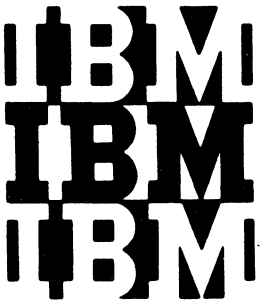


File Number S360-30
Order Number GY28-6604-5

Program Logic

OS SAM Logic

Release 21

Program Number 360S—DM—508

This manual contains a general description of the Get, Put, Read, Write, and associated modules for QSAM, BSAM, and BPAM. SAM executors and appendages are also described.

The manual is intended for use by persons involved in programming maintenance and system programmers who are altering the program design.

The SAM routines used for optical character readers, magnetic character readers, and optical reader sorters are discussed in separate publications. They are indexed in the *IBM System/360 and System/370 Bibliography,* GA22–6622.

Sixth Edition (February 1972)

This manual corresponds to OS Release 21. Changes to the information in this book may be made at any time. They will be reported in subsequent editions or technical newsletters. Before using this book in conjunction with the operation of IBM systems, consult the latest *SRL Newsletter,* GN20–0360, for the editions that are applicable and current.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.

Changes are indicated by a vertical line to the left of the change.

Forms for reader's comments appear at the back of this publication. They may be mailed directly to IBM. Address any additional comments concerning this publication to the IBM Corporation, Programming Publications, Department D78, San Jose, California 95114.

PREFACE

The information in this manual is intended for programming–support customer engineers and programmers who require specific information about queued sequential access method (QSAM), basic sequential access method (BSAM), and basic partitioned access method (BPAM) routines.

The manual is organized into six sections. With the exception of the diagrams in Section 5, which are identified in an alphabetic sequence, all illustrations are consecutively numbered throughout the manual.

Figure numbers appear as marginal tabs in Section 2. The marginal tabs identify figures that list functionally related groups of modules, appendages, or executors.

"Section 1: Introduction." This section contains a brief description of the sequential access method (SAM) routines and a reference to Diagram A, Sequential Access Method — Overview in Section 5. This diagram lists the macro statements used with SAM programming techniques and directs the reader to appropriate diagrams and figures in other parts of the manual.

"Section 2: Method of Operation." The SAM routines are described in five categories. They are:

1. Queued sequential access method (QSAM) routines that cause storage and retrieval of data records arranged in sequential order.
2. Basic sequential access method (BSAM) routines that cause storage and retrieval of data blocks arranged in sequential order.
3. Basic partitioned access method (BPAM) routines that cause storage and retrieval of data blocks in a member of a partitioned data set. They can also construct entries and search for entries in the directory of a partitioned data set.
4. Executors that operate with input/output support routines.
5. Buffer–pool management routines that furnish buffer space in main storage.

"Section 3: Directory." The directory lists the names of the sequential access method modules in alphabetical order. Each entry contains the module name, type, CSECT name, SVC entry (if any), and references to figures and appendixes in other parts of the manual that contain information about the module.

"Section 4: Data Areas." This section shows how various control blocks are used in QSAM and BSAM. The access method save area for user totaling is also described. This section does not describe in detail all fields of the system control blocks referred to in this manual. For information about system control blocks, see the *OS System Control Blocks* manual.

"Section 5: Program Organization and Flow of Control." This section contains diagrams that describe the organization and flow of control of the SAM routines. Diagrams A, B, C, and D describe some of the general characteristics of the access–method routines and contains references to other diagrams and figures in the manual. The remaining diagrams contain flow of control information for the QSAM and BSAM/BPAM routines.

"Section 6: Appendixes." This section describes code conversion routines, BSAM/QSAM channel programs, update channel programs, chained scheduling

channel programs, BSAM channel programs, and contains the ABEND codes cross–reference table.

Prerequisite Reading

For general information about the operating system:

OS MVT Guide, GC28–6720

OS MFT Guide, GC27-6939

For information about processing sequential and partitioned data sets:

OS Data Management Services Guide, GC26–3746

Related Reading

For specific information about the macro instructions required to process sequential and partitioned data sets:

OS Data Management Macro Instructions, GC26–3794.

For specific information about Open, Close, and End–of–Volume routines:

OS Open/Close/EOV Logic, GY28–6609

For information on the sequential access method routines used by the IBM 1285, 1287, and 1288 optical character readers:

OS IBM 1285, 1287, and 1288 Optical Character Reader Logic, GY21–0013

For information on the sequential access method routines used by the IBM 1419 Magnetic Character Reader and IBM 1275 Optical Reader Sorter:

OS IBM 1419 Magnetic Character Reader, 1275 Optical Reader/Sorter Logic, GY21–0012

For information about system control blocks referred to in this manual:

OS System Control Blocks, GC28–6628

For information about the resident access method modules option:

OS Data Management for System Programmers, GC28–6550

CONTENTS

ILLUSTRATIONS

Figures

Diagrams

SUMMARY OF CHANGES FOR RELEASE 21

Item	Description	Areas Affected
3505/3525 Device Support	Associated Data Set Processing New EOB Modules IGG019FK IGG019FQ IFF019FU	Section 2. End–of–Block Routines
	Channel–end and Abnormal–end Processing New Appendage IGG019C6	Section 2. Appendages
	Line Control Functions for 3525 New Control Module IGG019FA	Section 2. QSAM Control Routines and BSAM Control Routines
	Open Executor Processing New Stage 1 Executors IGG0197L IGG0197M	Section 2. Open Executors
	New Stage 2 Executors IGG0197N IGG0197P IGG0197Q	
	Close Executor Processing New Close Executors IGG201P IGG201R	Section 2. Close Executors
OS/DOS Tape Compatibility (OS/DOS Interchange)	Provisions for handling DOS embedded checkpoint records	Section 2. Get routines, Synchro–nizing–and–Error–Processing Routines, BSAM Control Routines.
	Channel–end and Abnormal–end Processing New Appendages IGG019EI IGG019EJ	Section 2. Appendages
PLM Reorganization	Organization described in Preface	All

SECTION 1: INTRODUCTION

Sequential access methods (SAM) are programming techniques for transferring data arranged in sequential order between main storage and an input/output device. This manual describes five groups of sequential access method routines. They are:

- Queued sequential access method (QSAM) routines
- Basic sequential access method (BSAM) routines
- Basic partitioned access method (BPAM) routines
- Sequential access method executors
- Buffer–pool management routines

A processing program using QSAM routines works with records. For input, QSAM routines turn the blocks of data of the channel programs into a stream of input records for the processing program; for output, QSAM routines collect the successive output records of the processing program into blocks of data to be written by channel programs. See Diagram F (Section 5) for information about the flow of control for QSAM routines.

A processing program using BSAM routines works with blocks of data. For input, BSAM routines cause a channel program to read a block of data for the processing program; for output, BSAM routines cause a channel program to write a block of data for the processing program. BSAM routines are also used to read and write blocks of data for members of a partitioned data set. See Diagram G (Section 5) for flow of control information about BSAM routines.

A processing program using BPAM routines also works with blocks of data. For output, BPAM routines construct and cause writing of entries in the directory; for input, BPAM routines search for and read entries in the directory. To read and write the blocks of the members, a processing program uses the BSAM routines. Flow of control for the BPAM routines is shown in Diagram G.

Sequential access method executors are modules that operate with the Open, Close, and end–of–volume (O/C/EOV) routines. When a data control block is opened, an executor constructs control blocks and loads the access method routines, unless the resident access method (RAM) option is used. For additional information about the resident access method modules option, see *OS Data Management for System Programmers,* GC28–6550. If the RAM option is used, the selected QSAM or BSAM routines are permanently resident. When the end of a data set or volume is reached, an executor processes the pending input/output blocks. The executors described are:

- Open executors
- Close executors
- SYNAD/FEOV/EOV executors
- SETPRT executors

Buffer–pool management routines form buffers in main storage and return main–storage space (for buffers no longer needed) to available status. A buffer–pool management routine is entered when a GETPOOL, BUILD, GETBUF, FREEBUF, or FREEPOOL macro instruction is encountered in a program.

The GETPOOL and Build routines both form a pool of buffers in main storage. However, the GETPOOL routine also obtains the main–storage space for the buffer pool. Main–storage space must be provided by the processing program when the Build routine is used.

The GETBUF and FREEBUF routines handle individual buffers. GETBUF obtains a buffer from a buffer pool and FREEBUF returns a buffer to a buffer pool.

The FREEPOOL routine returns the main–storage space used for a buffer pool.

Diagram A (see Section 5) lists the macro statements that are used with sequential access method programming techniques. The chart also refers to figures in other portions of the manual that describe the SAM routines, appendages, and executors associated with each macro statement.

SECTION 2: METHOD OF OPERATION

Queued Sequential Access Method Routines

Queued sequential access method (QSAM) routines cause storage and retrieval of records and furnish buffering and blocking facilities. There are six types of QSAM routines:

- Get routines
- Put routines
- End–of–block routines
- Synchronizing–and–error–processing routines (including the track–overflow–asynchronous–error–processing routine)
- Appendage routines
- Control routines

Diagram F, QSAM Flow of Control (see Section 5) shows the relationship of QSAM routines to other portions of the operating system and the processing program.

Get Routines

The manner in which a Get routine performs its processing depends on the buffering mode. Simple buffering Get routines determine the address of the next record by referring to the DCB. Exchange buffering Get routines determine the address of the next record by referring to the channel program.

The update mode Get routine determines the address of the next input record by referring to the DCB. The next output record is the last input record.

If the American National Standard Code for Information Interchange (ASCII) is used, the Get routine (providing it is specified in the DCB) will accept a record with a block prefix. The Get routines do not present the block prefix to the processing program. The block prefix is specified by the BUFOFF option in the DCB. For more information on block prefix and record formats for ASCII, see *OS Data Management Services Guide,* GC26–3746.

The Get routine descriptions that follow are accordingly grouped as:

- Simple–Buffering Get Routines
- Exchange–Buffering Get Routines
- Update–Mode Get Routine

Simple–Buffering Get Routines

Simple–buffering Get routines use buffers whose beginning and ending addresses are in the data control block (DCB). The beginning address is in the DCBRECAD field (address of the next record); the ending address is in the DCBEOBAD field (address of the end of the buffer). In each pass through a routine, it determines:

- The address of the next record
- Whether an input buffer is empty and ready to be scheduled for refilling
- Whether a new full input buffer is needed

If the records are unblocked, the address of the next record is always that of the next buffer.

If the records are blocked, a Get routine determines the address of the next record by adding the length of the last record to the address of the last record. The address of the last record is in the DCBRECAD field of the data control block (DCB). If the records are fixed–length blocked records, the length of each record is in the DCBLRECL field. If the records are variable–length blocked records, the length of each record is in the length field of the record itself.

A Get routine determines whether a buffer is empty and ready for refilling and whether a new full buffer is needed by testing for an end–of–block (EOB) condition.

When a buffer is empty, a Get routine passes control to an end–of–block routine to refill the buffer. The buffers are filled for the first time by Open executor IGG01911. Thus, the buffers are primed for the first entry into a Get routine.

When a new full buffer is needed, a Get routine obtains it by passing control to the input–synchronizing–and–error–processing routine, module IGG019AQ. The synchronizing routine updates the DCBIOBA field, thus pointing to the new buffer, and returns control to the Get routine. A Get routine updates the DCBRECAD field by inserting in it the starting address of the buffer from the channel program associated with the new IOB. To update the DCBEOBAD field, a Get routine adds the actual length of the block read to the buffer starting address. These two fields, DCBRECAD and DCBEOBAD, define the available buffer.

For unblocked records, an EOB condition exists after every entry into the Get routine. For blocked records, an EOB condition exists when the values in the DCBRECAD and DCBEOBAD fields are equal. In the move operating mode, the buffer can be scheduled for refilling as soon as the last record is moved out; thus, an EOB test is made after moving each record, so that the buffer can be scheduled for refilling as soon as possible. Another EOB test is made on the next entry to the routine to determine whether a new full buffer is needed. In the locate mode, the empty buffer is scheduled when the routine is entered, if the last record was presented in the preceding entry; thus, an EOB test is made on entry into the routine to determine whether a buffer is empty and ready for refilling and also whether a new full buffer is needed.

When the processing program determines that the balance of the present buffer is to be ignored and the first record of the next buffer is desired, the processing program issues a RELSE macro instruction. Control passes to a RELSE routine which sets an EOB condition. When records are spanned, one or more blocks can be skipped to find the first record in a new block.

The Open executor primes (that is, schedules for filling) the buffers if QSAM is used with a DCB opened for input, update, or readback. For the locate mode, all buffers except one are primed; for the move mode, all buffers are primed. The Open executor also sets an end–of–block condition; the first time that a Get routine gains control, it processes this condition in the usual way.

Upon return from the synchronizing–and–error–processing routine, the Get routines, which may be loaded for tape data sets, tests to determine if the buffer contains a DOS checkpoint record. If a DOS checkpoint record is indicated, ECB posted X'50', the Get routine branches to the end–of–block routine to reschedule the buffer for refilling and then branches back to the synchronizing routine to test the next buffer.

Figure 1 lists the simple buffering Get routines and the conditions that cause a particular routine to be used. The Open executor selects one of the routines, loads it, and puts its address into the DCBGET field. Figure 1 shows, for example, that when the Open parameter list specifies input and the DCB specifies the GET macro instruction, simple buffering, the locate mode, and the fixed–length record format, routine IGG019AA is selected and loaded.

Access Method Options	**Selections**																
Input, Get, Simple Buffering	X	X	X	X	X	X	X	X					X	X	X	X	X
RDBACK, Get, Simple Buffering									X	X	X	X					
Locate operating mode	X	X	X						X	X				X	X		
Move operating mode				X	X	X	X	X			X	X	X			X	
Data operating mode																	X
Fixed–length record format	X			X			X		X		X						
Undefined–length record format		X			X			X		X		X					
Variable–length or record format–D			X			X								X	X	X	X
Spanned records														X	X	X	X
Card reader, only a single, buffer CNTRL							X	X									
Character conversion for paper tape													X				
Logical record interface														X			
Get modules																	
IGG019AA	AA	AA															
IGG019AB			AB														
IGG019AC				AC	AC												
IGG019AD						AD											
IGG019AG							AG	AG									
IGG019AM									AM	AM							
IGG019AN											AN	AN					
IGG019AT[1]													AT				
IGG019BO														BO			
IGG019FB															FB		
IGG019FD																FD	
IGG019FF																	FF

[1] This module also includes the character–conversion and synchronizing–and–error–processing routine for paper–tape devices.

Figure 1. Module Selector — Simple Buffering Get Modules

Get Module IGG019AA: Module IGG019AA presents the processing program with the address of the next fixed–length or undefined–length record. The Open executor selects and loads this module if the Open parameter list specifies:

Input

and the DCB specifies:

Get

Simple buffering

Locate operating mode

Fixed–length (unblocked, blocked, or blocked standard) or undefined–length record format

The module consists of a Get routine and a RELSE routine.

The Get routine operates as follows:

- It receives control when a GET macro instruction is encountered in a processing program.
- It tests for an EOB condition to determine whether a buffer is empty and ready for refilling and if a new buffer is needed. When the Open executor primes the buffers, it schedules all buffers except one and sets an EOB condition.
- If no EOB condition exists, the Get routine determines the address of the next record, and then presents the address to the processing program and returns control to the processing program.
- If an EOB condition exists, the Get routine issues a BALR instruction to pass the present buffer to the end–of–block routine to be scheduled for refilling. The Get routine issues another BALR instruction to obtain a new full buffer through the input–synchronizing–and–error–processing routine, module IGG019AQ. The Get routine then presents the address of the first record of the new buffer to the processing program and returns control to the processing program.

The RELSE routine causes an EOB condition by setting the DCBRECAD and DCBEOBAD fields so that they are equal; it then returns control to the processing program.

Get Module IGG019AB: Module IGG019AB presents the processing program with the address of the next variable–length or format–D record. The Open executor selects and loads this module if the Open parameter list specifies:

Input

and the DCB specifies:

Get

Simple buffering

Locate operating mode

Variable–length or record format–D (unblocked or blocked), unspanned

The module consists of a Get routine and a RELSE routine.

The Get routine operates as follows:

- It receives control when a GET macro instruction is encountered in a processing program.
- It determines the address of the next record and tests for an EOB condition to determine whether a buffer is empty and ready for refilling and if a new buffer is

needed. When the Open executor primes the buffers, it schedules all buffers except one and sets an EOB condition.

- If no EOB condition exists, it presents the address of the next record to the processing program and returns control to the processing program.
- If an EOB condition exists, it issues a BALR instruction to pass the present buffer to the end–of–block routine to be scheduled for refilling. The Get routine issues another BALR instruction to obtain a new buffer through the input–synchronizing–and–error–processing routine, module IGG019AQ. The Get routine then presents the address of the first record of the new buffer to the processing program and returns control to the processing program.

The RELSE routine causes an EOB condition by setting the DCBRECAD and DCBEOBAD fields so that they are equal; it then returns control to the processing program.

Get Module IGG019AC: Module IGG019AC moves the next fixed–length or undefined–length record to the work area. The Open executor selects and loads this module if the Open parameter list specifies:

Input

and the DCB specifies:

Get

Simple buffering

Move operating mode

Fixed–length (unblocked, blocked, or blocked standard) or undefined–length record format

The DCB does not, however, specify the CNTRL macro instruction. The module consists of a Get routine and a RELSE routine.

The Get routine operates as follows:

- It receives control when a GET macro instruction is encountered in a processing program.
- It tests for an EOB condition to determine whether a new full buffer is needed. When the Open executor primes the buffers, it sets this EOB condition for the first GET macro instruction.
- If no EOB condition exists, the routine moves the next record to the work area.
- If an EOB condition exists, the routine issues a BALR instruction to obtain a new buffer through the input–synchronizing–and–error–processing routine, module IGG019AQ, and moves the first record of the new buffer to the work area.
- It tests for a new EOB condition to determine whether a buffer is empty and ready for refilling. For unblocked records, this condition exists at every entry into the routine.
- If no new EOB condition exists, the routine returns control to the processing program.

- If a new EOB condition exists, the routine issues a BALR instruction to pass the present buffer to the end–of–block routine to be scheduled for refilling and returns control to the processing program.

The RELSE routine sets a bit in the DCB so that the Get routine passes the buffer for refilling and obtains a new full buffer the next time the routine is entered.

Get Module IGG019AD: Module IGG019AD moves the next variable–length or format–D record to the work area. The Open executor selects and loads this module if the Open parameter list specifies:

Input

and the DCB specifies:

Get

Simple buffering

Move operating mode

Variable–length or record format–D (unblocked or blocked), unspanned

The DCB does not, however, specify the CNTRL macro instruction. The module consists of a Get and a RELSE routine.

Get

Simple buffering

Move operating mode

The Get routine operates as follows:

- It receives control when a GET macro instruction is encountered in a processing program.
- It tests for an EOB condition to determine whether a new full buffer is needed. When the Open executor primes the buffers, it also sets an end–of–block condition for the first GET macro instruction.
- If an EOB condition exists, the routine issues a BALR instruction to obtain a new buffer through the input–synchronizing–and–error–processing routine, module IGG019AQ, and moves the first record to the work area.
- If no EOB condition exists, the routine moves the next record to the work area.
- It tests for a new EOB condition to determine whether a buffer is empty and ready for refilling. For unblocked records, the condition exists after every entry to this routine.
- If no new EOB condition exists, the routine returns control to the processing program.
- If a new EOB condition exists, the routine issues a BALR instruction to pass the present buffer to the end–of–block routine to be scheduled for refilling and returns control to the processing program.

The RELSE routine sets a bit in the DCB so that the Get routine passes the buffer for refilling and obtains a new full buffer the next time the routine is entered.

Get Module IGG019AG (CNTRL — Card Reader): Module IGG019AG moves the next fixed–length or undefined–length record to the work area without scheduling the buffer for refilling. To refill the buffer, the processing program issues a CNTRL macro instruction. The Open executor selects and loads this module if the Open parameter list specifies:

Input

and the DCB specifies:

Get

Simple buffering

Move operating mode

Fixed–length (unblocked, blocked, or blocked standard) or undefined–length record format

CNTRL (card reader)

The module consists of a Get routine and a RELSE routine.

The Get routine operates as follows:

- It receives control when a GET macro instruction is encountered in a processing program.
- If an EOB condition exists, it resets the DCBRECAD and DCBEOBAD fields for the new buffer, issues a BALR to the input–synchronizing–and–error–processing routine, module IGG019AQ, and then tests for blocked records.
- If no EOB condition exists, it tests immediately for blocked records.
- For blocked records, it updates the DCBRECAD field, moves the present record to the work area, and returns control to the processing program.
- For unblocked records, it sets the DCBRECAD and DCBEOBAD fields so that they are equal, moves the present record to the work area, and returns control to the processing program.

The RELSE routine sets the value of the DCBEOBAD field equal to that of the DCBRECAD field to establish an EOB condition. Control then returns to the processing program.

Get Module IGG019AM (RDBACK): Module IGG019AM presents the processing program with the address of the next record when the data set is opened for backward reading. The Open executor selects and loads this module if the Open parameter list specifies:

RDBACK

and the DCB specifies:

Get

Simple buffering

Locate operating mode

Fixed–length (unblocked, blocked, or blocked standard) or undefined–length record format

The module consists of a Get routine and a RELSE routine.

The Get routine operates as follows:

- It receives control when a GET macro instruction is encountered in a processing program.
- It tests for an EOB condition.
- If no EOB condition exists, it determines the address of the next record by subtracting the DCBLRECL value from the DCBRECAD value. The routine presents the result to the processing program, and returns control to the processing program.
- If an EOB condition exists, it issues a BALR instruction to pass the present buffer to the end–of–block routine. The Get routine issues another BALR instruction to obtain a new buffer through the input–synchronizing–and–error–processing routine, module IGG019AQ. The Get routine then presents the address of the last record of the new buffer to the processing program, and returns control to the processing program.

The RELSE routine causes an EOB condition by setting the DCBRECAD and DCBEOBAD fields so that they are equal; it then returns control to the processing program.

Figure 2 illustrates the ordering of records using this module. When reading backwards under QSAM, each block is read from the tape from the end of the block to the beginning, each buffer is filled from the end of the buffer to the beginning, and the records are presented to the processing program in order of the record in the last segment of the buffer first, and the record in the first one last. In this manner of reading, buffering, and presenting, each record follows in backward sequence, from the record presented last out of one buffer to the record presented first out of the next buffer.

Get Module IGG019AN (RDBACK): Module IGG019AN moves the next record to the work area when the data set is opened for backward reading. The Open executor selects and loads this module if the Open parameter list specifies:

RDBACK

and the DCB specifies:

Get

Simple buffering

Move operating mode

Fixed–length (unblocked, blocked, or blocked standard) or undefined–length record format

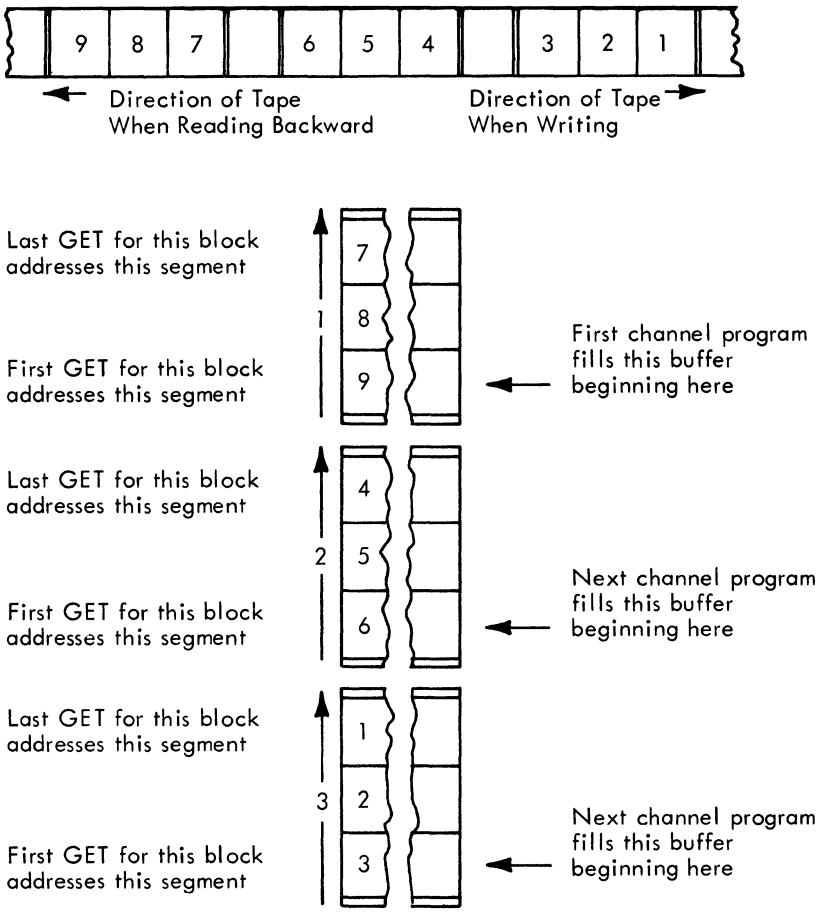


Figure 2. Order of Records Using Get Routines for Data Sets Opened for RDBACK (IGG019AM, IGG019AN)

The module consists of a Get routine and a RELSE routine.

The Get routine operates as follows:

- It receives control when a GET macro instruction is encountered in a processing program.
- It tests for an EOB condition.
- If no EOB condition exists, it moves the next record to the work area, and updates the DCBRECAD field by reducing it by the value of the DCBLRECL field.
- If an EOB condition exists, it issues a BALR instruction to obtain a new buffer through the input–synchronizing–and–error–processing routine, module IGG019AQ. The Get routine then moves the last record of the new buffer to the work area.
- It tests for a new EOB condition.

- If no new EOB condition exists, it returns control to the processing program.
- If a new EOB condition exists, it issues a BALR instruction to pass the present buffer to the end–of–block routine and then returns control to the processing program.

The RELSE routine issues a BALR instruction to pass the present buffer to the end–of–block routine and then returns control to the processing program.

Figure 2, described for Get module IGG019AM, also illustrates the ordering of records using this module.

Get Module IGG019AT (Paper Tape): Module IGG019AT converts paper tape characters into EBCDIC characters and moves them to the work area. The Open executor selects and loads this module and one of the code conversion modules (see "Appendix A: Code Conversion Routines" in Section 6) if the Open parameter list specifies:

Input

and the DCB specifies:

Get

Simple buffering

Move operating mode

Paper–tape–character–conversion

The module consists of a Get routine and a character–conversion and synchronizing–and–error–processing routine.

The Get routine operates as follows:

- It receives control when a GET macro instruction is encountered in a processing program.
- It converts the next character and moves it to the work area.
- It continues converting and moving until one of the following conditions is met, resulting in stated effect:

  The number of characters specified in the DCBBLKSI field of the DCB have been moved: The routine returns control to the processing program.

  An EOB condition is encountered: The routine passes control to the end–of–block routine to refill the buffer, and then enters the character–conversion and synchronizing–and–error–processing routine to obtain a new buffer.

  An end–of–record character is encountered (undefined–length records only): The routine returns control to the processing program.

  The tape is exhausted: The routine returns control to the processing program.

  A paper tape reader–detected error character is encountered: The routine moves the character to the work area without conversion and enters the character–conversion and synchronizing–and–error–processing routine.

- If one of the characters in the buffer is an undefined character, the module converts it to the hexadecimal character FF, moves it to the work area, and continues conversion. When one of the previous conditions is met, control passes to the character–conversion and synchronizing–and–error–processing routine.

The character–conversion and synchronizing–and–error–processing routine operates as follows:

- For an EOB condition, the routine finds the next buffer and returns control to the Get routine to resume converting and moving.
- For a reader–detected error character and for an undefined character, the routine passes control to the processing program's SYNAD routine. When control returns from the SYNAD routine, or if there is no SYNAD routine present, one of the error options is implemented.
- For the Accept–error option, the routine returns control to the processing program.
- For the Skip–error option, the routine fills the work area again.
- For the ABE–error option, or if no error option is specified, the routine issues the ABEND macro instruction.

The modules containing the tables used for code conversion are listed in Appendix A under "Code Conversion Routines."

Get Module IGG019BO: Module IGG019BO presents the processing program with the address of the next variable–length record. The Open executor selects and loads this module if the Open parameter list specifies:

Input

and the DCB specifies:

Get

Simple buffering

Locate operating mode

Variable–length spanned (unblocked or blocked) record format

Logical record interface

The module consists of a Get routine and a RELSE routine.

The Get routine operates as follows:

- It receives control when a GET macro instruction is encountered in the processing program.
- It determines the address of the next record and tests for an EOB condition to determine whether a buffer is empty and ready for refilling and if new buffer is needed. When the Open executor primes the buffers, it schedules all buffers except one and sets an EOB condition.
- If no EOB condition exists, it tests whether the next record segment contains a complete record.

- If it is a complete record, the routine presents the address of the next record to the processing program and returns control to the processing program.
- If it is the first segment of a spanned record, the routine moves the segment to the record area with the proper alignment, sets the EOB condition, and determines the address of the next record and whether a buffer is ready for refilling.
- If it is a segment that follows another segment of a spanned record, the routine moves the segment (without the segment descriptor word) next to the previous segment in the record area, and updates the count in the record area. This step continues until the entire logical record has been assembled in the record area. If an EOB condition occurs during this process, the routine determines the address of the next record and whether a buffer is ready for refilling. When the entire logical record is assembled, the routine sets the spanned record flag in the IOB, presents the address of the assembled record, and returns control to the processing program.
- If an EOB condition exists, it issues a BALR instruction to pass the present buffer to the EOB routine to be scheduled for refilling. The Get routine issues another BALR instruction to obtain a new buffer through the input–synchronizing–and–error–processing routine (module IGG019AO). The routine then obtains and interrogates the first record segment of the new buffer. If it is a complete record, the routine presents the address of the next record to the processing program and returns control to the processing program.

