no overflow
IGG019KF
Initiate conversion of relative block address to actual address; overflow
IGG019KR
Update spanned records; search on block ID
IGG019KK
Update nonspanned records, search on block ID
IGG019KQ
Add CCWs to channel program to verify written data
IGG019KR
Update spanned records; search on key
IGG019KI
Update nonspanned records; search on key
IGG019KW
Modify channel program when extended search specified
IGG019KN
Add format-VS record
IGG019KM
Add format-V or format-U record
IGG019KY
Modify channel program when extended search specified
IGG019KO
Add format-F record
IGG019LA
Modify channel program when extended search specified

Figure 37. Relationship among processing program, BDAM routines, and other parts of the operating system (Part 1 of 2)


Processing Program
I/O Interrupt
IOS
I/O request at top of request queue
IGG019LE or KL Start I/O Routine of dynamic buffering modules
IOS
Remove this request from IOS scheduled queue
no buffer available
Get buffer if dynamic buffering specified
IOS
Begin channel program
buffer available
I/O Interrupt
IOS
End of extent reached
IGG019LC End-of-Extent Appendage Module
Set up address of next extent
ERPs
Error Routine
A
I/O Interrupt
IOS
Channel Program Terminates
IGG019KU Channel End/Abnormal End Appendage Module
Check for error retry procedure requirement
if no
B
if yes
A
Exit effector of task supervisor
Schedule BDAM ASI routine
B
IGG019KA or KJ ASI routine of foundation module
Issue FREEDBUF SVC to release buffers
Provide or remove exclusive control of block
Schedule rest of WRITE-add channel program
COMPUTE feedback
Release IOB to pool
Set completion codes
Post request COMPLETION
Process segmented (spanned) records
- Process any subsequent segments
- Modify channel programs
- COMPUTE next address
Input/Output Supervisor
Supervisor
Supervisor
IGC0005G
SVC 57
IGG019KL or LE
Free dynamic buffer routine of dynamic buffering module
IGG019LG
Exclusive control module
C
if relex
D
IGG019KC
Track feedback
IGG019KG
Block feedback, no overflow
IGG019KH
Block feedback, track overflow
WAIT1
Supervisor
CHECK2
E
SYNAD
F
IGG019LI Check Module
Check for request completion
not complete
complete
Initiate a wait
Check for errors
no errors
errors
Release IOB
F
E
RELEX
IGC0005C Relex Module
Release block need under exclusive control
C
D
CLOSE
Data Management Close Routine
IGG0203A
Purge scheduled IOBs
Release IOB storage areas
Clear DCB fields built by BDAM
Release all storage areas for buffers and read-exclusive list

1. If a WAIT is encountered as indicated, the supervisor returns control to the processing program. If a WAIT is encountered before the request is posted as complete (that is, before the second indicated I/O interrupt), the processing program relinquishes control until posting occurs.

2. Either a WAIT or a CHECK may be specified.

Figure 37. Relationship among processing program, BDAM routines, and other parts of the operating system (Part 2 of 2)

GLOSSARY

The following terms are defined as they are used in this book. If you do not find the term you are looking for, refer to the index or to the *IBM Data Processing Glossary,* GC20–1699.

actual address: A pattern of characters that, without further modification, identifies a unique storage location.

actual extent: An area in the DEB containing data that describes the space occupied by an extent of a data set. BDAM module IGG0193A builds one actual extent for each extent in the data set.

BLKREF field: A field the user specifies in his program into which is put the data in the block address operand of the READ macro instruction. This data is either the relative or the actual address of the record the user wants access to. If it is the relative address, the BDAM address conversion routines convert it to an actual address (MBBCCHHR). Then the actual address is placed in the IOBSEEK field of the IOB so that the channel program can use the address to find a block.

buffer pool: A continuous area of main storage divided into buffers.

capacity record: The first block (block 0) on each track of a data set. It contains the ID of the last block on the track and the number of usable bytes remaining on the track.

dummy record: A record, created when BSAM builds a BDAM data set, whose purpose is to provide space in which new records can be added to the data set after it is created. The first byte in the key field of the dummy record contains X‘FF,’ and the first byte in the data field has a value indicating the position of the dummy record on the track (the R in MBBCCHHR).

extent: A continuous area of space on a direct–access device occupied by or reserved for a data set or part of a data set.

IOB buffer queue: The addresses of IOBs for requests for which a buffer is not available. The BCB contains the addresses of the first and last IOB in this chain and the IOBDQPTR field in each IOB in the chain contains the address of the next IOB.

period: A group of tracks in which the first track does not begin with an overflow block and the last track does not contain a block that overflows to another track.

processing program: Any program that is not a control program; synonymous with problem program.

read–exclusive list: An area of main storage containing the UCB address and actual address of blocks requiring exclusive control. This list is described in detail in the “Data Areas” section of this manual.

relative address: The position of a block in a data set relative to the first block of a data set. The relative address can be a relative track number or relative block number. See “relative track address” and “relative block address.”

relative block address: A 3–byte binary number that indicates the position of a block in relation to the first block of a data set. The first block of a data set always has a relative block address of 0.

relative extent: An area in the DEB containing either the number of blocks in each extent (if track overflow is in effect) or the number of blocks in each track (if track overflow is not in effect) of a data set. Module IGG0193E builds the relative extent area when relative block addressing is specified in the processing program.

relative track address: A 3–byte binary number in the form TTR where:

TT is the position of the track relative to the first track of a data set. The first track has a relative position of 0.

R is the number of the block relative to the first block on the track TT. The first block of data on a track has a relative value of 1.

search argument: The field of a block that contains data identifying the block as unique from any other block in the data set. Can be either the key field or the block ID in the count field.

search limit: The track following the last track that should actually be searched in a data set. The search limit is calculated and put in the IOBUPLIM field of the IOB when the request macro instruction specifies the extended search option for a block.

subroutine identification: The 2 low–order bytes of each module’s unique 8–byte name.

unposted queue: A queue of IOBs for requests for blocks whose addresses are currently on the read–exclusive list. The unposted queue contains only IOBs for the current task.

INDEX

Indexes to OS logic manuals are consolidated in the *OS Master Index to Logic Manuals,* GY28–6717. For additional information about any subject listed in this index, refer to other publications listed for the same subject in the *Master Index.*

READER'S COMMENT FORM

OS BDAM Logic

Order Number GY28-6617-6

Your comments about this publication will help us to produce better publications for your use. If you wish to comment, please use the space provided below, giving specific page and paragraph references.

Please do not use this form to ask technical questions about the system or equipment or to make requests for copies of publications. Instead, make such inquiries or requests to your IBM representative or to the IBM Branch Office serving your locality.

Reply requested

Yes □

No □

Name ______________________________

Job Title ______________________________

Address ______________________________

______________________________ Zip ______________

No postage necessary if mailed in the U S A

Order Number GY28-6617-6

YOUR COMMENTS, PLEASE . . .

This publication is one of a series which serves as a reference source for systems analysts, programmers, and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

fold fold

FIRST CLASS
PERMIT NO. 2078
SAN JOSE, CALIF.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY . . .

IBM Corporation
Monterey & Cottle Rds.
San Jose, California
95114

Attention: Programming Publications, Dept. D78

fold fold

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)

Order Number GY28-6617-6

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)