File No. S360-30
Form Y28-6617-3

Program Logic

IBM System/360 Operating System
Basic Direct Access Method
Program Logic Manual

Program Number 360S-DM-509

This publication describes the internal logic of the IBM System/360 Operating System basic direct access method (BDAM). The functions and structures of the routines are described, as are their relationships to other portions of the operating system.

The manual is intended for use by IBM customer engineers involved in program maintenance, and system programmers who are altering the program design. It can be used to locate specific areas of the program, and it enables the reader to relate these areas to the corresponding program listings. Program logic information is not necessary for the use and operation of the program; therefore, distribution of this publication is limited to those with the aforementioned requirements.

Fourth Edition (July 1967)

This publication corresponds to Release 12.

This publication is a major revision of Form Y28-6617-2 and makes that form and prior forms obsolete. In addition to the reorganization of some parts of the text, new material has been added to describe the enqueuing and dequeuing of requests that require either the read-exclusive option or the adding of new blocks of records of variable length or undefined length. Two figures have been added to Appendix A to describe the read-exclusive list. Figures 7 and 16 and Chart 02 have been modified, and Chart 05 has been deleted. Chart 04 has been replaced with a new chart for exclusive control under the MVT form of the operating system.

Changes (including deletions) in the text are indicated by a vertical line to the left of the change. Changes in illustrations are denoted by the symbol • to the left of the caption.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print chain.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N. Y. 12602

The information contained in this manual is intended for programmers engaged in maintenance of BDAM routines.

This publication is divided into three main parts. The first part describes the organization and function of the basic direct access method and its relationship to other portions of IBM System/360 Operating System. The second part describes the main components of the basic direct access method and their interaction. Part three contains reference material that is not necessary to an understanding of the logic of the access method but may be useful in understanding a storage dump or in otherwise analyzing the listings for this access method.

To provide the prerequisite knowledge for understanding the contents of this publication, the following publications are recommended:

For information regarding the primary control program, see:

<u>IBM System/360 Operating System: Introduction to Control Program Logic, Program Logic Manual</u>, Form Y28-6605

For information regarding the MVT configuration of the control program, see:

<u>IBM System/360 Operating System: Control Program Logic Summary</u>, Form Y28-6658

The following publications are listed as suggested reading:

<u>IBM System/360 Operating System: Supervisor and Data Management Services</u>, Form C28-6646

<u>IBM System/360 Operating System: Supervisor and Data Management Macro-Instructions</u>, Form C28-6647

<u>IBM System/360: Component Description,</u>

<u>2841 Storage Control Unit</u>

<u>2302 Disk Storage, Models 3 and 4</u>

<u>2311 Disk Storage Drive</u>

<u>2321 Data Cell Drive, Model 1</u>

<u>7320 Drum Storage</u>, Form A26-5988

This publication also makes references to routines that are described in one of the following publications:

<u>IBM System/360 Operating System: Input/Output Support (OPEN/CLOSE/EOV), Program Logic Manual</u>, Form Y28-6609

<u>IBM System/360 Operating System: Sequential Access Methods, Program Logic Manual</u>, Form Y28-6604

<u>IBM System/360 Operating System: Direct Access Device Space Management, Program Logic Manual</u>, Form Y28-6607

<u>IBM System/360 Operating System: MVT Supervisor, Program Logic Manual</u>, Form Y28-6659

ILLUSTRATIONS

FIGURES

TABLES

CHARTS

A listing of text page references for Figures, Tables, and Charts follows the index.

The basic direct access method (BDAM) consists of routines used in retrieving data from, and storing data onto, direct access devices. In this capacity, the BDAM routines are a part of the IBM System/360 Operating System control program, and they operate in conjunction with the input/output supervisor (I/O supervisor). The BDAM routines link a processing program to system supervisor routines in order to satisfy input/output requests for data in direct-organization data sets.

The BDAM routines are grouped into modules and placed in the supervisor call (SVC) library at system generation time. This library is part of the system residence library that resides on direct-access storage. When the BDAM routines are to be used by a processing program, the necessary modules are brought from the system residence volume and loaded into main storage. The loaded BDAM modules are linked by module storage addresses. Each address is placed in one of the following:

- The data control block (DCB) for the data set.
- Another previously loaded BDAM module.
- The BDAM appendage list that is in protected main storage.

Essentially, BDAM may be divided into three general sections: an opening section, a processing section, and a closing section.

The opening section initializes the access method by determining storage requirements, by determining and loading the required modules from system residence, and by building control blocks and control lists for further program usage.

The processing section does the following:

- Converts address to a form required by a channel program.
- Constructs, and initiates execution of, a channel program for fulfilling an input or an output request macro instruction.
- Maintains exclusive control of information in a block as long as requested.
- Assigns buffers for input/output operations.
- Indicates errors related to input/output operations.

The closing section performs the functions that are necessary to terminate normal processing of a BDAM data set.

Throughout this publication, references are made to control blocks or to fields of a control block. Appendix A contains a description of the major control blocks used by BDAM.

RELATIONSHIP OF THE BASIC DIRECT ACCESS METHOD TO THE OPERATING SYSTEM

When a data set to be processed using BDAM is opened, the open routine of data management (discussed in the publication IBM System/360 Operating System: Input/Output Support (OPEN/CLOSE/EOV), Program Logic Manual) gives control to one of the BDAM modules that initialize the access method routines for the type of processing indicated in the DCB.

The basic direct access method also has interfaces with the processing program and with the supervisor. When either a READ or a WRITE macro instruction in the processing program is encountered, the BDAM routines begin satisfying the request. The actual execution of a request requires a channel program that has been constructed by one or more BDAM routines.

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

Note: The BDAM start I/O appendage module is always given control just before the execution of a request begins. If dynamic buffering is specified, control is given to the BDAM dynamic buffering module.

When a data set is closed, the data management close routine gives control to the BDAM close executor module, which then completes the BDAM functions.

STRUCTURE OF THE BASIC DIRECT ACCESS METHOD

The modules of BDAM can be grouped into several categories that are related to the purpose or function of the module. These categories are:

- Opening a DCB.
- Controlling the processing.
- Converting addresses.
- Generating channel programs.
- I/O supervisor appendages.
- Maintaining exclusive control.
- Providing dynamic buffer allocation.
- Checking for request completion.
- Closing the DCB.

OPENING A DCB

To open a DCB for a BDAM application, two, and sometimes three, modules are required. These modules are the BDAM open executor modules, and they are used when a DCB for a direct organization data set is opened. The need for three modules depends on the options selected in the DCB macro instruction. The collective functions of these modules are to determine the need for other modules and to establish control blocks.

CONTROLLING THE PROCESSING

The module that controls the processing functions of BDAM is called the foundation module. This module is given control when either a READ or a WRITE macro instruction that uses BDAM is encountered in a processing program. It is used to complete the preparations necessary before a request can be executed and completed. The foundation module checks the validity of each request and contains the linkage to some of the other required modules.

An additional function of the foundation module is to complete the processing of a request after an input/output operation has terminated. (Refer to the section "Asynchronous Interrupt Component" for further information about this function.)

CONVERTING ADDRESSES

Five modules are used for address conversion. Two of these modules are used to convert a block address that has been specified as a relative block address into an actual device address. The choice of which one is to be used depends on whether or not track overflow is specified in the DCB macro instruction. Another of the five modules converts addresses from a relative track specification to an actual device address. The other two conversion modules are used only if the programmer specifies feedback with relative block addressing. Again, the choice depends on the track overflow specification.

GENERATING CHANNEL PROGRAMS

Several channel program generation modules are available as part of BDAM. At least one of these modules is used for every BDAM request that is normally completed. These channel programs either search auxiliary storage volumes for information to be brought into main storage or search auxiliary storage volumes for space on which to place information that is transferred from main storage. The selection of the required module is based on whether an existing block is being read or updated or a new block is being added to a data set. One additional channel program is used if it is required to verify data that is written on secondary storage volumes.

Reading or Updating Blocks

There are three modules used to generate channel programs for block reading or updating purposes. The choice of modules depends both on which part (either the key field or the block identification portion of the count field) of the data block is used as a search argument in the data retrieving function of the channel program and on whether an option has been selected to permit extending the search of a data set beyond a given track.

Adding New Blocks

There are three modules available to generate channel programs for adding new blocks to an existing data set. These modules are referred to as format modules since the use of a particular module depends on the block format in the data set being processed. The modules used for fixed-length (pre-format) blocks are different from those used for variable-length (self-format) blocks. Blocks whose lengths are undefined also are called self-format blocks. The number of modules required when adding new blocks depends on whether the extended search option has been specified.

Verifying Written Data

If written data is to be verified, there is one module used to generate the required channel command words. This channel program contains channel command words that are appended to the appropriate "write-type" channel program. The module is required if the write-validity-check option has been specified in the DCB.

I/O SUPERVISOR APPENDAGES

There are three BDAM modules to which I/O supervisor may pass control, depending on the stage of execution of a request.

The first is the start I/O appendage module. If the dynamic buffering option has been specified, the appendage module permits the allocation or release of buffer areas. I/O supervisor accordingly passes control to this module before beginning the channel program for such requests.

The second appendage module is the channel end appendage module. This module schedules further processing on a request after a channel program has terminated.

The third I/O supervisor appendage module is the end of extent appendage module. This module is loaded if the extended search option has been specified. It receives control if the channel program in control is required to switch from one extent to another while reading a block, writing an updated block, or adding a new block.

MAINTAINING EXCLUSIVE CONTROL

The exclusive control module provides protection for the data portion of a block requested under exclusive control by ensuring that subsequent duplicate requests for the block are not posted as complete before the initial request for the block has released control. A read-exclusive list is established to help ascertain if a given request is a duplicate request.

In a multi-task environment, this module places blocks on, and removes blocks from, an inter-task queue to provide block protection during updating. In this capacity, the module serves requests for adding new blocks as well as requests using the exclusive control option.

PROVIDING DYNAMIC BUFFER ALLOCATION

The dynamic buffer module obtains, assigns, and releases buffer areas for input/output requests that use the dynamic buffer option. A buffer control block and a buffer queue are established to handle the buffer requests.

CHECKING FOR REQUEST COMPLETION

The check module contains provisions both for determining if (and waiting, if necessary, until) a request has been posted as complete and for giving control to a user's error routine if errors have been indicated during the input or output operation.

CLOSING THE DCB

The BDAM close executor module is the only module in this category. This module is required to release, to the system, the main storage obtained by BDAM.

The second function of this module is to restore the fields of the DCB that have been changed by BDAM routines.

PROGRAM COMPONENT DESCRIPTION

The main components of BDAM are described in the following paragraphs. The modules required within each component, the conditions under which the modules are used, and the inter-module relationships, are discussed. Where appropriate, figures are used to illustrate the text description. Figure 16, following Appendix C, is a composite of Figures 1, 3, 4, 7, and 8 that are referred to in this section of the publication. It may be useful in gaining an overall general concept of BDAM and its relation to a processing program and to the operating system.

THE BDAM OPEN EXECUTOR PROGRAM (MODULES IGG0193A, IGG0193C, AND IGG0193E)

The BDAM open executor program consists of up to three modules that are given control during the opening of a data control block specifying BDAM. The routines in these modules obtain storage for, set up, and initialize control blocks used by BDAM routines, and load the required BDAM processing modules into main storage.

When the OPEN macro instruction is encountered in a processing program that specifies BDAM, the expansion of the macro instruction causes control to be passed to the open routine of data management. (See Figure 1.) This routine uses the BDAM open executor modules as subroutines. It gives control to module IGG0193A (referred to as phase 1 of the BDAM open executor program) to initiate the BDAM processing. Parameters specified in the DCB to which the OPEN macro instruction refers and in the data set control block (DSCB) for this data set enable phase 1 to determine the protected main storage requirements for the data extents to be added to the base of a data extent block (DEB). Phase 1 then obtains main storage for an I/O supervisor appendage list, a read-exclusive list (a list of the addresses of blocks still under exclusive control), and the DEB, and places necessary control information in the fields of the DEB. As each current DEB is being constructed, it is attached to the appropriate task control block for future reference.

In allocating storage for the DEB, phase 1 also provides space for the identification of BDAM modules that are required as a result of specifications given in the DCBMACRF field, the DCBOPTCD field, and the DCBRECFM field of the DCB.

After the preceding functions have been completed, phase 1 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 phase. If there are no more current requirements for the use of phase 1, control is given to module IGG0193C (referred to as phase 2 of the BDAM executor program).

Phase 2 loads into main storage the BDAM modules used for the particular application. The foundation module is loaded first. Phase 2 places in the foundation module the addresses of certain optional BDAM modules that have been selected. The addresses of the remaining BDAM modules are placed either in the DCB, in the I/O supervisor appendage list, or in other designated modules. Table 1 shows where the module addresses are placed. Each time BDAM is used, the addresses of the selected modules are placed in the same designated positions in the control blocks, modules, or lists.

The major activity of phase 2 is the selecting of required modules. There is a routine within the module for each of the following activities:

- Selecting and loading the proper addressing module.
- Determining and loading the proper channel program generating module(s).
- Determining and loading the optional module(s) required.
- Determining and loading the required appendage modules.

If main storage already contains some BDAM modules from a previous data set opening, phase 2 obtains their main storage addresses from the supervisor, and these modules are not reloaded. There are two situations to consider:

1. If BDAM modules have been included in the link pack area, then they are accessible to all jobs requesting them (since the BDAM modules are reenterable).
2. If a given task loads a BDAM module into a region assigned to that task, only that task or a subtask attached by that task has access to the module. If another main task requires the same

module, it must load the module into its own region.

The MVT Supervisor Program Logic Manual contains more detailed information concerning these situations.

As modules are loaded into main storage, their corresponding identifications are placed in the DEB. (After processing has been completed on a data set, the close routine uses this information for releasing storage areas.) Space for the module identifications was allotted by phase 1 of the BDAM open executors.

Phase 2 also initializes the read-exclusive list and some of the fields of the DCB.

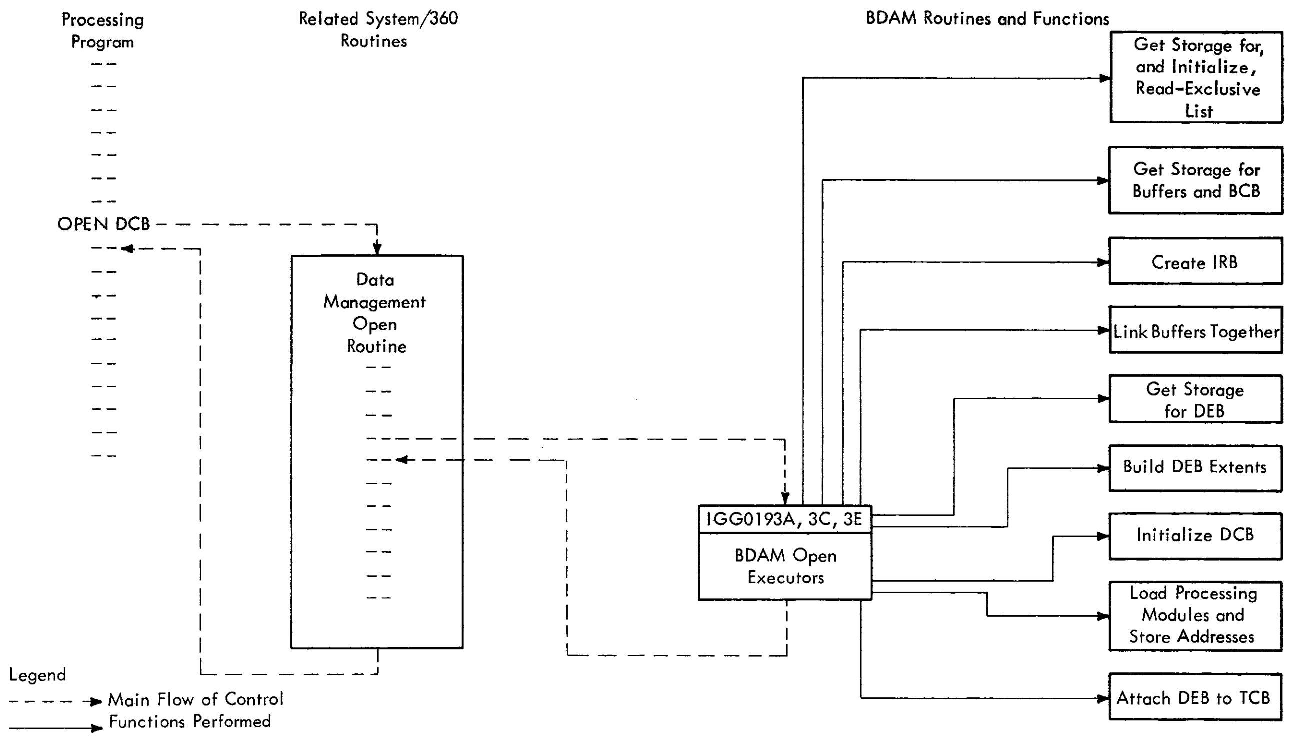

Processing Program
Related System/360 Routines
BDAM Routines and Functions
OPEN DCB
Data Management Open Routine
IGG0193A, 3C, 3E
BDAM Open Executors
Get Storage for, and Initialize, Read-Exclusive List
Get Storage for Buffers and BCB
Create IRB
Link Buffers Together
Get Storage for DEB
Build DEB Extents
Initialize DCB
Load Processing Modules and Store Addresses
Attach DEB to TCB
Legend
Main Flow of Control
Functions Performed

Figure 1. Relationship Among Processing Program, Data Management Open Routine, and BDAM at Open Time

Table 1. BDAM Module Addresses as Stored by Phase 2 of BDAM Open Executor Program

Module Name	Module, Control Block, or List in Which Address Is Placed
Foundation	DCB
Relative Track	Foundation Module
Relative Block	Foundation Module
Write-Verify	Foundation Module
Relative Block Feedback	DCB
Key	Foundation Module
Key Extended Search	Key Module
ID	Foundation Module
Self-Format	DCB
Self-Format Extended Search	Self-Format Module
Pre-Format	DCB
Pre-Format Extended Search	Pre-Format Module
Start I/O	Appendage List[1]
End of Extent	Appendage List
Channel End	Appendage List
Check	DCB
Dynamic Buffer	DCB
Exclusive Control[2]	DCB

[1]The address of the Appendage List is in the DEB.
[2]The address of the read-exclusive list is also in the DCB.

The remaining function of phase 2 of the open executor program is the branching to a supervisor routine to build the interrupt request block (IRB).

Before giving control to the next routine, phase 2 determines 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. Otherwise, two situations are considered:

1. If neither the dynamic buffer option nor the relative block addressing option has been specified in the DCB macro instruction, the data set is considered to be opened as far as BDAM is concerned, and control is returned either to an open executor routine for another data set or to the open routine of input/output support[1].

2. If the DCB macro instruction specifies either the relative block addressing option or the dynamic buffering option, or specifies both options, phase 2 gives control to module IGG0193E (phase 3 of the BDAM open executor program).

If it is indicated in the DCBOPTCD field of the data control block that BDAM should handle all buffer management for a given data set, module IGG0193E uses the buffer information in the DCB macro instruction to obtain the required amount of main storage for the buffers and a control block (the buffer control block) to contain buffer information. The buffer area in storage is then divided into the requested number of buffers, and the buffers are chained together so that the dynamic buffer module may satisfy the buffer requirements for individual read or write requests. (Refer to "Dynamic Buffering.")