The RELSE routine operates as follows:

- It receives control when a RELSE macro instruction is encountered in the processing program.
- It sets an EOB condition.
- It sets a release bit in the DCBRECAD of the DCB.
- It returns control to the processing program.

The RELSE routine sets a release bit in the DCB so that the Get routine passes the buffer for refilling and obtains a new full buffer the next time the routine is entered. After obtaining the new buffer as a result of RELSE, the Get routine interrogates the SDW of the first segment to determine if it is the first segment of a record (bit 6 in third byte of SDW must be 0); if not, the routine skips to the next SDW and checks it. This continues until an acceptable segment is found. The routine then processes the Get request in the usual way. The procedure may result in one or more additional blocks being passed.

Get Module IGG019FB: Module IGG019FB presents the processing program with the address of the next variable–length record. The Open executor selects and loads this module if the Open parameter list specifies:

Input

and the DCB specifies:

Get
Simple buffering
Locate operating mode
Variable–length format (unblocked or blocked) record, spanned

The module consists of a Get routine and a RELSE routine.

The Get routine operates as follows:

- It receives control when the processing program issues a GET macro instruction.
- It determines the address of the next record segment and tests for an EOB condition to determine whether a buffer is ready for refilling and also whether a new buffer is needed. When the Open executor primes the buffers, the executor schedules all buffers except one and sets an EOB condition.
- If no EOB condition exists, the routine presents the address of the next record segment to the processing program.
- If an EOB condition exists or if a DOS–type null segment (where the high–order bit of the record descriptor word is on) is encountered, the routine issues a BALR instruction to pass the current buffer to the EOB routine. The EOB routine schedules the buffer for refilling. The Get routine issues another BALR instruction to obtain a new buffer through the input–synchronizing–and–error–processing routine, module IGG019AQ. The Get routine then determines if the EOB routine was entered because of a RELSE macro instruction. If so, the Get routine checks the first record segment to determine if it is a member of a previous logical record. If it is, the Get routine continues to look for a record segment that is not a member of a previous record. Such a segment is considered the first record of the new buffer. (Note, however, that this could cause reentry into the EOB routine and result in one or more entire blocks being skipped.) The Get routine then presents the address of the first record segment of the new buffer to the processing program and returns control to the processing program.

The RELSE routine causes an EOB condition by setting the DCBRECAD and DCBEOBAD fields so that they are equal. It then sets the high–order 4 bits of DCBRECAD to 1s and returns control to the processing program.

Get Module IGG019FD: Module IGG019FD moves the next variable–length record to the work area. The Open executor selects and loads this module if the Open parameter list specifies:

Input

and the DCB specifies:

Get
Simple buffering
Move operating mode
Variable–length (unblocked or blocked) record format, spanned

The DCB does not, however, specify the CNTRL macro instruction. The module consists of a Get and a RELSE routine.

The Get routine operates as follows:

- It receives control when the processing program issues a GET macro instruction.
- It tests for an EOB condition to determine whether a new full buffer is needed. When the Open executor primes the buffers, the executor also sets an EOB condition for the first GET macro instruction.

- If an EOB condition exists, the routine issues a BALR instruction to obtain a new buffer through the input–synchronizing–and–error–processing routine, module IGG019AQ, and moves the first record segment to the user's work area.
- If no EOB condition exists, the routine moves the first record segment to the user's work area.
- If a DOS–type null segment (where the high–order bit of the record descriptor word is on) is encountered, that buffer is rescheduled by passing control to the EOB routine. Processing continues as if an EOB condition exists as described above.
- If more record segments are required, the routine moves them, without the segment descriptor words, to the part of the user's work area that is contiguous with the previous record segment. The routine also updates the DCBLRECL field and the logical–record–length field in the record descriptor word (RDW) in the user's work area. These fields then reflect the total logical–record length after additional record segments have been moved. This procedure continues until the routine has moved the entire logical record. An EOB condition can occur during this procedure.
- The routine tests for a new EOB condition to determine whether a buffer is empty and ready for refilling. For unblocked records, the EOB condition exists after every entry to the Get routine.
- If no new EOB condition exists, the routine returns control to the processing program.
- If a new EOB condition exists, the routine issues a BALR instruction to pass the present buffer to the EOB routine. The EOB routine then schedules the buffer for refilling and returns control to the processing program.

The RELSE routine sets the high–order 4 bits in the DCBRECAD field to 1s so that the Get routine passes the buffer for refilling and so that the next time the Get routine is entered, it obtains a new full buffer. After obtaining the new buffer, the Get routine interrogates the segment descriptor word (SDW) of the first record segment. The routine thus determines if the segment is the first segment of a record. If it is, bit 6 of the third byte of the SDW will be 0. If not, the Get routine skips to the next SDW and checks it. This procedure continues until an acceptable segment is found. Then the Get routine processes the GET macro instruction in the usual way. The procedure can result in one or more additional blocks being passed.

Get Module IGG019FF: Module IGG019FF moves the data portion of the next variable–length record to the work area. The Open executor selects and loads this module if the Open parameter list specifies:

Input

and the DCB specifies:

Get

Simple buffering

Data operating mode

Variable–length (unblocked or blocked) record format, spanned

The DCB does not, however, specify the CNTRL macro instruction. The module consists of Get and RELSE routines.

The Get routine operates as follows:

- It receives control when the processing program issues a GET macro instruction.
- It tests for an EOB condition to determine whether a new full buffer is needed. When the Open executor primes the buffers, the executor also sets an EOB condition for the first GET macro instruction.
- If an EOB condition exists, the routine issues a BALR instruction to obtain a new buffer through the input–synchronizing–and–error–processing routine, module IGG019AQ, and moves the data portion of the first record segment to the work area.
- If no EOB condition exists, the routine moves the data portion of the first record segment to the user's work area.
- If more segments are required, the routine moves them, without the segment descriptor word, to the part of the user's work area that is contiguous with the previous record segment. The routine also updates the DCBLRECL field to reflect the current total logical record length. This procedure continues until the routine has moved the entire logical record. An EOB condition can occur during this procedure.
- The routine tests for a new EOB condition to determine whether a buffer is ready for refilling. For unblocked records, the condition exists after every entry to this routine.
- If no new EOB condition exists, the routine returns control to the processing program.
- If a new EOB condition exists, the routine issues a BALR instruction to pass the present buffer to the EOB routine. The EOB routine then schedules the buffer for refilling and returns control to the processing program.

The RELSE routine sets the high–order 4 bits in the DCBRECAD field to 1s so that the Get routine passes the buffer for refilling and so that the next time the Get routine is entered, it obtains a new full buffer. After obtaining the new buffer, the Get routine interrogates the segment descriptor word (SDW) of the first record segment. The routine thus determines if the segment is the first segment of a record. If it is, bit 6 of the third byte of the SDW will be 0. If not, the Get routine skips to the next SDW and checks it. This procedure continues until an acceptable segment is found. Then the Get routine processes the GET macro instruction in the usual manner. The procedure can result in one or more additional blocks being passed.

Exchange Buffering Get Routines

Exchange buffering Get routines use buffers whose addresses and lengths are stated in the channel program. For unblocked records, the buffer address and length are in one channel command word (CCW). For blocked records, the addresses of the buffer segments are in successive CCWs (though the segments themselves are not necessarily

located next to one another). In each pass through an exchange buffering Get routine, it determines:

- The address of the next record
- Whether an input buffer is empty and ready to be scheduled for refilling
- Whether a new full input buffer is needed

If the records are unblocked, a Get routine finds the address of the next record in the Read CCW for the next input buffer.

If the records are blocked, a Get routine finds the address of the next record in the next Read CCW for the same buffer. The next CCW is found by adding 8 to the address of the previously current CCW (the value stated in the DCBCCCW field in the DCB).

If an input buffer is empty and ready to be scheduled for refilling, a Get routine passes control to an end–of–block routine. The end–of–block routine passes control to the I/O supervisor to have it schedule the buffer. After scheduling, the I/O supervisor returns control to the end–of–block routine, and it returns control to the Get routine.

If a new full buffer is needed, a Get routine passes control to a synchronizing–and–error–processing routine. The synchronizing routine enters the address of the input/output block (IOB) that points to that channel program into the DCBIOBA field in the DCB.

If an end–of–block condition exists, either an input buffer is empty and ready to be scheduled for refilling, or a new buffer is needed. An end–of–block condition exists for unblocked records during each pass through a routine; for blocked records it exists if the values in the DCBCCCW (the address of the current CCW) and DCBLCCW (the address of the last CCW) fields are equal.

In the locate operating mode, the empty buffer is scheduled when the routine is entered if the last record was presented in the preceding entry; accordingly, a test for an end–of–block condition is made on entry to the routine to determine both whether a buffer is empty and if a new buffer is needed.

In the substitute operating mode, the buffer can be scheduled for refilling as soon as a work area has been substituted for the last buffer segment. An end–of–block test is made before leaving the routine to determine whether the buffer is empty, and another end–of–block test is made on entry to the routine to determine if a new buffer is needed.

A RELSE routine sets an end–of–block condition. This end–of–block condition is processed so that, when the Get routine is entered next, it operates as usual.

The Open executor primes (that is, schedules for filling) the buffers if QSAM is used with a DCB opened for Input. For the locate mode, all buffers except one are primed; for the substitute mode, all buffers are primed. The Open executor also sets an end–of–block condition; the first time that a Get routine gains control, it processes this condition in the usual way.

There are four exchange buffering Get routines. Figure 3 lists the available routines and the conditions that cause a particular routine to be used. The Open executor selects one of the routines, loads it, and places its address into the DCBGET field. The table shows, for example, that if input, Get, exchange buffering, locate mode, and fixed–length blocked record format are specified, module IGG019EA is selected.

Access Method Options	Selections						
Get, Input, Exchange	X	X	X	X	X	X	X
Locate	X	X	X	X			
Substitute					X	X	X
Fixed–length	X	X			X		X
Variable–length			X				
Undefined–length				X		X	
Unblocked		X	X	X	X	X	
Blocked	X						X
Get Modules							
IGG019EA	EA						
IGG019EB		EB	EB	EB			
IGG019EC					EC	EC	
IGG019ED							ED

Figure 3. Module Selector — Exchange–Buffering Get Modules

Get Module IGG019EA: Module IGG019EA uses the locate mode to present the processing program with the address of the next fixed–length blocked record. The Open executor selects and, loads this module if the Open parameter list specifies:

Input

and the DCB specifies:

Get

Exchange buffering

Locate operating mode

Fixed–length blocked record format

The module checks the channel program for a rotational position sensing (RPS) channel program and, if one is found, decreases the last CCW pointer to account for the Read–sector CCW at the end of the channel program.

The module consists of a Get routine and a RELSE routine.

The Get routine operates as follows:

- It receives control when a GET macro instruction is encountered in a processing program.
- It tests for an end–of–block condition to determine whether a buffer is empty and ready for refilling and if a new full buffer is needed. When the Open executor primes the buffers, it schedules all buffers except one and sets an end–of–block condition.
- If no end–of–block condition exists, it presents the address of the next record (found in the next CCW) and returns control to the processing program.

- If an end–of–block condition exists, the routine passes control to the end–of–block routine to cause scheduling of the buffer for refilling. On return of control, the Get routine passes control to the input–synchronizing–and–error–processing routine, module IGG019AQ, to obtain a new full buffer. On return of control, the Get routine then presents the address of the first record and returns control to the processing program.

The RELSE routine causes an end–of–block condition by setting the DCBCCCW and DCBLCCW fields equal and returns control to the processing program.

Note: If an input DCB using this module is paired with an output DCB using module IGG019EF (Output, Put, Exchange), a PUTX macro instruction addressed to the output DCB causes an exchange of the addresses of the current buffer segments of each DCB. These are found in the CCWs pointed to by the input and output DCBs.

Get Module IGG019EB: Module IGG019EB uses the locate mode to present the processing program with the address of the next unblocked record. The Open executor selects and loads this module if the Open parameter list specifies:

Input

and the DCB specifies:

Get

Exchange buffering

Locate operating mode

Unblocked record format (fixed– , variable– , or undefined–length)

The module consists of a Get routine and a RELSE routine.

The Get routine operates as follows:

- It receives control when a GET macro instruction is encountered in a processing program.
- It passes control to the end–of–block routine to cause scheduling of the previous buffer for refilling.
- It passes control to the input–synchronizing–and–error–processing routine, module IGG019AQ, to obtain the next full buffer. When the Open executor primes the buffers, it schedules all buffers except one.
- It presents the address of the record and returns control to the processing program. For variable–length or undefined–length records, the routine also presents the record length.

The RELSE routine returns control without performing any processing.

Note: If an input DCB using this module is paired with an output DCB using module IGG019EE (Output, Put, Exchange), a PUTX macro instruction addressed to the output DCB causes an exchange of the addresses of the current buffer segments of each DCB. These addresses are found in the CCWs pointed to by the DCBCCCW fields in the input and output DCBs.

Get Module IGG019EC: Module IGG019EC uses the substitute mode to present the processing program with the address of the next unblocked record. The Open executor selects and loads this module if the Open parameter list specifies:

Input

and the DCB specifies:

Get

Exchange buffering

Substitute operating mode

Unblocked record format (fixed– or undefined–length)

The module consists of a Get routine and a RELSE routine.

The Get routine operates as follows:

- It receives control when a GET macro instruction is encountered in a processing program.
- It passes control to the synchronizing routine to obtain the next full buffer.
- It exchanges the address of the work area and the address of the buffer.
- It passes control to the end–of–block routine to cause the work area offered by the processing program to be scheduled for filling.
- It presents the address of the new record and returns control to the processing program. (When the Open executor primes the buffers, it schedules all buffers.)
- When undefined–length records are specified, the routine also presents the record length.

The RELSE routine returns control without performing any processing.

Get Module IGG019ED: Module IGG019ED uses the substitute mode to present the processing program with the address of the next fixed–length blocked record. The Open executor selects and loads this module if the Open parameter list specifies:

Input

and the DCB specifies:

Get

Exchange buffering

Substitute operating mode

Fixed–length blocked record format

The module checks the channel program for a rotational position sensing (RPS) channel program and, if one is found, decrements the last CCW pointer to account for the Read Sector CCW at the end of the channel program.

The module consists of a Get routine and a RELSE routine.

The Get routine operates as follows:

- It receives control when a GET macro instruction is encountered in a processing program.
- It tests for an end–of–block condition to determine if a new full buffer is needed. When the Open executor primes the buffers, it schedules all buffers and sets an end–of–block condition.
- If no end–of–block condition exists, it exchanges the address of the work area for the address stated in the current CCW. The current CCW is found by adding 8 to the value of the DCBCCCW field.
- If an initial end–of–block condition exists, it passes control to the input–synchronizing–and–error–processing routine, module IGG019AQ, to obtain the next full buffer. It then exchanges the address of the work area for the address stated in the first Read CCW of the channel program.
- It tests for a new end–of–block condition to determine if a buffer is empty and ready for refilling.
- If no new end–of–block condition exists, it presents the address of the next record and returns control to the processing program.
- If a new end–of–block condition exists, it passes control to the end–of–block routine to cause scheduling of the empty buffer for refilling. It then presents the address of the next record and returns control to the processing program.

The RELSE routine sets an end–of–block condition and passes control to the end–of–block routine to cause scheduling of the buffer for refilling. It then returns control to the processing program.

Update Mode Get Routine

The update mode Get routine differs from other Get routines in that it shares its buffers, as well as the DCB and the IOBs, with the update mode Put routine. The QSAM update mode of access uses simple buffering in which the buffer is defined by the start and end addresses of the buffer.

If a PUTX macro instruction addresses a record in a block, the update mode Get routine determines, when the end of the block is reached, that that buffer is to be emptied (that is, that the block is to be updated) before being filled with a new block of data. If no PUTX macro instruction addresses a record in a block, the update mode Get routine determines, when the end of the block is reached, that the buffer is to be refilled only; that is, that the last block need not be updated and the buffer can be filled with a new block of data. These characteristics of the buffer — simple buffering, sharing the buffer with the Put routine, and emptying the buffer before refilling — influence the manner in which the update mode Get routine determines:

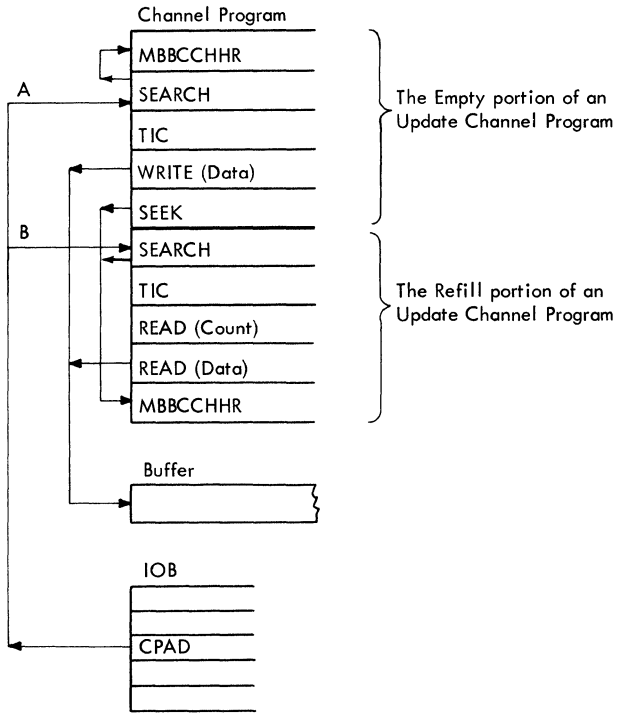
- The address of the next record.
- Whether the buffer can be scheduled.
- Whether a new buffer is needed.
- Whether to schedule the buffer for empty–and–refill or for refill–only.

The first three of these determinations are made at every pass through the routine. The last determination is made after the routine establishes that the buffer can be scheduled.

If the records are unblocked, the address of the next record is the address of the next buffer.

If the records are blocked, the address of the next record is found by adding the record length, found in the DCBLRECL field, to the value in the DCBRECAD field.

Whether the buffer can be scheduled and whether a new buffer is needed is determined by whether an end–of–block condition exists. In the update mode, one determination that an end–of–block condition exists causes both the last buffer to be scheduled and a new buffer to be sought. An end–of–block condition exists for unblocked records at every pass through the routine; for blocked records it exists if the values in the DCBRECAD (the address of the current record) and the DCBEOBAD (the address of the end of the block) fields are equal. To cause scheduling of the buffer, the Get routine passes control to the end–of–block routine. To obtain a new buffer, the Get routine passes control to the update–synchronizing–and–error–processing routine, module IGG019AF.



Legend:

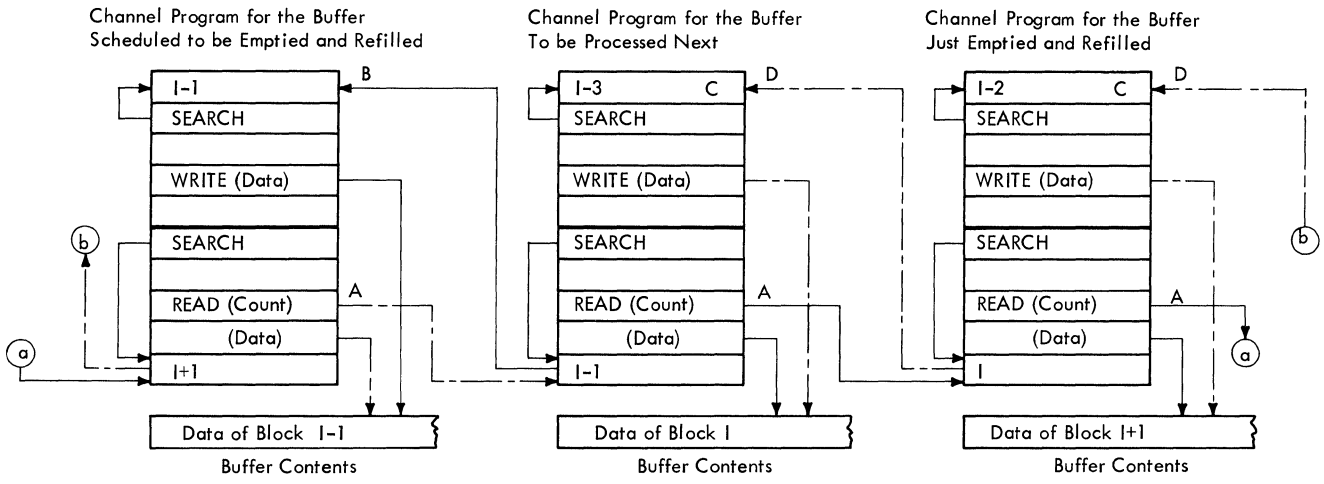
A - Address of channel program (CPAD) used to empty and refill the buffer. (A PUTX macro instruction was addressed to a record in this buffer.)

B - Address of channel program (CPAD) used only to refill the buffer. (No PUTX macro instruction was addressed to any record in this buffer.)

Figure 4. The Two Parts of an Update Channel Program (Empty, Refill)

To cause scheduling of the buffer for either empty–and–refill or refill–only, the update mode Get routine sets the IOB to point to the beginning of either one of two parts (empty and refill) of the QSAM update channel program. Empty writes out of the

buffer and reads into that same buffer (see Figure 4). If execution of a QSAM update channel program begins with the empty part, execution of the refill part always follows. Each part of the QSAM update channel program addresses a different location in auxiliary storage: the empty part addresses the location from which the block to be updated was read; the refill part addresses the location from which the last block was read. Addressing the last known block and skipping over its data field leads to the beginning of the next block, regardless of its address. This method of addressing a Search command to the block read previously to address a Read (count, key, and data) command to the next block is known as the search–previous technique. It makes the count field of the present block being read the Seek address of the refill portion of the next channel program. When a buffer is to be emptied (back to the original location of the block in auxiliary storage), the update mode Get routine obtains the block address from the Seek address of the refill part of the next channel program. It copies the address so that it becomes the Seek address for the empty part of the present channel program (see Figure 5). For a description of the processing for a refill–only QSAM update channel program, refer to the description of the update SIO appendage.



Legend:
A - The Refill portion reads the count field of the block being read into the search argument of the next Refill portion.

B - To empty the buffer, the search argument of the next Refill portion is used as the search argument of this Empty portion.

C - To empty the buffer, the search argument of the next Refill portion was copied before the last time this buffer was scheduled.

D - To empty the buffer, the search argument of the next Refill portion will be copied before the next time this buffer is scheduled.

———— Present entries — - — - — Future entries

Figure 5. Relation of Seek Addresses in Three Successive QSAM Channel Programs

Whether to schedule the buffer for empty–and–refill or for refill–only depends on whether the block is to be updated. If the block is to be updated, the PUTX routine will have set the update flag on in the IOB; otherwise, the flag is off. To schedule the buffer for empty–and–refill, the Get routine sets the IOB to point to the empty portion of the channel program and obtains the Seek address of the block to be updated from the refill portion of the next channel program. To schedule the buffer for refill–only,

the Get routine sets the IOB to point to the refill portion of the channel program. The end–of–block condition which triggers this processing also causes control to pass to the end–of–block routine, module IGG019CC, for issuing the EXCP macro instruction and to the update–synchronizing–and–error–processing routine, module IGG019AF, for obtaining the next buffer.

The PUTX routine sets the update flag in the IOB and returns control to the processing program. The RELSE routine sets an end–of–block condition and returns control to the processing program.

The Open executor primes (that is, schedules for filling) all the buffers except one if QSAM is used with a DCB opened for update. The Open executor also sets an end–of–block condition; the first time that the update mode Get routine gains control, it processes this condition in its normal manner.

Figure 6 shows the update mode Get routines and the access conditions that must be specified in the DCB to select a particular routine. The Open executor loads the selected routine and places its address into the DCBGET field of the DCB.

Access Method Options	**Selections**						
Update, Get	X	X	X	X	X	X	X
Fixed–length record format	X	X					
Variable–length record format			X	X		X	X
Undefined–length record format					X		
Blocked record format	X		X			X	
Unblocked record format		X		X	X		X
Locate operating mode						X	X
Logical record interface						X	X
Get Modules							
IGG019AE[1]	AE	AE	AE	AE	AE		
IGG019BN						BN	BN

[1]This module also carries the Update–Mode PUTX routine.

Figure 6. Module Selector — Update–Mode Get Modules

Get Module IGG019AE: Module IGG019AE presents the processing program with the next input record, flags the IOB if the block is to be updated (emptied and refilled), and sets the IOB to address a QSAM update channel program for either empty–and–refill or refill–only. The Open executor selects and loads this module if the Open parameter list specifies:

Update

and the DCB specifies:

Get

With the rotational position sensing (RPS) feature, the new CCWs are bypassed when necessary.

The module consists of a Get routine, a RELSE routine, and a PUTX routine.

The Get routine operates as follows:

- It receives control when a GET macro instruction is encountered in a processing program.
- It tests for an end–of–block condition to determine whether the buffer can be scheduled and if a new buffer is needed. When the Open executor primes the buffers, it schedules all buffers except one and sets an end–of–block condition.
- If no end–of–block condition exists, it presents the address of the next record, and returns control to the processing program. For variable–length, format–D, and undefined–length records, it also determines the length of the record and places it in the DCBLRECL field in the DCB.
- If an end–of–block condition exists, it tests whether the buffer is to be emptied and refilled or is to be refilled only.
- If it is to be refilled only, it sets the IOB to point to the start of the Read portion of the update channel program and passes control to the end–of–block routine to cause scheduling of the buffer.
- If it is to be emptied and refilled, it sets the IOB to point to the start of the update channel program. The routine obtains the auxiliary storage address to be used by the Write portion of the channel program by copying the address used by the Read portion of the channel program associated with the next IOB. The routine then passes control to the end–of–block routine to cause scheduling of the buffer.
- On return of control from the end–of–block routine, the Get routine passes control to the update–synchronizing–and–error–processing routine, module IGG019AF, to obtain a new full buffer.
- On return of control from the synchronizing routine, the Get routine updates the DCBLRECL field, presents the address of the next record, and returns control to the processing program.

The RELSE routine operates as follows:

- It receives control when a RELSE macro instruction is encountered in the processing program.
- It sets an end–of–block condition.
- It returns control to the processing program.

The PUTX routine operates as follows:

- It receives control when a PUTX macro instruction is encountered in the processing program.
- It sets the update flag in the IOB to show that the buffer is to be emptied before being refilled.
- It returns control to the processing program.

Get Update Module IGG019BN: Module IGG019BN presents the processing program with the next input record, flags the IOB if the block or a spanned record is to be updated (that is, emptied and refilled), and sets the IOB to address a QSAM update channel program for either empty–and–refill or refill–only. The Open executor selects and loads this module if the Open parameter list specifies:

Update

and the DCB specifies:

Get

Locate operating mode

Variable–length spanned (blocked or unblocked) record format

Logical record interface

The module consists of a Get routine, a RELSE routine, and a Put routine.

The Get routine operates as follows:

- It receives control when a GET macro instruction is encountered in a processing program.
- It tests whether EOV has occurred while processing a spanned record.
- If the record is not to be updated, it sets a bit in the DCBIOBAD field of the DCB to indicate that the old DEB, whose address was saved by the EOV routine, can be freed. It then issues an FEOV macro instruction to free the main storage assigned to this DEB.
- If the record is to be updated, it restores the address to read back the block that contains the beginning segment of the record. The current IOB is modified to function as if only one IOB exists. It then issues an FEOV macro instruction to cause the previous volume to be mounted and the data management count to be reset.
- On return of control from the FEOV routines, it operates as if no EOV has occurred.
- If EOV has not occurred, it continues on to the next step.
- It tests whether a spanned record is to be updated.
- If it is not to be updated, it obtains the length of the previous record segment from the DCBLRECL field in the DCB or the SDW if it was a spanned record.
- It determines the address of the next record segment and tests for an EOB condition to determine whether the buffer can be scheduled and if a new buffer is needed. (When the Open executor primes the buffers, it schedules all buffers except one and sets an EOB condition.)
- If no EOB condition exists, it tests the next record segment for a complete record.
- If it is a complete record, the routine presents the address of the next record, determines the length of the record, places it in the DCBLRECL field, and returns control to the processing program.

- If it is the first segment of a spanned record, the routine saves the track address of the block that contains this segment, the position of the segment in the block, and the alignment of the segment in the record area. The routine obtains the track address of the block by copying the address used by the Read portion of the channel program associated with the next IOB, the position of the segment by subtracting the buffer address from the current record address, and the alignment of the segment by using the low–order byte of the current record address. The routine then moves the first segment to the record area and sets the EOB condition. It determines the address of the next record, whether a new buffer can be scheduled, and if a new buffer is needed.
- If it is a segment that follows another segment of a spanned record, the routine combines the segment (without the SDW) contiguous with the previous segment in the record area. The count in the record descriptor word (RDW) in the record area is updated to include the total count. This process continues until the entire logical record has been assembled. An EOB condition may occur during this process, in which case the routine determines the address of the next record, whether a new buffer can be scheduled, and if a new buffer is needed. When the entire logical record has been assembled, the routine sets the spanned–record flag in the IOB, presents the address of the assembled record in the record area, places the length of the record (which is obtained from the RDW in the record area) in the DCBLRECL field, and returns control to the processing program.
- If an EOB condition exists, it tests whether the buffer is to be emptied and refilled or is to be refilled only.
- If it is to be refilled only, it sets the IOB to point to the start of the read portion of the update channel program and passes control to the EOB routine to cause scheduling of the buffer.
- If it is to be emptied and refilled, it sets the IOB to point to the start of the update channel program. The routine obtains the auxiliary storage address to be used by the Write portion of the channel program by copying the address used by the Read portion of the channel program associated with the next IOB. The routine then passes control to the EOB routine to cause scheduling of the buffer.
- On return of control from the EOB routine, the routine passes control to the update–synchronizing–and–error–processing routine, module IGG019BQ, to obtain a new full buffer.
- On return of control from the synchronizing routine, the routine interrogates the next record segment and saves the track address of the block that contains the record, the position of the segment in the block, and the alignment of the segment in the record area. The routine then moves the first segment to the record area and sets the EOB condition.
- If a spanned record is to be updated, the routine restores the track address to read back the block that contains the beginning segment of the record. The current IOB is modified to function as if only one IOB exists.
- It sets the IOB to point to the start of the read portion of the update channel program and passes control to the EOB routine to cause scheduling of the buffer.
- On return of control from the EOB routine, the routine passes control to the update–synchronizing–and–error–processing routine, module IGG019BQ, to obtain a new full buffer.

- On return of control from the synchronizing routine, the routine repositions the pointers to the beginning segment of the record and moves that portion of the record from the record area to the segment in the buffer. (A count is kept of the number of bytes of data moved.)
- If more segments are to be updated, the routine moves that portion of the record from the record area to the succeeding segments in the buffer. (The total count of the data moved is updated with each move.) This process continues until the entire logical record has been segmented. If an EOB condition occurs during this process, the routine tests whether a spanned record is to be updated. When the entire logical record has been segmented, the routine turns off the spanned–record flag in the IOB, restores the link field in the IOB, obtains the address of the next record segment, and determines whether a new buffer can be scheduled and is needed.

The RELSE routine operates as follows:

- It receives control when a RELSE macro instruction is encountered in the processing program.
- It sets an EOB condition.
- It sets a release bit in the DCBRECAD field of the DCB.
- It returns control to the processing program.

The RELSE routine sets a release bit in the DCB so that the Get routine passes the buffer for refilling and obtains a new full buffer the next time the routine is entered. After obtaining the new buffer as a result of RELSE, the Get routine interrogates the SDW of the first segment to determine if it is the first segment of a record (bit 6 in the third byte of the SDW must be 0); if not, the routine skips to the next SDW and checks it. This continues until an acceptable segment is found. The routine then processes the Get in the usual way. This procedure may result in one or more additional blocks being passed.

The PUTX routine operates as follows:

- It receives control when a PUTX macro instruction is encountered in the processing program.
- It sets the update flag in the IOB to show that the buffer is to be emptied before being refilled.
- It returns control to the processing program.

Note: When a RELSE macro instruction is issued after a spanned record is written with a PUTX macro instruction, this routine branches to the Get routine to write the last record (the spanned record) and then releases the block that contains the last segment of that spanned record.

Put Routines

Some of the general characteristics of the Put routines are described in Diagram B, QSAM Get and Put Routines, in Section 5. A specific Put routine is selected for each data set on the basis of access method options specified by the processing program. The options examined are in the Open statement parameter list and the data set attributes described in the DCB.

The Open executors (see Diagram D, SAM Open Executors, in Section 5) select and load the modules that are required for a particular data set.

The access method options that determine which Put modules are selected when Simple buffering is used are described in Figure 7. The options that determine which Put modules are selected when exchange buffering is used are described in Figure 8. For update mode, the PUTX routine resides in the Get module for update mode. See Figure 6 (under "Update Mode Get Routine") for information about the update mode PUTX routine.

For information about the flow of control through the QSAM routines, see Diagram F, QSAM Flow of Control, in Section 5.

Simple Buffering Put Routines

Simple buffering Put routines use buffers whose ending address and the address of the next or current record is pointed to by the DCB. The address of the next record is in the DCBRECAD field (address of the next record); the ending address is in the DCBEOBAD field (address of the end of the buffer). In each pass through a routine, it determines:

- The address of the next buffer segment
- Whether an output buffer is to be scheduled for emptying
- Whether a new empty buffer is needed

These three determinations are made at every pass through a Put routine.

If the records are unblocked, the address of the next available buffer segment is always that of the next buffer.

If the records are blocked, a Put routine determines the address of the next available buffer segment by adding the length of the last record to the address of the last buffer segment. The address of the last buffer segment is in the DCBRECAD field of the data control block (DCB). If the records are fixed–length blocked records, the length of each record is in the DCBLRECL field. If the records are variable–length blocked records, the length of each record is in the length field of the record itself.

A Put routine determines that a buffer is ready for emptying and a new empty buffer is needed by establishing that an end–of–block (EOB) condition exists.

If an output buffer is to be scheduled for emptying, a Put routine passes control to an end–of–block routine, to cause the present buffer to be scheduled for output.

If a new empty buffer is needed, a Put routine obtains a new buffer by passing control to the output–synchronizing–and–error–processing routine, module IGG019AR. For a buffer that was emptied without error, the synchronizing routine updates the DCBIOBA field (thus pointing to the new buffer) and returns control to the Put routine. The Put routine updates the DCBRECAD field by inserting the starting address of the buffer from the channel program associated with the new IOB. To update the DCBEOBAD field, the routine adds the length of the block stated in the DCBBLKSI field to the buffer starting address. These two fields, DCBRECAD and DCBEOBAD, define the available buffer.

An EOB condition is established by different criteria for different record formats and operating modes.

For unblocked records, an EOB condition exists after each record is placed in the buffer. If the move operating mode is used, a Put routine establishes that an EOB condition exists for the present buffer after the routine has moved the record into the buffer. If the locate operating mode is used, a Put routine establishes that an EOB condition exists for the present buffer on the next entry to the routine, after the processing program has moved the record into the buffer.

For blocked records, the time that an EOB condition occurs depends on the record format.

For fixed–length blocked records, an EOB condition occurs when the DCBRECAD field equals the DCBEOBAD field. The DCBRECAD field shows the address of the segment for the next record. The DCBEOBAD field shows a value equal to one more than the address of the end of the buffer. If the move operating mode is used, the Put routine moves the last fixed–length record into the buffer, updates the DCBRECAD field, and establishes that an EOB condition exists for the present buffer. If the locate operating mode is used, the processing program moves the last fixed–length record into the buffer. On the next entry to the Put routine, the routine updates the DCBRECAD field and establishes that an EOB condition exists for the present buffer.

For variable–length blocked records, unspanned, an EOB condition occurs when the length of the next record exceeds the buffer balance; that is, when the record length exceeds the space remaining in the buffer. If the user has specified move mode for unspanned records, the Put routine establishes that an EOB condition exists when the record length stated in the first word of the record exceeds the buffer balance. If the user has specified locate mode for unspanned records, the Put routine establishes that an EOB condition exists when the value stated in the DCBLRECL field exceeds the buffer balance.

For variable–length blocked records, spanned, the next record is segmented. The first record segment is used to fill the buffer when five or more bytes remain in the buffer. When fewer than five bytes remain in the buffer, an EOB condition occurs.

A TRUNC routine sets an end–of–block condition to empty the buffer. This end–of–block condition is processed so that the next entry to the Put routine permits it to operate as usual. Successive entries to a TRUNC routine without intervening entries to a Put routine cause the TRUNC routine to return control without performing any processing.

To permit a Put routine to operate normally when it is entered for the first time, the Open executor initializes the DCB fields DCBRECAD and DCBEOBAD. For an output data set using QSAM and simple buffering, the values entered in these fields depend on the operating mode. For locate mode routines, it sets them to show the beginning and end of the first buffer; for move mode routines, it sets an end–of–block condition.

Figure 7 lists the Put routines and the conditions that cause a particular routine to be read. The Open executor selects one of the routines, loads it, and places its address into the DCBPUT fields.

Put Module IGG019AI: Module IGG019AI presents the processing program with the address of the next available buffer segment for a fixed–length or undefined–length record. The Open executor selects and loads this module if the Open parameter list specifies:

Output

and the DCB specifies:

Put

Simple buffering

Locate operating mode

Fixed–length (unblocked, blocked or blocked standard) or undefined–length record format

Access Method Options	**Selections**									
Output, Put/PUTX, Simple buffering	X	X	X	X	X	X	X	X	X	X
Locate operating mode	X	X	X				X		X	
Move operating mode				X	X	X				
Data operating mode								X		X
Fixed–length record format	X			X						
Undefined–length record format		X			X					
Variable–length or record format–D			X			X	X	X	X	X
Spanned records							X	X	X	X
Logical record interface							X			
Put Modules										
IGG019AI	AI	AI								
IGG019AJ			AJ							
IGG019AK				AK	AK					
IGG019AL						AL				
IGG019BP							BP			
IGG019FG								FG		
IGG019FJ									FJ	
IGG019FL										FL

Figure 7. Module Selector — Simple Buffering Put Modules

The module consists of a Put routine and a TRUNC routine.

The Put routine operates as follows:

- It receives control when a PUT macro instruction is encountered in a processing program.
- It determines the address of the next buffer segment using the value in the DCBLRECL field.
- It tests for an EOB condition to determine whether a buffer is full and ready for emptying and if a new empty buffer is needed.
- If no EOB condition exists, it presents the address of the next buffer segment to the processing program and returns control to the processing program.
- If an EOB condition exists, it issues a BALR instruction to pass the present buffer to the end–of–block routine. The Put routine issues another BALR instruction to obtain a new buffer through the output–synchronizing–and–error–processing routine, module IGG019AR, and determines the address of the first segment of the new buffer. The Put routine then presents this address and returns control to the processing program.

The TRUNC routine causes an EOB condition by setting the DCBRECAD and DCBEODAD fields so that they are equal; it then returns control to the processing program.

Put Module IGG019AJ: Module IGG019AJ presents the processing program with the address of the next available buffer segment for a variable–length or format–D record. The Open executor selects and loads this module if the Open parameter list specifies:

Output

and the DCB specifies:

Put

Simple buffering

Locate operating mode

Variable–length or record format D (unblocked or blocked), unspanned

The module consists of a Put routine and a TRUNC routine.

The Put routine operates as follows:

- It receives control when a PUT macro instruction is encountered in a processing program.
- It determines the address of the next buffer segment using the length field of the record moved by the processing program into the buffer segment located last.
- It tests for an EOB condition to determine whether a buffer is ready for emptying and if a new empty buffer is needed, by using the value placed into the DCBLRECL field by the processing program.
- If no EOB condition exists, it tests for blocked records.
- If blocked records are specified, it presents the address of the next buffer segment to the processing program and returns control to the processing program.

- If an EOB condition exists or if unblocked records are specified, it issues a BALR instruction to pass the present buffer to the end–of–block routine. The Put routine issues another BALR instruction to obtain a new buffer through the output–synchronizing–and–error–processing routine, module IGG019AR, and determines the address of the first segment of the new buffer. The Put routine then presents this address to the processing program and returns control to the processing program.

The TRUNC routine causes an EOB condition by setting the DCBRECAD and DCBEOBAD fields so that they are equal; it then returns control to the processing program.

Put Module IGG019AK: Module IGG019AK moves the present fixed–length or undefined–length record into the next available buffer segment. The Open executor selects and loads this module if the Open parameter list specifies:

Output

and the DCB specifies:

Put

Simple buffering

Move operating mode

Fixed–length (unblocked, blocked, blocked standard) or undefined–length record format

The module consists of a Put routine, a PUTX routine, and a TRUNC routine.

The Put routine operates as follows:

- It receives control when a PUT macro instruction is encountered in a processing program.
- If an EOB condition exists, it issues a BALR instruction to obtain a new buffer through the output–synchronizing–and–error–processing routine, module IGG019AR, and then moves the record from the work area into the first buffer segment.
- If no EOB condition exists, it moves the record from the work area into the next buffer segment.
- It tests for blocked records.
- If blocked records are specified, it determines the address of the next segment and tests for a new EOB condition.
- If unblocked records are specified or if a new EOB condition exists, it issues a BALR instruction to pass the present buffer to the end–of–block routine and then returns control to the processing program.
- If no new EOB condition exists, it returns control to the processing program.

The PUTX routine operates as follows:

- It receives control when a PUTX macro instruction is encountered in a processing program.

- It obtains the DCBRECAD value of the input DCB, which points to the present record in the input buffer.
- It enters the Put routine at the start. The Put routine then uses the input DCBRECAD value in place of the work area address.

The TRUNC routine operates as follows:

- It receives control when a TRUNC macro instruction is encountered in a processing program.
- It simulates an EOB condition.
- It issues a BALR instruction to pass the present buffer to the end–of–block routine.
- On return of control from the end–of–block routine it returns control to the processing program.

Put Module IGG019AL: Module IGG019AL moves the present variable–length or format–D record into the next available buffer segment. The Open executor selects and loads this module if the Open parameter list specifies:

Output

and the DCB specifies:

Put

Simple buffering

Move operating mode

Variable–length or record format–D (unblocked or blocked), unspanned

The module consists of a Put routine, a PUTX routine, and a TRUNC routine.

The Put routine operates as follows:

- It receives control when a PUT macro instruction is encountered in a processing program.
- It determines the address of the next buffer segment and compares the length of the next record with the remaining buffer capacity.
- If the record fits into the buffer, it moves the record, updates the length field of the block, and tests for blocked records.
- If blocked records are specified, it returns control to the processing program.
- If the record does not fit into the buffer or if unblocked records are specified, it issues a BALR instruction to pass the present buffer to the end–of–block routine. It issues another BALR instruction to obtain a new buffer through the output–synchronizing–and–error–processing routine, module IGG019AR. The Put routine then moves the record from the work area to the buffer, updates the block–length field, and returns control to the processing program.

The PUTX routine operates as follows:

- It receives control when a PUTX macro instruction is encountered in a processing program.
- It obtains the DCBRECAD value of the input DCB, which points to the present record in the input buffer.
- It enters the Put routine at the start. The Put routine then uses the input DCBRECAD value instead of the work area address.

The TRUNC routine operates as follows:

- It receives control when a TRUNC macro instruction is encountered in a processing program.
- It issues a BALR instruction to pass control of the present buffer to the end–of–block routine.
- It issues another BALR instruction to obtain a new buffer through the output–synchronzing–and–error–processing routine, module IGG019AR.
- It determines the address of the first segment of the new buffer and then returns control to the processing program.

Put Module IGG019BP: Module IGG019BP presents the processing program with the address of the next available buffer segment for a variable–length record. The Open executor selects and loads this module if the Open parameter list specifies:

Output

and the DCB specifies:

Put

Simple buffering

Locate operating mode

Variable–length spanned (unblocked or blocked) record format

Logical record interface

The module consists of a Put routine and a TRUNC routine.

The Put routine operates as follows:

- It receives control when a PUT macro instruction is encountered in a processing program.
- It tests whether a spanned record was to have been written.
- If the last record written was not a spanned record, it determines the address of the next buffer segment using the length field of the last record segment moved by the processing program.
- It checks the value placed into the DCBLRECL field to determine if a buffer is ready for emptying and if a new empty buffer is needed.
- If no EOB condition exists, it tests for blocked records.

- If blocked records are specified, it presents the address of the next buffer segment to the processing program and returns control to the processing program.
- If unblocked records are specified, it issues a BALR instruction to pass the present buffer to the EOB routine. The Put routine issues another BALR instruction to obtain a new buffer through the output–synchronizing–and–error–processing routine, module IGG019AR, and determines the address of the first segment of the new buffer. The Put routine tests whether the present record to be written can fit entirely in the new buffer.
- If the record fits, the Put routine then presents this address to the processing program and returns control to the processing program.
- If the record does not fit, the routine saves the record address in the record area, obtains the address within the record area with the proper alignment, sets the spanned–record flag in the IOB, presents the address in the record area to the processing program, and returns control to the processing program.
- If an EOB condition exists, it tests whether a minimum record segment (at least 5 bytes) can fit in the present buffer.
- If it fits, the routine saves the record address, obtains the address within the record area, sets the spanned–record flag in the IOB, presents the address to the processing program, and returns control to the processing program.
- If it does not fit, the routine issues a BALR instruction to pass the present buffer to the EOB routine. The routine then issues another BALR instruction to obtain a new buffer through the output–synchronizing–and–error–processing routine, IGG019AR, and determines the address of the first segment of the new buffer. The routine tests whether the present record can fit entirely in the new buffer.
- If a spanned record was to be written out, it restores the record address, determines the length of the segment that can fit in this buffer, moves the segment from the record area to the buffer, and sets the proper flags for the segment.
- If more segments are required, the routine issues a BALR instruction to pass the present buffer to the EOB routine. The Put routine issues another BALR instruction to obtain a new buffer through the output–synchronizing–and–error–processing routine, module IGG019AR, and determines the address of the first segment of the new buffer. It moves the remaining bytes of data from the record area to the buffer and sets the proper flags for the segment. This step continues until the entire spanned record has been segmented. The routine then turns off the spanned–record flag and determines the address of the next buffer segment.

The TRUNC routine causes an EOB condition by setting the DCBRECAD and DCBEOBAD fields so that they are equal. It then returns control to the processing program.

When a TRUNC macro instruction is issued after a spanned record was written, this routine branches to the Put routine to write out the last record (the spanned record) and then truncates the block that contains the last segment of that spanned record.

Put Module IGG019FG: Module IGG019FG moves the data portion of the variable–length record into the next available buffer segment. The Open executor selects and loads this module if the Open parameter list specifies:

Output

and the DCB specifies:

Put

Simple buffering

Data operating mode

Variable–length (unblocked or blocked) record format, spanned

The module consists of a Put routine and a TRUNC routine.

The Put routine operates as follows:

- It receives control when the processing program issues a PUT macro instruction.
- It determines the possible location of the next buffer segment by adding the length of the previous record or record segment to the previous buffer segment address. This address is in the DCBRECAD field.
- It then compares the length of the next record with the remaining buffer capacity.
- If the record will fit, the routine moves the record, updates the length field of the block descriptor word (BDW), and checks for blocked records.
- If blocked records are specified, the routine returns control to the processing program. If unblocked records are specified, the routine issues a BALR instruction to pass the current buffer to the EOB routine. The Put routine issues another BALR instruction to obtain a new buffer through the output–synchronizing–and–error–processing routine, module IGG019AR. The Put routine then builds a new block descriptor word (BDW) and returns control to the processing program.
- If the record will not fit, the routine determines whether there are 5 or more unused bytes remaining in the buffer. If there are, the Put routine breaks the current record so that the first segment fills the buffer. The remaining segment will be placed in subsequent buffers. The length field in the segment descriptor word (SDW) of the first segment is updated to reflect the length of the segment. The third byte of this SDW is set to X'01' to indicate that this segment is the first of a multisegment record. After writing the buffer, the Put routine does not return control to the processing program until the entire record has been processed. The routine forms the remainder of the current record into a new segment. The new segment is constructed in a new buffer; the third byte of the SDW of the newly created segment is set to X'02' if this segment is the last of a multisegment record. If there are other segments, the third byte is set to X'03' to indicate that this segment is neither the first nor the last of a multisegment record. Newly created segments are processed as any other record.

The TRUNC routine operates as follows:

- It receives control when a TRUNC macro instruction is encountered in a processing program.
- It issues a BALR instruction to pass control of the present buffer to the end–of–block routine.
- It issues another BALR instruction to obtain a new buffer through the output–synchronizing–and–error–processing routine, module IGG019AR.
- It determines the address of the first segment of the new buffer and then returns control to the processing program.

Put Module IGG019FJ: Module IGG019FJ presents the processing program with the address of the next available buffer segment for a variable–length record. The Open executor selects and loads this module if the Open parameter list specifies:

Output

and the DCB specifies:

Put

Simple buffering

Locate operating mode

Variable–length (unblocked or blocked) record format, spanned

The module consists of a Put routine and a TRUNC routine.

The Put routine operates as follows:

- It receives control when the processing program issues a PUT macro instruction.
- It determines the address of the next buffer segment by adding the address of the last record or record segment moved to the buffer and the length of that record or record segment. The length of the record segment is in the SDW.
- It checks the buffer to see if there are five or more unused bytes.
- If there are 5 or more unused bytes remaining in the buffer, the Put routine places their address into register 1 for the processing program. The Put routine places the exact number of bytes left in the buffer into register 0 for the processing program. The Put routine then returns control to the processing program.
- If the buffer contains fewer than 5 unused bytes, the routine issues a BALR to the EOB routine. The Put routine issues another BALR instruction to obtain a new buffer through the output–synchronizing–and–error–processing routine, module IGG019AR, and determines the address of the first segment of the new buffer. The Put routine then builds a new block descriptor word (BDW) and returns control to the processing program.

The TRUNC routine causes an EOB condition by setting the DCBRECAD and DCBEOBAD fields so that they are equal. It then returns control to the processing program.

Put Module IGG019FL: Module IGG019FL moves the current variable–length record into the next available buffer segment. The Open executor selects and loads this module if the Open parameter list specifies:

Output

and the DCB specifies:

Put

Simple buffering

Move operating mode

Variable–length (unblocked or blocked) record format, spanned

The module consists of a Put routine and a TRUNC routine.

The Put routine operates as follows:

- It receives control when the processing program issues a PUT macro instruction.
- It determines the possible location of the next buffer segment by adding the length of the previous record or record segment to the previous buffer segment address. This address is in the DCBRECAD field.
- It then compares the length of the next record with the remaining buffer capacity.
- If the record will fit, the routine moves the record, updates the length field of the block descriptor word (BDW), and checks for blocked records.
- If blocked records are specified, the routine returns control to the processing program. If unblocked records are specified, the routine issues a BALR instruction to pass the current buffer to the EOB routine. The Put routine issues another BALR instruction to obtain a new buffer through the output–synchronizing–and–error–processing routine, module IGG019AR. The Put routine then builds a new block descriptor word (BDW) and returns control to the processing program.
- If the record will not fit, the routine determines whether there are five or more unused bytes remaining in the buffer. If there are, the Put routine breaks the current record so that the first segment fills the buffer. The remaining segment is placed in subsequent buffers. The length field in the segment descriptor word (SDW) of the first segment is updated to reflect the length of the segment. The third byte of this SDW is set to X'01' to indicate that this segment is the first of a multisegment record. After writing the buffer, the Put routine does not return control to the processing program until the entire record has been processed. The routine forms the remainder of the current record into a new segment, which is constructed in a new buffer. The third byte of the SDW of the newly created segment is set to X'02' if this segment is the last of a multisegment record. If there are other segments, the third byte is set to X'03' to indicate that this segment is neither the first nor the last of a multisegment record. Newly created segments are processed as any other record.

The TRUNC routine operates as follows:

- It receives control when a TRUNC macro instruction is encountered in a processing program.
- It issues a BALR instruction to pass control of the present buffer to the end–of–block routine.
- It issues another BALR instruction to obtain a new buffer through the output–synchronizing–and–error–processing routine, module IGG019AR.
- It determines the address of the first segment of the new buffer and then returns control to the processing program.

Exchange Buffering Put Routines

Exchange buffering Put routines use buffers whose addresses and lengths are in the channel program. For unblocked records, a buffer address and length are in one channel command word (CCW). For blocked records, addresses of buffer segments are in successive CCWs (though the segments themselves are not necessarily located next to one another). In each pass through an exchange buffering Get routine, it determines:

- The address of the next buffer segment.
- Whether an output buffer is to be scheduled for emptying.
- Whether a new empty buffer is needed.

These three determinations are made at every pass through a Put routine.

If the records are unblocked, a Put routine finds the address of the next buffer in the Write CCW for the next buffer.

If the records are blocked, a Put routine finds the address of the next buffer segment in the next Write CCW. The next CCW is found by adding 8 to the address of the previous CCW, the value in the DCB field DCBCCCW.

If an output buffer is to be scheduled for emptying, a Put routine passes control to an end–of–block routine to cause scheduling of the buffer. An end–of–block routine passes control to the I/O supervisor to have it schedule the buffer. After scheduling, the I/O supervisor returns control to the end–of–block routine, and it returns control to the Put routine.

If a new empty buffer is needed, a Put routine passes control to the output–synchronizing–and–error–processing routine. If the channel program for the next buffer has been executed without error, the synchronizing routine enters the address of the input/output block (IOB) that points to that channel program into the DCBIOBA field in the DCB.

An output buffer is to be scheduled for emptying and a new buffer is needed if an end–of–block condition exists. When using exchange buffering with an output data set, the buffer can be scheduled for emptying when the address of the last record has been placed in the last CCW or a record has been moved into the last segment. Accordingly, an end–of–block test is made before leaving the routine. This test determines whether the buffer is to be scheduled. Another test is made on entry to determine whether a new buffer is needed. An end–of–block condition exists for unblocked records each time the routine is entered; for blocked records, it exists if the

address of the current CCW (in DCBCCCW field) and the address of the last CCW (in DCBLCCW field) are the same.

A TRUNC routine sets an end–of–block condition to empty the buffer. This end–of–block condition is processed so that the next entry to the Put routine permits it to operate as usual. Successive entries to a TRUNC routine without intervening entries to a Put routine cause the TRUNC routine to return control without performing any processing.

The processing performed by the Open executor for an output data set using QSAM and exchange buffering includes setting an end–of–block condition. On the first entry to an exchange buffering Put routine, it processes this condition as usual.

There are two exchange buffering Put routines. Figure 8 lists both of these routines and the conditions that cause either routine to be used. The Open executor selects one of the routines, loads it, and places its address into the DCBPUT field. The table shows, for example, that if output, Put, exchange, move, and unblocked record format are specified, module IGG019EE is selected for use as the Put routine.

Access Method Options	**Selections**						
Output, Put/PUTX, Exchange	X	X	X	X	X	X	X
Move mode	X	X	X			X	
Substitute mode				X	X		X
Unblocked record format	X	X	X	X	X		
Blocked record format						X	X
Fixed–length record format	X			X		X	X
Variable–length or record format–D		X					
Undefined–length record format			X		X		
Put Modules							
IGG019EE	EE	EE	EE	EE	EE		
IGG019EF						EF	EF

Figure 8. Module Selector — Exchange Buffering Put Modules

Put Module IGG019EE: Module IGG019EE puts an unblocked record into the next buffer. The Open executor selects and loads this module if the Open parameter list specifies:

Output

and the DCB specifies:

Put, PUTX

Exchange buffering

Unblocked record format

Move operating mode and fixed–length, format–D, variable–length, or undefined–length record format; or substitute operating mode and fixed– or undefined–length record format

The module consists of a Put routine, a PUTX routine, and a TRUNC routine.

The Put routine operates as follows for the move mode:

- It receives control if a PUT macro instruction is encountered in the processing program.
- It passes control to the output–synchronizing–and–error–processing routine, module IGG019AR, to obtain the next buffer.
- It determines the address of the Write–Data CCW, enters the length in the CCW, and finds the buffer address.
- It moves the record from the work area into the buffer.
- It passes control to the end–of–block routine to cause scheduling of the buffer.
- It returns control to the processing program.

The Put routine operates as follows for the substitute mode:

- It receives control when a PUT macro instruction is encountered in a processing program.
- It passes control to the output–synchronizing–and–error–processing routine, module IGG019AR, to obtain the next buffer.
- It determines the address of the Write–Data CCW, enters the length in the CCW, and finds the buffer address.
- It exchanges the address of the work area for the address of the buffer area.
- It passes control to the end–of–block routine to cause scheduling of the buffer for output.
- It returns control and the address of the buffer to the processing program.

The PUTX routine operates as follows if the input DCB specifies simple buffering:

- It receives control when a PUTX macro instruction is encountered in a processing program.
- It passes control to the output–synchronizing–and–error–processing routine, module IGG019AR, to obtain the next buffer.
- It finds the address of the input buffer in the DCBRECAD field of the input DCB and the input buffer length in the DCBLRECL field.
- It moves the record from the input buffer to the output buffer and enters the length in the Write–Data CCW.
- It passes control to the end–of–block routine to cause scheduling of the buffer for output.
- It returns control to the processing program.

The PUTX routine operates as follows if the input DCB specifies exchange buffering:

- It receives control when a PUTX macro instruction is encountered in a processing program.
- It passes control to the output–synchronizing–and–error–processing routine, module IGG019AR, to obtain the next buffer.

- It finds the address of the Read CCW and the length of the buffer in the DCBCCCW and DCBLRECL fields of the input DCB. It also finds the address of the Write CCW in the DCBCCCW field of the output DCB.
- It exchanges the buffer addresses and enters the length into the Write CCW.
- It passes control to the end–of–block routine to cause scheduling of the buffer for output.
- It returns control to the processing program.

The TRUNC routine receives control when a TRUNC macro instruction is encountered in a processing program; it returns control to the processing program without performing any processing.

Put Module IGG019EF: Module IGG019EF puts a blocked record into the next buffer segment. The Open executor selects and loads this module if the Open parameter list specifies:

Output

and the DCB specifies:

Put, PUTX

Exchange buffering

Move or substitute operating mode

Fixed–length blocked record format

The module consists of a Put routine, a PUTX routine, and a TRUNC routine.

The Put routine operates as follows:

- It receives control when a PUT macro instruction is encountered in the processing program.
- If there is an end–of–block condition on entry to the routine, it passes control to the output–synchronizing–and–error–processing routine, module IGG019AR, to obtain the next buffer.
- If the move mode is used and either there is no end–of–block condition or control has returned from the synchronizing routine, the Put routine moves the record from the work area into the next buffer segment.
- If the substitute mode is used and either there is no end–of–block condition or control has returned from the synchronizing routine, the Put routine exchanges the current buffer segment address of the output DCB for either the current buffer segment address of the input DCB or the address of a work area.
- It tests for another end–of–block condition to determine if the buffer is to be scheduled for output.
- If there is no end–of–block condition, it returns control to the processing program.
- If there is an end–of–block condition, it passes control to the end–of–block routine to cause scheduling of the buffer. On return of control to the Put routine, it returns control to the processing program.

The PUTX routine operates as follows:

- It receives control when a PUTX macro instruction is encountered in the processing program.
- If there is an end–of–block condition on entry to the routine, it passes control to the output–synchronizing–and–error–pprocessing routine, module IGG019AR, to obtain the next buffer.
- If the input DCB uses simple buffering, and either there is no end–of–block condition or control has returned from the synchronizing routine, the PUTX routine moves the record from the input buffer segment into the next output buffer segment.
- If the input DCB uses exchange buffering, and either there is no end–of–block condition or control has returned from the synchronizing routine, the PUTX routine exchanges the buffer segment addresses of the current output and input CCWs.
- It tests for another end–of–block condition to determine if the buffer is to be scheduled for output.
- If there is no end–of–block condition, it returns control to the processing program.
- If there is an end–of–block condition, it passes control to the end–of–block routine to cause scheduling of the buffer for output. On return of control to the PUTX routine, it then returns control to the processing program.