To gain access to a data set block located on a direct-access device, the block's actual location on the device must be known. If a programmer has specified that the blocks in a data set are to be referred to by relative block number (i.e., relative block addressing), then these numbers must be converted to actual block locations (i.e., device addresses). For use in this conversion, phase 3 of the BDAM open executor program constructs a set of fields (known as relative extent fields) in the DEB. These fields of the DEB are described in Appendix A. There is a one-to-one correspondence between the actual data set extents in the DEB and the relative data set extents. (Refer to "Relative Block Conversion.")

If relative block addressing and track overflow are indicated in the DCBOPTCD and DCBRECFM fields of the DCB respectively, phase 3 constructs a modified relative extent portion of the DEB. In addition, phase 3 inserts two other 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 are related to the concept of a period as discussed in the following paragraphs.

Periods of an Extent

When the basic sequential access method (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, and for this to occur, at least one byte of the data portion of a block must fit on a track in order for the block to overflow onto the next track. For purposes of calculation, the track on which a block begins is considered to contain the block. When a track is reached in which block length and track conditions do not permit at least one byte of the data portion of a new block to be written, the end of a period has been reached.

Thus, a period constitutes that group of tracks containing a group of blocks such that the first track does not begin with an overflow block from another track 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

---
[1]The next routine is determined from the where-to-go table. (Refer to the publication IBM System/360 Operating System: Input/Output Support (OPEN/CLOSE/EOV), Program Logic Manual.)

- There are no inter-record gaps on the tracks. (This assumption is to simplify the calculations in the example.)

- Part of the given data set occupies an extent consisting of contiguous tracks beginning with track 47 on this device.

From the second assumption, it is determined that at least the first 115 bytes of a block must fit on a track in order for track overflow to occur. Figure 2 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 block identifications 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 2, track overflow occurs from track 47 to track 48 (via block B5), from track 48 to track 49 (via block B9), and so on till 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, track overflow cannot occur using block B31. 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 the section "Relative Block Conversion"), the number of blocks on each track is given in Figure 2 as the 'Track Block Count'.

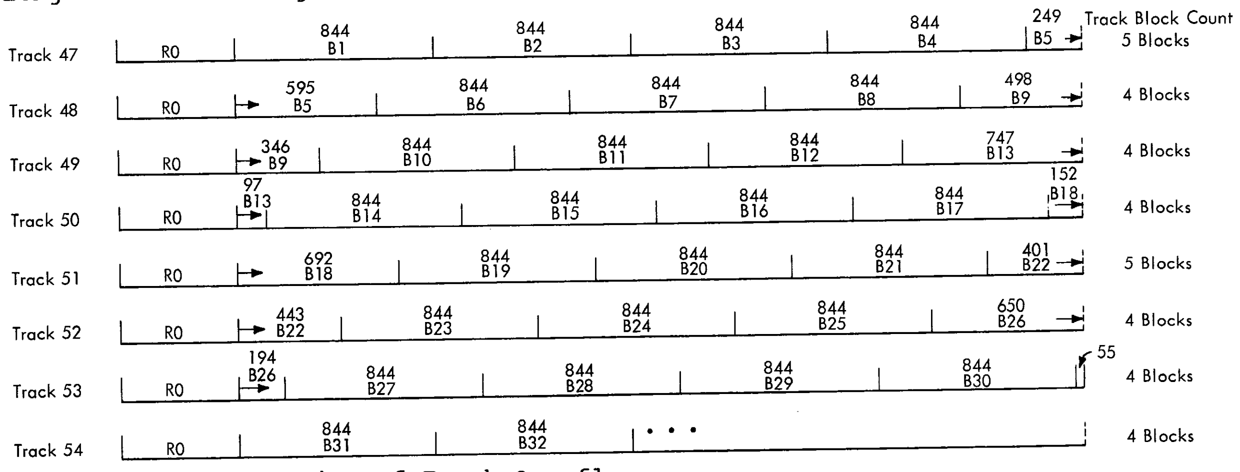
Based on block characteristics (key length and data length) and device characteristics, the third phase of the BDAM open program computes the size of the period for the data set. Phase 3 then computes both the number of blocks in a period and the number of tracks in a period and places the computed values in the two fields of the overflow section of the DEB. These fields are each one word in length, and they occur only once for a given data set. The values placed in these fields are constant for a given data set.

Since the allocation of actual extents to members of a data set is performed by space management routines (refer to the publication IBM System/360 Operating System: Direct Access Device Space Management, Program Logic Manual), and since the period is a concept used by BDAM, the boundaries of extents and periods may not always coincide. 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.

Before returning control to the next routine, phase 3 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 considered to be opened as far as BDAM is concerned, and control is returned either to an open executor routine for another data set or to the open routine of input/output support.[1]

---

[1]The next routine is determined from the where-to-go table. (Refer to the publication IBM System/360 Operating System: Input/Output Support (OPEN/CLOSE/EOV), Program Logic Manual.)


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 2. Illustration of Track Overflow

THE BDAM FOUNDATION MODULE (MODULE IGG019KA)

When either a READ or a WRITE macro instruction in the processing program is encountered, the BDAM foundation module receives control. (See Figure 3.) This module, IGG019KA, is the basic module of BDAM. As such, it is required in all processing that requires the basic direct access method. This module provides the initialization, housekeeping, and controlling functions for the BDAM processing routines. The foundation module consists of three main functional components:

- The base component.
- The asynchronous interrupt component.
- The error component.

BASE COMPONENT

The base routine is the first main component of the foundation module. All requests made through either a READ or a WRITE macro instruction enter BDAM in the base routine. (See Chart 01.) The base routine has two primary functions:

- Establishing the validity of options that have been specified in the request macro instruction for the application.

- Combining the options specified in the type field of the READ or the WRITE macro instruction with the options specified in the DCBOPTCD field of the DCB. The result of the combination is placed in the data event control block (DECB) and later transferred to the input/output block (IOB) for future reference by the BDAM routines.

The base routine then tests the type field of the DECB that results from the expansion of either the READ or the WRITE macro instruction, to determine the form of the channel program to be constructed. This routine then determines the main storage requirements for the IOB to be used by the request. (The IOB used by a request for the BDAM program is described in Appendix A.) The base routine then either (a) obtains an available IOB of the necessary size from an existing pool of IOBs or, (b), obtains an amount of main storage, through the use of the GETMAIN macro instruction, in which to construct a new IOB for the request and builds the IOB.

The base routine next determines the type of block addressing that has been specified for the data set. If actual addressing has not been specified, control is given to one of the BDAM modules that are used to convert the address form used to an actual address. (The main storage address of the required address conversion module was placed in the foundation module by the BDAM open executor program when the data set was opened.) The conversion module returns control to the base routine.

The base routine again determines the form of channel program required, this time by checking fields in the IOB and in the DCB. The indicated module is given control and generates the proper channel program. If the extended search option has been specified, the channel program will reflect the search limits established in the DCBLIMCT field of the DCB. At the completion of channel program generation, the base routine again receives control.

After establishing an expected end for the channel program, the base routine issues a request to the I/O supervisor by means of the execute channel program (EXCP) routine. After the I/O supervisor has scheduled the request, control is returned to the base routine, which then returns control to the processing program.

ASYNCHRONOUS INTERRUPT COMPONENT

The asynchronous interrupt (ASI) routine is the second main component of the foundation module. Before a request is considered completed, certain processing functions must be performed by the foundation module. When the processing program is interrupted by the completion of an input/output operation, the I/O supervisor gives program control to the BDAM channel end appendage routine. The control relationships involved at this time are shown in Figure 4. The ASI routine is scheduled by the supervisor after the BDAM channel end appendage module, IGG019KU, has indicated the need for the ASI routine. The appendage module then returns control to the I/O supervisor which, in turn, returns control to the supervisor.

The main functions (see Chart 02) of the ASI routines are:

- To determine the cause of interruptions.

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

- If the request currently being processed is a request to write a block and if the dynamic buffer option has been specified, then the ASI routine will use the dynamic buffer module as a subroutine to free the buffer that has been allocated for the corresponding 'read' request.
- If either the exclusive control option has been specified or a request to add new blocks of variable or undefined length has been specified, the ASI routine uses the exclusive control module either to place blocks on a queue or to remove blocks from a queue.
- To release the request's IOB to the pool of IOBs. (This function is performed by the check module if both a CHECK macro instruction is encountered and the DCB macro instruction specifies the check function.) After the IOB has been released, the request is posted as complete by means of the post routine, and control is returned to the supervisor.

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

ERROR COMPONENT

The third major component of the foundation module consists of error routines both for processing invalid requests and for processing errors resulting from abnormal completion of a request. Figures 3 and 4 indicate the error-processing functions.

Invalid Requests

Invalid requests are based upon differences between the parameters specified in the DECB related to an individual READ or WRITE macro instruction and the parameters specified in the DCB at the time it was opened for processing with BDAM. Invalid requests can occur in the base routine of the foundation module, in the module for converting a relative track address to an actual address, in the module for converting a relative block address to an actual address, and in the module for generating a write-type channel program to add a block to a data set of fixed-length (or preformatted) records.

When an invalid request is encountered, an error routine releases the IOB associated with the request, posts an indication in the DECB that an invalid request has occurred, and returns control to the processing program.

Abnormal Completion of a Request

There are two situations in which abnormal completion of a request is related to BDAM. The first situation results from device errors. Errors included in this category are those relating either to the actual input/output devices and control units or to end-of-data-set conditions (which are received by the channel end appendage module as unit exception conditions). An error condition also results from not finding a dummy record in the case of a request to add a fixed-length block to an existing data set. An indication of abnormal completion is established when the I/O supervisor enters the exceptional end routine of the BDAM channel end appendage module, IGG019KU.

The second situation can occur when a request is given 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 an abnormal completion is encountered, the error routine releases the related IOB to the IOB pool associated with the data control block, posts an indication of the type of error and an indication of the completion of the request, and returns control to the supervisor.

Note: The IOB is released to the IOB pool by the check module if applicable. (Refer to the section "Asynchronous Interrupt Component.")

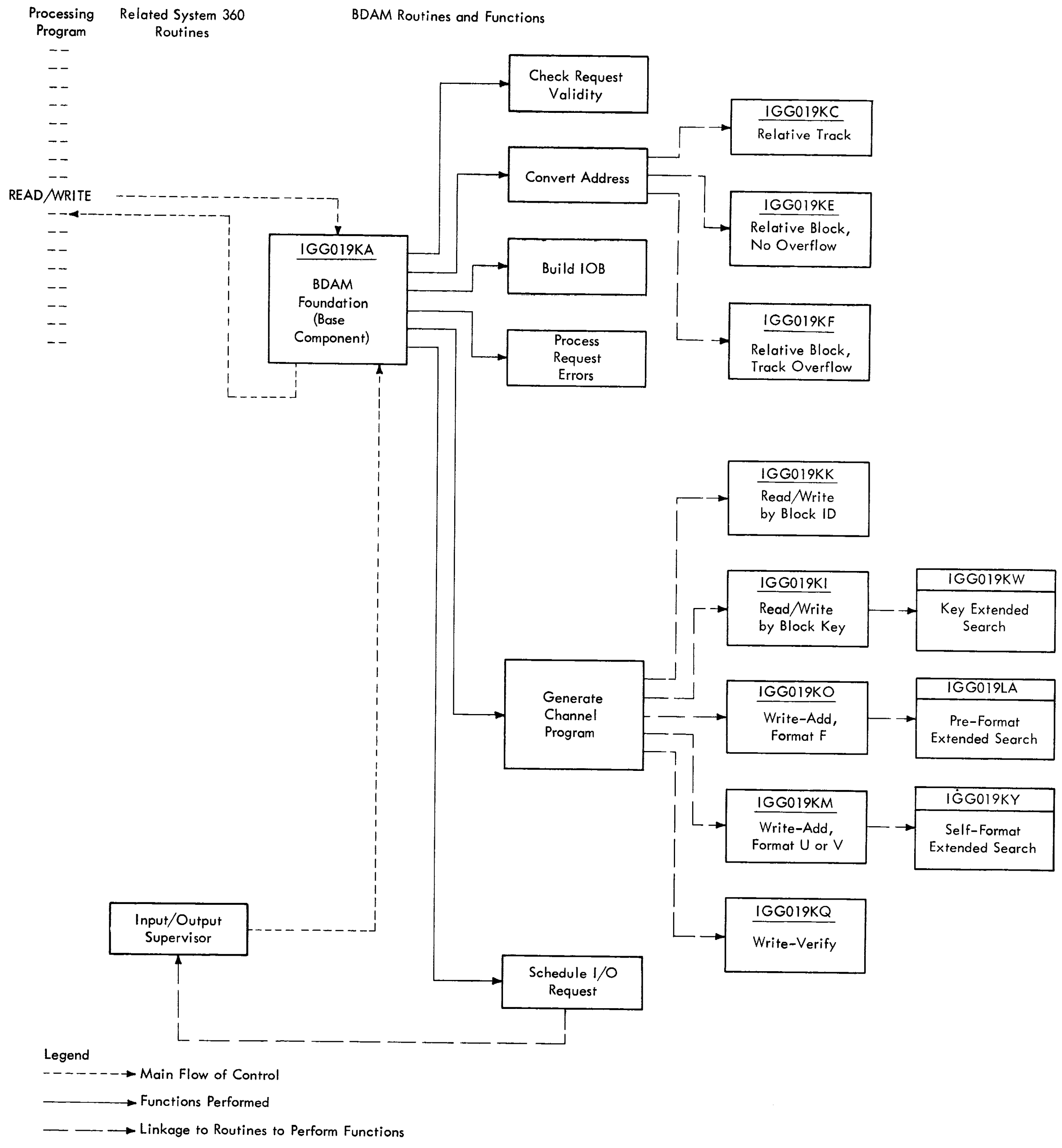
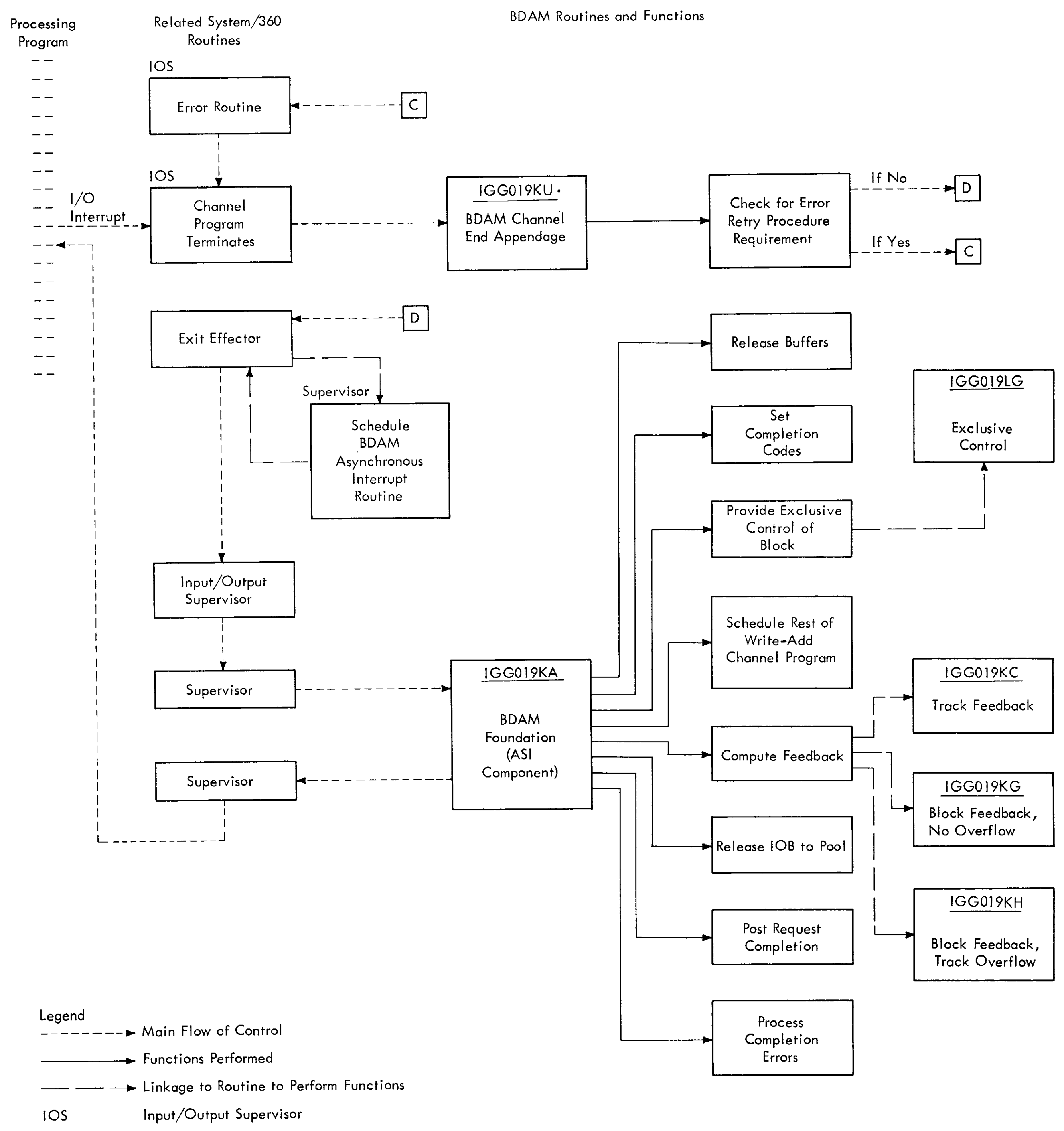


Figure 3. Relationship Among Processing Program, I/O Supervisor, and BDAM for Processing a Request


Processing Program
Related System/360 Routines
BDAM Routines and Functions
IOS
Error Routine
C
IOS
I/O Interrupt
Channel Program Terminates
IGG019KU
BDAM Channel End Appendage
Check for Error Retry Procedure Requirement
If No
D
If Yes
C
Exit Effector
D
Supervisor
Schedule BDAM Asynchronous Interrupt Routine
Release Buffers
Set Completion Codes
IGG019LG
Exclusive Control
Provide Exclusive Control of Block
Input/Output Supervisor
Schedule Rest of Write-Add Channel Program
Supervisor
IGG019KA
BDAM Foundation (ASI Component)
IGG019KC
Track Feedback
Compute Feedback
Supervisor
IGG019KG
Block Feedback, No Overflow
Release IOB to Pool
IGG019KH
Block Feedback, Track Overflow
Post Request Completion
Process Completion Errors
Legend
Main Flow of Control
Functions Performed
Linkage to Routine to Perform Functions
IOS Input/Output Supervisor

• Figure 4. Relationship Among Processing Program, Related System/360 Routines, and BDAM When a Request is Completed

ADDRESS CONVERSION

If the user specifies either relative track or relative block addressing, one of the BDAM relative address conversion modules is used. Each conversion module initiates transformation of the user's block address specification into an actual device location for the block so that the channel program can use the location when searching for the block. If the block address is given in terms of a relative track position, the relative track conversion module (IGG019KC) is used. If the address of a block is given as a relative block number within a data set, one of the two relative block conversion modules (IGG019KE or IGG019KF) is used. If required, an address conversion module is loaded into main storage by the BDAM open executor phase 2 module at the time the data control block is opened.