The TRUNC routine operates as follows:

- It receives control when a TRUNC macro instruction is encountered in a processing program.
- It returns control to the processing program without any further processing if the buffer was scheduled for output on the preceding entry into the Put or PUTX routine.
- It turns off the chain–data bit in the CCW used in the preceding pass through the Put or PUTX routine. (The chain–data bit is set on in every CCW in the normal course of operation of the Put or PUTX routine to offset any possible prior truncation.)
- It passes control to the end–of–block routine to cause scheduling of the buffer for output. On return of control, the TRUNC routine then returns control to the processing program.

Update Mode PUTX Routine

The update mode PUTX routine differs from other Put routines in that it shares its buffers (as well as the DCB and the IOBs) with the update mode Get routine. It is the update mode Get routine that determines the address of the segment, when the end of the buffer is reached and a new buffer is needed. Thus, all that remains for the PUTX routine is to flag the block for output.

There is one update mode Put routine; it is part of module IGG019AE, which is described under "Update Mode Get Routine" (see Figure 6).

End–of–Block Routines

The end–of–block routines are selected for use with a particular data set on the basis of the access conditions specified by the processing program for that data set.

Unless INOUT or OUTIN is specified in the Open parameter list, one end–of–block routine is selected. If INOUT or OUTIN are specified, two end–of–block routines may be required. When user–totaling is specified, a special user–totaling routine is executed in conjunction with one of the end–of–block routines.

An end–of–block routine receives control from a Get or a Put routine (when using QSAM), or from a Read or Write routine (when using BSAM).

Flow of control for QSAM end–of–block routines is shown in Diagram F, QSAM Flow of Control, in Section 5.

Control passes from an end–of–block routine to the I/O supervisor, except when a channel program is chained to another one not yet executed. End–of–block routines provide device–oriented entries for the channel program, such as control characters and auxiliary storage addresses.

If the American National Standard Code for Information Interchange (ASCII) is used, routines IGG019CC and IGG019CW issue an XLATE macro instruction which translates the entire buffer from EBCDIC to ASCII before writing the buffer. If format–D records are specified, the record descriptor words are converted from binary form to decimal form prior to translation.

End–of–block routine descriptions are grouped as follows:

- Ordinary end–of–block routines. These routines perform device–oriented processing when normal channel–program scheduling is used, except when it is used with an output data set with track overflow.
- Chained channel–program scheduling end–of–block routines. These routines perform device–oriented processing and attempt to chain channel programs when chained channel–program scheduling is used.
- Track–overflow, end–of–block routine. This routine performs device–oriented processing. It computes segment lengths and constructs count fields when track overflow, which uses normal channel–program scheduling, is used with an output data set.
- User–totaling routine. This routine moves the contents of the user's totaling area to the user–totaling save area pointed to by the DEB.

Ordinary End–of–Block Routines

Ordinary end–of–block routines process channel programs for all devices. This processing is independent of the progress of a previous channel program and causes access to proceed one channel program at a time. In the case of output data sets on direct–access devices, the routines limit the size of the block to the track capacity. For direct–access devices, an ordinary end–of–block routine computes auxiliary storage addresses for output data sets and input data sets with fixed–length standard record format to avoid end–of–track interruptions. For unit–record devices, these routines process control characters and PRTOV macro instructions. For an input data set with track overflow, progression from track to track is controlled by the track–overflow bit in the overflowing segment, not by computation of the end–of–block routine nor by an entry in the channel program.

Access Method Options	Selections																							
Normal channel program scheduling	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Input, or					X	X				X	X	X	X	X	X									
Update				X					X															
Output, or													X		X									
INOUT, OUTIN					X					X		X	X	X	X					X				
Card reader or paper tape reader	X	X					X																	
Printer or Card Punch																X	X	X	X	X				
Print (3525)																						X	X	
Interpret Punch (3525)																								X
Data Protection Image (3525)																					X			
Magnetic tape			X					X																
Direct–access storage					X					X		X	X	X	X									
Track Overflow						X					X													
Record format not fixed–length standard					X					X														
Record format is fixed–length standard												X		X										
No control character																X								
Machine control character																	X							
ANSI control character																			X					
PRTOV–No user exit																X	X		X					
label = (,,,IN) or LABEL = (,,,OUT) on DD card [1]																				X				
User totaling facility							X	X	X	X	X			X	X									
Associated Data Set (3525)		X																X		X		X		
End of Block Modules																								
IGG019AX[2]							AX	AX	AX	AX	AX			AX	AX									
IGG019CC	CC	CC	CC	CC	CC	CC																		
IGG019CD												CD	CD											
IIGG019CE																CE	CE	CE						
IGG019CF																			CF	CF				
IGG019CT[3]																					CT			
IGG019FK																						FK		
IGG019FQ																							FQ	FQ
IGG019FU																								FU
IGG019TC							TC	TC	TC	TC	TC													
IGG019TD														TD	TD									

[1] When either of these LABEL subparameters is specified and the data set is opened for INOUT or OUTIN, the OPEN executor calls module IGG019CT, in addition to one of the other end–of–block routines

[2] This module is described later in this section under "Track Overflow and User Totaling Save Routines"

Figure 9. Module Selector — Ordinary End–of–Block Modules

Figure 9 lists the routines available and the conditions that cause a particular routine to be used. For QSAM, the Open executor selects one of the routines, loads it and places its address into the DCBEOB field. For BSAM and BPAM, the Open executor selects one of the routines, loads it, and places its address into both the DCBEOBR and DCBEOBW fields. If INOUT or OUTIN is specified, a second end–of–block routine may be selected and loaded. Its address replaces one of the duplicate addresses in the DCB. Figure 9, for example, shows that when normal channel–program scheduling is used and the device type is magnetic tape, routine IGG019CC is selected and loaded for use as the end–of–block routine for that DCB.

End–of–Block Module IGG019CC: Module IGG019CC causes a channel program to be scheduled.

If ASCII coding is used, the entire output buffer is translated from EBCDIC to ASCII.

The Open executor selects and loads this module if one of the following conditions exists:

The DCB specifies normal channel–program scheduling and magnetic tape, card reader, or paper tape as the device type.

The data set is opened for Input, and the DCB specifies normal channel–program scheduling, direct–access storage device, and a record format other than fixed–length standard.

The data set is opened for INOUT or OUTIN, and the DCB specifies normal channel–program scheduling, direct–access storage device and a record format other than fixed–length standard. The address of this module is placed in the DCBEOBR field.

The data set is opened for Update.

An associated data set is being used during a 3525 operation.

The module operates as follows:

- It receives control when a Get or Put routine finds that a buffer is ready to be scheduled, or at the conclusion of the processing performed by a Read or Write routine.
- If the device type is magnetic tape, record format is variable, control is received from a Put or Write routine, and a check is made to see if at least 18 bytes are to be written. If not, the record is padded with binary zeros up to 18 bytes or blocksize, whichever is less; however, with the ASCII feature, format–D records are padded with the ASCII padding character, X'5F', instead of the zeros. An EXCP macro instruction is issued and control is returned to the Put or Write routine.
- If the device type is magnetic tape and either the record format is not variable or control is not gained from a Put or Write routine, an EXCP macro instruction is issued and control is returned to the Get, Put, Read, or Write routine.
- If the device type is direct access and more than one IOB is associated with the DCB, the module issues an EXCP macro instruction and returns control to the Get or Read routine.

- If a 3525 associated data set is being used, a test is made to determine the status of the Read–sequence flag.

  a. If the Read–sequence flag (DCBQSWS field) is on and the associated data set is not Read and Print, a WTP message is issued, which indicates that either the Get or Read sequence is invalid. An ABEND (003) is issued with a return code of 01. If the Read–sequence flag is off, the macro sequence is assumed to be valid and the Read–sequence flag is turned on.

  b. Tests are made to determine if the associated data set is either Read, Punch, and Print, or Read and Punch.

  c. If either Read, Punch, and Print, or Read and Punch is specified in the FUNC parameter, a test is made to determine the status of the Punch–sequence flag. If the Punch–sequence flag (DCBQSWS field) is on, it is turned off. (This indicates to modules IGG019CE and IGG019CF that their calling routine is in the proper sequence.)

  d. If the associated data set is not Read, Punch, and Print, or Read and Punch, it is assumed that Read and Print is being used.

  e. A test is made to determine the status of the Print–sequence flag (DCBQSWS).

  f. If the Print–sequence flag is on, it is assumed that the Print command has been issued. It is turned off so that proper sequencing may continue. If the Print–sequence flag is off, it is assumed that the Print command has not been issued.

- If the device type is direct access and only one IOB is associated with the DCB, the module copies the DCBFDAD field in the DCB into the IOBSEEK field in the IOB, issues an EXCP macro instruction, and returns control to the Get or Read routine.

End–of–Block Module IGG019CD: Module IGG019CD schedules a channel program after determining that the next block fits on a track within the allocated extents.

The Open executor selects and loads this module if one of the following conditions exists:

The data set is opened for output and the DCB specifies normal channel–program scheduling, no track–overflow, and direct–access storage as the device type.

The data set is opened for input and the DCB specifies normal channel–program scheduling with direct–access storage as the device type.

The data set is opened for INOUT or OUTIN and the DCB specifies direct–access device storage. If the record format (also specified in the DCB) is other than fixed–length standard, the address of this module is placed in the DCBEOBW field. If the record format is fixed–length standard, the address of this module is placed in both the DCBEOBR and DCBEOBW fields.

The module operates as follows:

- It receives control when a Get or Put routine finds that a buffer is ready to be scheduled, or at the conclusion of the processing performed by a Read or Write routine.

- It calculates the block length using the overhead value for the last record. (This value is found in the resident I/O device table. The address of the table is in the DCBDVTBL field.) It compares the calculated block length with the value in the DCBTRBAL field of the DCB.
- If the block length is equal to or less than the DCBTRBAL field value, the module determines that the block fits on the track.
- If the block length exceeds the DCBTRBAL field value, the module finds the next track as follows:

  It converts the full device address (MBBCCHHR) of the present track into a relative address (TTR) by passing control to the IECPRLTV routine.

  It adds 1 to the value of TT.

  It passes control to the IECPCNVT routine, which converts the relative address of the next track into the full device address.
- If there is another track in the allocated extents, its full address has been entered in the DCBFDAD field and the block fits on the track.
- If there is no other track in the allocated extents (as shown by the error return code from IECPCNVT routine), an EOV condition exists. The module sets the DCBCIND1 field in the DCB and the CSW field in the IOB to show this, and returns control to the Get, Put, Read, or Write routine without issuing an EXCP macro instruction. The EOV condition is eventually recognized and processed in QSAM by the synchronizing routine and in BSAM by the Check routine.
- When the module determines that the block fits on the track, the module calculates the actual block length, using the overhead value for other than the last record. (This value is found in the resident I/O device table.) It adjusts the value in the DCBTRBAL field by this amount and updates the DCBFDAD field and the ID field of the count area of the block which is located immediately after the channel program. It then issues an EXCP macro instruction and returns control to the Get, Put, Read, or Write module.

End–of–Block Module IGG019CE: Module IGG019CE, if necessary, modifies channel programs for unit record output devices when ANSI control characters are not used. The module then causes scheduling of the channel program, whether it was modified or not. The Open executor selects and loads this module if the DCB specifies:

Normal channel–program scheduling

Punch, or printer

Machine control character, or no control character

The module operates as follows:

- It receives control when a Put routine finds that a buffer is ready to be scheduled, or at the conclusion of the processing performed by a Write routine.
- It adjusts, in the channel program, the length and starting address either for the length field of variable–length records or for a control character. If there are variable–length records and a control character, the module adjusts for both.
- If a control character is present, it inserts it as the command byte of the Write channel command word (CCW).

- It tests the DCB field at location DCBDEVT+1 for a PRTOV mask. If a PRTOV mask is present, the module temporarily inserts it into the length field of the NOP CCW and sets the first bit in the IOB. The PRTOV appendage IGG019CL tests for the presence of the IOB bit and the CCW mask.

- If an associated data set is being used, a test is made to determine the status of the Punch–sequence flag.

  a. If the Punch–sequence flag (DCBQSWS) is on and the associated data set is not Punch and Print, a WTP message is issued which indicates that either the Put or Write sequence is invalid. An ABEND (003) is issued with a return code of 02. If the Punch–sequence flag is off, the macro sequence is assumed to be valid and the Punch–sequence flag is turned on.

  b. A test is made to determine if the associated data set is Read, Punch, and Print. If Read, Punch, and Print is specified in the FUNC parameter, a test is made to determine the status of the Read–sequence flag.

  c. If the Read–sequence flag is on, it is turned off. This allows proper sequencing to continue. If the Read–sequence flag is off, an ABEND is issued.

  d. A test is made to determine the status of the Print–sequence flag.

  e. If the Print–sequence flag is on, proper sequencing continues. If it is off, modules IGG019CE and IGG019CF continue with their normal functions.

  f. If the associated data set is Punch and Print, the status of the Print–sequence flag is determined as previously explained for module IGG019CC.

- It issues an EXCP macro instruction and returns control to the Put or Write routine.

End–of–Block Module IGG019CF: Module IGG019CF modifies channel programs for unit record output devices when an American National Standards Institute (ANSI) control character is present. The module then causes scheduling of the channel program, whether it was modified or not. The Open executor selects and loads this module if the DCB specifies:

Normal channel–program scheduling

Punch or printer

ANSI control character

The module operates as follows:

- It receives control when a Put routine finds that a buffer is ready to be scheduled, or at the conclusion of the processing performed by a Write routine.

- It adjusts, in the channel program, the length and starting address for the control character, and for the length field of variable–length records.

- It translates the control character and inserts it as the command byte of the control channel command word (CCW) which precedes the Write CCW.

- It tests the DCB field at location DCBDEVT+1 for a PRTOV mask. If a PRTOV mask is present, the module inserts it into the length field of the control CCW and sets the first bit in the IOB. The PRTOV appendage IGG019CL tests for the presence of the IOB bit and the CCW mask.
- If an associated data set is being used, a test is made to determine the status of the Punch–sequence flag.

  a. If the Punch–sequence flag (DCBQSWS) is on and the associated data set is not Punch and Print, a WTP message is issued which indicates that either the Put or Write sequence is invalid. An ABEND (003) is issued with a return code of 02. If the Punch–sequence flag is off, the macro sequence is assumed to be valid and the Punch–sequence flag is turned on.

  b. A test is made to determine if the associated data set is Read, Punch, and Print. If Read, Punch, and Print is specified in the FUNC parameter, a test is made to determine the status of the Read–sequence flag.

  c. If the Read–sequence flag (DCBQSWS) is on, it is turned off. This allows proper sequencing to continue. If the Read–sequence flag is off, an ABEND is issued.

  d. A test is made to determine the status of the Print–sequence flag (DCBQSWS).

  e. If the Print–sequence flag is on, proper sequencing continues. If it is off, modules IGG019CE and IGG019CF continue with their normal functions.

  f. If the associated data set is Punch and Print, the status of the Print–sequence flag is determined, as previously explained for module IGG019CC.
- It issues an EXCP macro instruction and returns control to the Put or Write routine.

End–of–Block Module IGG019CT: Module IGG019CT sets error indicators in the user's DCB and IOB. The Open executor selects and loads this module if the following conditions exist:

The data set is opened for INOUT and the DD card specifies LABEL=(,,,IN)

or

The data set is opened for OUTIN and the DD card specified LABEL=(,,,OUT)

The module operates as follows:

- It receives control and sets error indicators in the user's DCB and IOB when either of the following conditions exists:

  The DD card specifies LABEL=(,,,IN), the data set is opened for INOUT, and a WRITE macro instruction is issued,

  The DD card specifies LABEL=(,,,OUT), the data set is opened for OUTIN, and a READ macro instruction is issued.

End–of–Block Module IGG019FK: Module IGG019FK causes a channel program to be scheduled. The Open executor selects and loads this module, if the following conditions are described in the DCB:

Data protection image (DPI) is specified for the 3525 with a Read and Punch, or Read, Punch, and Print file with normal channel–program scheduling.

The module operates as follows:

- It receives control when a Put routine finds that a buffer is ready to be scheduled, or at the conclusion of the processing performed by a Write routine.
- If the Read associated data set has been opened, a test is made to determine the status of the Read–sequence flag.
- If the Read associated data set has not been opened, or if the Read–sequence flag is off, a WTP message is issued which indicates that the sequence is invalid. An ABEND (003) is then issued with a return code of 02. If the Read–sequence flag is on (indicating proper sequencing), it is turned off.
- A test is then made to determine the status of the Punch–sequence flag (DCBQSWS field). If the Punch–sequence flag is on, a WTP message is issued, followed by an ABEND (003). If the Punch–sequence flag is off, it is turned on so that proper sequencing may continue.
- It then establishes the buffer area (for the punch operation) according to the format of the data protection image. If a byte in the DPI is blank (X'40'), the module blanks out the corresponding byte in the output punch buffer. If the byte is not blank, the output buffer is not altered. Both areas are 80 bytes in length.
- It returns control to either the Put or Write routine that called it.

End–of–Block Module IGG019FQ: Module IGG019FQ causes a channel program to be scheduled to the 3525 Printer. The Open executor selects and loads this module, if the following conditions exist:

A Print; Read, Punch, and Print; Read and Print; or Punch and Print file is specified for the 3525 with either a machine control character, an ANSI control character, or no control character at all with normal channel–program scheduling.

The module operates as follows:

- It receives control when a Put routine finds that a buffer is ready to be scheduled, or at the conclusion of the processing performed by a Write routine.
- If either a Read, Punch, and Print or Punch and Print associated data set has been specified, a test is made to determine the status of the Print sequence flag. If the Print–sequence flag is on, the CCW pointer is modified to point to the Print CCW.
- If both the Print– and Punch–sequence flags are off, a WTP message is issued which indicates that the sequence is invalid. An ABEND (003) is then issued with a return code of 03.

- If the Print–sequence flag is off, but the Punch–sequence flag is on, the module locates the Punch DCB and turns off the Punch–sequence flag. The CCW pointer is then modified to point to the Print CCW and the Print–sequence flag is turned on.
- If a Read and Print associated data set is specified and the Print–sequence flag is on, the CCW pointer is modified to point to the Print CCW.
- If the Print–sequence flag is off, but the Read–sequence flag is on, the Read DCB is located and the Read–sequence flag is turned off. The CCW pointer is then modified to point to the Print CCW and the Print–sequence flag is turned on.
- After sequence checking is completed, the module tests for ANSI and machine control characters. If ANSI is specified, the control character is analyzed to determine which line the data is to be printed on. An OR operation is then performed on that line number and the Print CCW.
- If ANSI control characters are not specified, the module tests for record format and machine control characters. If machine control characters are specified, they are inserted into the CCW and the buffer address is increased by one.
- If no control character is specified, and two–line printing is specified in the FUNC parameter, the module tests to determine line positioning on the card. This is reflected in the operation code of the Print CCW.
- If no control character is specified, and multiline printing is specified, tests are again made to determine line positioning. (Output lines are printed on successive lines.)
- If no control characters are specified, or if they are specified and have been processed, or if either two–line or multiline positioning is complete, the module establishes the Write CCW and stores the Start address of the CCW for the input/output supervisor (IOS).
- If the PRTOV macro instruction is specified, a check is made for either channel 9 or 12 (depending on which channel is specified in the PRTOV macro instruction).
- The channel program is then executed and a Wait command is issued. It returns control (via register 14) to either the Put or Write routine that called it.

End–of–Block Module IGG019FU: Module IGG019FU causes a channel program to be scheduled. The Open executor selects and loads this module if one of the following conditions exists:

INTERPRET PUNCH is specified for the 3525 with normal channel–program scheduling.

INTERPRET PUNCH is specified for the 3525 with first control character for stacker selection or with no control character at all.

The module operates as follows:

- It retrieves the data address from the Write CCW.
- It tests for record format to determine if machine control characters or ANSI control characters are being used.

- If either machine or ANSI control characters are being used, the data address is increased by one and the control character is inserted into the command byte of the Write CCW.
- If machine control characters are not specified, the data address remains unchanged.
- The module blanks out a print buffer. (The print buffer is a 64–byte area located 64 bytes past the beginning of the IOB). It then moves the final 16 characters of the output punch buffer into the last 16 bytes of the print buffer.
- The channel program start address is stored in the IOB.
- The channel program is then scheduled for execution.
- It returns control (via register 14) to either the Put or Write routine that called it.

End–of–Block Module IGG019TC: The Open executor selects and loads this module if the user specified the user–totaling facility (that is, if bit 6 is 1 in DCBOPTCD) for his data set and if one of the following conditions exists:

The DCB specifies normal channel–program scheduling and magnetic tape, card reader, or paper tape as the device type.

The data set is opened for Input, and the DCB specifies normal channel–program scheduling, direct–access storage device, and a record format other than fixed–length standard.

The data set is opened for INOUT or OUTIN, and the DCB specifies normal channel–program scheduling, direct–access storage device and a record format other than fixed–length standard. The address of this module is placed in the DCBEOBR field.

The data set is opened for Update.

The module operates as follows:

- It receives control when a Get or a Put routine finds that a buffer is ready to be scheduled, or at the conclusion of the processing performed by a Read or Write routine.
- If the device type is magnetic tape, paper tape, or card reader, the module issues an EXCP macro instruction and returns control to the Get, Put, Read, or Write routine.
- It issues a BALR instruction to the user–totaling save routine, IGG019AX, to place the user's total in the user–totaling save area, which is pointed to by the DEB.
- If the device type is direct access and more than one IOB is associated with the DCB, the module issues an EXCP macro instruction, and returns control to the Get or Read routine.
- If the device type is direct access and only one IOB is associated with the DCB, the module copies the DCBFDAD field in the DCB into the IOBSEEK field in the IOB, issues an EXCP macro instruction, and returns control to the Get or Read routine.

End–of–Block Module IGG019TD: Module IGG019TD schedules a channel program after determining that the next block fits on a track within the allocated extents.

The Open executor selects and loads this module if the user specified the user totaling facility (that is, if bit 6 is 1 in DCBOPTCD) for his data set and if one of the following conditions exists:

The data set is opened for Output, and the DCB specifies normal channel–program scheduling, no track–overflow, and direct–access storage as the device type.

The data set is opened for Input, and the DCB specifies normal channel–program scheduling with direct–access storage as the device type.

The data set is opened for INOUT or OUTIN, and the DCB specifies direct–access storage device. If the record format (also specified in the DCB) is other than fixed–length standard, the address of this module is placed in the DCBEOBW field. If the record format is fixed–length standard, the address of this module is placed in both the DCBEOBR and the DCBEOBW fields.

The module operates as follows:

- It receives control when a Get or a Put routine finds that a buffer is ready to be scheduled, or at the conclusion of the processing performed by a Read or Write routine.
- It issues a BALR instruction to the user–totaling save routine, IGG019AX, to place the user's total in the user–totaling save area, which is pointed to by the DEB.
- It calculates the block length using the overhead value for the last record. (This value is found in the resident I/O device table. The address of the table is in the DCBDVTBL field.) It compares the calculated block length with the value in the DCBTRBAL field of the DCB.
- If the block length is equal to or less than the DCBTRBAL field value, the module determines that the block fits on the track.
- If the block length exceeds the DCBTRBAL field value, the module finds the next track as follows:

  It converts the full device address (MBBCCHHR) of the present track into a relative address (TTR) by passing control to the IECPRLTV routine.

  It adds 1 to the value of TT.

  It passes control to the IECPCNVT routine, which converts the relative address of the next track into the full device address.
- If there is another track in the allocated extents, its full address has been entered in the field DCBFDAD and the block fits on the track.
- If there is no other track in the allocated extents (as shown by the error return code from routine IECPCNVT), an EOV condition exists. The module sets the DCBCIND1 field in the DCB and the CSW field in the IOB to show this, and

returns control to the Get, Put, Read, or Write routine without issuing an EXCP macro instruction. The EOV condition is eventually recognized and processed — in QSAM by the synchronizing routine and in BSAM by the Check routine.

- When the module determines that the block fits on the track, the module calculates the actual block length, using the overhead value for other than the last record. (This value is found in the resident I/O device table.) It adjusts the value in the DCBTRBAL field by this amount, and updates the DCBFDAD field and the ID field of the count area of the block (located immediately after the channel program). It then issues an EXCP macro instruction and returns control to the Get, Put, Read, or Write module.

Chained Channel–Program Scheduling End–of–Block Routines

Chained channel–program scheduling consists of joining the channel programs before execution and disconnecting and posting the channel programs after execution. Joining is performed by the end–of–block routines and mainly uses the input/output block (IOB); disconnecting and posting is performed by appendages and uses the interruption control block (ICB). (For a description of the disconnecting process, refer to the program controlled interruption (PCI) appendages.) The IOB constructed by the Open executor when chained channel–program scheduling is used differs from the IOB used in normal channel–program scheduling. These differences are illustrated in Figure 10 and tabulated in Figure 12.

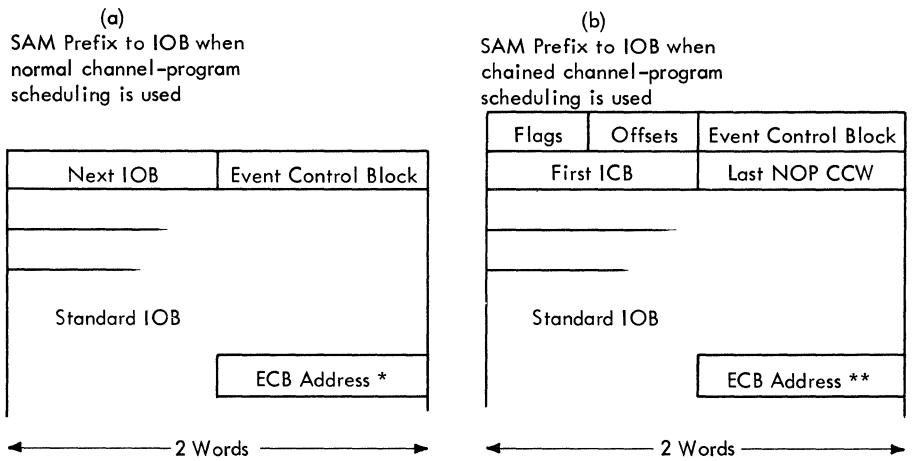


Figure 10. IOB SAM Prefixes for Normal and for Chained Scheduling

These routines join channel programs so that the channel executes successive channel programs without interruption as if they were one continuous channel program. To join the present channel program to one already scheduled, the end–of–block routine finds the last CCW of the preceding channel program by referring to the IOB and changes that CCW from a NOP command to a TIC command. If this joining is performed before the channel attempts to execute (more precisely, before it fetches) that CCW, the joining process is successful. If the execution of the preceding channel program is completed while the routine is operating, the joining is unsuccessful. The

Access Method Options	**Selections**										
Chained channel program scheduling	X	X	X	X	X	X	X	X	X	X	X
Input, or	X	X		X							X
Output			X	X		X	X	X	X	X	
Card reader	X										
Printer or card punch								X	X	X	
Magnetic tape		X	X	X							
Direct–access storage					X	X	X				
No control character								X			
Machine control character									X		
ANSI control character										X	
User–totaling facility				X			X				X
End–of–Block Modules											
IGG019AX[1]				AX			AX				AX
IGG019CV						CV					
IGG019CW	CW	CW	CW		CW						
IGG019CX								CX	CX		
IGG019CY										CY	
IGG019TV							TV				
IGG019TW				TW							TW

[1]This module is described later in this section under Track Overflow and User–Totaling Save Routines.

Figure 11. Module Selector — Chained Channel–Program Scheduling End–of–Block Modules

routine tests the success or failure of the joining by testing whether the IOB has been posted as completed. If the IOB is not posted as completed, control is returned to the calling program. If the IOB is posted, the routine tests the ICB for the current channel program. If completed, control returns to the calling program; if not completed, the routine resets the IOB for the EXCP macro instruction and passes control to the I/O supervisor.

The chained scheduling end–of–block routines, like the ordinary end–of–block routines, provide device–oriented entries for channel programs. For direct–access devices they compute auxiliary storage addresses; for unit–record devices they process control characters. (No processing is performed for the PRTOV macro instruction since it and chained scheduling are mutually exclusive.) There are six chained scheduling end–of–block routines, each of which performs joining and channel program entry processing for a different set of access condition options. Figure 11 lists the available routines and the conditions that cause a particular routine to be used.

For QSAM, the Open executor selects one of the routines, loads it, and places its address into the DCBEOB field. For BSAM and BPAM, the Open executor selects one of the routines, loads it, and places its address into both the DCBEOBR and

DCBEOBW fields. If INOUT or OUTIN is specified, a second end–of–block routine may be selected and loaded. Its address replaces one of the duplicate addresses in the DCB.

Figure 11 shows that when chained scheduling is used, the open mode is Input, the device type is magnetic tape, and routine IGG019CW is selected and loaded for use as the end–of–block routine for the DCB.

End–of–Block Module IGG019CV: Module IGG019CV computes from the track balance (and from further allocated extents on this volume, if necessary) a valid storage address for a channel program for an output data set on a direct–access device and attempts to join the channel program to the preceding one. The Open executor selects and loads this module if the Open parameter list specifies:

Output

and the DCB specifies:

Chained channel–program scheduling

Direct–access storage

The module operates as follows:

- It receives control from a Put routine when that routine finds that a buffer is ready to be scheduled, or from a Write routine at the conclusion of its processing.
- It calculates the block length using the overhead value for a last block on a track. (This value is found in the resident I/O device table. The address of the table is in the DCBDVTBL field.) It compares the calculated block length with the value in the DCBTRBAL field of the DCB.