RELATIVE TRACK CONVERSION (MODULE IGG019KC)

The relative track conversion module, IGG019KC, is entered from either the base component or the ASI component of the foundation module. (See Figures 3 and 4.) Entry is from the base component if the purpose is to initiate a conversion from a relative track address to an actual device address. Entry is from the ASI component if the purpose is to initiate a conversion from a block's actual device address to a relative track address. The latter conversion is for purposes of feedback if specified by the user.

Entry From Base Component: The actual converting of track addresses is done by a conversion routine (referred to as the convert-to-actual routine) of the basic partitioned access method (BPAM). (Refer to the publication IBM System/360 Operating System: Sequential Access Methods, Program Logic Manual.) This routine is resident in main storage. When the BDAM conversion module gives control to the BPAM routine, it supplies the address of the location at which the actual address is to be placed (it is placed in the IOBSEEK field for use by the channel program), the relative track number that the user has indicated in the blkref field of the request macro instructions , and the address of the DEB so that the BPAM routine can refer to the actual extents in the DEB.

The DEB extents contain the cylinder and track information for the various sections of the data set (which is stored on a direct access device); from the extents, 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 whose address is to be converted. This address is then placed in the specified location (i.e., IOBSEEK), and the BPAM routine returns control to the BDAM relative track conversion module, which gives control to the base component of the foundation module.

If the extended search option has been specified, the foregoing procedure is used to determine the actual address of the upper limit of the search. 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 Component: The ASI routine receives program control after the channel program ends. If the user requests relative track feedback, the ASI routine gives control to the BDAM relative track conversion module, which gives control to another resident BPAM conversion routine (referred to as the convert-to-relative routine). The BDAM module gives the BPAM routine both the address of the location of the actual block address (i.e., the address of IOBSEEK) and the address of the DEB. After the BPAM routine completes the address conversion and places the converted address in a parameter register, program control is returned to the relative track conversion module. This module stores the relative track address in the blkref area of the processing program and gives control to the ASI routine.

RELATIVE BLOCK CONVERSION (MODULES IGG019KE AND IGG019KF)

Each of the relative block conversion modules is entered from the foundation module to initiate the conversion of a relative block address to an actual device address. The data control block is examined to determine if track overflow has been specified. If it has not been specified, module IGG019KE is used. If it has been specified, module IGG019KF is used. (See Figure 3.) In either case, the actual conversion requires two routines. The first routine is a BDAM routine that converts a relative block address to a relative track address, and the second routine is a BPAM routine that converts a relative track address to an actual track address.

To convert relative block addresses to relative track addresses, the BDAM modules use information from the relative extent areas of the DEB. For each actual extent area in the DEB, there is a relative extent area whose fields (or field, if track overflow is specified) contain information related to a specific actual extent. One field contains the number of blocks on a track for the device used, and the other field contains the number of blocks in the actual extent. If track overflow is specified, the first of these two fields is not present.

Track Overflow Not Specified (Module IGG019KE)

To determine the relative track address from a relative block number given in the blkref field of a READ macro instruction, module IGG019KE goes through a repeating cycle of reducing the relative block number by a given number of blocks and recording the corresponding number of tracks as follows.

The relative block number reduction process involves successively subtracting the number of blocks in each extent of the data set until an extent that contains more blocks than needed to reach the blkref field value is reached. For each full extent that can be subtracted, the number of tracks in the extent is added to a cumulative total of tracks representing previously subtracted extents. The first full extent that cannot be subtracted as indicated is referred to as a terminal extent.

When the terminal extent is reached, there will remain a number of blocks equal to the difference between the blkref value and the number of blocks already subtracted. This remaining number is divided by the "blocks per track" field of the DEB. The quotient in this division is the number of full tracks to be added to the cumulative total of tracks. The remainder in this division represents the number of blocks from the next track (called the terminal track) that are required to reach the value indicated by the blkref parameter.

The relative track address, in the form of a TTR address (where TT is the relative track number and R is the block number of track TT), is composed of (1) the sum of the tracks required from all extents (if any) up to the terminal extent plus the number (the quotient in the above division) of full tracks required from the terminal extent and (2) the number of remaining blocks (if any) required from the (terminal) track following the last full track.

Control is then given to the BPAM convert-to-actual routine that is resident in main storage. This is the routine required for the relative track address conversion process, and it requires the same information that was needed for the relative track conversion process. (Refer to "Relative Track Conversion.")

The BPAM conversion routine places the converted address in the IOBSEEK field of the IOB and returns control to the BDAM relative block conversion module, which then gives control back to the foundation module.

In a manner similar to that described in the section "Relative Track Conversion," an extended search limit, if necessary, is computed for relative-block-addressing conditions. The actual address of this upper limit also is placed in the IOBUPLIM field.

Following is an example of calculating a relative track address for the case of a data set without track overflow when a relative block number is given in the blkref parameter of a READ macro instructions .

Example 2: Assume the 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 can be summarized in Table 2 below:

Table 2. DEB Information for Example 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_4=96

If the blkref field contains a relative block number of 284, the calculations to find the relative track address are indicated below:

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
$R_3 - B_4 = R_4$	
$28 - 96 < 0$	

Since R_4 is less than zero, the full extent (IV) cannot be subtracted. Extent IV is called 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 equivalent to 35 tracks (the TT value) and 4 blocks (the R value) from the beginning of the data set.

Track Overflow Specified (Module IGG019KF)

To determine the relative track address from a relative block number given in the blkref field of a READ macro instruction, module IGG019KF uses the overflow section of the DEB as well as the single field in each required relative extent area of the DEB. The track overflow option implies that overflow blocks may be present in an extent. For purposes of block addressing, the track on which an overflow block begins is considered to contain the block. The DEB information used for calculating the relative track address of an overflow block is contained in the following fields of the DEB:

- Tracks per extent
- Tracks per period
- Blocks per period
- Blocks per extent

The process of converting a relative block number to a relative block address (in the TTR format) involves successively subtracting the number of blocks in each data set extent until an extent that contains more blocks than needed to reach the blkref field value is reached. For each full extent that can be subtracted, the corresponding number of full tracks is added to a cumulative total of tracks representing previously subtracted extents. The first full extent that cannot be subtracted as indicated is called the terminal extent.

Upon reaching the terminal extent, the periods in that extent are considered as follows. The 'blocks per period' value is subtracted successively until reaching a period, the block count of which cannot be subtracted as indicated. This period is the terminal period. For each period for which the full number of blocks can be subtracted, the number of tracks is added to the cumulative total of tracks.

The individual tracks in the terminal period are then added in successively until a track containing a number of blocks that is equal to or greater than the remaining number of blocks needed to equal the blkref value is reached. This track is the terminal track. The total of all the tracks (taken in considering full extents, full periods, and partial periods) and blocks (taken in considering the blocks on the terminal track in the terminal period) determines the relative track address corresponding to the relative block number given in the blkref field.

After the relative track address has been determined for a block where the track overflow specification exists, control is given to the BPAM convert-to-actual routine and processing proceeds as described for the no-overflow case.

Following is an example of calculating a relative track address for the case of a data set having overflow blocks when the relative block number is given in the blkref field.

Example 3: Assume the 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 phase 3 of the BDAM open program 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 two tracks each contain 6 blocks and the third track contains 5 blocks.

The information needed from the DEB for this data set is summarized in Table 3 below:

Table 3. DEB Information for Example 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

If the blkref field of a READ macro instruction contains a relative block number of 217, the calculations to find the relative track address are indicated below:

blkref value - B_1 = R_1 (remainder)

217 - 114 = 103	The 114 blocks (from Extent I) are on <u>20 tracks</u>
R_1 - B_2 = R_2	
103 - 57 = 46	The 57 blocks (from Extent II) are on <u>10 tracks</u>
R_2 - B_3 = R_3	
46 - 153 < 0	

Since R_3 is less than zero, the full 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 <u>3 tracks</u>
R_3 - IIIb = R_4	
29 - 17 = 12	The 17 blocks (from period IIIb) are on <u>3 tracks</u>
R_4- IIIc = R_5	
12 - 17 < 0	

Since R_5 is less than zero, the full period (IIIc) cannot be subtracted. Period IIIc becomes the terminal period. Now, the blocks on the tracks in the terminal period are subtracted to arrive at the final required number of equivalent tracks and/or blocks to equal the relative block number. In this example, the 12 remaining blocks (the R_4 value) are equivalent to one track (of 6 blocks) plus six 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 one 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 routine to obtain an actual address for the block.

FEEDBACK FOR RELATIVE BLOCK ADDRESSING (MODULES IGG019KG AND IGG019KH)

The feedback modules, IGG019KG and IGG019KH, are used when feedback and relative block addressing have been specified. (See Figure 4.) Entry to the proper module is from the ASI routine at the completion of a channel program. The actual address of the block was obtained when the channel program made access to the block.

The actual device address of the block is given to the BPAM convert-to-relative routine by the feedback module. The relative track address determined by the BPAM routine is given to the appropriate BDAM feedback module. Using information that is contained in the actual extents, the relative extents, and, if track overflow has been specified, the overflow section of the DEB, the feedback module constructs the relative block address for the block and places the result in the blkref area specified in the DECB. The method used in obtaining the relative block number is basically a reversal of the technique used to convert to an actual address. The feedback module then gives program control to the ASI routine.

<u>CHANNEL PROGRAMS FOR BDAM</u>

To perform the input and output operations required by BDAM processing, several channel programs are available. For each request to read or write a block, the appropriate channel program is constructed dynamically when the base routine of the foundation module gives control to a BDAM channel program generating module. (See Figure 3.) Appendix C contains the actual channel programs that are generated.

The foundation module uses parameters specified in the DCB and in the individual request macro instruction to determine which channel program is to be constructed. The type field of the IOB is tested for this purpose. The channel command words, of which the channel programs are constructed, contain the following types of operation codes: write information, read information, search for equal argument, transfer in channel, and track seek. The channel programs permit BDAM to read or write blocks based on either a key or a block identification search argument.

As a channel program is constructed, it becomes a logical part of the IOB associated with the request. (The structure of an IOB as it relates to BDAM is given in Appendix A.) There are three categories of channel programs: update programs, format programs, and the verification program.

UPDATE PROGRAMS (MODULES IGG019KI, IGG019KK, AND IGG019KW)

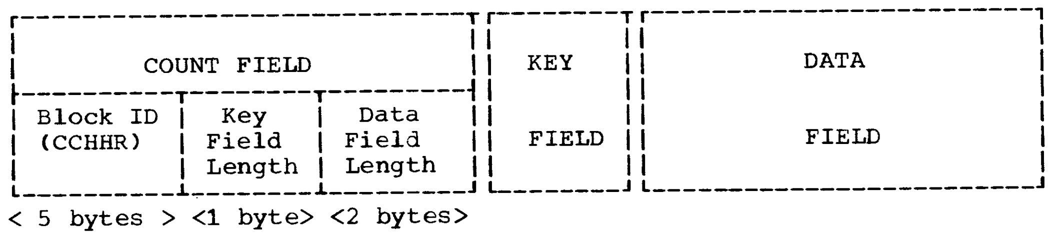
The update programs are those that read or write information for purposes other than adding a new block to an existing data set. Control is given to the proper channel program generating module by the foundation module.

Basically, there are three BDAM channel programs available for reading or for updating an existing block. Each channel program can take one of two forms depending on whether it is generated in response to a read request or in response to a write (update) request. These forms are shown in Table 4. The search arguments indicated in this table correspond to the fields of a block as it appears on a direct-access storage device. (See Figure 5.) If the extended search option has been specified, module IGG019KW modifies the channel program that has been generated by module IGG019KI, to search additional tracks or blocks beyond that which is specified in a READ or a WRITE macro instruction.

For channel programs to satisfy a write request for which the write-validity-check option has been specified, control is given to the write-verify module, IGG019KQ, at the completion of the channel-program generating function of the appropriate update program module. After generating the verification channel program, the write-verify module returns control to the foundation module.

Table 4. Channel Programs for Reading or Writing an Updated Block

Channel Program Form	Search Argument	Extended Search Option Specified	Generating Module(s)
Read Data or Write Data	Key Field	No	IGG019KI
Read Data or Write Data	Key Field	Yes	IGG019KI and IGG019KW
Read Data or Write Data	Block ID of Count Field	Not a Factor	IGG019KK



CCHHR gives the physical position of the block on the device.
The Key Field Length specification may be from 0 to 255 bytes.
The Data Field Length specification may be from 0 to 32760 bytes.

Figure 5. Structure of a Block on a Direct-Access Storage Device

Each channel-program generating module returns control to the foundation module's base component after the last channel command word has been generated and placed in the IOB for the request. Chart 03 summarizes the flow of control among BDAM modules for updating (or reading) information.

FORMAT PROGRAMS

There are three BDAM format channel programs. These programs are used for writing a new block from main storage onto a direct-access device. For adding fixed-length blocks to an existing data set, two of the three channel programs are available. The choice of which one will be used depends on whether the extended search option has or has not been specified. Since the block length is known or predetermined, these channel programs are called pre-format programs.

If the processing program specifies adding either variable-length blocks or blocks of an undefined length to an existing data set, the remaining format channel program is required. When the extended search option is specified, this channel program is repeated as many times as necessary until either track space for the record is found or the search limit is reached. Since BDAM itself must establish the space requirements for writing these blocks, these channel programs are called self-format programs.

The format channel programs are sometimes referred to as 'write-add' programs to distinguish them from the write programs described as update programs.

As with the update channel programs, the format channel programs incorporate channel command words for searching either on the key field or on the count field of a block, and they may or may not require the verification channel program (generated by module IGG019KQ). The pre-format and self-format programs are respectively generated by either the pre-format modules or the self-format modules, depending on the block format of the data set.

Table 5 shows the factors that determine which format modules are required to generate the format channel programs. Note that when the extended search option is specified for fixed-length blocks, two modules are required. The second module listed is given control by, and returns control to, the first module. The function of the second module is to modify the channel program generated by the first module. The foundation module gives control to either IGG019KO or IGG019KM (depending on the block format), and when the channel programs have been generated, the format module returns control to the foundation module.

Pre-Format Channel Programs (Modules IGG019KO and IGG019LA)

The pre-format channel programs are used when a new fixed-length block is to be added to an existing data set. In order to add fixed-length blocks, the user must have initially placed his data blocks on the direct-access device by means of the basic sequential access method (BSAM) write routine for creating a direct organization data set. As blocks were placed on the direct-access device, dummy records may have been provided to allow the BDAM channel program to search for an area in which to place a new block. A dummy record is one in which the first byte of the key field is a hexadecimal FF, and the first byte of data has a value indicating the position of the dummy record on the track.

Table 5. Requirements for Channel Programs to Add New Blocks to an Existing Data Set

Block Length	Channel Program Format	Extended Search Option Specified	Generating Module(s)
Fixed	Pre-format	No	IGG019KO
		Yes	IGG019KO, IGG019LA
Variable or Undefined	Self-format	No	IGG019KM
		Yes	IGG019KM

Without Extended Search Option: If the extended search option has not been specified, the pre-format channel program is generated by module IGG019KO. This module is entered from, and returns control to, the foundation module's base component. The module constructs the necessary channel command words and, by incrementing a base address in the IOB, positions the fields of the channel command words as they are developed.

The channel program first searches an indicated track for a dummy record. This search starts at the first block encountered on the proper track. If a dummy record is not found on the indicated track, the search is not satisfied. An indication of 'no-space-found' is then given to the processing program. This indication appears in the DECB. The search is successful if a dummy record is encountered.

After the key field of a dummy record has been found, the first byte of the data field is read into the IOB to provide the position of the dummy record on the track. The same dummy record is next located by a search on the ID portion of the count field. The new block key and data fields can then be written to replace those of the dummy record.

With Extended Search Option: If the extended search option has been specified, the pre-format channel program is generated by modules IGG019KO and IGG019LA. The function of module IGG019LA is to modify the channel program generated by module IGG019KO so that the search for a dummy record will extend across multiple tracks (up to the limit specified in the IOBUPLIM field of the IOB). If the extended search comes to an end and a dummy record is not found, the procedure is the same as described for the case without the extended search option.

If a dummy record is found, the search is successful and the dummy record's position is read into the IOB as before. The channel program then continues as in the case without the extended search option.

If the block that is being written by the request must be verified (because of an option specified in the DCB macro-instruction), the appropriate pre-format channel program is enlarged to include additional channel command words. Module IGG019KQ is given control to perform this function. (Refer to "Verification Program.")

Self-Format Channel Programs (Modules IGG019KM and IGG019KY)

The self-format channel programs are used when new blocks of either undefined length (format U) or variable length (format V) are to be added to an existing data set. As with fixed-length records, the basic sequential access method's write routine for creating a direct-organization data set must have been used to initially place the data blocks on the direct access device. When the data set is initially put on the direct access device, a capacity record (block 0) is also placed on each track. The capacity record contains both the ID of the last block on the track and the number of usable bytes remaining on the track. These are the available bytes that may be used for new blocks. Figure 6 illustrates the data portion of the capacity record.

```
┌─────────────┬─────────────┬─────────────┐
│   ID of     │Usable bytes │   Unused    │
│ last block  │remaining on │             │
│             │   track     │             │
└─────────────┴─────────────┴─────────────┘
<---5 bytes--> <--2 bytes--> <---1 byte--->
```

Figure 6. Data Field of a Capacity Record for a BDAM Data Set

The foundation module gives control to BDAM module IGG019KM to generate the channel program for both format U and format V records. The channel program consists of (1) one section that reads the capacity record from the indicated track into main storage and (2) another section that writes the new block and updates the capacity record to reflect inclusion of the new block on the track. Generation of the self-format channel program involves moving constants representing elements of channel command words into assigned positions of the request's IOB to form the required channel command words. If record verification is required, the self-format channel program is enlarged by module IGG019KQ to include the verification program channel command words.

The last channel command word of the section of the channel program that reads the capacity record does not include a command chaining flag. This permits the I/O supervisor to give control to the channel end appendage module after the capacity record has been read. The appendage module then branches to a supervisor routine to schedule the ASI routine.

In the performance of two or more tasks, concurrent requests to add a block to the same track on a direct-access device may be made. All but one of these requests (called 'write-add' requests) must be placed on an inter-task queue to prevent undesired interference. This interference would result from other requests obtaining the same track capacity record for interrogating and updating before a first request was finished with it. Concurrent requests for the same track may also occur in the performance of a single task.

When the ASI routine (1) (numbers in () refer to points indicated on Figure 6A) gets control after the track capacity record has been read for a write-add request, it, in turn, gives control to the exclusive control module IGG019LG (2). Module IGG019LG either places the record on an inter-task queue by using the ENQ macro instruction (3) or places the IOB for the record on the unposted queue (4). (The functions of the exclusive control module are described more fully in the section "Exclusive Control.") In either case, control is then given to the supervisor.

After the channel program reads the capacity record in connection with a given write-add request, the supervisor again 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 indicated track.

If the track can contain the new block, calculations are made to update the capacity record. 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.

If the unposted queue does not contain an IOB waiting for the capacity record that was just updated, the capacity record is no longer needed for the performance of the current task. Module IGG019LG issues a DEQ macro instruction to release the record to other tasks that may require it, and control is given to module IGG019KM (8). If an IOB for this capacity record is on the unposted queue, both the address of the IOB and program control are given to module IGG019KM.

The self-format module determines whether the block was placed on the track (i.e., if 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 IOB field (IOBDBYTR...see Appendix A) that 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 same 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 has been kept from 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 one and proceeds according to the conditions given in the following paragraphs:

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 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 beyond the current extent, control is given to the BDAM end-of-extent module, IGG019LC. This module determines if more extents are available for searching. There are two possibilities for consideration.

   1. If there are more available extents 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 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 either the search limit has been reached or there are no more available extents, 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, commencing with giving control 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 (point 9 on Figure 6A), processing of that IOB then continues from point D in Figure 6A.

VERIFICATION PROGRAM (MODULE IGG019KQ)

If the processing program specifies the write-validity-check option in the DCB 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 write-type channel program. These channel command words are added to the existing channel program.

If required, the BDAM open executor program brings the write-verify module into main storage at the time the data set is opened. This module is entered from the module that generates the particular type of writing channel program required by 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 field. 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 that developed by the read-back. If the two codes are not equal, a data check indication is set. The skip flag-command-code is set to 1 (on) in the last channel command word of the verification program so that the data that is read back is not placed into main storage.

The write-verify module returns control to the base component of the foundation module.

APPENDAGES

The basic direct access method contains appendage modules to which program control is given at various stages during the execution of a read or a write request. The combined functions of the several appendage modules are to make tests, to set switches, to schedule the BDAM ASI routine, and to obtain new extents for extended search operation.

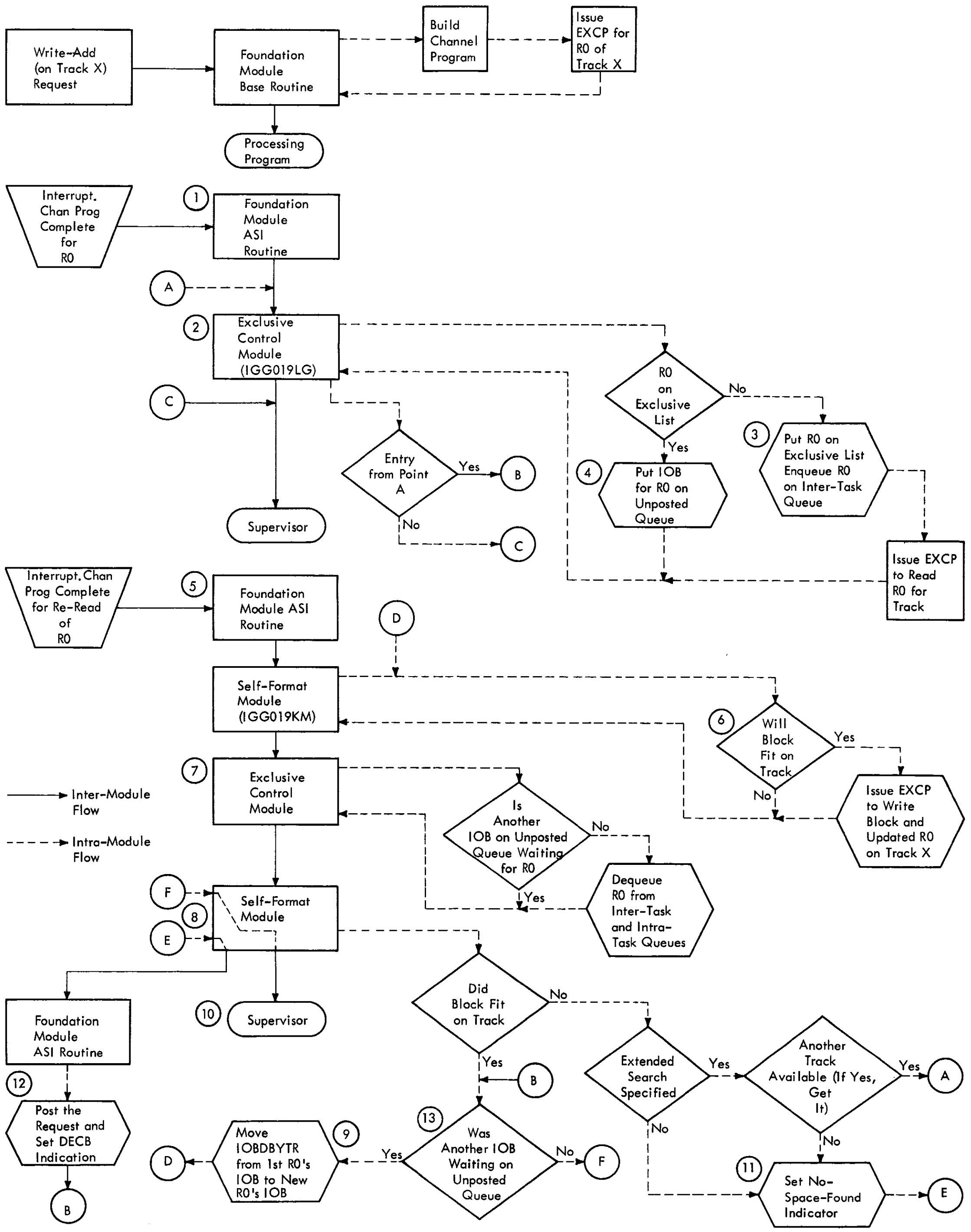
When the DCB is opened, phase 2 of the BDAM open executor routine issues the LOAD macro instruction to load the appendage modules into main storage and places the addresses of the modules into an appendage list, which phase 1 has built in an area of protected main storage. (See Figure 1.) This list, which is also referred to as the IOS Appendage Address Table, is located in subpool 254 of the supervisor queue area.

The BDAM program uses three appendage modules: the start I/O appendage module, the channel end appendage module, and the end-of-extent appendage module. (See Chart 01.)

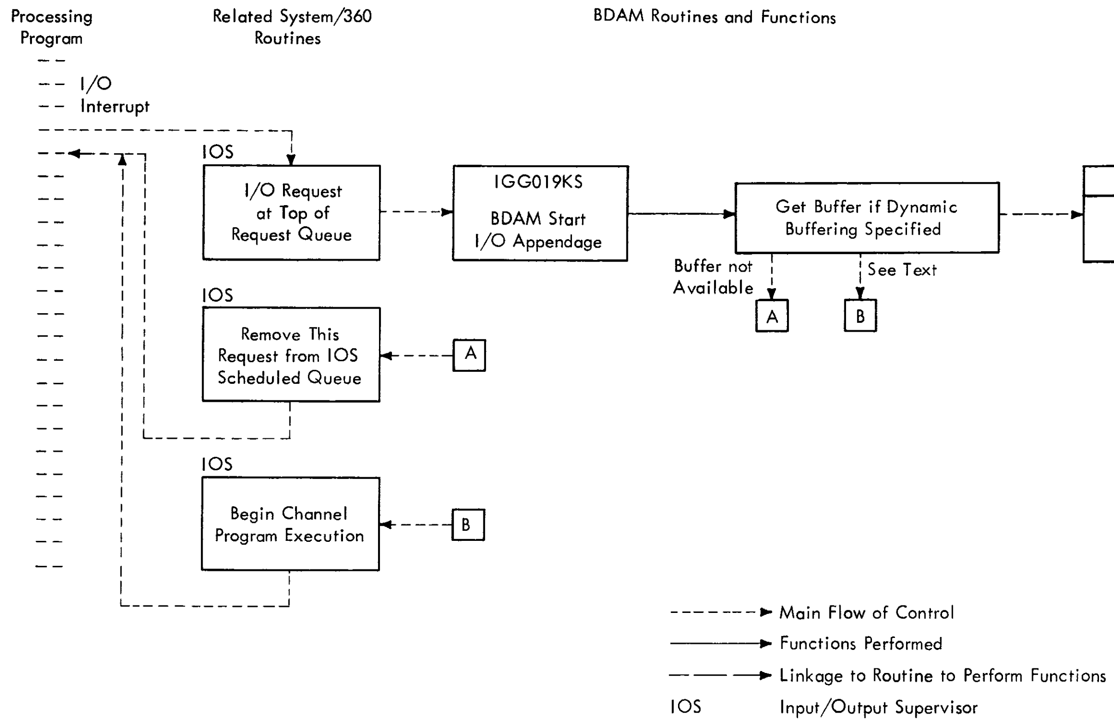
The start I/O appendage module and the channel end appendage module are entered from the I/O supervisor and return control to the I/O supervisor. The end-of-extent appendage module may be entered from either the BDAM self-format module or the I/O supervisor. On returning from the end-of-extent appendage module, control may be given to either the BDAM self-format module or the I/O supervisor.

START I/O APPENDAGE (MODULE IGG019KS)

The BDAM start I/O appendage module, IGG019KS, is placed in main storage if the dynamic buffering option has been specified. This module is entered from the I/O supervisor before a channel program is initiated. (See Figure 7.)


Write-Add (on Track X) Request
Foundation Module Base Routine
Build Channel Program
Issue EXCP for R0 of Track X
Processing Program
Interrupt. Chan Prog Complete for R0
Foundation Module ASI Routine
Exclusive Control Module (IGG019LG)
R0 on Exclusive List
No
Yes
Put R0 on Exclusive List Enqueue R0 on Inter-Task Queue
Put IOB for R0 on Unposted Queue
Entry from Point A
Yes
No
Supervisor
Issue EXCP to Read R0 for Track
Interrupt. Chan Prog Complete for Re-Read of R0
Foundation Module ASI Routine
Self-Format Module (IGG019KM)
Will Block Fit on Track
Yes
No
Issue EXCP to Write Block and Updated R0 on Track X
Inter-Module Flow
Intra-Module Flow
Exclusive Control Module
Is Another IOB on Unposted Queue Waiting for R0
No
Yes
Dequeue R0 from Inter-Task and Intra-Task Queues
Self-Format Module
Supervisor
Foundation Module ASI Routine
Did Block Fit on Track
No
Yes
Extended Search Specified
Yes
No
Another Track Available (If Yes, Get It)
Yes
No
Post the Request and Set DECB Indication
Move IOBDBYTR from 1st R0's IOB to New R0's IOB
Was Another IOB Waiting on Unposted Queue
Yes
No
Set No-Space-Found Indicator
1
2
3
4
5
6
7
8
9
10
11
12
13
A
B
C
D
E
F

•Figure 6A. Module Relationships for Write-Add Requests in Multi-Task Environment



Processing Program
Related System/360 Routines
BDAM Routines and Functions
I/O Interrupt
IOS
I/O Request at Top of Request Queue
IGG019KS
BDAM Start I/O Appendage
Get Buffer if Dynamic Buffering Specified
IGG019LE
Dynamic Buffering
Buffer not Available
See Text
A
B
IOS
Remove This Request from IOS Scheduled Queue
IOS
Begin Channel Program Execution
Main Flow of Control
Functions Performed
Linkage to Routine to Perform Functions
IOS Input/Output Supervisor

• Figure 7. Relationship Among Processing Program, I/O Supervisor, and BDAM for Executing a Request

<u>Buffer Needed</u>

When a request with the dynamic buffer specification is ready to be executed, I/O supervisor gives control to the BDAM start I/O appendage module. This module determines if a buffer has already been assigned to this request or if it is necessary to obtain a buffer from a buffer queue. If a buffer must be obtained, module IGG019KS gives control to the dynamic buffer module, IGG019LE. If the buffer request does not have to be placed on a queue by the dynamic buffer module, the start I/O appendage module will receive control after a buffer has been allocated. The request is then ready for execution, and control passes to the I/O supervisor for channel program execution.

CHANNEL END APPENDAGE (MODULE IGG019KU)

When a channel program is terminated, either normally or abnormally, the I/O supervisor gives control to the channel end appendage module, IGG019KU. (See Figure 4.) This module is always placed in main storage by the BDAM open executor phase 2 module. There are two general sections to the channel end appendage module. The sections are entered under the conditions described in the following paragraphs:

<u>Entry to First Section:</u> The first section is entered under one of the following conditions:

- A channel program terminates normally.
- A channel program encounters a unit exception condition, which is interpreted by the BDAM program as an end-of-data-set condition.
- The execution of a channel program results in a block length different from that specified in the READ or WRITE macro instruction.

If the channel program terminated normally, the channel end appendage module schedules the BDAM asynchronous interrupt routine and then gives control to the I/O supervisor.

<u>Note:</u> To schedule the ASI routine, the channel end appendage module branches to the exit effector routine of the task supervisor. (Refer to the publication <u>IBM System/360 Operating System: MVT Supervisor, Program Logic Manual</u>.) The exit effector routine then schedules the ASI routine and returns control to the channel end appendage module.

For an end-of-data-set condition, the channel end appendage module sets indicators in the IOB, schedules the BDAM asynchronous interrupt routine, and returns control to the I/O supervisor. (Refer to preceding note.)

If the channel end appendage module is entered because of an incorrect-length indication (given in the IOB when the number of bytes read or written is not equal to the number of bytes specified in the channel program), a test is made to determine the type of request being processed. Three cases are possible:

- 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 is specified by the first two 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 length values are equal, the incorrect-length indication 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 read request for format-U records, the incorrect-length indication 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, or if the block lengths in the first case are unequal, the ASI routine is not scheduled, the incorrect-length indication is left set at 1 (on), and the channel end appendage module gives control to the I/O supervisor.

Entry to the Second Section: The second section of the channel end appendage module is entered when either a device-type error or a permanent error has been encountered. 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 channel end appendage module sets an indicator in the IOB, schedules the BDAM asynchronous interrupt routine, and returns program control to the I/O supervisor.

END OF EXTENT APPENDAGE (MODULE IGG019LC)

The BDAM end-of-extent appendage module is required if the extended search option has been specified in the DCB macro instructions . At the time the data set is opened, phase 2 of the BDAM open executor program determines the need for this end-of-extent module and loads it if necessary. There are two extended-search-type situations that require this module. In one situation, the module functions as an I/O supervisor appendage. In the other situation, the module appears as a BDAM routine without supervisory functions.

Supervisory Mode

The I/O supervisor gives control to the BDAM end-of-extent module when a channel program comes to the end of a data set extent while searching for a block to be read or written or while searching for a dummy record in which to write a new pre-format type block. Under either of these situations, module IGG019LC establishes the address of the next extent to be searched. Note that if a search has begun at some point other than the beginning of the first extent in the DEB, the address of the next extent may, at some point in the search, become that of the first extent.

If the search limit (as determined from the LIMCT parameter in the DCB macro instruction) is not in this next (or new) 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, indicates the need for the BDAM asynchronous interrupt routine to reschedule the channel program using a search address in the new extent.

Note: The search limit is found in the IOBUPLIM field of the IOB. The limit is defined as the first track beyond the last actual track that may be searched for the given data set.

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 but the new search address equals the search limit, the search has ended unsuccessfully. An indication is then set to show that either no space was found or no block was found, and control is given to I/O supervisor, which, in turn, will go to the abnormal end component of the BDAM channel end appendage.

Non-Supervisory Mode

The BDAM self-format extended search module may at some time, in the process of establishing search addresses, recognize that the end of a given data extent has been reached. If this happens, the extended search module gives control to the end-of-extent module.

As when in the supervisory mode, module IGG019LC determines the availability of other extents to be searched, establishes a new search address, and determines whether or not the search limit has been reached. If the search limit has not been reached, the end-of-extent module uses the new search address related to a new extent and reschedules the channel program (to read in the capacity record of the next track) and then 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)

If a programmer has specified that the exclusive control feature be applied to blocks that are read and that may or may not be subsequently written, the ASI routine gives control to module IGG019LG to handle both the block queuing and the block dequeuing that is required with this feature.

In addition, module IGG019LG is used to place records on a queue when the processing program encounters a request to add a new block of either variable-length records or records of undefined length. (Refer to Chart 04.)

With exclusive control in effect, a given block may not be updated (or otherwise acted upon) by processing associated with other requests until exclusive control for that block has been removed. 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 Appendix A) consists of an 80-byte segment of main storage obtained by phase 1 of the BDAM open executor program. This segment contains space for nine entries, each entry consisting of the UCB pointer and the device 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-byte segments, each of which 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 intra-task list of device addresses of blocks (i.e., capacity records and data blocks) that are 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 device 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 on the list, the IOB for the request is placed on a queue called the unposted queue. This is an intra-task queue of IOBs representing requests for blocks whose addresses are currently on the read-exclusive list and are associated with the current task. The data control block contains the addresses of the first and the last IOBs on 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, it signifies that this is the first request for the record for the current task. The IOB contains the block address. The UCB pointer and the CCHHR bytes of the device address of the block are put on the read-exclusive list. Since the same block may be required in the performance of another concurrent task, 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 inter-task 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 by a routine associated with the task for which the entry had been enqueued. Since a block on this inter-task queue cannot be used in the performance of one task until it is disassociated from another task, a waiting period of indeterminate length may result.) Module IGG019LG then issues an EXCP macro instructions for the re-reading of the block that has just been enqueued. Control is then given to the ASI routine which, in turn, gives control to the supervisor.

Note: For systems operating under the primary control program, there is no need for an inter-task queue. Therefore, when either the enqueue routine or the dequeue routine is given control, the routine returns control directly to the routine that had invoked it (i.e., the effect is the same as a no-operation). The Supervisor and Data Management publications contain further information regarding 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 place the block address.

Releasing Blocks Under Exclusive Control

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 (i.e., their data fields remain unchanged).

RELEASE BY WRITING: When a request (called a write-exclusive request) to write a block that has been previously read under exclusive control is encountered, 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 that 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 intra-task requests for the block, module IGG019LG clears the block's address from the read-exclusive list. This permits another entry to be made to 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 that is being written out, it is necessary that these other requests be provided with the most 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 "duplicate" request on the queue. Control then passes to the ASI routine so that the first of these "duplicate" 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 "duplicate" requests. 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 encountered. After initialization, determination of the type of addressing that has been specified, and conversion of a block address to an actual address if it is not already in that form, module IGC0005C gives control to the exclusive control module. If the block specified for release from exclusive control is not found by searching the addresses on the read-exclusive list, an error condition is indicated; 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.

Table 6 summarizes the exclusive control module's main functions as they have been described in the preceding paragraphs.

Table 6. Functions of the Exclusive Control Module for Specified Conditions

Macro instruction That Requests Action	Block Address Already on Read-Exclusive List	Other IOBs for Same Block on Unposted Queue	Action Taken
1. READ (Exclusive)	Yes	--	Place request's IOB on unposted queue. Go to supervisor.
2. READ (Exclusive)	No	--	Add block's address to read-exclusive list. Enqueue block on inter-task queue. Schedule the block for reading. Go to supervisor
3. WRITE (Exclusive)	No	--	Error exit to ASI.
4. 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.
5. WRITE (Exclusive)	Yes	No	Go to ASI routine to post write request and free the IOB. Remove entry from read-exclusive list and from inter-task queue. Go to supervisor.
6. RELEX	No	--	Return to RELEX routine with error code.
7. RELEX	Yes	Yes	Remove read request from queue. Go to ASI routine to post read request and free the IOB. Return to RELEX routine.
8. RELEX	Yes	No	Remove block from read-exclusive list and from inter-task queue. Return to RELEX routine.

DYNAMIC BUFFERING (MODULE IGG019LE)

The handling of all buffer requirements for BDAM requests is done by module IGG019LE if the dynamic buffering feature is specified. These requirements include: obtaining and assigning buffers into which data may be read; placing buffer requests on a queue if there are no available buffers; and releasing buffers for other uses either after a request has been completed or when the FREEDBUF macro instruction is used.