Prefix Parameter	Normal Scheduling	Chained Scheduling
Number of IOBs	As many as there are buffers or channel programs	Only 1 (there are as many ICBs as there are buffers or channel programs)
Size of SAM prefix	2 words	4 words
Contents of link address field	Address of the next IOB	Flags Offsets
Use of ECB field	Used in QSAM to post channel program execution (in BSAM, the ECB in the DECB is used)	Used in QSAM and BSAM to post a channel program execution that is terminated by channel–end interruption (that is, channel program chaining has been broken)
Contents of IOBFICB field	Field does not exist	Address of the first ICB
Contents of IOBLNOP field	Field does not exist	Address of NOP CCW of last scheduled channel program

Figure 12. Comparison of the IOB SAM Prefixes for Normal and for Chained Scheduling

- If the block is equal to or less than the DCBTRBAL field value, the module determines that the block fits on the track.
- If the block length exceeds the DCBTRBAL field value, the module calculates the next sequential track address and compares it with the end address of the current extent shown in the data extent block (DEB).
- If no end–of–extent condition exists, it determines that the block fits on the track.
- If an end–of–extent condition exists, it seeks a new extent in the DEB.
- If a new extent exists, it updates the DCBFDAD and DCBTRBAL fields and determines that the block fits on the track.
- If there is no further extent, an EOV condition exists. The module sets the DCBCIND1 field in the DCB and the CSW field in the IOB to show end–of–volume, and returns control to the Get, Put, Read, or Write routine without issuing an EXCP macro instruction. The EOV condition is eventually recognized and processed — in QSAM by the synchronizing routine, and in BSAM by the Check routine.

- If the module determines that the block fits on the track, the module calculates the actual block length using the overhead value for a block that is not the last on a track. (This value is found in the resident I/O device table.) It adjusts the value in the DCBTRBAL field by this amount and updates the DCBFDAD field and the ID field of the count area of the block located immediately after the channel program.

- If the block fits on the track, the module next attempts to join the channel program for the current buffer to the preceding channel program (that is, chain schedule) by:

  Setting the ICB to not–complete.

  Inserting the address of either the Write or the Search CCW of this channel program into the NOP CCW of the preceding channel program. The address of the Write CCW is inserted if the present and the preceding channel programs address the same track. The address of the Search CCW is inserted if the present and the preceding channel programs address different tracks. In this case, the Search CCW addresses record zero of the next track.

  Changing the NOP CCW in the preceding channel Program to a TIC CCW.

  Updating the SAM IOB prefix block to point to the end of the current channel program.

- It determines whether the joining was successful by testing the ECB (pointed to by the IOB) to see if the I/O supervisor has posted the I/O event as completed.

- If the I/O supervisor did not post the event as completed, the joining was successful and the routine returns control to the calling routine.

- If the I/O supervisor did post the event as completed, the routine tests the ICB for the present channel program to find whether the joining was successful or not.

- If the present ICB remains unposted, the present channel program was not joined to the preceding one. The routine prepares to cause restart of the channel by copying the Seek address and the channel program start address from the current ICB into the IOB and uses the EXCP macro instruction to schedule the channel program. It then returns control to the calling routine.

- If the present ICB is posted as completed, the present channel program was joined successfully. (The routine was interrupted long enough, between the joining and the testing, for the channel program to be executed and for the channel–end appendage to post the ICB.) The routine returns control to the calling routine.

End–of–Block Module IGG019CW: Module IGG019CW attempts to join the present channel program to the last one in the chain of scheduled channel programs. If ASCII is used, the entire output buffer is translated from EBCDIC to ASCII. The Open executor selects and loads this module if one of the following conditions exists:

The Open parameter list specifies Input and the DCB specifies chained channel–program scheduling and any device.

The Open parameter list specifies Output and the DCB specifies chained channel program scheduling and magnetic tape.

The module operates as follows:

- It receives control from a Get or Put routine when the routine finds that a buffer is ready to be scheduled, or from a Read or Write routine at the conclusion of its processing.
- If the device type is magnetic tape, the routine determines the increment value and stores it in the ICB.
- If the device is magnetic tape, the record format is variable, and control is received from a Put or Write routine, a check is made to see if at least 18 bytes are to be written. If not, the record is padded with binary zeros up to 18 bytes or blocksize, whichever is less; however, with the ASCII feature, format–D records are padded with the ASCII padding character, X'5F', instead of zeros.
- The module attempts to join the channel program for the current buffer to the preceding channel program (that is, chain schedule) by:

  Setting the ICB to not–complete.

  Inserting the address of the current channel program into the NOP CCW of the preceding channel program.

  Changing the NOP CCW in the preceding channel program to a TIC CCW.

  Updating the SAM IOB prefix block to point to the end of the current channel program.
- It determines whether the joining was successful by testing the ECB (pointed to by the IOB) for a completion posting by the I/O supervisor.
- If the I/O supervisor did not post the event as completed, the joining was successful and the routine returns control to the calling routine.
- If the I/O supervisor did post the event as completed, the routine tests the ICB for the present channel program to find whether the joining was successful or not.
- If the present ICB remains unposted, the present channel program was not joined to the preceding one. The routine prepares to cause restart of the channel by copying the channel program Start address (and the Seek address, if direct–access storage) from the current ICB into the IOB, and uses the EXCP macro instruction to cause scheduling of the channel program. It then returns control to the calling routine.
- If the present ICB is posted as completed, the present channel program was joined successfully. (The routine was interrupted long enough, between the joining and the testing, for the channel program to be executed and for the channel–end appendage to post the ICB.) The routine returns control to the calling routine.

End–of–Block Module IGG019CX: Module IGG019CX, if necessary, modifies channel programs for unit–record output devices when ANSI control characters are not used. The module then attempts to join the current channel program to the preceding one. The Open executor selects and loads this module if the DCB specifies:

Chained channel–program scheduling

Printer or card punch

No control character, machine control character

The module operates as follows:

- It receives control from a Put routine when the routine finds that a buffer is ready for scheduling, or from a Write routine at the conclusion of its processing.
- It adjusts the length entry and the start address entry in the channel program for either a control character or a variable–length block length field or for both, if both are present.
- It inserts the control character, if present, as the command byte of the Write channel command word (CCW).
- It attempts to join the channel program for the current buffer to the preceding channel program (that is, chain schedule) by:

  Setting the ICB to not–complete.

  Inserting the address of the current channel program into the NOP CCW of the preceding channel program.

  Changing the NOP CCW in the preceding channel program to a TIC CCW.

  Updating the SAM IOB prefix block to point to the end of the current channel program.
- It determines whether the joining was successful by testing the ECB (pointed to by the IOB) to see if the I/O supervisor has posted the I/O event as completed.
- If the I/O supervisor did not post the event as completed, the joining was successful and the routine returns control to the calling routine.
- If the I/O supervisor did post the event as completed, the routine tests the ICB for the present channel program to find whether the joining was successful or not.
- If the present ICB remains unposted, the present channel program was not joined to the preceding one. The routine prepares to cause restart of the channel by copying the channel program Start address from the current ICB into the IOB, and uses the EXCP macro instruction to cause scheduling of the channel program. It then returns control to the calling routine.

End–of–Block Module IGG019CY: Module IGG019CY modifies channel programs for unit record output devices when ANSI control characters are used. The module then attempts to join the current channel program to the preceding one. The Open executor selects and loads this module if the DCB specifies:

Chained channel–program scheduling

Printer or card punch

ANSI control character

The module operates as follows:

- It receives control from a Put routine that finds a buffer is to be scheduled, or from a Write routine at the conclusion of its processing.
- It adjusts the length entry and the Start–address entry in the channel program for either the control character or a variable–length block length field or for both, if both are present.
- It translates the control character and inserts it as the command byte of the Control CCW (which precedes the Write CCW).
- It attempts to join the current channel program to the preceding one (that is, chain schedule) by:

  Setting the ICB to not–complete.

  Inserting the address of the current channel program into the NOP CCW of the preceding channel program.

  Changing the NOP CCW in the preceding channel program to a TIC CCW.

  Updating the SAM IOB prefix block to point to the end of the current channel program.
- It determines whether the joining was successful by testing the ECB (pointed to by the IOB) to see if the I/O supervisor has posted the I/O event as completed.
- If the I/O supervisor did not post the event as completed, the joining was successful and the routine returns control to the calling routine.
- If the I/O supervisor did post the event as completed, the routine tests the ICB for the present channel program to find whether the joining was successful or not.
- If the present ICB remains unposted, the present channel program was not joined to the preceding one. The routine prepares to cause restart of the channel by copying the channel program Start address from the current ICB into the IOB, and uses the EXCP macro instruction to cause scheduling of the channel program. It then returns control to the calling routine.
- If the present ICB is posted as completed, the present channel program was joined successfully. (The routine was interrupted long enough, between the joining and the testing, for the channel program to be executed and for the channel–end appendage to post the ICB.) The routine returns control to the calling routine.

End–of–Block Module IGG019TV: Module IGG019TV computes from the track balance (and from further allocated extents on this volume, if necessary) a valid storage address for a channel program for an output data set on a direct–access device and attempts to join the channel program to the preceding one. The Open executor selects and loads this module if the user specified the user–totaling option (that is, if bit 6 is 1 in DCBOPTCD) for his data set and if the Open parameter list specifies:

Output

and the DCB specifies:

Chained channel–program scheduling

Direct–access storage

The module operates as follows:

- It receives control from a Put routine that finds a buffer is ready to be scheduled, or from a Write routine at the conclusion of its processing.
- It issues a BALR instruction to the user–totaling save routine, IGG019AX, to place the user's total in the user–totaling save area, which is pointed to by the DEB.
- It calculates the block length using the overhead value for a last block on a track. (This value is found in the resident I/O device table. The address of the table is in the DCBDVTBL field.) It compares the calculated block length with the value in the DCBTRBAL field of the DCB.
- If the block length is equal to or less than the DCBTRBAL field value, the module determines that the block fits on the track.
- If the block length exceeds the DCBTRBAL field value, the module calculates the next sequential track address and compares it with the end address of the current extent shown in the data extent block (DEB).
- If no end–of–extent condition exists, it determines that the block fits on the track.
- If an end–of–extent condition exists, it seeks a new extent in the DEB.
- If a new extent exists, it updates the DCBFDAD and the DCBTRBAL fields and determines that the block fits on the track.
- If there is no further extent, an EOV condition exists. The module sets the DCBCIND1 field in the DCB and the CSW field in the IOB to show end–of–volume, and returns control to the Get, Put, Read, or Write routine without issuing an EXCP macro instruction. The EOV condition is eventually recognized and processed — in QSAM by the synchronizing routine and in BSAM by the Check routine.
- If the module determines that the block fits on the track, the module calculates the actual block length using the overhead value for a block that is not the last on a track. (This value is found in the resident I/O device table.) It adjusts the value in the DCBTRBAL field by this amount and updates the DCBFDAD field and the ID field of the count area of the block located immediately after the channel program.
- If the block fits on the track, the module next attempts to join the channel program for the current buffer to the preceding channel program (that is, chain schedule) by:

  Setting the ICB to not–complete.

  Inserting the address of either the Write or the Search CCW of this channel program into the NOP CCW of the preceding channel program. The address of the Write CCW is inserted if the present and the preceding channel programs address the same track. The address of the Search CCW is inserted if the present and the preceding channel programs address different tracks. In this case, the Search CCW addresses record zero of the next track.

  Changing the NOP CCW in the preceding channel program to a TIC CCW.

  Updating the SAM IOB prefix block to point to the end of the current channel program.

- It determines whether the joining was successful by testing the ECB (pointed to by the IOB) to see if the I/O supervisor posted the I/O event as completed.
- If the I/O supervisor did not post the event as completed, the joining was successful and the routine returns control to the calling routine.
- If the I/O supervisor did post the event as completed, the routine tests the ICB for the present channel program to find whether the joining was successful or not.
- If the present ICB remains unposted, the present channel program was not joined to the preceding one. The routine prepares to cause restart of the channel by copying the Seek address and the channel program Start address from the current ICB into the IOB, and uses the EXCP macro instruction to schedule the channel program. It then returns control to the calling routine.
- If the present ICB is posted as completed, the present channel program was joined successfully. (The routine was interrupted long enough, between the joining and the testing, for the channel program to be executed and for the channel–end appendage to post the ICB.) The routine returns control to the calling routine.

End–of–Block Module IGG019TW: Module IGG019TW attempts to join the present channel program to the last one in the chain of scheduled channel programs. The Open executor selects and loads this module if the user specifies the user–totaling option (that is, if bit 6 is 1 in DCBOPTCD) for his data set and if either of the following conditions exists:

The Open parameter list specifies Input and the DCB specifies chained channel–program scheduling and any device.

The Open parameter list specifies Output and the DCB specifies chained channel program scheduling and magnetic tape.

The module operates as follows:

- It receives control from a Get or a Put routine when the routine finds that a buffer is ready to be scheduled, or from a Read or Write routine at the conclusion of its processing.
- It issues a BALR instruction to the user–totaling save routine, IGG019AX, to place the user's total in the user–totaling save area, which is pointed to by the DEB.
- If the device type is magnetic tape, the routine determines the increment value and stores it in the ICB.
- The module attempts to join the channel program for the current buffer to the preceding channel program (that is, chain schedule) by:

  Setting the ICB to not–complete.

  Inserting the address of the current channel program into the NOP CCW of the preceding channel program.

  Changing the NOP CCW in the preceding channel program to a TIC CCW.

  Updating the SAM IOB prefix block to point to the end of the current channel program.

- It determines whether the joining was successful by testing the ECB (pointed to by the IOB) to see if the I/O supervisor posted the I/O event as completed.
- If the I/O supervisor did not post the event as completed, the joining was successful and the routine returns control to the calling routine.
- If the I/O supervisor did post the event as completed, the routine tests the ICB for the present channel program to find whether the joining was successful or not.

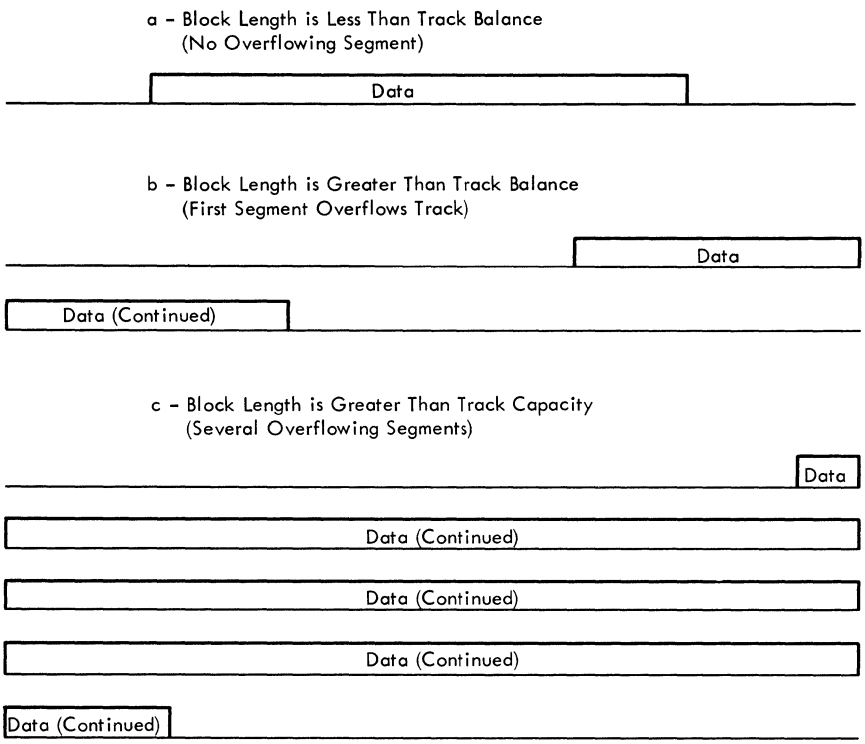


Figure 13. Track–Overflow Records

- If the present ICB remains unposted, the present channel program was not joined to the preceding one. The routine prepares to cause restart of the channel by copying the channel program Start address (and the Seek address, if direct–access storage) from the current ICB into the IOB and uses the EXCP macro instruction to cause scheduling of the channel program. It then returns control to the calling routine.
- If the present ICB is posted as completed, the present channel program was joined successfully. (The routine was interrupted long enough, between the joining and the testing, for the channel program to be executed and for the channel–end appendage to post the ICB.) The routine returns control to the calling routine.

Track–Overflow and User–Totaling Save Routines

The track–overflow, end–of–block routine processes channel programs for output data sets whose blocks may overflow from one track onto another (see Figure 13). Such a block is written by a channel program consisting of a channel program segment for each track to be occupied by a segment of the block. The track–overflow, end–of–block routine computes the address of each track written on; to progress from track to track (to continue writing successive segments of one block), the channel program uses the Search command with the multiple–track (M/T) mode.

There are two track–overflow, end–of–block routines, modules IGG019C2 and IGG019T2; they are used with output data sets if the access conditions shown in Figure 14 are specified for a DCB. The Open executor selects one of these routines, loads it, and places its address into the DCBEOB or DCBEOBW field. (For an input data set with track–overflow, end–of–block module IGG019CC is used.)

The user–totaling save module is also shown in Figure 14. This module saves an image of the user's totaling area in the sequential access method totaling save area.

Access Method Options	Selections	
Output, INOUT, OUTIN	X	X
Track Overflow	X	X
User–Totaling Facility		X
End–of–Block Modules		
IGG019AX		AX
IGG019C2	C2	
IGG019T2		T2

Figure 14. Module Selector — Track–Overflow, End–of–Block Modules

User–Totaling Save Module IGG019AX: Module IGG019AX saves an image of the user's totaling area in the sequential access method totaling save area. This save area is described in Figure 42 (see Section 4).

The Open executor selects and loads this module if the user–totaling option is specified in the DCB (that is, if bit 6 is 1 in the DCBOPTCD field).

The module operates as follows:

- It receives control from one of the end–of–block routines — IGG019TC, IGG019TD, IGG019TV, IGG019TW, or IGG019T2.
- It retrieves the address of the sequential access method totaling save area from the access method portion of the DEB.
- The sequential access method totaling save area contains a pointer to the user's totaling area. An image of the user's total is saved in the next available segment of the sequential access method totaling save area. Then the save area control block is updated so that the pointer identifies the current entry.
- It returns control to the end–of–block routine that called it.

End–of–Block Module IGG019C2: Module IGG019C2 performs device–oriented processing when track overflow is permitted with an output data set. The Open executor selects and loads this module if the Open parameter list specifies:

Output, INOUT, or OUTIN

and the DCB specifies:

Track overflow

- If the entire block fits on this track, the module completes a channel program (consisting of one channel program segment) for writing the block, updates the track balance, and passes control to the I/O supervisor.
- If at least a 1–byte data field fits on this track, the module completes a channel program segment for the segment of the block that fits on the track (by entering the Seek address, main storage address, and count field for the channel program segment) and tests if there is another track in the same extent.
- If the next track is in this extent, it compares the remaining block length with the track capacity.
- If the remainder of the block exceeds the track capacity, the module proceeds as it does when at least one byte fits on the track.
- If the remainder of the block is less than the track capacity, the module completes the final channel program segment for the final segment of the block, updates the track balance, and passes control to the I/O supervisor.
- If the next track is not in this extent, the module passes control to the track balance routine using an SVC 25 instruction. That routine erases all tracks in the current extent that were found insufficient for the block to be written. On return of control from the track balance routine, the module tests for another extent.
- If there is another allocated extent on this volume, the module reconstructs the channel program by proceeding as it does when at least one byte fits on a track.
- If there is no other allocated extent on this volume, an end–of–volume condition exists. The module sets the DCBCIND1 field in the DCB and the CSW field in the IOB to show end–of–volume, and returns control to the Put or Write routine without issuing an EXCP macro instruction. The EOV condition is eventually recognized and processed in QSAM by the synchronizing routine, and in BSAM by the Check routine.

End–of–Block Module IGG019T2: Module IGG019T2 performs device–oriented processing when track overflow is permitted with an output data set. The Open executor selects and loads this module if the user specifies the user–totaling option (that is, if bit 6 in DCBOPTCD is 1) for his data set and if the Open parameter list specifies:

Output, INOUT, or OUTIN

and the DCB specifies:

Track overflow

The module operates as follows:

- It receives control from a Put routine when the routine finds that a buffer is to be scheduled, or from a Write routine at the conclusion of its processing.

- It issues a BALR instruction to the user–totaling save routine, IGG019AX, to place the user's total in the user–totaling save area, which is pointed to by the DEB.
- It compares the block length with the space remaining on the track last written on.
- If the entire block fits on this track, the module completes a channel program (consisting of one channel program segment) for writing the block, updates the track balance, and passes control to the I/O supervisor.
- If at least a 1–byte data field fits on this track, the module completes a channel program segment for the segment of the block that fits on the track (by entering the Seek address, main storage address, and count field for the channel program segment) and tests for another track in the same extent.
- If the next track is in this extent, the module compares the remaining block length with the track capacity.
- If the remainder of the block exceeds the track capacity, the module proceeds as it does when at least one byte fits on the track.
- If the remainder of the block is less than the track capacity, the module completes the final channel program segment for the final segment of the block, updates the track balance, and passes control to the I/O supervisor.
- If the next track is not in this extent, the module passes control to the track balance routine using an SVC 25 instruction. That routine erases all tracks in the current extent that were found insufficient for the block to be written. On return of control from the track balance routine, the module tests for another extent.
- If there is another allocated extent on this volume, the module reconstructs the channel program by proceeding as it does when at least one byte fits on a track.
- If there is no other allocated extent on this volume, an end–of–volume condition exists. The module sets the DCBCIND1 field in the DCB and the CSW field in the IOB to show end–of–volume, and returns control to the Put or the Write routine without issuing an EXCP macro instruction. The EOV condition is eventually recognized and processed — in QSAM by the synchronizing routine, and in BSAM by the Check routine.

Synchronizing–and–Error–Processing Routines

A synchronizing–and–error–processing routine synchronizes execution of the processing program with execution of the channel programs, and performs error–processing to permit continued access to the data set after an error was encountered during the execution of a channel program.

There are six synchronizing–and–error–processing routines. Four of the six routines:

Are unique to QSAM

Both synchronize and process errors

Receive control from a Get or a Put routine

Are pointed to by an address in the DCB

The track–overflow and 3211 Printer Retry asynchronous–error–processing routines:

Are shared between QSAM and BSAM

Only process errors

Receive control by being scheduled by an abnormal–end appendage

To synchronize, the QSAM input and output synchronizing–and–error–processing routines, modules IGG019AQ and IGG019AR, return control to the Get or Put routine immediately if the channel program completed processing without error, or use the WAIT macro instruction if the channel program has not yet completed processing. To process errors, these routines pass control to the SYNAD/EOV executor (using an SVC 55 instruction) to distinguish between the processing necessary for unit check (that is, a permanent error) and unit exception (that is, an end–of–volume condition).

For a unit check, the executor returns control to the synchronizing routine, which in turn passes control to the SYNAD routine. On return of control from the SYNAD routine, the synchronizing routine again passes control to the executor to implement the error options. For the Accept and Skip options, control returns once more to the synchronizing routine. It now operates as when it is first entered.

For a unit exception, the executor causes end–of–volume processing by the end–of–volume routine of I/O support. That routine passes control to the EOV/new volume executor. The executor returns control to the synchronizing routine. It now operates as it did when first entered.

To synchronize, the paper–tape–character–conversion–synchronizing routine (contained in the paper tape Get module IGG019AT) returns control to the Get routine immediately if the channel program executed without error, or uses the WAIT macro instruction if the channel program has not yet executed. To process errors, the routine passes control to the SYNAD routine. When control returns from the SYNAD routine to the synchronizing routine, the latter implements the error option. (The equivalent of an end–of–volume condition is handled by the paper tape Get routine.)

To synchronize, the update–synchronizing–and–error–processing routine, module IGG019AF, returns control to the Get routine immediately if the channel program executed without error, or uses the WAIT macro instruction if the channel program has not yet executed. To process an end–of–volume condition, the routine suspends volume–switching until processing on the old volume is finished. To process permanent errors, the routine interprets the error option to ensure that neither a buffer nor a block is skipped.

If the American Standard Code for Information Interchange (ASCII) is used, synchronizing Module IGG019AQ will issue an XLATE macro instruction which converts the entire buffer from ASCII to EBCDIC when the input buffer is full. If format–D records are specified, the record descriptor words are converted from decimal to binary code.

The error processing performed by the track–overflow, asynchronous–error–processing routine, module IGG019C1, distinguishes two kinds of errors—those in the block being read and those in the block being skipped over to read the next one. For errors in the block being read, the routine sets the channel program to permit the processing program to continue reading the segments and blocks beyond the one in error. For errors in the block being skipped, the routine resets the channel program and uses the EXCP macro instruction, so that the processing program is unaware of the error.

For an error whose character and occurrence the processing program must know about (errors in segments of the block being read into the buffer), the track–overflow routine addresses the IOB to the next track and its channel program and causes control to return to the processing program using the TCB queue. For errors whose correction does not affect the processing program (errors in segments of the block being skipped over), the module uses the EXCP macro instruction to skip around the defective segment to present the processing program with the block it expects to obtain. This latter condition only holds if an error occurs on a Read–data CCW with the Skip bit on for a segment that is not the last or only segment on an alternate track. In that case, control returns to the processing program when the desired block is in the buffer in its entirety. For errors that do not permit reading the entire block in one pass without error, control returns to the processing program with the IOB set to a track and channel program that permits reading the segments following the defective one. The defective segment and the preceding good segments of the block are in the buffer at the time control is returned to the processing program.

Five of the six routines described here (those enumerated in Figure 15) are unique to QSAM. One of these routines gains control when a Get or a Put routine finds that it needs a new buffer. Figure 15 lists the routines available and the conditions that cause a particular routine to be used. The Open executor selects one of the routines, loads it, and puts its address into the DCBGERR/PERR field.

The fifth routine (identified in Figure 16) is shared between QSAM and BSAM. It gains control by being scheduled for eventual execution by track–overflow abnormal–end appendage IGG019C3. The Open executor loads it and enters its address in an interruption request block (IRB); the address of the IRB is in the DEB. (If QSAM is used, module IGG019AQ is also used.)

Access Method Options	**Selections**					
Get	X	X		X	X	
Put			X			
Input, Readback		X				
Output			X			
Update	X				X	
Paper–tape character conversion				X		
Variable–length record format					X	
Spanned records					X	
Locate operating mode					X	
Modules						
IGG019AF	AF					
IGG019AQ		AQ				
IGG019AR			AR			
IGG019AT[1]				AT		
IGG019BQ					BQ	

[1]This module includes both the paper–tape–synchronizing–and– error–processing routine and the paper tape Get routine. Both routines are described in "Simple Buffering Get Routines" (see Figure 1).

Figure 15. Module Selector — Synchronizing and Error–Processing Modules

Synchronizing Module IGG019AF (Update): Module IGG019AF finds the next buffer and ensures that it has been refilled. If a unit status prevented refilling the buffer, the module processes the pending channel programs according to whether they are empty–and–refill or refill–only channel programs. The Open executor selects and loads this module if the Open parameter list specifies:

Update

and the DCB specifies:

Get

The module operates as follows if no error occurred:

- It receives control when the update Get routine finds that a new buffer is needed. It also receives control after the FEOV (force–end–of–volume) macro instruction is encountered in a processing program, once from the update Get routine (when the FEOV executor schedules the last buffer) and once directly from the FEOV executor (when it awaits execution of the scheduled buffers.)
- If the next buffer has been refilled, the module returns control to the update Get routine.
- If the channel program for the next buffer has not yet completed processing, the module issues a WAIT macro instruction.

The module operates as follows if an end–of–volume condition is encountered:

- It receives control when the update Get routine finds that a new buffer is needed or when the FEOV executor awaits execution of the scheduled buffers.
- If the channel program for the next buffer encountered an end–of–volume condition, or if this module has control due to an FEOV macro instruction, the module finds the IOBs flagged for output. It then turns off the command–chain flag at the end of the empty portion of the channel program, and schedules the empty channel programs for execution by means of an EXCP macro instruction.
- If all empty channel programs have been executed, or if none are pending, the module passes control to the SYNAD/EOV executor by way of an SVC 55 instruction. If this module has control due to an FEOV macro instruction, control returns to the routine that passed control.
- If a permanent error is encountered during execution of empty channel programs for an end–of–volume condition or for an FEOV macro instruction, control passes to the SYNAD routine, if one is present. The SYNAD routine returns control to this module.
- The module then processes the error option as follows:

  Accept or Skip option: The pending empty channel programs are rescheduled for execution using an EXCP macro instructions.

  Terminate option: Control passes to the EOV routine to request an ABEND macro instruction.

The module operates as follows if a permanent error was encountered:

- It receives control when the Update Get routine finds a new buffer is needed.
- If the channel program for the next buffer encountered a permanent error and a SYNAD routine is present, the module passes control to the SYNAD routine.
- If control returns from the SYNAD routine, or if there is no SYNAD routine, the module processes the error option in the following manner:

  Accept Option: If the error occurred in the empty portion of a channel program, the module resets the IOB to point to the refill portion of the channel program and issues an EXCP macro instruction for it and all following IOBs.

  If the error occurred in the refill portion of a channel program, the module posts the current IOB as complete without error and issues an EXCP macro instruction for all the IOBs except the present one.

  The module ensures refilling of the buffer associated with the first IOB and then returns control to the update Get routine.

  Skip Option: If the error occurred in the empty portion of a channel program, the module operates as it does for the Accept option.

  If the error occurred in the refill portion of a channel program, the module issues an EXCP macro instruction for all IOBs.

  The module ensures refilling of the buffer associated with the first IOB and then returns control to the update Get routine.

  Terminate Option: If the error occurred in the empty portion of a channel program, the module passes control to the ABEND routine.

  If the error occurred in the refill portion of a channel program, the module finds the end of the empty portion of any pending empty–and–refill channel programs, turns off the commmand–chain flag, and issues an eXCP macro instruction for these empty channel programs. On execution of all the channel programs, the module passes control to the EOV routine to request an ABEND.

Synchronizing Module IGG019AQ (Input): Module IGG019AQ finds the next input buffer, determines its status, and passes a full buffer to the Get routine. If ASCII is used, the entire input buffer is translated from ASCII to EBCDIC. The Open executor selects and loads this module if the Open parameter list specifies:

Input, Readback

and the DCB specifies:

Get

The module operates as follows:

- It receives control when a Get routine determines that a new buffer is needed.
- It finds the next IOB and tests the status of the channel program associated with that IOB.

- If the channel program has not yet completed processing, the module issues a WAIT macro instruction.
- If the channel program has been executed normally, the module updates the DCBIOBA field to point to this IOB and returns control to the Get routine.
- If the channel program has been completed normally, and if the buffer contains a DOS checkpoint record, tape files only, the module returns control to the Get routine.
- If an error occurred during the execution of the channel program, the module issues an SVC 55 instruction to pass control to the SYNAD/EOV executor, module IGC0005E. For an EOV condition, control eventually passes to the end–of–volume routine of I/O support and returns after the next volume has been found and the purged channel programs have been rescheduled. For a description of the flow of control from the SYNAD/EOV executor for a permanent error condition, refer to SYNAD/EOV Executors in Figure 31.

Synchronizing Module IGG019AR (Output): Module IGG019AR finds the next output buffer, determines its status, and passes an empty buffer to the Put routine. The Open executor selects and loads this module if the Open parameter list specifies:

Output

and the DCB specifies:

Put

The module operates as follows:

- It receives control when a Put routine determines that a new buffer is needed.
- It finds the next IOB and tests the status of the channel program associated with that IOB.
- If the channel program is not yet executed, the module issues a WAIT macro instruction.
- If the channel program has been executed normally, the module updates the DCBIOBA field to point to this IOB and returns control to the Put routine.
- If the output device is a 3211 Printer and three or more buffers are being used, the synchronizing module waits for two channel programs to be completed before updating the DCBIOBA field.
- If an error occurred during the execution of the channel program, the module issues an SVC 55 instruction to pass control to the SYNAD/EOV executor, module IGC0005E. For an EOV condition, control eventually passes to the end–of–volume routine of I/O support and returns after a new volume or more space has been found and the purged channel programs have been rescheduled. For a description of the flow of control from the SYNAD/EOV executor for a permanent error condition, refer to SYNAD/EOV Executors in Figure 31.

Synchronizing Module IGG019BQ (Update): Module IGG019BQ finds the next buffer and ensures that it has been refilled. If a unit status prevented refilling of the buffer, the module processes the pending channel programs according to whether they are

empty–and–refill or refill only channel programs. The Open executor selects and loads this module if the Open parameter list specifies:

Update

Locate operating mode

and the DCB specifies:

Get

Variable–length spanned (blocked or unblocked) record format

The module operates as follows if no error occurred:

- It receives control when the update Get routine finds that a new buffer is needed. It also receives control after an FEOV macro instruction is encountered in a processing program, once from the update Get routine (when the FEOV executor schedules the last buffer) and once directly from the FEOV executor (when it awaits execution of the scheduled buffers).
- If the next buffer has been refilled, the module returns control to the update Get routine.
- If the channel program for the next buffer has not yet executed, the module awaits its execution.

The module operates as follows if an EOV condition is encountered:

- It receives control when the update Get routine finds that a new buffer is needed or when the FEOV executor awaits execution of the scheduled buffers.
- If the channel program for the next buffer encountered an EOV condition, the module tests whether assembling or updating of a spanned record is in process.
- If updating is in process, the module delays the normal EOV processing by turning off the error flags in the DCB and then returns control to the update Get routine.
- If assembling is in process, the module sets the spanned record flag in the IOB and continues to the next step.
- If assembling is in process or if this module has control due to an FEOV macro instruction, the module finds the IOBs flagged for output. It then resets the command–chain flag at the end of the empty portion of the channel program and schedules the empty channel programs for execution by means of an EXCP macro instruction.
- If all empty channel programs have been executed, or if none are pending, the module passes control to the SYNAD/EOV executor using an SVC 55 instruction. If this module has control due to an FEOV macro instruction, control returns to the routine that passed control.
- If a permanent error is encountered during execution of empty channel programs for an EOV condition or for an FEOV macro instruction, control passes to a SYNAD routine if one is present. The SYNAD routine returns control to this module.

- The module then processes the error option as follows:

  Accept or Skip: The pending empty channel programs are rescheduled for execution using EXCP macro instructions.

  Terminate: Control passes to the ABEND routine.

- On return of control from the SYNAD/EOV executor, the module tests whether assembling of a spanned record is in process. If it is being processed, the module turns off the spanned spanned–record flag in the IOB and returns control to the update Get routine.

The module operates as follows if a permanent error is encountered:

- It receives control when the update Get routine finds that a new buffer is needed.
- If the channel program for the next buffer encountered a permanent error and a SYNAD routine is present, the module passes control to the SYNAD routine.
- If control returns from the SYNAD routine, or if there is no SYNAD routine, the module processes the error option in the following manner:

  Accept: If the error occurred in the empty portion of a channel program, the module resets the IOB to point to the refill portion of the channel program and issues an EXCP macro instruction for it and all following IOBs.

  If the error occurred in the refill portion of a channel program, the module posts the current IOB as complete without error and issues an EXCP macro instruction for all the IOBs except the present one.

  The module ensures refilling of the buffer associated with the first IOB and then returns control to the update Get routine.

  Skip: If the error occurred in the empty portion of a channel program, the module operates as it does for the Accept option.

  If the error occurred in the refill portion of a channel program, the module treats this as a RELSE condition and issues an EXCP macro instruction for all IOBs.

  The module ensures refilling of the buffer associated with the first IOB and then returns control to the update Get routine.

  Terminate: If the error option occurred in the empty portion of a channel program, the module passes control to the ABEND routine.

  If the error occurred in the refill portion of a channel program, the module finds the end of the empty portion of any pending empty–and–refill channel programs, resets the command–chain flag, and issues an EXCP macro instruction for these empty channel programs. On the execution of all the channel programs, the module passes control to the ABEND routine.

Track–Overflow, Asynchronous–Error–Processing Module IGG019C1: IGG019C1, used in both QSAM and BSAM, processes error conditions that are encountered in the execution of a channel program for an input data set with track overflow. It processes error conditions asynchronously with the execution of the channel program, the I/O supervisor, or the processing program. It receives control by being scheduled for execution by the track–overflow abnormal–end appendage IGG019C3. It passes control to the processing program through the supervisor. The module determines the

Access Method Options	Selections			
Get	X			
Read		X		
Input, INOUT, OUTIN	X	X		
Track Overflow	X	X		
3211 Printer			X	
Modules				
IGG019C1	C1	C1		
IGG019FS			FS	

Figure 16. Module Selector — Track Overflow/3211 Printer

Seek address for reading the segments and blocks beyond the segment in error and inserts it in the IOBSEEK field. If the error occurred in a segment of the block being read into the buffer, the segment following the segment in error is read, if the processing program chooses the Accept option in the SYNAD routine. If the error occurred in a segment in the block preceding the block to be read into the buffer (that is, the error occurred in the block being skipped over to find the block to be read into the buffer), the desired block is in the buffer when the processing program obtains the buffer.

The Open executor selects and loads this module and places its address into an IRB pointed to in the DEB if the Open parameter list specifies:

Input, INOUT, or OUTIN

and the DCB specifies:

Track overflow

Get or Read

The module operates as follows if the error occurred in a CCW other than a Read–data CCW:

- It receives control from the supervisor.
- It increases the track address in the IOB by 1, posts the ECB with the error code, and causes control to return to the processing program.

The module operates as follows if the error occurred in a Read–data CCW without a Skip bit on:

- It receives control from the supervisor.
- If the segment in error is the last or only segment of the block, the module posts the ECB with the error code and causes control to return to the processing program.
- If the segment in error is not the last segment and it is not on an alternate track, the module sets the IOB to address the track following the track in error, posts the ECB with the error code, and causes control to return to the processing program.

- If the segment in error is not the last segment and it is on an alternate track, the module increases the track address in the IOB by 1, posts the ECB with the error code, and causes control to return to the processing program.

The module operates as follows if the error occurred in a Read–data CCW with the Skip bit on:

- It receives control from the supervisor.
- If the segment in error is the final or only segment of a block and it is not on an alternate track, the module sets the IOB to address the track in error, changes the Read–data command to a NOP command and issues an EXCP macro instruction for the changed channel program.
- If the segment in error is the final or only segment of a block and it is on an alternate track, the module sets the IOB to address the track following the one originally addressed, posts the ECB with the error code, and causes control to return to the processing program. (In the case of an error in a final or only segment on an alternate track, the remaining segment or blocks on that track are not read.)
- If the segment in error is not the last one and it is not on an alternate track, the module sets the IOB to address the track following the one in error and issues an EXCP macro instruction for the readdressed channel program.
- If the segment in error is not the last one and it is on an alternate track, the module successively increases the track address in the IOB by 1 and issues an EXCP macro instruction for the readdressed channel program.
- When control returns from the I/O supervisor, this module awaits execution of the channel program by using a WAIT macro instruction. On channel program execution, the module restores the purged IOBs (and the Read–Skip command, if it was changed to a NOP command) and causes control to return to the processing program.

3211 Printer Asynchronous–Error–Processing Module IGG019FS (Print Line Buffer Error — Retry): Module IGG019FS is device–dependent and scheduled asynchronously by the 3211 abnormal–end appendage (IGG019FR). The module retries operations that result in print line buffer parity errors or UCS buffer parity errors whenever possible. When an operation cannot be retried, the printer is reset and control is returned to the calling program.

The module operates as follows:

- It initializes registers to point to the DCB, ECB, and IOB. It then examines the sense bytes in the IOB to check whether one of the error conditions for which a retry is possible has occurred.
- If a UCS buffer parity error is indicated (ECB posted in error with an X'41' or X'44' and the command retry bit on in sense byte 1), the UCS image ID is obtained from the UCB, located in SYS1.IMAGELIB, and loaded into main storage. (Failure to locate the UCS image in SYS1.IMAGELIB causes a skip to channel 0 command to be issued. This resets the printer and the module returns to the calling program.) An IOB and channel program to load the UCS image into the UCS buffer on the 3211 are constructed and executed. If a permanent I/O error occurs during an attempt to load the UCS buffer, a skip to channel 0

command is issued to reset the printer. The UCS field in the UCB is also set to 0 and control is returned to the calling program. If the UCS buffer is loaded successfully, a check is made to determine the access method (BSAM or QSAM) is being used.

- When QSAM is being used, a check is made to see whether three or more buffers were specified in the BUFNO field of the DCB macro instruction. (This is a condition necessary to retry a print line.) After either UCS buffer parity errors or print line buffer parity errors, the type of scheduling is determined. For normal scheduling, the IOB associated with the failing print line is located and the channel program for that IOB is reissued once. If the channel program is now successful, the next IOB is rescheduled if necessary, and control is returned to the problem program as though no error had occurred. If the channel program is unsuccessful, a skip to channel 0 command is issued to reset the control unit and the module returns to the calling program. For chained channel scheduling, the portion of the channel program associated with the failing print line is reissued. If it is successful, a check is made to see whether another chain needs to be restarted before the return to the problem program. If the retry was unsuccessful, a skip to channel 0 command is issued and the module returns to the calling program.
- For BSAM, or QSAM with fewer than three buffers specified, a skip to channel 0 command is issued and the module returns to the calling program.

Appendages

Appendages are access method routines that receive control from and return control to the I/O supervisor. They operate in the supervisor state. The same appendages are used in QSAM as in BSAM.

An appendage receives control from the I/O supervisor and tests and may alter the channel status word (CSW). The I/O supervisor uses the CSW to post the event control block (ECB). If the SIO appendage receives control from the I/O supervisor before the latter starts execution of the channel program, it may alter channel commands just before channel program execution. The relationship of the I/O supervisor and the appendages are shown in Diagram F in Section 5.

The I/O supervisor permits an appendage to gain control at certain exit points. At that time the I/O supervisor refers to the entry associated with that exit in the appendage vector table, whose address is in the data extent block (DEB). If an entry contains the address of an appendage, control passes to it; otherwise, control remains with the I/O supervisor. The five I/O supervisor exits where appendages receive control are:

- End–of–extent
- SIO
- Channel end
- PCI
- Abnormal end

Appendages differ from other sequential access method routines that are loaded by the Open executor into processing program main storage. They differ because they operate with the CPU disabled for interruptions in the supervisor state and asynchronously with the processing program. The events that cause appendages to gain control depend on the progress of the channel program, not on the progress of the processing program.

The Open executor selects and loads all the appendages to be used with a DCB. No appendage, one appendage, or several appendages may be used with one DCB. The Open executor places the addresses of the required appendages into the various fields of the appendage vector table. Figure 17 lists the appendages and the conditions that cause the different appendages to be used. The appendages are grouped according to the condition detected by the I/O supervisor before control is passed to the appendage. Note that some appendages have entry points for more than one of the conditions checked by the I/O supervisor.

End–of–Extent Appendages

End–of–extent appendages gain control of the central processing unit (CPU) if the EXCP supervisor finds an end–of–extent condition. This condition exists if the direct–access device storage address associated with a channel program is outside of the extent currently pointed to in the data extent block (DEB).

Five end–of–extent appendages are provided for use with sequential access method routines:

- IGG019AW processes an end–of–extent condition for QSAM update mode channel programs.
- IGG019BM processes an end–of–extent condition for BSAM update mode channel programs.
- IGG019CH processes an end–of–extent condition when neither the update mode nor chained channel–program scheduling is specified.
- IGG019CZ processes end–of–extent conditions when chained channel–program scheduling is used.
- IGG019C4 is loaded in for standard format–F records and verifies whether an extent violation is valid. It is also the end–of–extent appendage for the search–direct option.

Appendage IGG019AW (End–of–Extent — Update — QSAM): Appendage IGG019AW readdresses the refill portions of all QSAM update channel programs to a new extent. The Open executor selects and loads this module for use as the end–of–extent appendage if the Open parameter list specifies:

Update

and the DCB specifies:

Get

The appendage operates as follows:

- When using the rotational position sensing (RPS) feature and when the Seek address is updated to reflect the beginning of the next extent, the Set–sector byte is reset to 0. A test is made to determine whether the record–ready feature is present and the correct offset is used.
- It receives control from the EXCP supervisor under one of the following conditions:

  A refill portion of QSAM update channel program attempts to read the first block beyond the present extent.

Access Method Options	Selections																							
Input, INOUT, OUTIN					X	X	X																	
Readback						X																		
Update	X	X		X																				
SYSIN								X																
Get	X																							
Read		X																						
Offset Read (BDAM)				X																				
Record format is fixed–length										X														
Record format is fixed–length blocked						X																		
Record format is variable–length							X																	
Record format is variable–length spanned			X	X																				
Record format is not fixed–length standard					X																			
Create–BDAM spanned record			X																					
Direct–access storage			X	X	X								X											
Printer									X															
Paper tape										X														
Chained scheduling											X		X											
Track–overflow												X												
3211 printer											X			X										
Magnetic Tape (OPTCD = H)															X									
3525 Associated Data Sets																X								
Search Direct (OPTCD = Z)					X				X								X							
Appendages entered from End–of–Extent Exit																								
IGG019AW	AW																							
IGG019BM		BM																						
IGG019CH					CH																			
IGG019CZ													CZ											
IGG019C4					C4																			
Appendages entered from SIO Exit																								
IGG019CG				CG																				
IGG019CL									CL															
IGG019FN									FN															
Appendages entered from Channel–End Exit																								
IGG019BT			BT																					
IGG019BV				BV																				
IGG019CI						CI																		
IGG019CJ							CJ																	
IGG019CK								CK																
IGG019CS										CS														
IGG019CU[1]											CU													
IGG019C6																C6								
IGG019EI															EI									
IGG019EJ															EJ									
IGG019FP																	FP							
Appendages entered from PCI Exit																								
IGG019CU											CU													
IGG019C3[2]				C3																				
Appendages entered from Abnormal End Exit																								
IGG019CU[1]											CU													
IGG019C3														C3										
IGG019EI[3]															EI									
IGG019EJ[3]															EJ									
IGG019FR														FR										

[1] Module has multiple entry points Description appears in Program Controlled Interruption Appendages

[2] Module has multiple entry points Description appears in Abnormal–end Appendages

[3] Module has multiple entry points Description appears in Channel–end Appendages

Figure 17. Module Selector — Appendages

The remaining channel programs attempt to refill their buffers from the new extent.

- If there is no other extent, the appendage sets error indications in the IOB and the DCB (to show an end–of–volume condition) and returns control to the EXCP supervisor. The EXCP supervisor then issues a PURGE macro instruction for that channel program. The update synchronizing routine ensures writing out of the empty portions of pending channel programs.
- If the interruption occurred in a Read–count CCW and there is a new extent, the appendage builds a Seek address for the new extent using the starting address from the DEB. It then copies this new Seek address into the IOB and UCB (unit control block) and updates the M value in the refill portion of each channel program.
- If the interruption occurred in a Seek CCW, the appendage copies the Seek address from the refill portion of the present channel program into the IOB and UCB.
- It resets the IOB and UCB to address the next track and its channel program and returns control to the I/O supervisor.

Appendage IGG019BM (End–of–Extent — Update — BSAM): Appendage IGG019BM readdresses channel programs to a new extent for a DCB opened for Update and using BSAM. The Open executor selects and loads this appendage for use as the end–of–extent appendage if the Open parameter list specifies:

Update

and the DCB specifies:

Read

The appendage operates as follows:

- When the Seek address is updated to reflect the beginning of the next extent and the rotational position sensing (RPS) feature has been specified, the Set–sector byte is reset to zero. A test is made to determine whether the record–ready feature is present and the correct offset is used.
- It receives control from the EXCP supervisor when a channel program to refill a buffer attempts to read the first block beyond the present extent.
- If there is no other extent for a Refill channel program, the appendage sets error indications in the IOB and the DCB to show an end–of–volume condition and returns control to the EXCP supervisor.
- If there is a new extent for a Refill channel program, the appendage adds 1 to the value of M in the DCBFDAD field and in the Seek address of each refill channel program for the DCB. It places the new Seek address into the current IOB and into the UCB and returns control to the EXCP supervisor. The supervisor restarts the channel program.

Appendage IGG019CH (End–of–Extent — Ordinary): Appendage IGG019CH finds a new extent when the EXCP supervisor finds an end–of–extent condition. The Open executor selects and loads this appendage for use as the end–of–extent appendage if the Open parameter list specifies:

Input, INOUT, or OUTIN

and the DCB specifies:

Direct–access storage device

Record format other than fixed–length standard

Normal channel–program scheduling

The appendage operates as follows:

- It receives control when a channel program attempts to read a block beyond the present extent.
- The appendage examines the DEB for another extent.
- If there is another extent, the appendage enters the new full device address in the DCB, the unit control block (UCB), the IOBs, and returns control to the EXCP supervisor. The EXCP supervisor restarts the channel program.
- If there is no other extent, the appendage sets error indications in the IOB and the DCB to show an end–of–volume condition and returns control to the EXCP supervisor. The EXCP supervisor then issues a PURGE macro instruction for that channel program.

Appendage IGG019CZ (End–of–Extent — Chained Channel–Program Scheduling): Appendage IGG019CZ readdresses the chain of channel programs to a new extent when the EXCP supervisor finds an end–of–extent condition. The Open executor selects and loads this appendage for use as the end–of–extent appendage if the DCB specifies:

Chained channel–program scheduling

Direct–access storage device

The appendage operates as follows:

- It receives control when an end–of–track condition interrupts the chained scheduling and the I/O supervisor finds that the next track is not in the current extent.
- If there is another extent, the appendage enters the new Seek address in the DCB, IOB, unit control block (UCB), updates the Seek addresses of the remaining ICBs, and returns control to the I/O supervisor to reschedule the channel program for execution.
- If there is no other extent, the appendage sets a volume–full indication in the DCB, IOB, and ICB and returns control to the I/O supervisor to skip further scheduling for this DCB.

Appendage IGG019C4 (End–of–Extent): Appendage IGG019C4 finds a new extent when the EXCP supervisor finds an end–of–extent condition. The Open executor selects and loads this appendage for use as the end–of–extent appendage if the Open parameter list specifies:

Input, INOUT, or OUTIN

and the DCB specifies:

Direct–access storage device

Record format other than fixed–length standard

Normal channel–program scheduling

The appendage operates as follows:

- It receives control when a channel program attempts to read a block beyond the present extent.
- If another extent is not available, it tests the IOBSEEK field to see whether the Seek is on cylinder FFxx. If it is, the appendage tests to see whether the M/T Read–count following the Read–data is causing a Seek beyond the current extent. If so and the unit check is not on, it clears all error and status flags in the IOB, sets CHAN and DEV END, sets 7F in the IOB completion code, clears the DCB flags, and sets an X'FF' in the HI cylinder byte of the next IOBSEEK field (for multiple IOBs) or the DCBFDAD field (for one IOB).
- For an error on any other command causing a Seek beyond the current extent if an X'FF' is found, it sets the volume–full bit in the DCBCIND1, sets the unit–exception bit in the CSW, and turns off the unit check.

Start I/O (SIO) Appendages

Start I/O (SIO) appendages, if present, gain CPU control when the start I/O subroutine of the EXCP supervisor reaches the start I/O appendage exit. The following appendages set channel program entries:

- IGG019CG. This appendage makes the Seek address accessible to the I/O supervisor for QSAM and BSAM update channel programs that refill buffers. (This is necessary because the Seek address for such a channel program is read by the preceding channel program into a location unknown to the I/O supervisor.)
- IGG019CL. This appendage causes the next line to print at the top of a new page if a printer overflow condition was encountered in the execution of the last channel program.
- IGG019FN. This appendage checks the search argument for a record value of zero and sets the sector value to zero when one is found.

Appendage IGG019CG (SIO — Update): Appendage IGG019CG resets the IOB to the Seek address and channel program for refilling for a refill–only update channel program. The Open executor selects and loads this appendage for use as the SIO appendage if the Open parameter list specifies:

Update

The appendage operates as follows:

- It receives control whenever the EXCP supervisor reaches the SIO appendage exit.
- It tests the IOB to determine whether the buffer is to be emptied and refilled or to be refilled only.
- If the buffer is to be emptied and refilled, the module turns on the PCI flag in the Read–data CCW and returns control to the EXCP supervisor.
- If the buffer is to be refilled only, the module resets the IOB to the refill portion of the channel program and its Seek address, resets the PCI flag in the Read–data CCW and returns control to the EXCP supervisor.
- With the rotational position sensing (RPS) feature, the offset to the special FDAD from the Read CCW is not the same for record–ready. A test is made for record–ready and the correct offset used. When the special FDAD has been partially or completely destroyed, the channel program Start address is set to point to a TIC so the IOB will ultimately be marked in error. The offset of the TIC is different for record–ready.

Appendage IGG019CL (SIO — PRTOV): Appendage IGG019CL causes a skip to the top of a new page with the first channel program following a printer overflow condition. The Open executor selects and loads this appendage for use as the SIO appendage if the DCB specifies:

Printer

The appendage operates as follows:

- The appendage tests the IOB to determine whether a PRTOV macro instruction was issued with this PUT or WRITE macro instruction.
- If a PRTOV macro instruction was not issued, the appendage returns control to the EXCP supervisor immediately.
- If the PRTOV macro instruction was issued, the appendage resets the PRTOV bit in the IOB and tests the DCBIFLGS field to determine whether a printer–overflow condition has occurred.
- If printer–overflow has not occurred, the appendage returns control to the EXCP supervisor.
- If printer–overflow has occurred, the appendage resets the DCBIFLGS field, inserts the "skip–to–1" command byte into the channel program, updates the IOB channel program start–address field and the channel address word (location 72), and returns control to the EXCP supervisor.

Appendage IGG019FN (SIO — Rotational Position Sensing): Appendage IGG019FN checks the search argument for a record value of zero if the rotational position sensing (RPS) feature is specified.

The appendage operates as follows:

- For seeks on nonzero records, control is immediately returned to the supervisor.
- For RPS, the sector value pointed to by the Set–sector is initialized to zero.

- For seeks on record zero, the second TIC in the channel program is changed to an M/T Read–count. If search–direct has been implemented, the count is read into an 8–byte location that is 8 bytes beyond the last byte of the last CCW in the Read portion of the channel program.

Channel–End Appendages

Channel–end appendages, if present, gain CPU control when the I/O interruption supervisor reaches the channel–end appendage exit. For a SYSIN data set, the SYSIN appendage recognizes the delimiter characters. For other data sets, other appendages distinguish between valid and invalid block lengths by computation.

When the rotational position sensing (RPS) function is implemented in the channel programs, the Read–sector follows the Read–data command and wipes out the Residual count for the Read. The implementation of RPS Requires reading in the full 8 bytes of the count field of the record to use the DLDL plus the K of the count for checking the number of bytes read. The count is read into an 8–byte area following the channel program. In the channel–end appendages, it is necessary to move the CCHHR into the next search argument or DCBFDAD+3 depending on the number of IOBs.

Two sector bytes are needed for each opened DCB. The first byte is used to Set–sector and the other to Read–sector. The 2 bytes are necessary because the error recovery procedures require that no channel program modify itself.

In the channel–end appendages, the sector value from the last Read–sector is moved to the byte for the Set–sector command.

The channel–end appendages are:

- IGG019BT. This appendage schedules the writing of successive blocks when a record has to be segmented.
- IGG019BV. This appendage distinguishes between valid wrong–length blocks and variable–length blocks.
- IGG019CI. This appendage distinguishes between wrong–length and truncated blocks when fixed–length blocked records are being read using normal channel program scheduling.
- IGG019CJ. This appendage distinguishes between wrong–length and variable–length blocks when variable–length records are being read using normal channel program scheduling.
- IGG019CK. This appendage recognizes SYSIN delimiter characters.
- IGG019CS. This appendage distinguishes between valid and invalid wrong–length indications when paper tape is being read.
- IGG019CU. This appendage, which also appears at the PCI and abnormal–end exits, disconnects executed channel programs that were scheduled by chaining, and posts the completions. For channel–end channel status, this appendage distinguishes between wrong–length and truncated blocks when fixed–length blocked records are being read using chained channel–program scheduling.

Refer to the section on the PCI appendage for a discussion of separation of chained channel–programs and a description of appendage IGG019CU.

- IGG019FP. This appendage does length checking for all formats supported by the search–direct feature.
- IGG019C6. This module receives control when any combination of Read, Punch, and Print is specified for the 3525 for either BSAM or QSAM.
- IGG019EI. This appendage distinguishes between fixed, fixed–blocked, and undefined user blocks and embedded DOS checkpoint records. In the case of fixed–length blocked records it also distinguishes between wrong–length and truncated blocks.
- IGG019EJ. This appendage distinguishes between wrong–length and variable–length blocks and embedded DOS checkpoint records.

Appendage IGG019BT (Channel End — Create BDAM): Module IGG019BT schedules the writing of successive blocks when a record has to be segmented. The Open executor selects and loads this module if the DCB specifies:

Write (Load)

Variable–length spanned record

The module operates as follows:

- It receives control when the I/O supervisor arrives at the channel–end exit.
- It determines whether the WRITE was WRITE–SZ. If it was WRITE–SZ, the routine returns control to the I/O supervisor.
- When the WRITE–SF is issued, it determines whether the block was spanned record. If not, the routine returns control to the I/O supervisor.
- When a spanned record is being processed, the routine determines whether the entire record has been written. If the record has been written, the routine returns control to the I/O supervisor. When the entire record has not been written, the routine schedules the asynchronous exit routine. The asynchronous exit routine will schedule an EXCP to write a middle segment or the last segment of the record.

Channel–end Appendage IGG019BV (Offset Read): Appendage IGG019BV distinguishes between valid wrong–length blocks and variable–length blocks. It also performs an offset read function when necessitated by spanned records. The Open executor selects and loads this appendage and the associated Read module (IGG019BU), if the Open parameter list specifies:

Input

and the DCB specifies:

BFTEK=R

Variable–length spanned record format

(Under these conditions, the SLI flag is off in the Read CCW.)

The appendage operates as follows:

- It receives control from the I/O supervisor at the channel–end exit.
- If the appendage finds a unit exception bit on in the CSW, it returns to the I/O supervisor immediately.
- If the unit check bit is on, the Abnormal routine is branched too. The abnormal channel–end appendage returns to the I/O supervisor immediately if it finds a cylinder–end or file–protect condition. Otherwise, the current channel program is changed back to Read–key–and–data, and control is returned to the I/O supervisor.
- If a key was not read (Read–data CCW), the command is changed back to Read–key and data.
- If a key was expected (Read–key–and–data CCW) and there was no key to read (key length=0 in count just read), then the Read CCW must be rescheduled with an offset.
- The appendage calculates the length of the block and compares it to the block length field.
- If the lengths are equal, it resets error indicators in the ECB.
- If the lengths are unequal and the current channel program is changed to Read–key–and–data, control is returned to the I/O supervisor. The I/O supervisor then sets the ECB to show that the channel program executed with an error condition.
- The appendage checks the SDW to see if another segment is to follow. If there is, the next channel program is changed to Read–data.

Appendage IGG019CI (Channel End — Fixed–Length Blocked Record Format): Appendage IGG019CI distinguishes between valid wrong–length blocks and truncated blocks. The Open executor selects and loads this appendage if the Open parameter list specifies:

Input, Readback, INOUT, or OUTIN

and the DCB specifies:

Fixed–length blocked records

The appendage operates as follows:

- It performs length checking for fixed–length records. If the record format is fixed standard or the track–overflow feature is used with record–ready, the SILI bit is left off in the Read–data CCW. If a wrong length record is read, the command chaining bit is turned off and the CSW reflects channel end and wrong length indication. The channel–end appendage determines whether the record is a valid short block. For standard format–F records with a valid short block, the module turns on the EOV bit in the DCB and ECB.
- For nonstandard format–F records with the track–overflow feature, a short block is treated as a valid record and the sector value for this last record is used for the next READ.
- Length checking for nonstandard format–F records without the track–overflow feature is performed in the following manner.

The module searches the channel program for a Read–count command, picks up the address of the count to locate the data length and key length, and adds them together. The appendage then compares this value to the block size to determine whether a short block was read. (The SILI bit is on so the channel program will not be terminated with the Read–data CCW.) If a short block has been read, the appendage divides the data length plus key length by the LRECL to determine whether the record is a multiple of the LRECL. If it is, the appendage continues processing using code common to non–RPS. If the DD is not a multiple of the LRECL, the incorrect length bit in the CSW is turned on and processing continues with code common to both RPS and non–RPS.