Buffer Assignment

As each request to read data is about to be executed, the start I/O appendage module, IGG019KS, checks the IOBDTYPE field of the IOB to see if there is a requirement for a buffer to be assigned to the request.

If a buffer is required, a check is made to determine if a buffer has already been assigned to the request. Note that requests for dynamic buffer assignment may occur whether or not the exclusive control option is specified for read requests. If a request has an assigned buffer, the channel program may be initiated. If a buffer is needed, the start I/O module gives control to the dynamic buffer module.

The dynamic buffer module ascertains whether all buffers in the buffer pool (the buffer pool was established and initialized by the open executor program) for the data set have been allocated to other requests or if one is available. If a buffer is available for the current request, it is assigned to the request, and the entries in the buffer pool are updated to reflect the effective removal of the buffer. The associated channel program is completed by placing in it the buffer addresses into which information is to be read, and then the channel program is ready for execution.

If no buffers are available to satisfy a buffer request, the IOB is placed on a queue of requests waiting for buffer assignment. The elements of this queue are chained to each other through addresses given in the IOB. The addresses of both the first and the last request on the queue are given in the buffer control block. As buffers subsequently become available, they are allocated to the requests on the queue. As each request is added to the queue, it becomes the last request of the queue. When a buffer becomes available, it is allocated to the request currently at the top of the queue (i.e., the first request on the queue), and that request is removed from the queue.

Releasing Buffers

When a request using a dynamically-assigned buffer has been completed (either successfully or unsuccessfully), the buffer that has been assigned to the request may be made available for other requests. This can happen in one of two ways: control may be given to the dynamic buffer routine either by the ASI routine or as the result of a FREEDBUF macro instruction being encountered in the processing program. Note that a buffer assigned to a read request for a block that is not to be updated will be released when the request is completed only if the FREEDBUF macro instruction is issued for that request's buffer. For a block that is to be updated, a buffer is retained until freed by a corresponding write request that specifies dynamic buffering.

After the ASI routine receives control at the completion of a write request, it gives control to the dynamic buffer module if dynamic buffering is being used. If a dynamic buffer routine finds an IOB on the buffer queue waiting for a buffer, it assigns the buffer from the just-completed request to the top request of the queue. The buffer queue is updated by moving each request up one position on the queue. The channel program for the selected request is then completed (as described in the section "Buffer Assignment"), and the dynamic buffer module issues an EXCP request to execute the channel program.

If there are no entries on the buffer queue waiting for buffers, the buffer from the completed request is placed on the list of available buffers. The buffer control block and its entries are then updated as required to include the added buffer. The dynamic buffer module then gives control back to the ASI routine.

When a FREEDBUF macro instruction is encountered in the processing program, the supervisor gives control to the dynamic buffer module. A routine in this module then checks for other queued requests and assigns the freed buffer or makes it available for future buffer requests as discussed for the entrance from the ASI routine. The dynamic buffer module then gives control back to the supervisor.

CHECK MODULE (MODULE IGG019LI)

To ensure that a given read or write request is completed before a certain point in the associated processing program, either a CHECK macro instruction or a WAIT macro instruction must be coded following the request 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 same DCB macro instruction that contains the check specification.

When the check module receives control, it establishes a wait condition if the associated request has not been posted as complete. If the request is complete at this point or is subsequently 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. The error types are placed in a register and control is given to a SYNAD routine if one is present. The publication _IBM System/360 Operating System: Supervisor and 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.

THE BDAM CLOSE EXECUTOR PROGRAM (MODULE IGG0203A)

The BDAM close executor program consists of module IGG0203A. This module is given control during the closing of a DCB that specifies BDAM. (See Figure 8.) When the CLOSE macro instruction is encountered in a processing program, the expansion of the macro instruction causes program control to be given to the data management close routine. This routine uses the BDAM close module as a subroutine.

The main purpose of the BDAM close executor program 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.

The first step consists of removing from the I/O supervisor scheduled queue of requests any requests that have been scheduled but whose operations (i.e., 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 on the list of IOBs. The IOB list also includes the IOBs that are on 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 on the unposted queue. These IOBs were placed on the queue by the exclusive control module.

As a fourth step, the storage that has been allotted to IOBs remaining on the buffer queue is released. (The IOBs on the buffer queue are those waiting for buffers to be made available to them.) The main storage areas that have been obtained both for the buffer control block and for buffers to be assigned dynamically are also released in this fourth step.

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

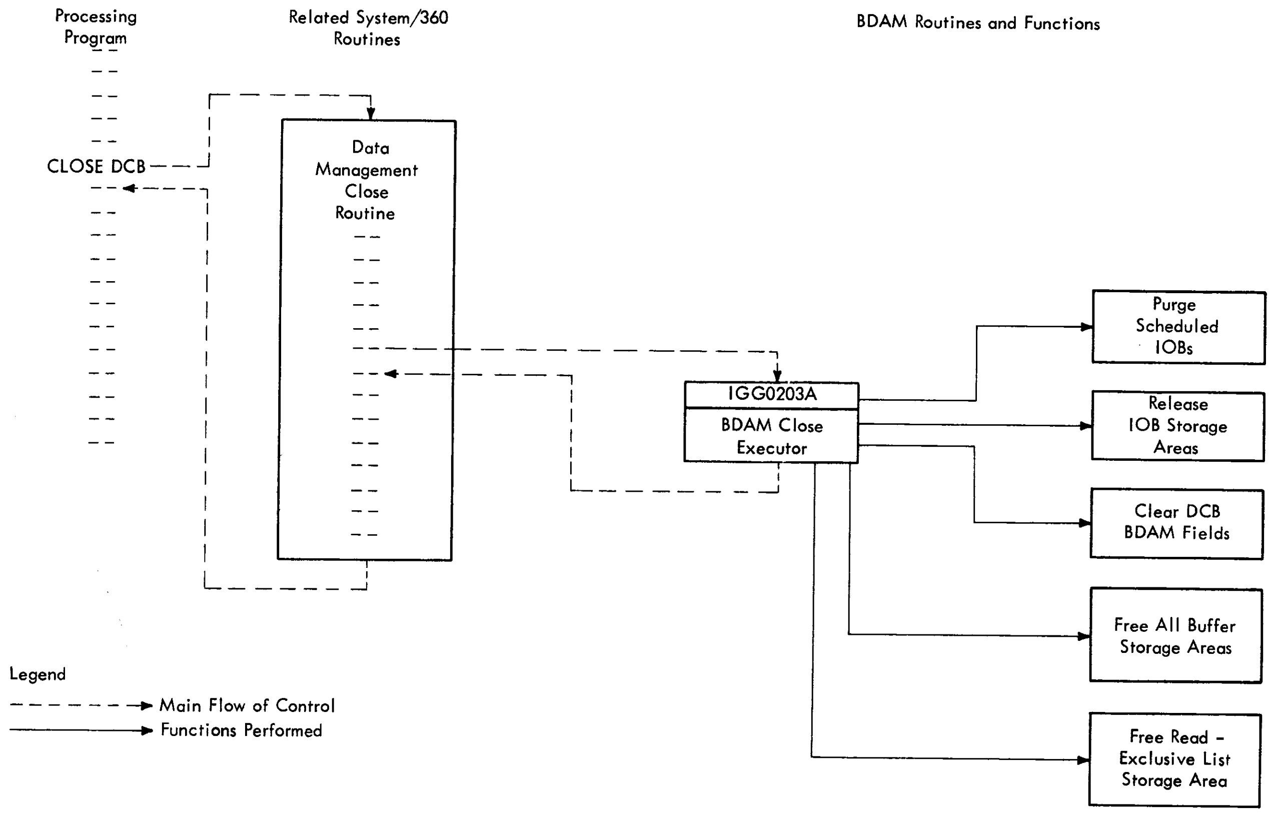
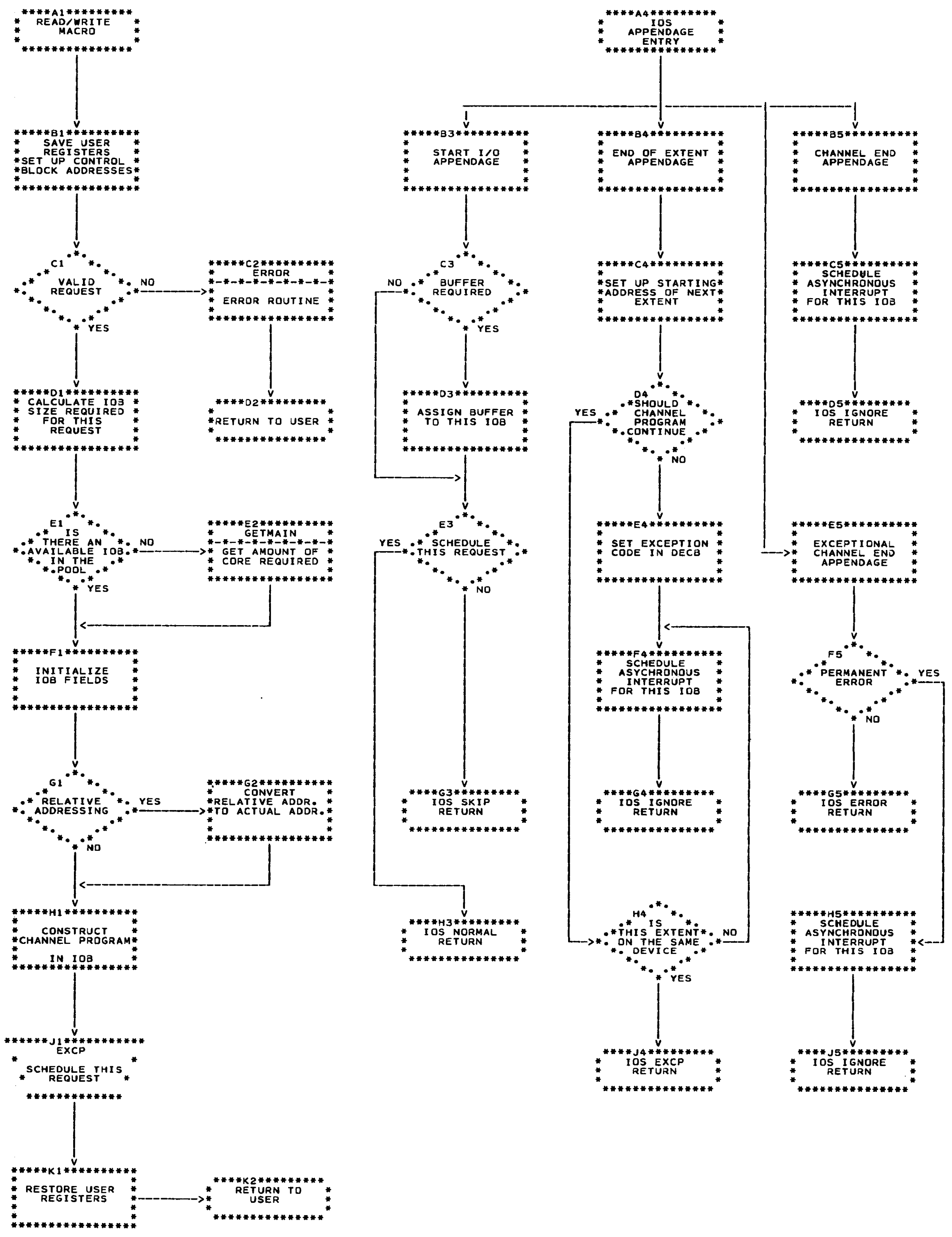

Processing Program
Related System/360 Routines
BDAM Routines and Functions
CLOSE DCB
Data Management Close Routine
IGG0203A
BDAM Close Executor
Purge Scheduled IOBs
Release IOB Storage Areas
Clear DCB BDAM Fields
Free All Buffer Storage Areas
Free Read - Exclusive List Storage Area
Legend
Main Flow of Control
Functions Performed

Figure 8. Relationship Among Processing Program, Data Management Close Routine, and BDAM at Close Time

•Chart 01. BDAM High Level Flow


A1 READ/WRITE MACRO
B1 SAVE USER REGISTERS SET UP CONTROL BLOCK ADDRESSES
C1 VALID REQUEST
NO
YES
C2 ERROR ERROR ROUTINE
D1 CALCULATE IOB SIZE REQUIRED FOR THIS REQUEST
D2 RETURN TO USER
E1 IS THERE AN AVAILABLE IOB IN THE POOL
NO
YES
E2 GETMAIN GET AMOUNT OF CORE REQUIRED
F1 INITIALIZE IOB FIELDS
G1 RELATIVE ADDRESSING
YES
NO
G2 CONVERT RELATIVE ADDR. TO ACTUAL ADDR.
H1 CONSTRUCT CHANNEL PROGRAM IN IOB
J1 EXCP SCHEDULE THIS REQUEST
K1 RESTORE USER REGISTERS
K2 RETURN TO USER
A4 IOS APPENDAGE ENTRY
B3 START I/O APPENDAGE
C3 BUFFER REQUIRED
NO
YES
D3 ASSIGN BUFFER TO THIS IOB
E3 SCHEDULE THIS REQUEST
YES
NO
G3 IOS SKIP RETURN
H3 IOS NORMAL RETURN
B4 END OF EXTENT APPENDAGE
C4 SET UP STARTING ADDRESS OF NEXT EXTENT
D4 SHOULD CHANNEL PROGRAM CONTINUE
YES
NO
E4 SET EXCEPTION CODE IN DECB
F4 SCHEDULE ASYCHRONOUS INTERRUPT FOR THIS IOB
G4 IOS IGNORE RETURN
H4 IS THIS EXTENT ON THE SAME DEVICE
NO
YES
J4 IOS EXCP RETURN
B5 CHANNEL END APPENDAGE
C5 SCHEDULE ASYNCHRONOUS INTERRUPT FOR THIS IOB
D5 IOS IGNORE RETURN
E5 EXCEPTIONAL CHANNEL END APPENDAGE
F5 PERMANENT ERROR
YES
NO
G5 IOS ERROR RETURN
H5 SCHEDULE ASYCHRONOUS INTERRUPT FOR THIS IOB
J5 IOS IGNORE RETURN

• Chart 02. BDAM High Level Flow (Continued)