- For record–ready channel programs, the sector value that was read upon the execution of the preceding channel program is moved to the address of the Set–sector command.
- For record–ready, if there is only one IOB, the CCHHR of the count is moved into the DCBFDAD+3. If there is more than one IOB, the CCHHR of the count is moved into the IOBSEEK+3 of the next IOB or, in the case of update, into the next special FDAD+3.

Appendage IGG019CJ (Channel End — Variable–Length Record Format): Appendage IGG019CJ distinguishes between valid wrong–length blocks and variable–length blocks. The Open executor selects and loads this appendage if the Open parameter list specifies:

Input, INOUT, OUTIN

and the DCB specifies:

Variable–length records

(Under these conditions, the SLI flag is off in the Read CCW.)

The module performs length checking for variable–length records. When the track–overflow option is used with rotational position sensing (RPS), no length checking is performed because the count that would be read in is the count of the first segment only. When the record–ready feature is used without track–overflow, all 8 bytes of the count of the record are read into main storage. The DLDL plus the K of the count is compared with the LL of the record. If they are equal, the module branches via the return register. If they are not equal, it turns on the wrong–length indicator, dummies up the residual count, and continues processing with code common to both RPS and non–RPS.

For record–ready, if there is only one IOB, the CCHHR of the count is moved into the DCBFDAD+3. If there is more than one IOB, the CCHHR of the count is moved into the IOBSEEK+3 of the next IOB or, in the case of update, into the next special FDAD+3.

To save the sector value for the write for update channel program, the first sector value is moved from the Set–sector byte for the Write channel programs. To prepare for executing the next channel program, the contents of the Read–sector byte are moved to the Set Sector byte of the Read channel program.

The appendage operates as follows:

- It receives control when the I/O interruption supervisor arrives at the channel–end exit.

- If the appendage finds a unit–exception bit on in the channel status word, it returns control to the I/O interruption supervisor immediately.
- The appendage calculates the length of the block and compares it to that in the block length field.
- If the lengths are equal, the appendage turns off error indications in the ECB and DCB and returns control to I/O interruption supervisor.
- If the lengths are not equal and the device is magnetic tape, a check is made to see if the block has been padded up to 18 bytes or blocksize, whichever is less. If so, the appendage turns off the error indicators in the ECB and DCB and returns control to the I/O supervisor. If the device is not magnetic tape or the block is not padded, control is returned to the I/O interruption supervisor immediately. The I/O interruption supervisor then sets the ECB to show that the channel program executed with an error condition.

Appendage IGG019CK (Channel End — SYSIN): Appendage IGG019CK translates the delimiter character for a SYSIN data set into an end–of–data–set indication for the access method routine. The Open executor selects and loads this appendage if the device assigned to this DCB is SYSIN.

The appendage operates as follows:

- It receives control when the I/O interruption supervisor arrives at the channel–end exit.
- The appendage tests the buffer for the SYSIN delimiter characters /*.
- If the characters read are not delimiter characters, the appendage returns control to the I/O supervisor.
- If the characters read are delimiter characters, the appendage turns on the unit–exception bit in the channel status word (CSW) and the error flag in bits 0 and 1 of the DCBIFLGS field of the DCB, indicating an end–of–data set condition. The appendage clears bit 6 of byte 187 (CVTSTUSA) of the control vector table (CVT) and returns control to the I/O supervisor. This bit indicates whether the /* delimiter is expected.

Appendage IGG019CS (Channel End — Paper Tape): Appendage IGG019CS distinguishes between valid wrong–length blocks and the wrong–length indication characteristic when paper tape is being read. The Open executor selects and loads this appendage if the DCB specifies:

Fixed–length record format

Paper tape

The appendage operates as follows:

- It receives control when the I/O interruption supervisor arrives at the channel–end exit.
- If the channel status word (CSW) residual count is zero, the appendage turns off error indications in the IOB and the DCB and then returns control to the I/O supervisor.
- If the channel status word (CSW) residual count is not zero, the appendage returns control to the I/O supervisor immediately.

Appendage IGG019CU. (Channel End — Chained Channel–Program Scheduling): This appendage is entered for all three I/O interruption supervisor exits. Refer to "Program Controlled Interruption Appendage" for a description of appendage IGG019CU and a discussion of disconnecting chained channel–programs.

Appendage IGG019C6 (Channel End, Abnormal End — 3525 Associated Data Set): This module is loaded when any combination of Read, Punch, and Print is specified for the 3525 for either BSAM or QSAM.

The module operates as follows:

- DCBQADFL1 (Bit 3 at X'50') is used to indicate whether entry was via a channel end or an abnormal–end interrupt.
- DCBIFLGS (Bit 1 at X'2C') is used to indicate that error correction is in progress.
- Control is transferred via register 14.

Appendage IGG019EI (Channel End, Abnormal End — Fixed–length or Undefined–length Record Format): Appendage IGG019EI distinguishes between fixed, fixed–blocked, and undefined user blocks and embedded DOS checkpoint records. In the case of fixed–length blocked records it also distinguishes between wrong–length and truncated blocks. The Open executor selects and loads this appendage if the Open parameter list specifies:

Input, Readback

and the DCB specifies:

OPTCD=H (specified in JCL)

Magnetic tape

Fixed, fixed–blocked, or undefined–length blocks

The appendage operates as follows:

- It receives control from the I/O interruption supervisor when the I/O interruption supervisor arrives at the channel–end and abnormal–end appendage exits.
- Upon encountering a checkpoint header record, bit 0 in the DEBTFLGS field of the DEB is turned on. It is turned off when the checkpoint trailer record is encountered. This provides the means to differentiate between the user's data records and the embedded checkpoint records.
- In the channel–end entry into this appendage the number of bytes read is tested. If 20 bytes were not read and the bypass–flag bit in the DEBTFLGS field is off, the appendage takes the normal exit to the I/O interruption supervisor for fixed–length and undefined–length formats and performs the necessary record length check for fixed–block records. If 20 bytes were read, the record is tested to determine if it is a checkpoint header record. If it is not a checkpoint header record, the normal exit to the I/O interruption supervisor is taken for fixed–length and undefined–length formats, and record length checking for fixed–block formats is performed.

- When a checkpoint header record is encountered, the bypass–flag bit in the DEBTFLGS field is turned on, the DCBBLKCT field is decremented by the value in the IOBINCAM field, the "Flags 1–3" fields of the IOB are reinitialized, and the IOBERRCT field is set to zero. For QSAM, the IOB completion code is set to X'50' and the normal exit is taken to the I/O interruption supervisor. The bypassing of the checkpoint records is performed in the QSAM routines. For BSAM, the re–EXCP exit is taken to the I/O interruption supervisor.
- The appendage is reentered when the reexecuted channel program ends for BSAM or when the rescheduled channel program ends for QSAM and, finding the bypass flag on, tests for the checkpoint trailer record. If the record read is not the trailer record, the DCBBLKCT field is decreased, the IOB–flag fields reinitialized, and the IOBERRCT field is set to zero. For BSAM, the re–EXCP exit is taken to the I/O interruption supervisor. For QSAM, the IOB completion code is set to X'50', and the normal exit is taken to the I/O interruption supervisor. This process continues until the trailer record is read. When the trailer record is read, the bypass flag is turned off, and the above procedure is followed. The next entry to this channel–end appendage follows the reading of the record immediately following the embedded checkpoint records.
- The appendage is entered in the event of an abnormal condition arising. If this entry is the result of any condition other than a data error, control is returned to the I/O interruption supervisor by way of the normal exit.
- If it is a data error, a test is then performed to determine if a checkpoint header/trailer record was read. This test is comprised of an initial 12 byte comparison of the record's first 12 bytes with the checkpoint identifier

  /// CHKPT //

  Should this comparison fail, a byte–by–byte comparison is performed. If 10 or more bytes compare successfully, it is then assumed that a header or trailer record has been encountered and the appendage returns control to the I/O interruption supervisor.

Appendage IGG019EJ (Channel End, Abnormal End — Variable–length Record Format): Appendage IGG019EJ distinguishes between variable–length and wrong–length blocks and embedded DOS checkpoint records. The Open executor selects and loads this appendage if the Open parameter list specifies:

Input

and the DCB specifies:

OPTCD=H (via JCL)

Magnetic Tape

Variable–length blocks

The appendage operates as follows:

- It receives control from the I/O interruption supervisor when the I/O interruption supervisor arrives at the channel–end and abnormal–end appendage exits.
- Upon encountering a checkpoint header record, bit 0 in the DEBTFLGS field of the DEB is turned on. It is turned off when the checkpoint trailer record is

encountered. This provides the means to differentiate between the user's data records and the embedded checkpoint records.

- In the channel–end entry into this appendage the first two bytes of the record are tested to determine if it is a valid block. (The first two bytes of a variable–length physical record specify the block length and are used in performing length–checking.) The first 12 bytes of a checkpoint header or trailer record (which are identical and 20 bytes in length) identify it as a header/trailer record. These 12 bytes are

  /// CHKPT //

  The first two bytes of the checkpoint header record do not satisfy the length check as a variable–length record. If the first two bytes do satisfy the length check, the appendage takes the normal exit to the I/O interruption supervisor for variable–length records. If the first two bytes do not satisfy the length check for a variable–length record, the number of bytes read is computed. If 20 bytes were not read and the bypass–flag bit in the DEBTFLGS field is off, the appendage returns to the I/O interruption supervisor. If 20 bytes are read, the record is tested to determine if it is a checkpoint header record. If it is not a checkpoint header record, the normal exit to the I/O interruption supervisor is taken for variable–length formats.

- When a checkpoint header record is encountered, the bypass–flag bit in the DEBTFLGS field is turned on, the DCBBLKCT field is decremented by the value in the IOBINCAM field of the IOB, the "Flags 1–3" fields of the IOB reinitialized, and the IOBERRCT field set to zero. For QSAM, the IOB completion code is set to X'50' and the normal exit is taken to the I/O interruption supervisor. The bypassing of the checkpoint records is performed in the QSAM routines. For BSAM, the re–EXCP exit is taken to the I/O interruption supervisor.

- The appendage is reentered when the reexecuted channel program ends for BSAM or when the rescheduled channel program ends for QSAM and, finding the bypass flag on, tests for the checkpoint trailer record. If the record read is not the trailer record, the DCBBLKCT field is decremented, the IOB flag fields re–initialized, and the IOBERRCT field is set to zero. For BSAM, the re–EXCP exit is taken to the I/O interruption supervisor. For QSAM, the IOB completion code is set to X'50', and the normal exit is taken to the I/O interruption supervisor. This process continues until the trailer record is read. When the trailer record is read, the bypass–flag is turned off and the above procedure followed. The next entry to this channel–end appendage follows the reading of the record immediately following the embedded checkpoint records.

- The appendage is also entered in the event that an abnormal condition arises. If this entry is the result of any condition other than a data error, control is returned to the I/O interruption supervisor by way of the normal exit.

- If it is a data error, a test is then performed to determine if a checkpoint header/trailer record was read. This test is comprised of an initial 12–byte comparison of the record's first 12 bytes with the checkpoint identifier

  /// CHKPT //

Should this comparison fail, a byte by byte comparison is performed. If 10 or more bytes compare successfully, it is then assumed that a header or trailer record has been encountered, and the appendage returns control to the I/O interruption supervisor.

Appendage IGG019FP (Channel End — Search–direct): Appendage IGG019FP receives control at channel–end time or if an incorrect length has been given.

The appendage operates as follows:

- For the search–direct feature, the SILI bit is on for all Read CCWs. The appendage does length checking for all formats supported by the search–direct feature.
- If the rotational position sensing (RPS) feature is present, the appendage moves the sector value from the Read–sector address to the Set–sector address.
- If the second TIC in the channel program is a multitrack Read–count CCW, the appendage moves the 2–byte data length of the count field, pointed to by the address of the multitrack Read–count CCW, to the right half of the second TIC location.
- For format–V records, it compares the data length to the record descriptor word that is pointed to by the Read–data command.
- For format–U records, no length checking is provided.
- It finds the multitrack Read–count CCW following the Read-data and moves the data length of the count field, pointed to by the Read–count CCW, to the right half of the second TIC of the next IOB (or to that of the same IOB, if only one IOB exists).
- It moves the CCHHR portion of the count, pointed to by the multitrack Read–count CCW following the Read–data CCW, to the next IOBSEEK field (for multiple IOBs) or to the DCBFDAD (for one IOB).
- It changes the multitrack Read–count CCW, preceding the Read–data CCW to a TIC CCW to the Read–data CCW.
- For exchange buffering, the value in the DCBBLKSI field is used instead of the data length specified in the Read–data command.

Program Controlled Interruption (PCI) Appendage (Execution of Channel Programs Scheduled by Chaining)

There is one program controlled interruption (PCI) appendage. If chained channel–program scheduling is used, its address is placed into the appendage vector table for all three I/O interruption supervisor exits: PCI, channel end, and abnormal end.

A program controlled interruption (PCI), in the sequential access methods, signals the normal execution of a channel program that was scheduled by chaining. The interruption occurs when control of the channel has passed to the next channel program. If the only channel status is PCI, the I/O supervisor performs no processing; if other channel conditions are also present, the I/O supervisor processes these in the usual way after it regains CPU control from the PCI appendage.

This appendage performs the following three functions:

- It performs the channel status analysis usually done by the I/O interruption supervisor. The interruption is caused by a condition in the logic of the channel program rather than a condition in the channel or the device. The condition is meaningful only to the processing program (in this case, the access method routines, or, more specifically, the appendage) and has no meaning to the I/O supervisor.
- It repeats this process for preceding channel programs whose PCIs were lost. PCIs are not stacked. If a channel remains masked from the time of one PCI until after another PCI, only one PCI occurs.
- It performs processing normally necessary for other interruptions (for example, channel end). Interruptions other than PCIs may terminate execution of chained channel programs.

Accordingly, a PCI appendage not only does the processing implicit for the logical condition that the interruption signals (namely, that the preceding channel program executed normally), but also extends this processing back to any preceding channel programs whose PCI may have been masked and, finally, takes CPU control at other I/O interruption supervisor appendage exits if chained channel–program scheduling is used.

Appendage IGG019CU (PCI, Channel End, Abnormal End — Chained Channel–Program Scheduling): Appendage IGG019CU disconnects (parts) chained channel programs that have executed and posts their completion; in addition, it performs normal channel–end and abnormal–end appendage processing. (For a description of the joining process of chained channel–program scheduling, refer to the chained channel–program scheduling end–of–block routines.) The Open executor selects and loads this appendage for use as the channel end, PCI, and abnormal–end appendage if the DCB specifies:

Chained channel–program scheduling

The appendage operates as follows:

- It receives control from the I/O interruption supervisor when the latter arrives at the PCI, channel end, and abnormal–end appendage exits.
- It checks the channel program for a rotational position sensing (RPS) program and, if one is found, moves the ICB's channel program address to the main IOB's TIC, which has been offset by the Set Sector CCW, and updates the sector values (that is, moves the previously read sector to the Set Sector address).
- It tests to determine if the CSW and the "First ICB" field in the IOB, point to the same channel program.
- If they do, the appendage returns control to the I/O supervisor, unless a channel–end condition exists.
- If they do not, the appendage disconnects (parts) the channel program (pointed to by the ICB) from the next channel program in the chain as follows:

  For input, the appendage tests the IOB for an end–of–volume condition. If it exists, the appendage continues as it would for a channel–end interruption with a permanent error.

For output, or for input without an associated end–of–volume condition, the appendage resets the command in the last CCW from TIC to NOP and the address to the beginning of the next channel program.

If the device is magnetic tape, it updates the DCBBLKCT field in the DCB.

If a WAIT macro instruction was addressed to this channel program, the appendage causes the Post routine to perform its processing and to return control to the appendage.

It posts the ICB with the completion code and with channel end and updates the IOB SAM prefix to point to the next ICB.

It repeats this disconnecting process until the IOB and the CSW point to the same channel program.

The appendage continues as follows if channel–end processing occurred without an error:

- It sets the IOB and the ICB to show that the channel program completed without an error, and resets the IOB to point to the next channel program and ICB.
- If there are more channel programs to be executed, the appendage resets the IOB to not–complete and passes control to the EXCP supervisor to schedule these channel programs.
- If there are no more channel programs to be executed, the appendage returns control to the I/O supervisor.

The appendage continues as follows if the channel–end interruption occurred with a wrong–length indication:

- It determines whether a truncated block has been read.
- If a truncated block has been read in a data set with fixed–length blocked standard record format, it sets:

  The DCB to show an end–of–volume condition

  The current ICB to complete–without–error

  The next ICB to complete–with–error

  The CSW in the next ICB to show channel end and unit exception

  It returns control to the I/O interruption supervisor.
- If a truncated block has been read in a data set with fixed–length blocked record format, the appendage sets the ICB to complete–without–error and resets the IOB to point to the next ICB and its channel program. The appendage causes control to pass to the EXCP supervisor to restart the channel.
- If a block with wrong–length data has been read, the appendage continues as it would for permanent errors.

The appendage continues as follows if channel–end processing occurred with an error:

- It isolates the channel program in error by disconnecting it from the next one.
- It sets the IOB to point to the channel program in error.
- It sets the DCB to show that the channel program is being retried.
- It returns control to the I/O interruption supervisor. That routine then processes the channel program in the error–retry procedure.

The appendage continues as follows if channel end occurred with a permanent error:

- It receives control after the I/O supervisor error-retry procedure is found unsuccessful in correcting the error.
- For a 3211 printer, it tests to see whether further retry is necessary. If the ECB is posted in error with an X'41' or X'44' and the command–retry bit in sense byte 1 is on, then it schedules the asynchronous–error–processing module, IGG0197H, and exits.
- It posts the ICB to show that the channel program was completed in error.
- It disconnects the channel program in error from the following one.
- It resets the IOB to point to the channel program after the one in error.
- It returns control to the I/O interruption supervisor.

Appendage IGG019C3 (PCI — QSAM Update): This appendage is described in the section "Abnormal–End Appendages."

Abnormal–End Appendages

Abnormal–end appendages receive control from the I/O interruption supervisor when the latter finds a unit check condition in the channel status word (CSW). The appendages for this exit are a track–overflow appendage and a chained channel–program execution appendage shared with the channel–end and PCI exits. The shared appendage is described under the PCI appendage.

A unit check status in a channel addressing an input data set with track overflow may indicate a permanent error in one segment of a block. If there are further good segments, or if the segment in error is being skipped over to find the next block, the sequential access methods attempt to continue access beyond the segment in error. The processing necessary to accomplish this is performed by the track–overflow asynchronous–error–processing routine, (module IGG019C1, described in "Synchronizing and Error–Processing–Routines"), rather than by the appendage. To permit other I/O operations to continue, the appendage suspends further processing of the condition by the I/O supervisor, schedules the asynchronous error–processing routine and returns control to the I/O supervisor.

Appendage IGG019CU (Abnormal End — Chained Channel–Program Scheduling): This appendage is entered for all three I/O interruption supervisor exits. Refer to Program Controlled Interruption Appendages for a description of appendage IGG019CU and a discussion of disconnecting chained channel–programs.

Appendage IGG019C3 (Abnormal End — Track Overflow, PCI–QSAM Update): Appendage IGG019C3 schedules the track–overflow–asynchronous–error–processing routine if a permanent error occurs in a channel program for an input data set with track overflow. The Open executor selects and loads this appendage for use as the abnormal–end appendage if the Open parameter list specifies:

Input, INOUT, or OUTIN

and the DCB specifies:

Track overflow

or if the Open parameter list specifies:

UPDAT

and the DCB specifies:

QSAM

The abnormal end appendage operates as follows:

- It receives control from the I/O interruption supervisor when the latter reaches the abnormal–end appendage exit.
- If the CSW that caused this appendage to gain control addresses a Read–Data CCW (without a Skip bit) and shows a unit–exception channel status, the appendage returns control to the I/O interruption supervisor without further processing. After control returns to the processing program, the synchronizing or Check routine processes this channel status as an end–of–volume condition.
- If the CSW that caused this appendage to gain control addresses a Read–Data CCW (with a Skip bit on) and shows a unit exception or a unit check channel status, the appendage passes control to the exit–effector routine together with the entry point address of I/O supervisor that causes the I/O supervisor not to post the ECB and to retain the request element for the channel program. The exit–effector routine schedules the track–overflow, asynchronous–error–processing routine for eventual execution and passes control to the given entry point.

The PCI appendage operates as follows:

- Updates the IOBSTART address of the update IOB to point to the refill portion of the channel program
- Returns to the I/O supervisor

Appendage IGG019FR (Abnormal End — 3211 Printer): Appendage IGG019FR schedules the asynchronous error–processing routine IGG019FS when a print line buffer (PLB) parity error or a UCS buffer parity error occurs.

The appendage operates as follows:

- The module receives control before and after the error recovery procedure (ERP).
- The first time the abnormal–end appendage is entered, it is returned to the I/O supervisor to schedule the 3211 ERP.

- The second time the appendage is entered, a return to the I/O supervisor is made when any of the following occurs:

  Command retry bit in sense byte 1 is off.

  Error persists after the print line operation was retried.

  Otherwise, the abnormal–end appendage obtains the address of the interruption request block (IRB) from the DEB and the address of the interruption queue element (IQE) from the IRB. The IQE address is placed in register 1 in complement form. The address of the stage 2 exit effector is obtained from the communications vector table (CVT) and a branch is taken to that address. The stage 2 exit effector schedules the asynchronous routine, which retries the print line. It is then returned to the I/O supervisor.

QSAM Control Routines

These control routines, shared by QSAM and BSAM, consist of both modules loaded by the Open executor and macro expansions. The selection and loading of one of the modules is performed by the Open executor and depend on the access conditions; the presence of macro expansions depends solely on the use of the corresponding macro instruction in the processing program and is independent of the presence or absence of modules.

If a CNTRL macro instruction is encountered in a processing program using QSAM or BSAM, control passes to a control routine. The PRTOV macro expansions place the code to be executed inline in the processing program. CNTRL routines pass control to the I/O supervisor; the macro expansions return control to the processing program. The CNTRL routine for the card reader causes execution of a channel program that stacks the card just read into the selected stacker. The CNTRL routine for the printer causes execution of a channel program with a command to space or to skip. The printer overflow macro expansions cause testing for the printer–overflow condition.

Access Method Options		**Selections**	
CNTRL	X	X	X
Printer	X		
Card Reader, Single Buffer		X	
3525 Printer			X
Modules			
IGG019CA		CA	
IGG019CB	CB		
IGG019FA			FA

Figure 18. Module Selector — Control Modules

There are two CNTRL routines in QSAM; they are load modules. Figure 18 lists the routines available and the conditions that cause a particular routine to be used. The Open executor selects one of the modules, loads it, and puts its address into the DCBCNTRL field.

There are two PRTOV routines, which are macro expansions. Whenever the assembler encounters either of the two macro instructions shown in Figure 19, it substitutes the corresponding macro expansion in the processing program object module.

Macro Instruction	Number of Macro Expansions
PRTOV — User exit	1
PRTOV — No user exit	1

Figure 19. Control Routines that Are Macro Expansions

Control Module IGG019CA (CNTRL — Select Stacker — Card Reader): Module IGG019CA permits stacker selection on the card reader. The Open executor selects and loads this module if the DCB specifies:

CNTRL

Card reader

One buffer

The module operates as follows:

- It receives control when the CNTRL macro instruction is encountered in a processing program.
- For QSAM, the module schedules a channel program that stacks the card just read, reads the next card into the buffer, forces an EOB condition to be recognized by the Get routine, and returns control to the processing program. (Card reader Get module IGG019AG depends on the use of this routine to refill empty buffers.)
- For BSAM, the module schedules a channel program that stacks the card just read and then returns control to the processing program. The Read/Write module IGG019BA causes a channel program to be scheduled that reads the next card into the buffer.
- If the 3505 or 3525 is specified, processing continues for stacker 1 or 2 (whichever is specified in the CNTRL macro instruction of the user's program).

  A test is made to determine if either OMR or RCE is being used.

  If either OMR or RCE is specified, the OMR/RCE bit is turned on in the operation codes of the CCWs.

Control Module IGG019CB (CNTRL — Space, Skip — Printer): Module IGG019CB causes printer spacing and skipping by use of macro instructions; the spacing or skipping to be performed are specified as operands of the macro instruction. The Open executor selects and loads this module if the DCB specifies:

CNTRL

Printer

The module constructs a channel program to control the device, issues an EXCP macro instruction and then returns control to the processing program.

Control Module IGG019FA: This module performs line control functions if:

The 3525 is specified.

A print file is specified.

CNTRL is specified.

The module operates as follows:

- The line counter total (DCBLNP) in the DCB is increased, according to the specifications in the CNTRL instruction.
- I/O macro sequencing is performed when using this module and a 3525 associated data set. If an error is detected, an ABEND (003) is issued with a return code of 03.
- If a skip to a channel on the next card is issued by the user, this module issues an EXCP to feed the next card, issues a WAIT, and returns control to the user's program by way of register 14.

Printer–Overflow Macro Expansions: The PRTOV macro expansions permit processing program response to printer–overflow conditions.

The following macro expansions are created as inline coding during the expansion of the macro instruction.

PRTOV – User Exit: The coding operates as follows:

- A WAIT macro instruction is issued for the IOB pointed to by the DCBIOBA field.
- The DCBIFLGS field of the DCB is tested for an overflow condition.
- If an overflow condition exists, a BALR instruction is issued to pass control to the user's routine.
- If no overflow condition exists, control passes to the next instruction.

PRTOV – No User Exit: The coding creates a test mask in the DCB field located at DCBDEVT+1 and returns control to the processing program.

Note: The printer end–of–block routine temporarily stores the mask in the NOP channel command word (CCW) preceding the Write CCW, turns on a bit in the first byte of the IOB and resets the mask. The PRTOV appendage tests the IOB bit to determine whether to respond to or ignore an overflow condition and resets the bit.

Basic Sequential Access Method Routines

Basic sequential access method (BSAM) routines cause storage and retrieval of blocks of data. BSAM routines furnish device control, but do not provide blocking. There are six types of BSAM routines:

- Read routines
- Write routines
- End–of–block routines
- Check routines
- Appendage routines
- Control routines

Diagram G, BSAM/BPAM Flow of Control, in Section 5 shows the relationship of the BSAM routines to other portions of the operating system and to the processing program.

Control routines (not shown in Diagram G) permit the processing program to control the positioning of auxiliary storage devices. They receive control when the CNTRL (printer, tape, card reader), PRTOV, NOTE, POINT or BSP macro instructions are encountered in a processing program. The track balance routine receives control from a Write routine or the track–overflow, end–of–block routine.

The BSAM control routines are described later in this section of the manual. See Figures 22, 23, and 24 for information about control modules.

Read and Write Routines

A Read or Write routine receives control when the processing program issues a READ or a WRITE macro instruction. The Read and Write routines used with data sets organized for the sequential or partitioned access methods pass control to the end–of–block routines, which in turn pass control to the I/O supervisor. The Write routines, used to create data sets organized for later access by basic direct access method (BDAM) routines, include the end–of–block function within themselves, and so pass control to the I/O supervisor directly. A Read or Write routine processes parameters set by the processing program in the DECB to permit scheduling of the next channel program.

There are six Read/Write modules. Figure 20 lists the modules available and the conditions that cause a particular module to be used. The Open executor selects one of these routines, loads it, and puts its address into the DCBREAD/WRITE field. The figure shows, for example, that module IGG019BH is selected and loaded if update and the READ macro instruction are specified.

Read/Write Module IGG019BA: Module IGG019BA completes the channel program to be scheduled next, and relates control blocks used by the I/O supervisor to the channel program. The Open executor selects and loads this module if the Open parameter list specifies:

Input, Output, INOUT, or OUTIN

and the DCB specifies:

Read or Write

Access Method Options	Selections									
Input	X		X	X		X				
Output		X			X			X	X	X
INOUT, OUTIN	X	X								
Update							X			
Read	X		X	X			X			
Offset Read						X				
Write		X								
Write (Load mode) (Create–BDAM)					X			X	X	X
Paper–tape character–conversion			X	X						
Fixed–length record format			X					X		X
Undefined–length record format				X					X	
Variable–length record format					X	X			X	
Spanned records					X					
Track overflow										X
Read/Write Modules										
IGG019BA	BA	BA								
IGG019BF			BF	BF						
IGG019BH							BH			
IGG019BR					BR					
IGG019BU						BU				
IGG019DA								DA		
IGG019DB									DB	
IGG019DD										DD

Figure 20. Module Selector — Read and Write Modules

The module operates as follows:

- It receives control when a READ or WRITE macro instruction is encountered in a processing program.
- It enters the address of the IOB into the DECB to permit the Check routine to later test execution of the channel program.
- It completes the channel program by inserting the buffer address from the DECB, and the length from either the DECB (for undefined–length records), the DCB (for fixed–length records, and for input of variable–length records), or the record itself (for output of variable–length records).

- If a block is to be written on a direct–access storage device, the module tests the DCBOFLGS field in the DCB to establish the validity of the value in the DCBTRBAL field.
- If the DCBTRBAL value is valid, or if a block is to be written on a device other than direct–access storage, or if a block is to be read from any device, the module passes control to an end–of–block routine.
- If the DCBTRBAL value is not valid (that is, the preceding operation was a Read, Point, or Open for MOD), the module issues an SVC 25 instruction to pass control to BSAM control module IGC0002E to obtain a valid track balance. When control returns to this module, it passes control to an end–of–block routine.

Read Module IGG019BF (Paper–Tape Character Conversion): Module IGG019BF completes a channel program to read paper tape, awaits its execution, and converts the paper tape characters into EBCDIC characters. The Open executor selects and loads this module and one of the code–conversion modules listed in Appendix A, Section 6 if the DCB specifies:

Read

Fixed–length or undefined–length record format

Paper tape

The module operates as follows:

- It receives control when a READ macro instruction is encountered in a processing program.
- It enters the address of the IOB into the DECB to permit the Check routine to test execution of the channel program.
- It completes the channel program by inserting the buffer address from the DECB, and the length value from the DCBBLKSI field (for fixed–length record format) or the DECB (for undefined–length record format).
- It passes control to the end–of–block routine.
- When control returns from the end–of–block routine, the module issues a WAIT macro instruction to await execution of the channel program.
- It converts each character in the buffer until one of the following conditions is met, with the stated effect:

  Conversion has provided the number of characters specified in the length value: The module returns control to the processing program.