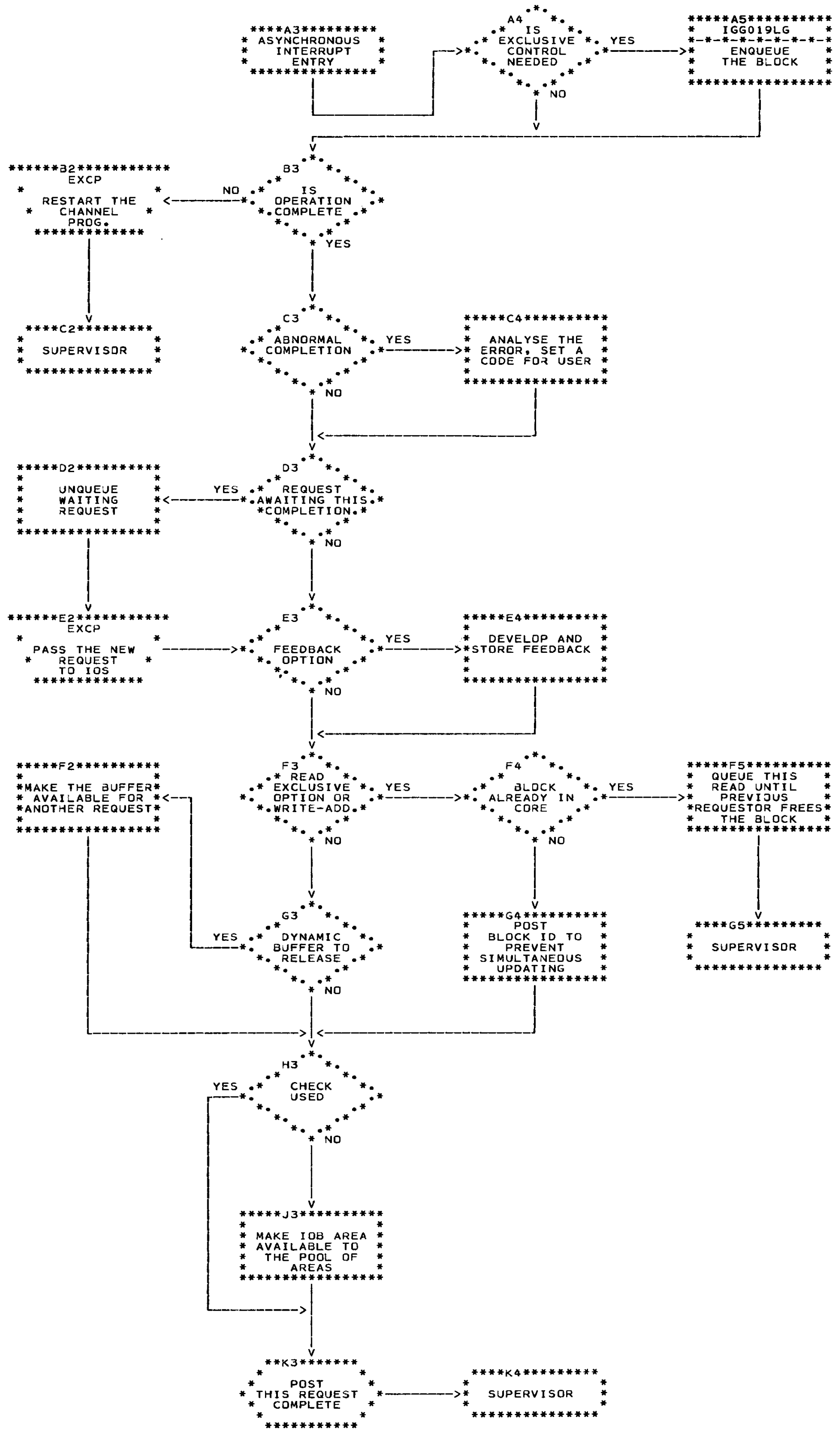
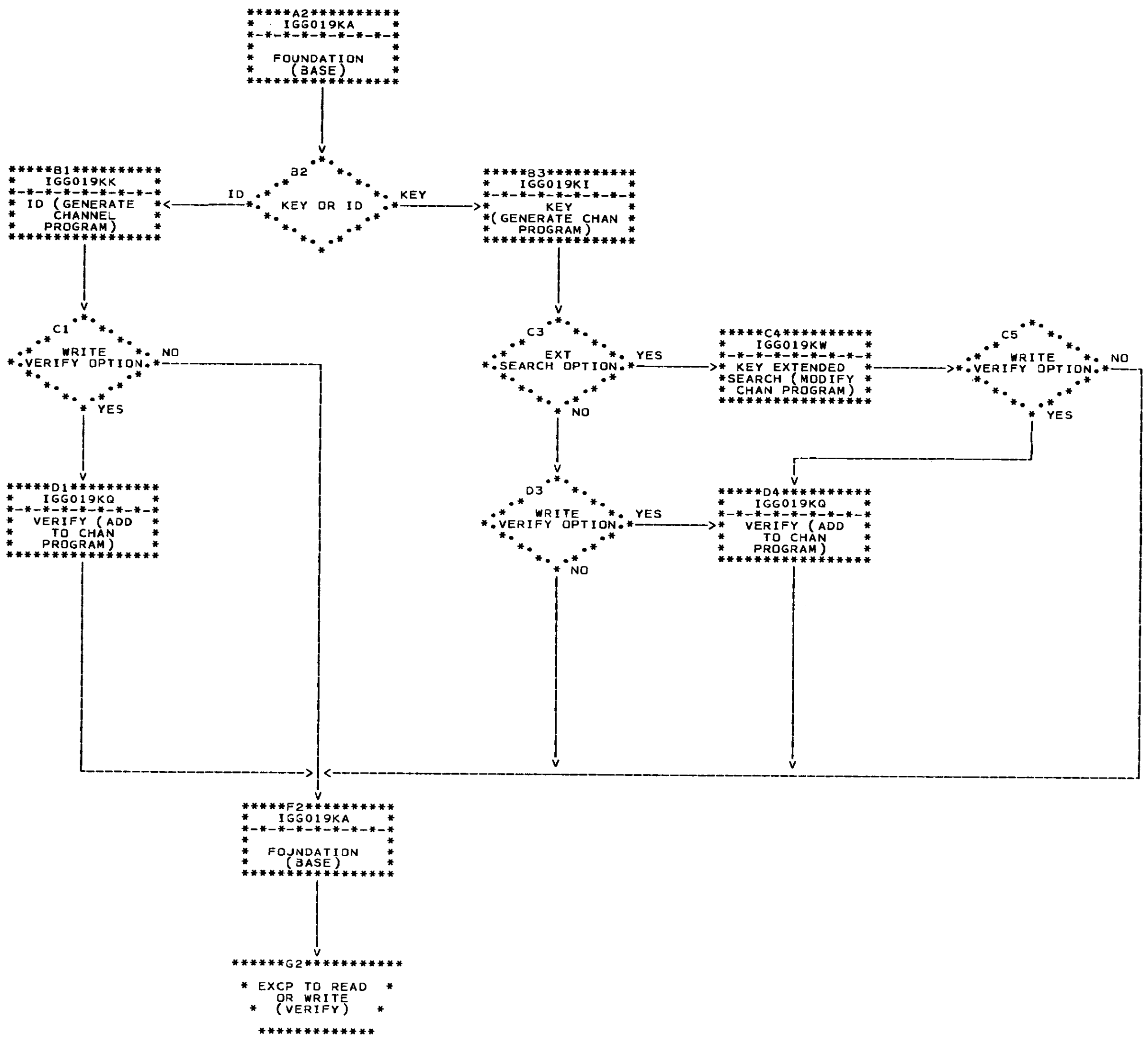
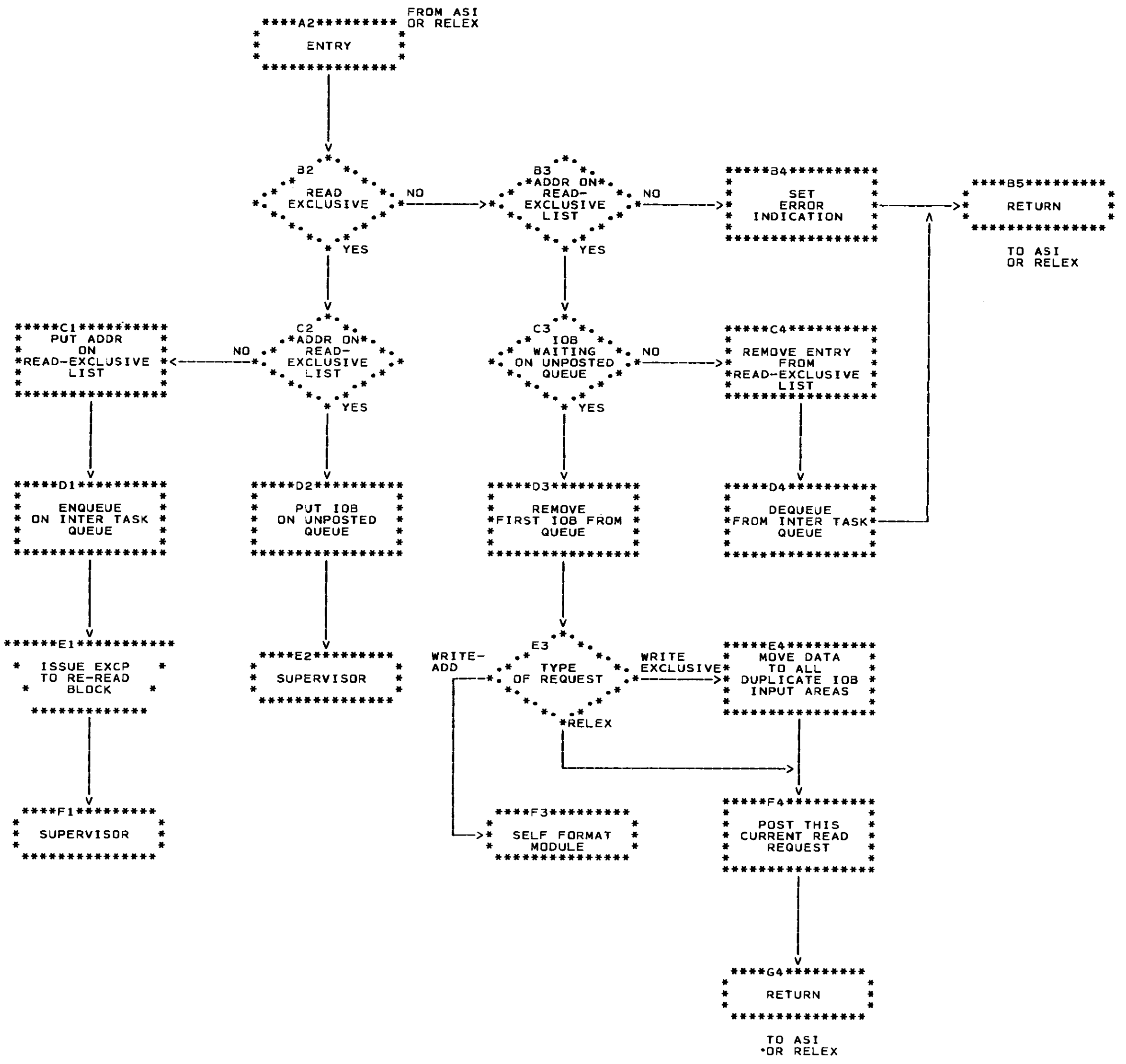

A3 ASYNCHRONOUS INTERRUPT ENTRY
A4 IS EXCLUSIVE CONTROL NEEDED
YES
NO
A5 IGG019LG ENQUEUE THE BLOCK
B2 EXCP RESTART THE CHANNEL PROG.
B3 IS OPERATION COMPLETE
NO
YES
C2 SUPERVISOR
C3 ABNORMAL COMPLETION
YES
NO
C4 ANALYSE THE ERROR, SET A CODE FOR USER
D2 UNQUEUE WAITING REQUEST
D3 REQUEST AWAITING THIS COMPLETION
YES
NO
E2 EXCP PASS THE NEW REQUEST TO IOS
E3 FEEDBACK OPTION
YES
NO
E4 DEVELOP AND STORE FEEDBACK
F2 MAKE THE BUFFER AVAILABLE FOR ANOTHER REQUEST
F3 READ EXCLUSIVE OPTION OR WRITE-ADD
YES
NO
F4 BLOCK ALREADY IN CORE
YES
NO
F5 QUEUE THIS READ UNTIL PREVIOUS REQUESTOR FREES THE BLOCK
G3 DYNAMIC BUFFER TO RELEASE
YES
NO
G4 POST BLOCK ID TO PREVENT SIMULTANEOUS UPDATING
G5 SUPERVISOR
H3 CHECK USED
YES
NO
J3 MAKE IOB AREA AVAILABLE TO THE POOL OF AREAS
K3 POST THIS REQUEST COMPLETE
K4 SUPERVISOR

Chart 03. Module Flow for Block Updating



•Chart 04. Exclusive Control - MVT System


FROM ASI OR RELEX
A2 ENTRY
B2 READ EXCLUSIVE
NO
YES
B3 ADDR ON READ-EXCLUSIVE LIST
NO
YES
B4 SET ERROR INDICATION
B5 RETURN
TO ASI OR RELEX
C1 PUT ADDR ON READ-EXCLUSIVE LIST
C2 ADDR ON READ-EXCLUSIVE LIST
NO
YES
C3 IOB WAITING ON UNPOSTED QUEUE
NO
YES
C4 REMOVE ENTRY FROM READ-EXCLUSIVE LIST
D1 ENQUEUE ON INTER TASK QUEUE
D2 PUT IOB ON UNPOSTED QUEUE
D3 REMOVE FIRST IOB FROM QUEUE
D4 DEQUEUE FROM INTER TASK QUEUE
E1 ISSUE EXCP TO RE-READ BLOCK
E2 SUPERVISOR
E3 TYPE OF REQUEST
WRITE-ADD
WRITE EXCLUSIVE
RELEX
E4 MOVE DATA TO ALL DUPLICATE IOB INPUT AREAS
F1 SUPERVISOR
F3 SELF FORMAT MODULE
F4 POST THIS CURRENT READ REQUEST
G4 RETURN
TO ASI OR RELEX

IOB

The IOB is constructed by the foundation module base component. The fields of the IOB are constructed dynamically 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 during the life of the associated request.

When a request is completed, the associated IOB is either replaced in the pool or assigned to the pool for the first time, depending on how the IOB was obtained for the request. If the IOB was obtained from the pool, it is returned to its former position in the pool by setting the availability byte in the IOB to '0'. If the IOB was obtained by the getmain routine, it is placed in the pool according to its 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 control system by the freemain routine at the time the DCB is closed.

Note: If the first usage of an IOB storage area occurs with an invalid request, then the area is returned to the system by using the freemain routine rather than being placed on the pool when the request completes.

Various BDAM routines use some of the fields in the IOB as temporary work areas until such time as these fields are filled in with information as described in Table 7.

There are three main sections to the IOB as used by the basic direct access method. (See Figure 9.) The first part is a standard 40-byte section and is described in the publication IBM System/360 Operating System: System Control Blocks, Form C28-6628. 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 second part of the IOB is a 40-byte section that contains information needed by BDAM to process the related request. The 11 fields in this part are described in Table 7.

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

```
         +0        +1        +2        +3        +4        +5        +6        +7

     ┌─────────┬─────────┬──────────────────┬─────────────────────────────────────┐     ^
  +0 │ Flag 1  │ Flag 2  │      Sense       │              IOBECBPT               │     |
     ├─────────┴─────────┴──────────────────┴─────────────────────────────────────┤     |
  +8 │                          Channel Status Word                               │     |
     ├─────────┬────────────────────────────┬─────────────────────────────────────┤     |
 +16 │         │     Channel-Program        │              IOBDCBPT               │ Standard
     │         │     Starting Address       │                                     │    IOB
     ├─────────┴────────────────────────────┼───────────────────┬─────────────────┤     |
 +24 │               IOBRESTR               │     IOBINCAM      │    IOBERRCT     │     |
     ├──────────────────────────────────────┴───────────────────┴─────────────────┤     |
 +32 │                                 IOBSEEK                                    │     |
     │   (M        B         B         C         C         H         H        R)  │     v
     ├───────────────────┬──────────────────┬─────────┬───────────────────────────┤     -
 +40 │     IOBDBYTR      │     IOBDIOBS     │ IOBDAVLI│          IOBDPLAD         │     ^
     ├───────────────────┼──────────────────┼─────────┴───────────────────────────┤     |
 +48 │     IOBDTYPE      │     IOBDSTAT     │              IOBDCPND               │     |
     ├───────────────────┼──────────────────┼─────────────────────────────────────┤     |
 +56 │     IOBDBYTN      │                  │              IOBDQPTR               │     |
     ├───────────────────┴──────────────────┴─────────────────────────────────────┤     |
 +64 │                                 IOBUPLIM                                   │   BDAM
     ├────────────────────────────────────────────────────────────────────────────┤ Extension
 +72 │                                 IOBDNRCF                                   │    to
     ├────────────────────────────────────────────────────────────────────────────┤    IOB
 +80 │                                 CHANNEL                                    │     |
   . │                                 PROGRAM                                    │     |
   . │                                                                            │     |
   . │                                                                            │     |
   . │ (Length varies according to channel program.  Refer to Appendix C.)        │     |
   . │                                                                            │     |
   . │                                                                            │     |
   . └────────────────────────────────────────────────────────────────────────────┘     v
```

Figure 9. Fields of the IOB for BDAM

Table 7. Fields, Field Size, and Field Contents of the IOB for BDAM (Part 1 of 4)

Field	Field Size (in bytes)	Field Contents and Comments
IOBDBYTR	2	Number of unused bytes remaining on a track on which a new variable-length block or a new 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 input/output operation is complet-ed.
IOBDIOBS	2	Overall size of the IOB, specified in bytes. The base component 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 a hexadecimal FF to indicate that the IOB is being used (i.e., unavailable). This is done by the base component of the foundation module. When an input/output operation is posted as complete, and an IOB is either returned to or placed on the pool of available IOBs, the value of the IOBDAVLI field is set to zero. Depending on the cause of the completion of the I/O operation, the zero value is set by either the asynchronous interrupt component or the invalid request routine of the foundation module.

Table 7. Fields, Field Size, and Field Contents of the IOB for BDAM (Part 2 of 4)

Field	Field Size (in bytes)	Field Contents and Comments
IOBDPLAD	3	Address of the next IOB area in the pool of IOBs attached to the current DCB. If there are no more IOBs on the pool, the value of this field is zero. Each IOB on the pool became a member of the pool after the first usage 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 request type and indication of any options specified in the DECB related to the request. The contents of the DECTYPE field (see DECB block) 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 the bit is set to '0', the interpretation is not in effect.) First Byte: Bit 0: Indicates verification of written block is desired. Bit 1: Indicates track overflow (i.e., overflow blocks are being used). (Refer to the publication IBM System/360 Operating System: Sequential Access Methods, Program Logic Manual.) Bit 2: Indicates extended search is desired. Bit 3: Indicates feedback of block address is desired. Bit 4: Indicates actual block addressing is being used. Bit 5: Indicates dynamic buffering is being used. Bit 6: Indicates exclusive control is being used. Bit 7: Indicates relative block addressing is being used. Note: If both bit four and bit seven are 0, the interpretation is that relative track addressing is being used. Second Byte: Bit 0: Indicates that an '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: Indicates that an 'S' has been specified in the length operand of the READ or WRITE macro instruction. The setting of this bit is ignored (i.e., not tested) when writing variable-length blocks since the block length is given in the first two bytes of the data field. Bit 2: Reserved for future use. Bit 3: Reserved for future use. Bit 4: Indicates a read request. (A '0' indicates a write request.) Bit 5: Indicates the search argument is the block key. (A '0' indicates the search argument is the block ID.) Bit 6: Indicates a write request to add a new block. Bit 7: Reserved for future use.
IOBDSTAT	2	Indication of status of the related 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 the bit is set to '0', the interpretation is not in effect.)

Table 7. Fields, Field Size, and Field Contents of the IOB for BDAM (Part 3 of 4)

Field	Field Size (in bytes)	Field Contents and Comments
IOBDSTAT (Cont.)	2	First Byte: Bit 0: Indicates an abnormal completion of the request. See second byte for details. Bit 1: On extended search, this indicates that the ASI routine is to issue the EXCP macro instruction after the end-of-extent appendage has determined that the next extent is on a new volume. The end-of-extent appendage 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 in order 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: Indicates that the read-exclusive request related to this IOB has been placed on an inter-task queue by the exclusive control module. Bit 5: Indicates that a buffer has been assigned to this IOB. Bit 6: Indicates that 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: Indicates to dynamic buffer module that it was entered from, and is to return control to, the start I/O appendage module.
IOBDSTAT	2	Second Byte: This byte contains indications of an abnormal completion of a request. When the request is posted as complete, these indications are placed in byte 1 (the second byte) of the DECSDECB field of the DECB. Bit 0: Indicates that the requested block was not found on the indicated track. Bit 1: Indicates that the length of the block was incorrect. (Refer to the section "Channel End Appendage Module.") Bit 2: Indicates that no space was found in which to write a new block. Bit 3: Reserved for future use. Bit 4: Indicates that a read operation (either to bring data into main storage or as a verification of written data) has resulted in a data check error that has not been corrected by the standard IOS error retry procedure. (Refer to the section "Verification Program.") Bit 5: Indicates that the request has been completed but that 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 zero). (Refer to the section "Channel End Appendage Module.") Bit 6: Indicates an error that cannot be attributed to any other cause as indicated by the bits in this byte. Bit 7: Indicates no match has been found on the read-exclusive list.

Table 7. Fields, Field Size, and Field Contents of the IOB for BDAM (Part 4 of 4)

Field	Field Size (in bytes)	Field Contents and Comments
IOBDCPND	4	The main storage address of the expected end of the related channel program if the program goes to a normal completion. This address is placed in IOBDCPND by the base component of the foundation module. At the completion of a request, the I/O supervisor routine places an address in the channel status word. This address is equal to the address of the last channel command word executed plus eight bytes. A normal completion is indicated if the two addresses are equal and there have been no error indications.
IOBDBYTN	4	Indication of 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 this module by the ASI routine.
IOBDQPTR	4	Address of the next IOB on the dynamic buffer queue of IOBs. This value is determined either by the dynamic buffer routine.
IOBUPLIM	8	Address to be used as the location in 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, this field indicates 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 either variable-length records or records of undefined length is to be added to a track.

DECB

The DECB results from the expansion of either a READ or a WRITE macro instruction. The contents of the fields of the DECB as they relate to BDAM are explained in the publication IBM System/360 Operating System: Supervisor and Data Management Macro Instructions. A summary of the contents of the fields is given in Table 8, with reference to Figure 10.

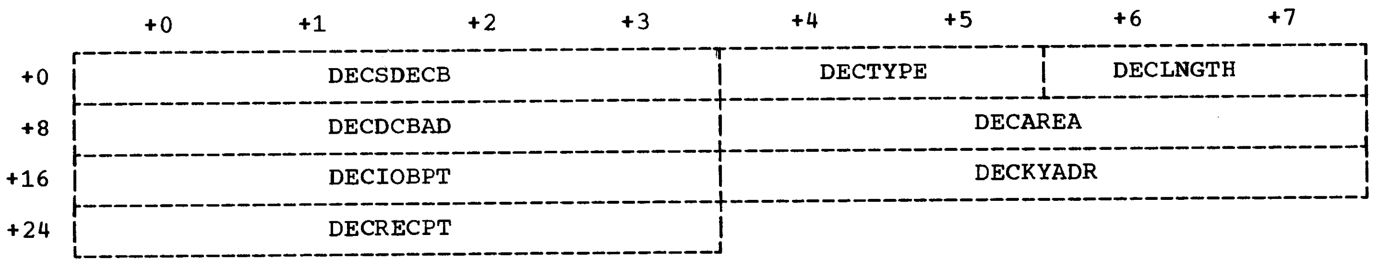


Figure 10. Fields of the DECB for BDAM

Table 8. Fields, Field Size, and Field Contents of the DECB for BDAM

Field	Field Size (in bytes)	Field Contents	
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	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 as the type of request to which the DECB refers.	
		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' of the dummy record.)
DECRECPT	4	Address of the blkref field.	

DEB

The DEB is constructed during Phase 1 of the BDAM open executor program. The fields of the DEB that are specifically related to BDAM are the DEBAMLNG field, the DEBNMTRK field, and the fields of the relative extent areas. (See Figure 11.) A more complete description of the DEB is contained in the publication IBM System/360 Operating System: System Control Blocks.

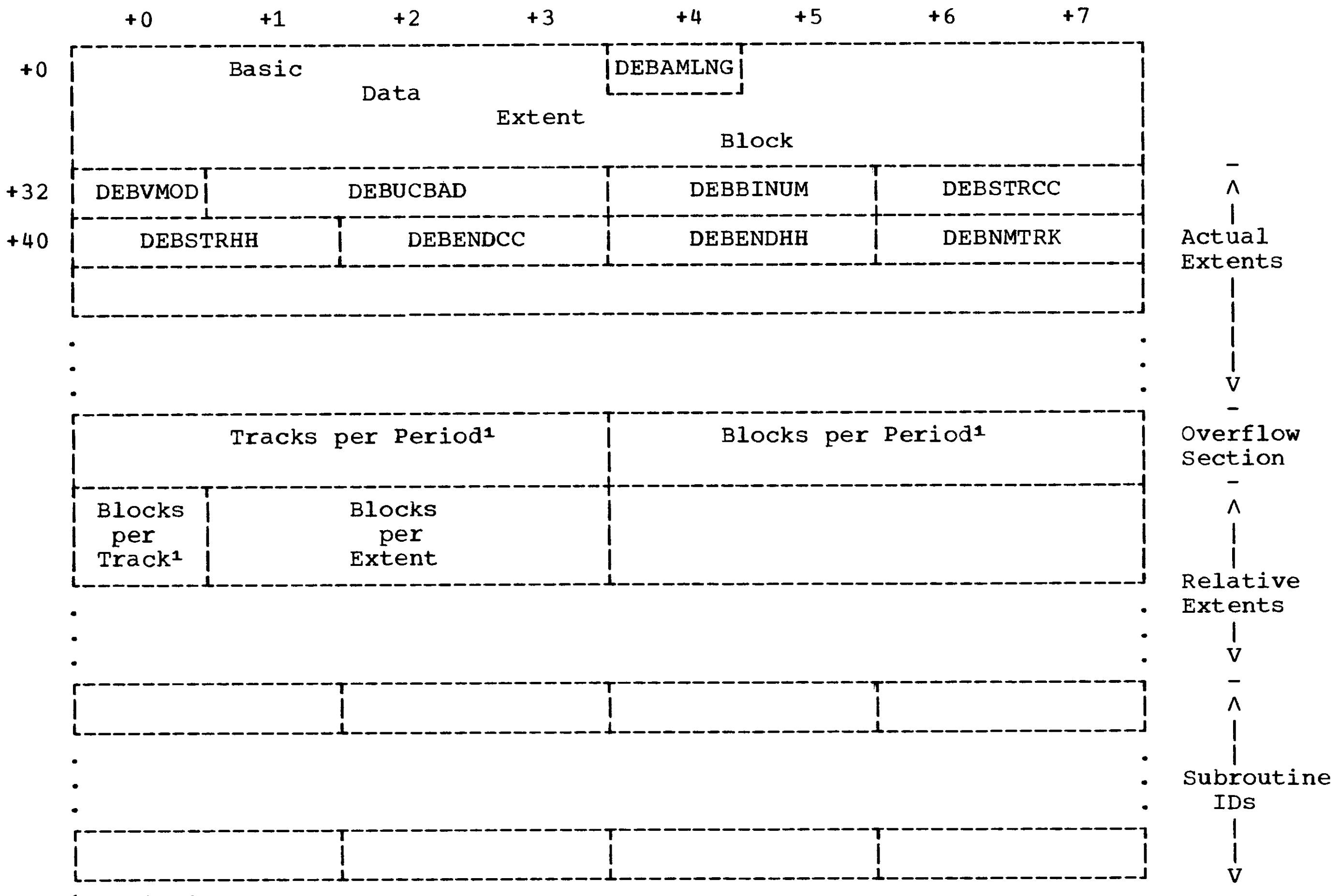
The DEBAMLNG field is a one-byte field containing the number of words of main storage that contain the relative extent areas.

The DEBNMTRK field is a two-byte field containing the number of tracks in the corresponding actual extent.

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

If track overflow has not been specified, each relative extent area consists of a one-byte field that contains the number of blocks on a track (for the device used) and a three byte field that contains the number of blocks in the extent. This latter value is obtained by multiplying the number of tracks in the extent (given as the value in the last two bytes of the associated actual extent) by the number of blocks on a track.

If track overflow has been specified, each relative extent area consists of only a three-byte "blocks per extent" field. The byte preceding each "blocks per extent" field is unused. In addition, two one-word fields constituting an overflow section are inserted between the last actual extent area and the first relative extent area. The values in these fields are based on the size of the period that is calculated by phase 3 of the BDAM open program.



¹See text.

Figure 11. Fields of the DEB for BDAM

DCB

The DCB (see Figure 12) contains information relating to the current use of a data set. A more complete description of the DCB is contained in the publication IBM System/360 Operating System: System Control Blocks. The fields of the DCB that are of particular interest to programmers concerned with BDAM applications are indicated in Table 9.

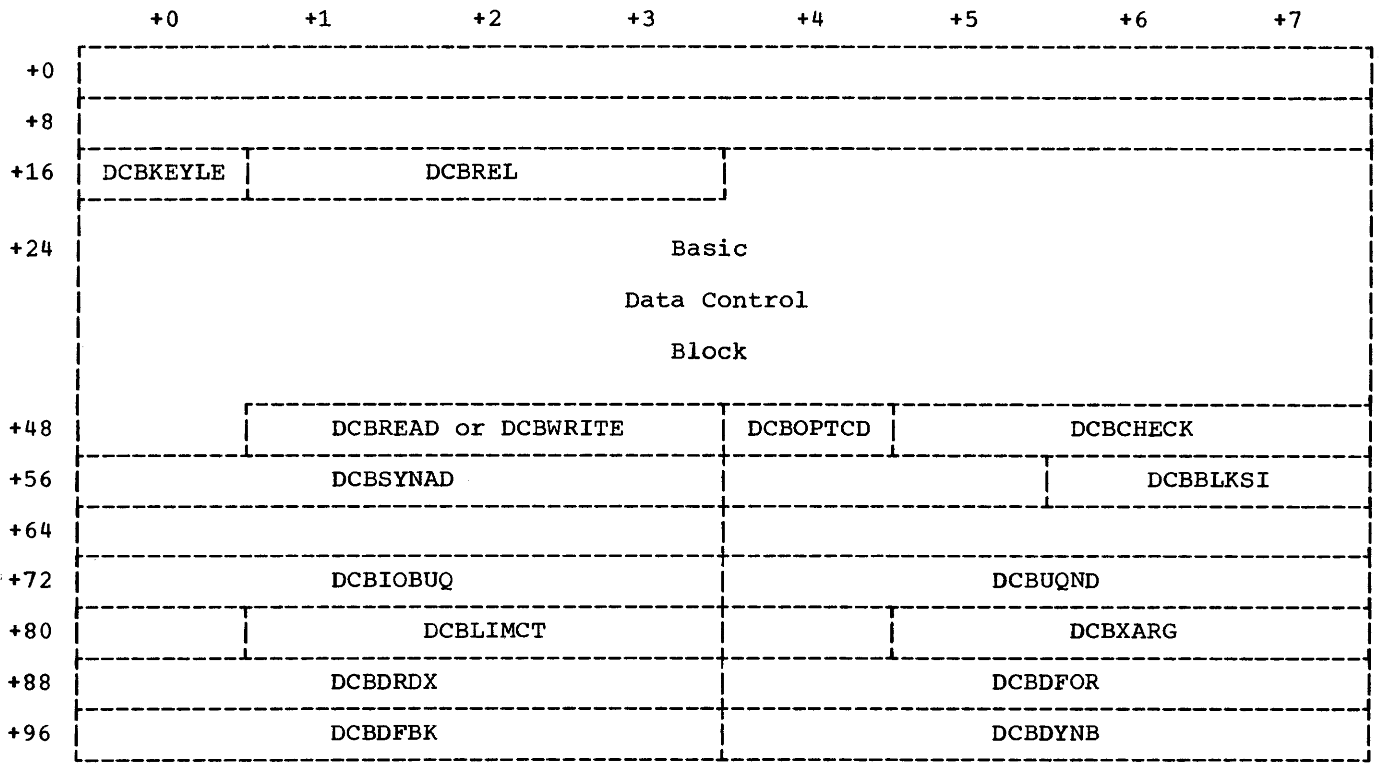


Figure 12. Fields of the DCB for BDAM

Table 9. Fields, Field Size, and Field Contents of the DCB for BDAM

Field	Field Size (in bytes)	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 the BDAM open executor phase 2 routine, and it can be used by the processing program in the process of conversion of a relative address.
DCBREAD or DCBWRITE	3	Address of the BDAM Foundation module, IGG019KA.
DCBOPTCD	1	Indication of options specified for the data set. Significant 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 the bit is set to '0', the interpretation is not in effect.)

(Continued)

Table 9. Fields, Field Size, and Field Contents of the DCB for BDAM (Continued)

Field	Field Size (in bytes)	Contents and Comments
DCBOPTCD (Cont'd.)		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.[1] 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.[1] [1]If neither actual addressing nor relative block addressing has been specified (i.e., if bits four and seven 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 on the unposted queue.
DCBUQND	4	Address of the last IOB on 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.
DCBDRDX	4	Address of the exclusive control module, IGG019LG.
DCBDFOR	4	Address of the format channel program generating module that is required for the block format indicated in the DCB macro instruction. This address is placed in DCBDFOR by the BDAM open executor phase 2 routine.
DCBDFBK	4	Address of the feedback module, IGG019KG. This address is used only for relative block feedback specification.
DCBDYNB	4	Address of the dynamic buffer module, IGG019LE.

The initial value of each of the address fields in Table 9 is '00000001.' When the data set is opened, the value of each address field 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.'

BCB

The buffer control block (BCB) is built if the dynamic buffering option has been specified. Phase 3 of the BDAM open executor routines obtains contiguous main storage for both the BCB and the required number of buffers. The BCB is initialized by the open executor phase 3 routine, but subsequent entries are placed in the BCB by the dynamic buffering module. The main storage area for both the BCB and the buffers is released by the BDAM close executor routine. Figure 13 depicts the fields of the BCB, and Table 10 describes the field contents.

	+0 +1 +2 +3	+4 +5 +6 +7
+0	BCBFRQT	BCBFRQB
+8	BCBNABFR	BCBTBFRS

Figure 13. Fields of the BCB for BDAM

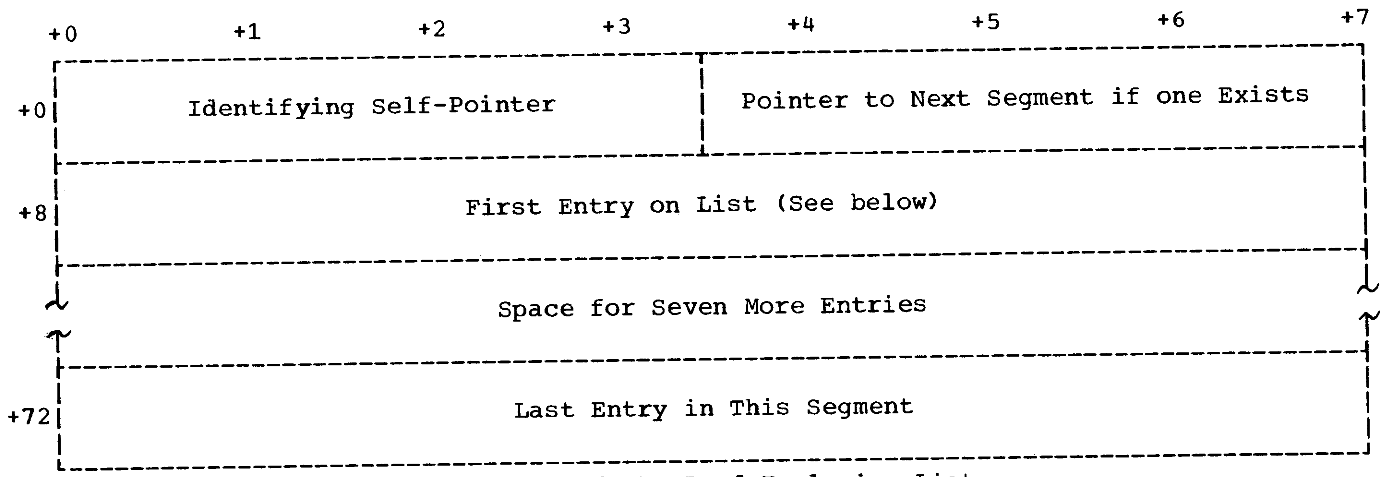
Table 10. Fields, Field Size, and Field Contents of the BCB for BDAM

Field	Field Size (in bytes)	Field Contents and Comments
BCBFRQT	4	Contains the address of the first IOB waiting to be assigned a buffer from the buffer queue. The dynamic buffer module inserts this address.
BCBFRQB	4	Contains the address of the last IOB waiting to be assigned a buffer from the buffer queue. The dynamic buffer module inserts this address.
BCBNABFR	4	Contains the address of the next buffer that is available for assignment to an IOB. Initially,. the open executor phase 3 module inserts this address. Subsequent addresses are inserted by the dynamic buffer module.
BCBTBRS	4	Contains an indication of the total size of the buffer pool and the buffer control block. The open executor phase 3 module inserts this value.

READ-EXCLUSIVE LIST SEGMENT

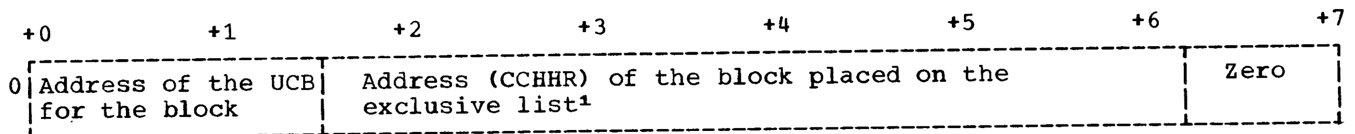
The read-exclusive list is composed of one or more 80-byte segments of storage. Each segment contains identifying and chaining information and has space for nine 8-byte entries that identify the blocks for which exclusive control is required. Phase 1 of the BDAM open executor routine requests storage for the initial 80-byte segment, and if additional segments are required, the read-exclusive module, IGG019LG, requests the storage. The BDAM close executor routine releases the storage area(s) that may have been obtained for the read-exclusive list segments.

Figure 14 indicates the contents of the fields of a typical segment of the read-exclusive list. Figure 15 indicates the contents of the fields of each of a segment's nine entries representing members of the read-exclusive list.



+0
+1
+2
+3
+4
+5
+6
+7
+0
Identifying Self-Pointer
Pointer to Next Segment if one Exists
+8
First Entry on List (See below)
Space for Seven More Entries
+72
Last Entry in This Segment

Figure 14. Description of a Segment of the Read-Exclusive List



+0
+1
+2
+3
+4
+5
+6
+7
0
Address of the UCB for the block
Address (CCHHR) of the block placed on the exclusive list[1]
Zero

[1]This is the address of the track capacity record (R0) in the case of write-add requests for variable length or undefined length blocks.

Figure 15. Description of an Entry in the Read-Exclusive List

In Table 11, the BDAM modules are listed alphabetically according to their coding (listing) names. Opposite each module coding name is the functional name of the module.

In Table 12, the module listed at the head of each column may use the module(s) listed underneath it as a subroutine. After the subroutine functions have been performed, the subroutine returns program control to either the module listed at the head of the column or, in the case of module IGG018KQ, to the foundation module.

In Table 13, options that may be specified in the MACRF field, the OPTCD field, and the RECFM field of a DCB macro instruction are related to the BDAM modules that are required to fulfill the options.

Table 11. Coding and Functional Names of BDAM Modules

Coding Name	Functional Name
IGG019KA	Foundation Module
IGG019KC	Relative Track Conversion Module
IGG019KE	Relative Block Conversion Module without Track Overflow
IGG019KF	Relative Block Conversion Module with Track Overflow
IGG019KG	Relative Block Feedback Module without Track Overflow
IGG019KH	Relative Block Feedback Module with Track Overflow
IGG019KI	Channel Program Generating Module for Searching by Block Key
IGG019KK	Channel Program Generating Module for Searching by Block ID
IGG019KM	Channel Program Generating Module for writing New Blocks of Variable- or Undefined-Length Records
IGG019KO	Channel Program Generating Module for writing New Blocks of Fixed-Length Records
IGG019KQ	Channel Program Generating Module for Verification
IGG019KS	Start I/O Appendage Module
IGG019KU	Channel End Appendage Module
IGG019KW	Key Extended-Search Module
IGG019KY	Self-Format Extended-Search Module
IGG019LA	Pre-Format Extended-Search Module
IGG019LC	End-of-Extent Appendage Module
IGG019LE	Dynamic Buffering Module
IGG019LG	Exclusive Control Module
IGG019LI	Check Module
IGG0193A	Open Executor Phase 1 Module
IGG0193C	Open Executor Phase 2 Module
IGG0193E	Open Executor Phase 3 Module
IGG0203A	Close Executor Module

Table 12. Passage of Control Among BDAM Modules

IGG019KA	IGG019KI	IGG019KK	IGG019KM	IGG019KO	IGG019KS	IGG019KW	IGG019KY	IGG019KH
IGG019KC	IGG019KW	IGG019KQ	IGG019KY	IGG019LA	IGG019LE	IGG019KQ	IGG019LC	IGG019KF
IGG019KE	IGG019KQ		IGG019KQ	IGG019KQ				
IGG019KF			IGG019LG					
IGG019KG								
IGG019KH								
IGG019KI								
IGG019KK								
IGG019KM								
IGG019KO								
IGG019LG								

Table 13. BDAM Modules Required to Satisfy DCB Macro Instruction Options

BDAM Modules Required ---> Operand	Option		IGG019KC	IGG019KE	IGG019KF	IGG019KG	IGG019KH	IGG019KI	IGG019KK	IGG019KM	IGG019KO	IGG019KQ	IGG019KW	IGG019KY	IGG019LA	IGG019LC	IGG019LE	IGG019LG	IGG019LI	Comments
MACRF	WA										X									Used when block length fixed
										X								X		Used when block length is variable or undefined.
	C																		X	When CHECK macro instruction is encountered
	RI or WI								X											
	RK or WK							X												
	RX																	X		
	RS																X			
OPTCD	A																			None of those listed
	F		X																	Feedback is to be in form of relative track
		RECFM is																		
		FT					X													Feedback is to be in form of relative block
		F				X														
	R	FT			X															
		F		X																
	W											X								
	Neither A nor R		X																	Relative track addressing is assumed
OPTCD is E	MACRF is	RECFM is																		This section is for the extended search options
	K							X					X			X				
	A	F									X				X	X				
	A	U or V								X				X		X		X		
	K AND A	F						X			X		X		X	X				
	K and A	U or V						X		X			X	X		X				

The channel program for each request using BDAM is constructed by the appropriate 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 of where data is to be placed or found; this is for a Read or a Write command word.
- The location of the search argument; this is for a Search command word.
- 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:

C. Command chaining.

D. Data chaining between gaps of a record.

K. Skip the transferring of data.

S. Suppress incorrect length indication.

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

The function or 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 (C) flag bit set on.

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

CCW No.	Command Code	Address	Flags		Count	Comments
1.	Search ID Equal	IOBSEEK + 3	C	000	5	Search for an equal CCHHR.
2.	TIC	CCW 1				Transfer if unequal.
3.[1]	Read (or Write) Key-Data	Key Address[2]	D	000	Key Length	Read (or Write) Key portion.
4.	Read (or Write) Data	Area Address[2]		000	Data Length	Read (or Write) Data portion.
5.[3] [4]	Seek cylinder head (CCHH)	IOBDNRCF	C	000	6	Seek back to track containing beginning of block.
6.	Search ID Equal	IOBSEEK + 3	C	000	5	Locate the block just updated.
7.	TIC	CCW 5				Transfer if unequal.
8.	Read Key-Data		S,K	000	256	Read to validity check.

[1]CCW 3 is omitted if either the field DCBKEYLE or the field DECKYADR is zero.
[2]This address is obtained from the DECB.
[3]CCWs 5-8 are included only if the field DCBOPTCD specifies the write-validity-check option.
[4]This CCW is present only if track overflow has been specified.

Channel Program for Reading or Writing by Block Key (Type DK)

CCW No.	Command Code	Address	Flags		Count	Comments
1.	Read Count	IOBDNRCF + 2	C,S	000	5	Read CCHHR for feedback.
2.	Search Key Equal	Key Address[1]	C,S	000	Key Length	Search for an equal Key.
3.	TIC	CCW 1				Transfer if unequal.
4.	Read (or Write) Data	Area Address[1]		000	Data Length	Read (or Write) Data portion.
5.[2] [3]	Seek Cylinder head (CCHH)	IOBDNRF	C	000	6	Seek back to track containing beginning of block.
6.	Search Key Equal	Key Address[1]	C,S	000	Key Length	Locate the block just updated.
7.	TIC	CCW 5				Transfer if unequal.
8.	Read Data		S,K	000	256	Read to validity check.

[1]This address is obtained from the DECB.
[2]CCWs 5-8 are included only if the field DCBOPTCD specifies the write-validity-check option.
[3]This CCW is present only if track overflow has been specified.

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

CCW No.	Command Code	Address	Flags		Count	Comments
1.	Read Count	IOBDNRCF + 2	C,S	000	5	Read CCHHR for feedback.
2.	Search Key Equal	Dummy Key	C,S	000	1	Search for a dummy record.
3.	TIC	CCW 1				Transfer if unequal.
4.	Read Data	IOBDNRCF + 6	C,S	000	1	Read dummy record's position (i.e., R)
5.[1]	Seek cylinder head (CCHH)	IOBDNRCF	C	000	6	Seek back to track containing beginning of block.
6.	Search ID Equal	IOBDNRCF + 2	C	000	5	Search for the dummy record.
7.	TIC	CCW 5				Transfer if unequal.
8.	Write Key-Data	Key Address[2]	C	000	Key Length	Update the Key portion.
9.	Write Data	Area Address[2]		000	Data Length	Update the Data portion.
10.[1] [3]	Seek cylinder head (CCHH)	IOBDNRCF	C	000	6	Seek back to track containing beginning of block.
11.	Search ID Equal	IOBDNRCF + 2	C	000	5	Locate the block just written.
12.	TIC	CCW 9				Transfer if unequal.
13.	Read Key-Data		S,K	000	256	Read to validity check.

[1]This CCW is present only if track overflow has been specified.
[2]This address is obtained from the DECB.
[3]CCWs 10-13 are included only if the field DCBOPTCD specifies the write-validity-check option.

Channel Program for Writing a New Block of Variable-Length Records or Undefined-Length Records (Type DA)

CCW No.	Command Code	Address	Flags		Count	Comments
1.	Search ID Equal	IOBSEEK + 3	C	000	5	Search for track capacity record.
2.	TIC	CCW 1				Transfer if unequal.
3.	Read Data	IOBDNRCF	S	000	7	Read capacity record into IOB.
4.	Search ID Equal	IOBUPLIM	C	000	5	Locate track capacity record.
5.	TIC	CCW 4				Transfer if unequal.
6.	Write Data	IOBSEEK + 3	C,S	000	7	Update capacity record.
7.	Search ID Equal	CCW 1[1]	C	000	5	Locate current last block on track.
8.	TIC	CCW 7				Transfer if unequal.
9.	Write Count-Key-Data	IOBDNRCF	D	000	8	Write new block.
10.[2]	Write Count-Key-Data	Key Address[3]	D	000	Key Length	
11.[4]	Write Count-Key-Data	Area Address[3]		000	Data Length	
12.[5]	Search ID Equal	IOBSEEK + 3	C	000	5	Locate new block.
13.	TIC	CCW 12				Transfer if unequal.
14.	Read Key-Data		C,S,K	000	256	Read to validity check the block.
15.	Read R0		K	000	16	Read to validity check the capacity record.

[1]The IOB area that initially contained CCW 1 has been overlaid with 5 bytes (the CCHHR part) of the capacity record that was read by CCW 3.
[2]CCW 10 is omitted if keys are not present in the block format.
[3]This address is obtained from the DECB.
[4]CCW 11 is omitted if Data Length is 0 (i.e., end-of-data-set mark).
[5]CCWs 12-15 are included only if the field DCBOPTCD specifies the write-validity-check option.

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

CCW No.	Command Word	Address	Flags		Count	Comments
1.	Search ID Equal	IOBSEEK + 3	C	000	5	Search for an equal CCHHR.
2.	TIC	CCW 1				Transfer if unequal.
3.	Multiple Track Search ID Equal	IOBUPLIM + 3	C	000	5	Stop search at limit.
4.	TIC	CCW 6				Transfer if unequal.
5.	NOP		S		1	Search limit reached.
6.	Search Key Equal	Key Address	C,S	000	Key Length	
7.	TIC	CCW 3				Transfer if unequal.
8.	Read Home Address[1]		C,S,K	000	1	
9.	Read Count	IOBDNRCF + 2	C,S	000	5	Read CCHHR for feedback.
10.	Search Key Equal	Key Address[2]	C	000	Key Length	Search for equal key.
11.	TIC	CCW 9				Transfer if unequal.
12.	Read (or Write) Data	Area Address[2]		000	Data Length	Read (or Write) data portion of block.
13.[3] [4]	Seek cylinder head (CCHH)	IOBDNRCF	C	000	6	Seek back to track containing beginning of block.
14.	Search Key Equal	Key Address[2]	C,S	000	Key Length	Locate the block just updated.
15.	TIC	CCW 13				Transfer if unequal.
16.	Read Data		S,K	000	256	Read to validity check.

[1]CCWs 8-11 are included if feedback is requested.
[2]This address is obtained from the DECB.
[3]This CCW is present only if track overflow has been specified.
[4]CCWs 13-16 are included only if the field DCBOPTCD specifies the write-validity-check option.

Channel Program for Writing a New Block of Fixed-Length Records Using Extended Search (Type DA)

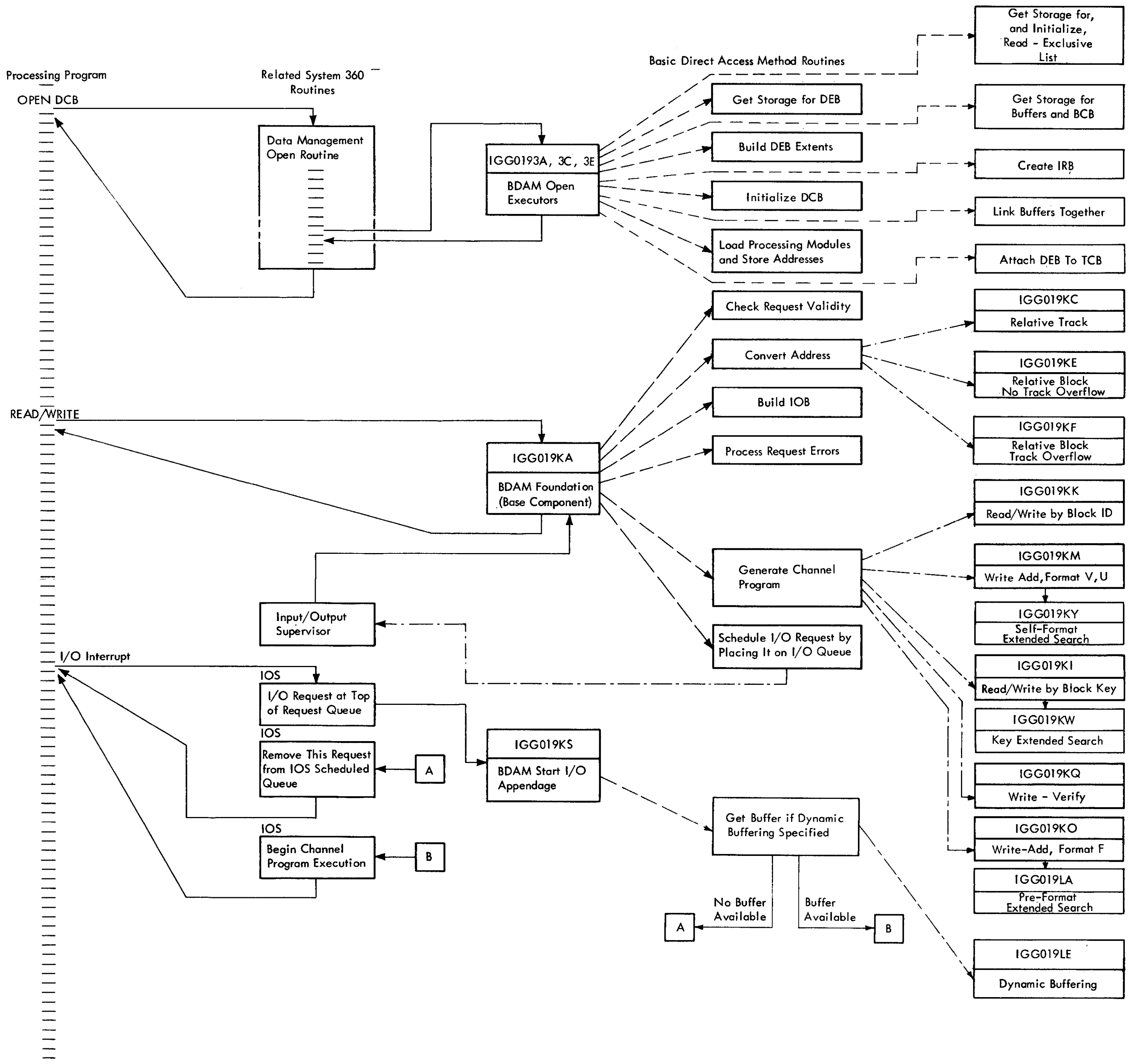
CCW No.	Command Code	Address	Flags		Count	Comments
1.	Search ID Equal	IOBSEEK + 3	C	000	5	Search for track capacity record.
2.	TIC	CCW 1				Transfer if unequal.
3.	Read Data	IOBDNRCF + 2	C,S	000	5	Read highest ID from capacity record.
4.	Search ID Equal	IOBDNRCF + 2	C	000	5	
5.	TIC	CCW 12				Transfer if unequal
6.	Search Key Equal	Dummy Key Address	C,S	000	1	Search for dummy record.
7.	TIC	CCW 9				Transfer if unequal.
8.	TIC	CCW 14				Transfer if equal.
9.	Multiple Track Search ID Equal	IOBUPLIM + 3	C	000	5	Stop search at limit.
10.	TIC	CCW 3				Transfer if unequal.
11.	NOP		S		1	Search limit reached.
12.	Search Key Equal	Dummy Key Address	C,S	000	1	
13.	TIC	CCW 4				Transfer if unequal.
14.	Read Data	IOBDNRCF + 6	C,S	000	1	Read dummy record's position (i.e., R).
15.[1]	Seek cylinder head (CCHH)	IOBDNRCF	C	000	6	Seek back to track containing beginning of block.
16.	Search ID Equal	IOBDNRCF + 2	C	000	5	Search for dummy record.
17.	TIC	CCW 15				Transfer if unequal.
18.	Write Key-Data	Key Address[2]	C	000	Key Length	Update the Key Portion.
19.	Write Data	Area Address[2]		000	Data Length	Update the Data portion.
20.[1] [3]	Seek cylinder head (CCHH)	IOBDNRCF	C	000	6	Seek back to track containing beginning of block.

(Continued)

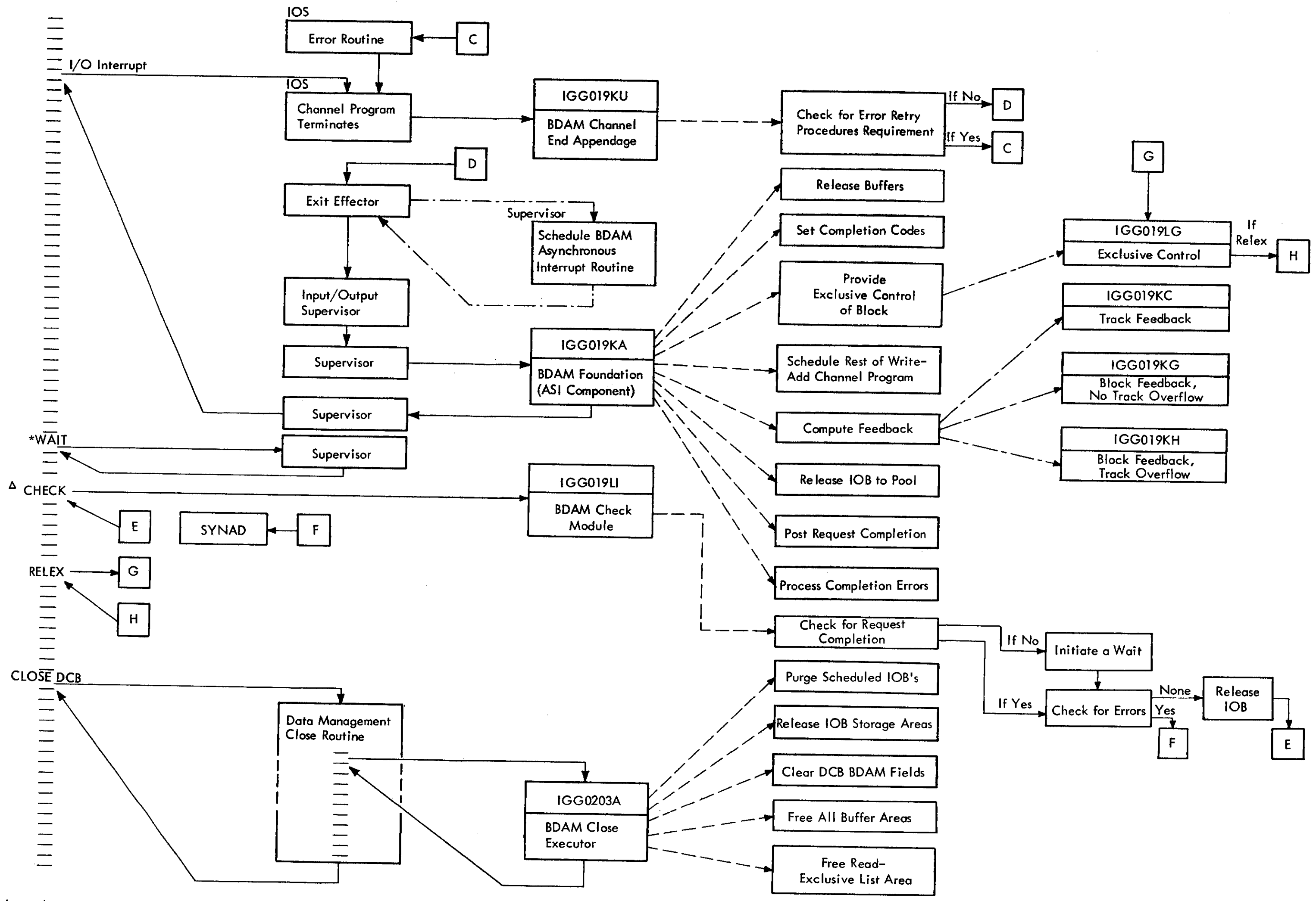
Channel Program for Writing a New Block of Fixed-Length Records Using Extended Search (Type DA) (Continued)

21.	Search ID Equal	IOBDNRCF + 2	C	000	5	Locate the block just written.
22.	TIC	CCW 19				Transfer if unequal.
23.	Read Key-Data		S,K	000	256	Read to validity check.

[1]This CCW is present only if track overflow has been specified.
[2]This address is obtained from the DECB.
[3]CCWs 20-23 are included only if the field DCBOPTCD specified the write-validity-check option.



● Figure 16. Relationship Among Processing Program, BDAM Routines, and Other Components of the Operating System



Legend

——→ Main flow of control
– – → Functions performed by main block
–·–·→ Linkage to routines that perform indicated functions and return to point of departure
IOS: Input/Output Supervisor

* 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, (i.e., before the second indicated I/O interrupt) the processing program relinquishes control until posting occurs.

△ Either a WAIT of a CHECK may be specified. (Refer to text at "Check Module".)

• Figure 16. Relationship Among Processing Program, BDAM Routines, and Other Components of the Operating System (Continued)

INDEX

TEXT REFERENCES FOR FIGURES, TABLES, AND CHARTS

Figure	Page
1	8,25
2	12
3	8,13,14,17,20
4	8,13,14,17,20,23,27
5	21
6	23
6A	24,25
7	8,25
8	33
9	39
10	44
11	45
12	46
13	48
14	49
15	49
16	8

Table	Page
1	8
2	18
3	20
4	21
5	22
6	23
7	39
8	44
9	46
10	48
11	51
12	51
13	51

Chart	Page
01	13,25
02	13
03	22
04	29

READER'S COMMENT FORM

IBM System/360 Operating System
Basic Direct Access Method
Program Logic Manual

Form Y28-6617-3

- Is the material:

	Yes	No
Easy to read?	☐	☐
Well organized?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Accurate?	☐	☐
Suitable for its intended audience?	☐	☐

- How did you use this publication?
  - ☐ As an introduction to the subject
  - ☐ For additional knowledge
  - Other ....

- Please check the items that describe your position:

☐ Customer personnel	☐ Operator	☐ Sales Representative
☐ IBM personnel	☐ Programmer	☐ Systems Engineer
☐ Manager	☐ Customer Engineer	☐ Trainee
☐ Systems Analyst	☐ Instructor	Other ....

- Please check specific criticism(s), give page number(s), and explain below:

☐ Clarification on page(s)	☐ Deletion on page(s) ....
☐ Addition on page(s)	☐ Error on page(s) ....

Explanation:

- Thank your for your cooperation. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS PLEASE . . .

This publication is one of a series which serves as reference 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. 81
POUGHKEEPSIE, N.Y.

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

POSTAGE WILL BE PAID BY

IBM Corporation
P.O. Box 390
Poughkeepsie, N.Y. 12602

Attention: Programming Systems Publications
Department D58

Fold Fold

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

Y28-6617-3

IBM S/360 Printed in U.S.A. Y28-6617-3

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]