  All the characters read have been converted, but into a smaller number of characters. Some input character codes have no corresponding EBCDIC translation in a specific code–conversion module. Therefore, after conversion of all characters in the buffer, the number of converted characters may be less than the length value: The module completes a channel program for the number of additional characters needed to fill the buffer, passes control to the end–of–block routine, which issues the EXCP macro instruction to schedule the channel program, and issues a WAIT macro instruction for the channel program. When control returns, the module resumes converting characters.

An end–of–record character is encountered (undefined–length record format only): The module returns control to the processing program.

The tape is exhausted: The module returns control to the processing program.

A paper tape reader–detected error character is encountered: If necessary because of compression, the module moves the character to the left, without conversion, and returns control to the processing program.

- If one of the characters in the buffer is an undefined character, the module converts the character to the hexadecimal character FF, sets an indication of this condition in the IOB for the paper–tape Check routine, and continues conversion until one of the other conditions is met.

The tables used for code conversion are listed along with the code conversion routines in Appendix A, Section 6.

Read/Write Module IGG019BH (Update): Module IGG019BH ascertains whether a buffer supplied by the processing program is to be written from or read into, and causes a corresponding BSAM update channel program to be executed. The Open executor selects and loads this module if the Open parameter list specifies:

Update

and the DCB specifies:

Read

With the rotational position sensing (RPS) feature, this module bypasses the new CCWs when necessary.

The module operates as follows:

- It gains control when the processing program uses a READ or WRITE macro instruction.
- If data is to be read into a buffer, the module flags the IOB for a Read operation, sets it to point to the Read channel program, and copies the length and buffer address from the DECB or the DCB into the Read CCW.
- If data is to be written from a buffer, the module flags the IOB for a Write operation, sets it to point to the Write channel program, copies the auxiliary storage address from the DCBFDAD field into the IOBSEEK field, and completes the length and buffer address entries in the Write CCW.
- The module passes control to end–of–block module IGG019CC. On return of control from that module, it returns control to the processing program.

Write Module IGG019BR (Create BDAM/VRE): Module IGG019BR writes variable–length spanned blocks and record–zero blocks for a data set that will later be processed by BDAM. The Open executor selects and loads this module if the DCB specifies:

Write (Load mode)

Variable–length spanned record

BFTEK=R

The module consists of three routines: one to write data blocks, one to write record–zero blocks, and an asynchronous exit routine.

To write a data block for BDAM, the routine operates as follows:

- It receives control from the processing program when it encounters a WRITE–SF macro instruction and from the EOV/new volume executor (to write the block not written into the previous volume) after the EOV routine of I/O Support has obtained another extent.
- It determines whether this block fits on the current track. If it does, the routine determines whether the new track balance is greater than 8 bytes. If the new track balance is equal to or less than 8 bytes, the routine adds Write–capacity–record CCWs to the Write–count–key–and–data CCWs. It then issues an EXCP.
- If the block does not fit on the current track, the routine determines whether the block fits on the current volume. If it does, this module constructs a channel program to write the first segment from a segment area associated with this IOB and to write the capacity record of this track. It then issues an EXCP. The asynchronous exit routine writes the successive segments. The DCBFDAD field has the address of the highest track on which the last segment of this record is written.
- If the block does not fit on the track or within the current volume, this routine constructs a channel program to write the capacity record of the track. It then issues an EXCP. The asynchronous exit routine will write the capacity records of all the tracks on this volume. The EOV executor (IGG0551A) will reschedule the Write request on the same volume spanning the extents, if the secondary allocation is on the same volume. When the secondary allocation is on a different volume, the Write request will be written on the new volume.

To write a record–zero block for BDAM, the routine operates as follows:

- It receives control when a WRITE–SZ macro instruction is encountered in the processing program or after the EOV routine has obtained another extent.
- It updates the record–zero area and the channel program to write the record–zero block and issues as EXCP macro instruction. The routine returns control to the processing program or to the EOV routine.
- If there are no data blocks on the track, the module modifies the channel program to clear the track after writing the record–zero block.

The asynchronous exit routine operates as follows:

- It receives control from the channel–end appendage through the exit effector when a spanned record is to be processed.
- If the record is a spanned record, it constructs a segment from the remaining part of the record and issues an EXCP macro instruction to write the segment.
- If the record is a spanned volume record, it issues an EXCP macro instruction to write capacity records up to the end of the extent.

Read Module IGG019BU: This module completes the channel program to read a direct data set, and relates the control blocks used by the I/O supervisor to the channel program. The Open executor selects and loads this module along with an associated channel–end appendage (IGG019BV) if the Open parameter list specifies:

Input

and the DCB specifies:

BFTEK=R

Variable–length spanned record format

The module operates as follows:

- It receives control when a READ macro instruction is encountered in the processing program.
- It enters the address of the IOB into the DECB to permit the Check routine to later test execution of the channel program.
- It completes the channel program by inserting the buffer address and the record length. The buffer address is obtained from the DECB. If there is no key with this segment (this is not the first segment), the buffer address is offset by the key length. This determination is made by checking to see if the CCW has been changed by module IGG019BV to Read–data. The record length is obtained from the DEB and modified by the key length if appropriate.
- The module then issues an EXCP macro instruction.

Write Module IGG019DA (Create–BDAM): Module IGG019DA writes fixed–length data blocks, fixed–length dummy blocks, and record–zero blocks for a data set to be processed later by BDAM. The Open executor selects and loads this module if the DCB specifies:

Write (Load mode)

Fixed–length record format

With the rotational position sensing (RPS) feature, this module tests the first CCW of a channel program created by IGG0199L. It tests for a Set–sector command to determine whether it should take any RPS CCWs into account when making modifications to the channel program.

The module operates as follows:

- It receives control from the processing program when it encounters a WRITE macro instruction and also from the EOV/new volume executor after the end–of–volume routine of O/C/EOV has obtained another extent.
- It connects the next available IOB to the DCB and the DECB.
- It determines, in the same manner as end–of–block routine IGG019CD, whether this block fits on the current track and updates the DCBTRBAL field.
- If this is neither the first nor the last block of a track, the module updates the full device address (FDAD) in the DCB and the IOB and issues an EXCP macro instruction. It then returns control to the processing program or the EOV/new volume executor from which it received control.
- If this is the last block of a track (that is, no other block fits on the track except the present block), the module updates the full device address (FDAD) in the DCB and the IOB, expands the channel program to write the record–zero block for that track as well as the last data block, and issues an EXCP macro instruction. The module then returns control to the routine from which it received control.

- If this is the first block of a new track and there is another track in the allocated extent, the module finds the next track in the allocated extent, updates the full device address (FDAD) in the DCB and the IOB, and issues an EXCP macro instruction. It then returns control to the routine from which it received control.
- If this is the first block of a new track and there is no other track in the allocated extent, the module sets an EOV condition indication and returns control to the processing program.

Write Module IGG019DB (Create–BDAM): Module IGG019DB writes variable–length and undefined–length blocks and record–zero blocks for a data set to be processed later by BDAM. The Open executor select and loads this module if the DCB specifies:

Write (Load mode)

Variable–length or undefined–length record format

The module consists of two routines: one to write data blocks and one to write record–zero blocks.

With the rotational position sensing (RPS) feature, the module tests for a Set–sector command in the first CCW of a channel program created by IGG0199L. If it is an RPS channel program, the module makes the necessary modifications to the channel program.

To write a data block for BDAM, the routine operates as follows:

- It receives control from the processing program when it encounters a WRITE–SF macro instruction and from the EOV/new volume executor (to write the block not written into the previous volume) after the end–of–volume routine of O/C/EOV has obtained another extent.
- It determines whether this block fits on the current track in the same manner as end–of–block routine IGG019CD and updates the DCBTRBAL field.
- If one of the following conditions exists, it returns control without any further processing to the processing program or to the EOV/new volume executor from which it received control:

  A block other than the first block on a track is to be written, but it does not fit on the balance of the track.

  The first block is to be written on a track, but the allocated extents are exhausted. For this condition, the module sets an EOV condition indication before it returns control.
- If either of the following conditions exists, the module updates the full device address (FDAD) in the DCB, the IOB, and the channel program, issues an EXCP macro instruction and then returns control to the routine from which control was received:

  A block other than the first block on the track is to be written and it fits on the balance of the track.

  The first block is to be written on a track and there is another track in the allocated extents.
- It returns control to the processing program or the end–of–volume routine.

To write a record–zero block for BDAM, the routine operates as follows:

- It receives control when a WRITE–SZ macro instruction is encountered in the processing program, or after the end–of–volume routine has obtained another extent.
- It updates the record–zero area and the channel program to write the record–zero block and issues an EXCP macro instruction. The routine returns control to the processing program or to the end–of–volume routine.
- If there are no data blocks on the track, the module modifies the channel program to clear the track after writing the record–zero block.

Write Module IGG019DD (Create–BDAM — Track Overflow): Module IGG019DD creates data sets (with track overflow) of fixed–length data and fixed–length dummy blocks that are subsequently to be processed by BDAM. The module segments the block, enters the segment lengths and buffer segment addresses in the channel program, updates storage addresses for the channel program, and updates count fields for the block to be written and for records–zero of the tracks. The Open executor selects and loads this module if the Open parameter list specifies:

Output

and the DCB specifies:

Write (Load mode)

Fixed–length record format

Track overflow

With the rotational position sensing (RPS) feature, the first CCW of a channel program created by IGG0191M is tested by this module for a Set–sector command code. If the code is present, alterations to the channel program are made accordingly.

The module operates as follows:

- It receives control from the processing program when the program issues a WRITE macro instruction, or from the end–of–volume routine of I/O support after that routine has obtained a new volume to write out any pending channel programs. (The end–of–volume routine receives control from the Check routine when that routine finds that a channel program did not execute because of an end–of–volume condition.)
- If no IOB is available, it returns control to the processing program.
- If an IOB is available, it stores its address in the DCB and the DECB.
- If the block last written was the last one for this extent, the module erases the balance of the extent.
- If the block last written filled the last track used, the module obtains the address of the next track.
- It sets the IOB and its channel program to write the block onto the next available track.
- If the block does not fill the track, the module completes the count field for this record and issues an EXCP macro instruction.

- If the block fills the track, the module sets the track–full indicator, completes record zero for this track, links the channel program that writes record zero to the channel program that writes the data record, and issues an EXCP macro instruction.
- If the block overflows the track, the module completes record zero for this track and completes a channel program to write record zero, completes the count field and channel program for the segment that fits on the track, and constructs the identification for record one of the next track.
- It repeats the preceding until a segment is left that does not overflow a track. For the final segment, the module operates as it would for a block that fits on the track.
- On return of control from the I/O supervisor, the module returns control to the routine from which it was received.

Check Routines

A Check routine synchronizes the execution of channel programs with that of the processing program. When the processing program issues a READ or WRITE macro instruction, control returns to the processing program from the Read or Write routine. This occurs when the channel program has been scheduled for execution or, if reading paper tape, when the buffer has been filled and the data converted. To determine the state of execution of the channel program, the processing program issues a CHECK macro instruction; control returns to the processing program from the Check routine if the channel program was executed successfully, or if it was executed successfully after the Check routine caused volume–switching. For permanent errors, control passes to the processing program's SYNAD routine. Reading or writing under BSAM, the SYNAD routine may continue processing the data set by returning control to the Check routine; writing in the create–BDAM mode, processing cannot be resumed.

If the American National Standard Code for Information Interchange (ASCII) is used and input is specified, the check module issues an XLATE macro instruction which translates the entire input buffer from ASCII form to EBCDIC form. If format–D records are specified, the record descriptor words are form converted from decimal to binary. For format–D records when BUFOFF $\neq$ F, the length of the record read is calculated and placed in the DCB LRECL field.

Figure 21 lists the available Check routines and the conditions that cause a particular module to be used. The Open executor selects one of the four routines, loads it, and places its address into the DCBCHECK field. For example, Figure 21 shows that module IGG019BG is selected and loaded if Read and paper–tape character conversion are specified.

Check Module IGG019BB: Module IGG019BB synchronizes the execution of the channel program to that of the processing program, and responds to any exceptional condition remaining after the I/O supervisor has posted execution of the channel program in the IOB. If ASCII coding is used, the entire input buffer is translated from ASCII to EBCDIC. The Open executor selects and loads this module of the Open parameter list specifies:

Input, Output, INOUT, or OUTIN

and the DCB specifies:

Read or Write

Access Method Options	Selections							
Input	X		X					
Output		X			X			
INOUT, OUTIN	X	X						
Update				X				
Read	X		X					
Write		X						
Write (Load) (Create–BDAM)					X	X		
Paper–tape character–conversion			X					
Variable–length spanned record format					X			
Check Modules								
IGG019BB	BB	BB						
IGG019BG			BG					
IGG019BI				BI				
IGG019BS					BS			
IGG019DC						DC		

Figure 21. Module Selector — Check Modules

The module operates as follows:

- It receives control when a CHECK macro instruction is encountered in a processing program.
- It tests the DECB for successful execution of the channel program.
- If the channel program was executed normally, the module returns control to the processing program.
- If the channel program is not yet executed, the module issues a WAIT macro instruction.
- If the channel program encountered an error condition in its execution, the module issues an SVC 55 instruction to pass control to the SYNAD/EOV executor IGC0005E. Two types of returns from the executor are possible:

  If the executor determines the error condition to be an EOV condition, the executor passes control to the end–of–volume routine of O/C/EOV for volume switching. That routine passes control to the EOV/new volume executor which reschedules the purged channel programs, this executor then returns control to the Check module.

  If the executor determines the error condition to be a permanent error, the executor returns control to the Check module immediately. Control is then passed to the processing program's SYNAD routine. If the SYNAD routine returns control to the Check routine, the routine issues a second SVC 55

instruction to pass control to the SYNAD/EOV executor IGC0005E again. The executor treats this as an ACCEPT error option, implements it, and returns control to the routine, which then returns control to the processing program.

Check Module IGG019BG (Paper–Tape Character–Conversion): Module IGG019BG processes error conditions detected by Read module IGG019BF.

This module is loaded if the DCB specifies the READ macro instruction and paper–tape character–conversion.

The module operates as follows:

- It receives control when a CHECK macro instruction is encountered in a processing program.
- If the Read routine filled the buffer with valid characters, the Check module returns control to the processing program.
- If the Read routine stopped converting because of a reader–detected error character, or if the Read routine encountered an undefined character, the Check module passes control to the processing program's SYNAD routine.
- If control returns from the SYNAD routine, the Check module returns control to the processing program.
- If the channel program encountered an EOV condition, the Check module issues an SVC 55 instruction. Control passes to the SYNAD/EOV executor, IGC0005E, then to the end–of–volume routine of O/C/EOV, and finally to the processing program's EODAD routine.

Check Module IGG019BI (Update): Module IGG019BI synchronizes the execution of a BSAM update channel program to the progress of the processing program. A BSAM update channel program either writes data from a buffer or reads data into a buffer. The module also processes permanent errors and end–of–volume conditions. The Open executor selects and loads this module if the Open parameter list specifies:

Update

and the DCB specifies:

Read

The module operates as follows:

- It receives control when the processing program uses the CHECK macro instruction.
- It tests the ECB in the DECB for successful execution of the channel program associated with that DECB.
- If the channel program has not yet completed processing, the module issues a WAIT macro instruction.
- If the channel program has been executed normally, the module returns control to the processing program.
- If the channel program encountered an error condition in its execution, the module tests to determine if the error is an EOV condition.

- If the error is an EOV condition, the module sets an indicator to show that this entry is from the Check module and passes control to the processing program's EODAD routine.
- If the error is not an EOV condition the module issues an SVC 55 instruction to pass control to the SYNAD/EOV executor, module IGC0005E.
- On return of control from the SYNAD/EOV executor, the Check module passes control to the processing program's SYNAD routine. If the SYNAD routine returns control to the Check routine, the routine issues a second SVC 55 instruction to pass control to the SYNAD/EOV executor (IGC0005E) again. The executor treats this as an Accept–error option, implements it, and returns control to this routine, which then returns control to the processing program.

Check Module IGG019BS (Create BDAM): Module IGG019BS synchronizes the execution of the channel program (to write a block for a BDAM data set) to the progress of the processing program, and responds to exceptional conditions encountered in the execution of the channel program. The Open executor selects and loads this module if the DCB specifies:

Write (Load mode)

Variable–length spanned record

BFTEK=R

The module operates as follows:

- It receives control when the processing program uses the CHECK macro instruction.
- If the channel program is not yet executed, the module issues a WAIT macro instruction.
- If a user specifies WRITE–SFR, the next record address (TTR) is supplied in the next address field of the DECB.
- If the execution of the channel program encounters a permanent error condition, the module passes control to the processing program's SYNAD routine. If control is returned from the SYNAD routine, or if there is no SYNAD routine, the module issues an ABEND macro instruction.
- If the Write routine encounters an EOV condition and therefore does not request scheduling of the channel program for execution, this module passes control to the SYNAD/EOV executor (IGC0005E) by issuing an SVC 55 instruction. On return of control, this module tests for completion of the channel program.

Check Module IGG019DC (Create—BDAM): Module IGG019DC synchronizes the execution of the channel program to write a block for a BDAM data set to the progress of the processing program, and responds to exceptional conditions encountered in the execution of the channel program. The Open executor selects and loads this module if the DCB specifies:

Write (Load mode)

The module operates as follows:

- It receives control when the processing program uses the CHECK macro instruction.

- If the channel program is not yet executed, the module issues a WAIT macro instruction.
- If the channel program executed without error, the module returns control to the processing program.
- If the execution of the channel program encountered a permanent error condition, the module passes control to the processing program's SYNAD routine. If control is returned from the SYNAD routine, or if there is no SYNAD routine, the module issues an ABEND macro instruction.
- If the Write routine encountered an EOV condition and therefore did not request scheduling of the channel program for execution, this module passes control to the SYNAD/EOV executor (IGG0005E) by issuing an SVC 55 instruction. On return of control this module tests for completion of the channel program.

BSAM Control Routines

A control routine receives control when a control macro instruction (for example, CNTRL, NOTE, POINT, BSP) is used in a processing program or in another control routine. BSAM control routines (which include those available in QSAM) pass control

Access Method Options	Selection							
Note/Point	X	X		X	X			
Update, Track Overflow				X				
Chained Scheduling				X	X			
CNTRL			X			X	X	X
Direct–Access Storage	X			X				
Magnetic Tape		X	X		X			
Card Reader						X		
Printer							X	
3525 Printer								X
Control Modules								
IGG019BC	BC							
IGG019BD		BD						
IGG019BE			BE					
IGG019BK				BK				
IGG019BL					BL			
IGG019CA[1]						CA		
IGG019CB[1]							CB	
IGG019FA[1]								FA

[1]These routines are also used in QSAM; see Figure 18 for description of these routines.

Figure 22. Module Selector — Control Modules Selected and Loaded by the Open Executor

to the I/O supervisor, another control routine, or return control to the processing program directly. BSAM control routines cause the physical or logical positioning of auxiliary storage devices.

There are three types of BSAM control routines:

- Routines that are loaded into processing program main storage by the Open executor (CNTRL, Note/Point).
- Routines that are loaded into supervisory transient area main storage by an SVC instruction in a processing program macro expansion or in another control routine (BSP, track balance).
- Routines that are inline macro expansions in the processing program (PRTOV).

Routines that are loaded by the Open executor are mutually exclusive; that is, only one of them can be used with one DCB. The PRTOV macro expansions result in instructions that set or test bits that cause branching in either the processing program or in an appendage.

Figures 22, 23, and 24 list the various kinds of control routines and the conditions that cause them to gain control. Figure 22 shows the access condition options that cause the Open executor to load a control routine for use with a DCB.

Figure 23 lists the SVC instructions that cause a control routine to be loaded at execution time. Figure 24 lists the different macro expansions constructed by the assembler.

SVC Number	Macro Instruction	Function	Module Number
25	(none)	Establish valid track balance. Erase balance of extent for track overflow.	IGC0002E
69	BSP	Device Independent Backspace (tape direct–access).	IGC0006I

Figure 23. Control Modules Loaded at Execution Time

Macro Instruction	Number of Macro Expansions
PRTOV — User exit	1
PRTOV — No user exit	1

Figure 24. Control Routines that Are Macro Expansions [1,2]

[1] These routines are also used in QSAM; see the QSAM section for a description of the routines.

[2] This table duplicates Figure 19; it is repeated here to identify all control routines available in BSAM.

Control Module IGG019BC (Note/Point — Direct Access): The Open executor selects and loads this module if the DCB specifies:

Point

Direct–access storage device

The module consists of two routines: Note and Point.

Note Routine: The Note routine in module IGG019BC converts the full direct–access device address (FDAD) for the last block read or written to a relative address of the form TTR, and presents that value to the processing program.

If the records are standard format without the track–overflow feature, the record number is passed to the resident sector routine to compute the sector value. If the record format is not standard F or if the track–overflow feature is used, the value X'FF' is placed in the byte used by the Set–sector CCW.

The Note routine operates as follows:

- It receives control when a NOTE macro instruction is encountered in a processing program.
- It obtains the FDAD value used by the channel program last executed. The address is found in either the IOB or the DCB depending on which macro instruction the last channel program implemented:

  If the macro instruction is READ and more than one buffer is used, the channel program last executed places the FDAD value into the IOBSEEK field in the IOB.

  If the macro instruction is READ and a single buffer is used, the channel program last executed places the FDAD value into the DCBFDAD field of the DCB.

  If the macro instruction is WRITE, the end–of–block routine places the FDAD value into the DCBFDAD field.
- It issues a BALR instruction to pass control to the IECPRLTV routine, which converts full addresses into relative addresses.
- It returns the address and control to the processing program.

Point Routine: The Point routine in module IGG019BC converts a relative address (of the form TTRZ) to the full direct–access device address (FDAD) used by the next channel program to read or write the block noted.

The Point routine operates as follows:

- It receives control when a POINT macro instruction is encountered in a processing program.
- It issues a BALR instruction to pass control to the IECPCNVT routine which converts the relative address to the full address and returns control to the Point routine. If the processing program passed an invalid relative address, the routine sets the DCBIFLGS and IOBECBCC fields to show that an addressing error occurred before returning control. (The Check routine finds the error and processes accordingly.)

- It establishes the actual value to be used by the next channel program by testing the fourth byte of the relative address TTRZ. If the value of Z is zero, the full address is decreased by one; if Z is one, the address calculated by the IECPCNVT routine is left unchanged. For an explanation of how the value of Z is set, refer to the description of the POINT macro instruction in the publication *OS Data Management Macro Instructions,* GC26–3794.
- It inserts the value in the DCBFDAD and IOBSEEK fields, sets the DCBOFLGS field to show that the contents of the DCBTRBAL field are no longer valid, and returns control to the processing program.

Control Module IGG019BD (Note/Point — Magnetic Tape): The Open executor selects and loads this module if the DCB specifies:

Point

Magnetic tape

This module consists of two routines: Note and Point.

Note Routine: The Note routine in module IGG019BD presents the contents of the DCBBLKCT field of the DCB to the processing program and returns control to the processing program.

Point Routine: The Point routine in module IGG019BD positions the tape at the block for which the NOTE macro instruction was issued.

The Point routine operates as follows:

- It receives control when a POINT macro instruction is encountered in a processing program.
- It constructs a channel program to read forward or backward one block.
- It tests for the bypassing embedded DOS checkpoint records option by testing bit 3 of the DCBOPTCD field. If the option is found to have been specified, the routine issues a GETMAIN to obtain 20 bytes and modifies the CCW to read the first 20 bytes of each block into the obtained main storage while performing recording positioning. The suppress–incorrect–length–indication bit is set in the CCW. The actual bypassing of any embedded DOS checkpoint records is performed by either channel–end appendage IGG019EI or IGG019EJ. Module IGG019BD uses the FREEMAIN macro instruction to obtain main storage prior to returning to the user.
- It passes the channel program for execution the number of times required to position the tape at the desired block.
- It follows the last Read channel program by a NOP channel program to obtain device end information for the last spacing operation.
- It returns control to the processing program, unless a tapemark, load point, or permanent error is encountered in one of the executions of the Read channel program. In that case, the routine sets the DCBIFLGS field to indicate a permanent error, before returning control to the processing program. (Subsequent processing by the Read or Write routine to cause scheduling of channel programs for execution results in their not being scheduled. On the next entry into the Check routine, it detects and processes the error condition.)

Control Module IGG019BE (CNTRL: Space to Tapemark, Space Tape Records): Module IGG019BE positions magnetic tape at a point within the data set specified by the CNTRL macro instruction. The Open executor selects and loads this module if the DCB specifies:

CNTRL

Magnetic tape

The module consists essentially of two routines: One for spacing forward or backward to the tapemark (the FSM/BSM routine), and one for spacing forward or backward a number of tape records (the FSR/BSR routine).

The FSM/BSM routine operates as follows:

- It receives control when a CNTRL macro instruction is encountered in a processing program.
- It constructs a channel program to space to the tapemark in the desired direction.
- It issues an EXCP macro instruction for the FSM or BSM channel program. Control returns to the routine at channel end for the FSM/BSM channel program.
- It issues an EXCP macro instruction for a NOP channel program to obtain device–end information from the FSM/BSM channel program.
- It issues an EXCP macro instruction for a BSR or FSR channel program to position the tape within the data set after the FSM/BSM channel program encounters a tapemark.
- It issues an EXCP macro instruction for a NOP channel program again to obtain device–end information from the BSR/FSR channel program. The routine then returns control to the processing program.

The FSR/BSR routine operates as follows:

- It receives control when a CNTRL macro instruction is encountered in a processing program.
- It constructs a channel program to space one record in the desired direction.
- It tests bit 3 of the DCBOPTCD field for the bypassing embedded DOS checkpoint records option. If the option is found to have been specified, the routine issues a GETMAIN to obtain 20 bytes and modifies the CCW to read the first 20 bytes of each block into the obtained main storage while performing record positioning. The suppress–incorrect–length indication bit is set in the CCW. The actual bypassing of any embedded DOS checkpoint records is performed by either channel–end appendage IGG019EI or IGG019EJ. Module IGG019BE uses the FREEMAIN macro instruction to obtain main storage prior to returning to the user.
- It reduces the count passed by the control macro instruction and issues an EXCP macro instruction for the FSR or BSR channel program.
- When the count is zero, it issues an EXCP macro instruction for a NOP channel program to obtain the device–end information from the last FSR/BSR channel program. The routine then returns control to the processing program.

- If a load point is encountered during spacing, the routine returns control to the processing program.
- If a tapemark is encountered during spacing, the routine repositions the tape to a point within the data set by reverse spacing one block and returns control to the processing program.
- If a permanent error is encountered during spacing, the routine issues a BALR instruction to pass control to the SYNAD routine, if one is present; if not, it issues an ABEND macro instruction.

Control Module IGG019BK (Note/Point — Direct Access — Special): This module contains the Note and Point routines for the special access conditions of chained scheduling, track overflow, and update. The Open executor selects and loads this module if the DCB specifies:

Point

Direct–access storage

Chained scheduling, track overflow, or the Open parameter is update.

Note Routine: The Note routine in module IGG019BK finds the full direct–access device address (FDAD) for the last block read or written, converts it to a relative address of the form TTR, and presents that value to the processing program.

If the records are standard F without the track–overflow feature, the record number is passed to the resident sector routine to compute the sector value. If the record format is not standard F or if the track–overflow feature is used, the value 255 is placed in the byte used by the Set–sector CCW.

The Note routine operates as follows:

- It receives control when a NOTE macro instruction is encountered in a processing program.
- It obtains the FDAD value used by the channel program last executed. The location of this address depends on which macro instruction the last channel program implemented:
- If the macro instruction is READ and more than one buffer is used, the channel program last executed places the FDAD value into the IOBSEEK field in the IOB if track–overflow or update is being used, and into the ICBSEEK field if chained scheduling is used.
- If the macro instruction is READ and only a single buffer is used, the channel program last executed places the FDAD value into the DCBFDAD field of the DCB.
- If the macro instruction is WRITE, the end–of–block routine places the FDAD value into the DCBFDAD field.
- It issues a BALR instruction to pass control to the IECPRLTV routine, which converts full addresses into relative addresses.
- It returns the address and control to the processing program.

Point Routine: The Point routine in module IGG019BK establishes the full direct–access device address (FDAD) used by the channel program to read or write the block noted.

The Point routine operates as follows:

- It receives control when a POINT macro instruction is encountered in a processing program.
- It issues a BALR instruction to pass control to the IECPCNVT routine which converts the relative address to the full address and returns control to the Point routine. If the processing program passed an invalid relative address, the executor sets the DCBIFLGS and the IOBECBCC fields to show that an addressing error occurred, before returning control. The Check routine finds the error and processes accordingly.
- It establishes the actual value to be used by the next channel program by testing the fourth byte of the relative address TTRZ. If the value of Z is zero, the full address is decreased by one; if Z is one, the address calculated by the convert routine is left unchanged. For an explanation of how the value of Z is set, refer to the description of the POINT macro instruction in the *OS Data Management Macro Instructions,* GC26–3794.
- It inserts the value into the DCBFDAD and IOBSEEK fields if track overflow or update is being used, and also into the ICBSEEK field if chained scheduling is used. It sets the DCBOFLGS field to show that the contents of the DCBTRBAL field are no longer valid and returns control to the processing program.

Control Module IGG019BL (Note/Point — Magnetic Tape — Chained Scheduling): Module IGG019BL is selected and loaded by the Open executor if the DCB specifies:

Point

Magnetic tape

Chained scheduling

The module consists of two routines: Note and Point.

Note Routine: The Note routine in module IGG019BL presents the contents of the DCBBLKCT field of the DCB to the processing program and returns control to the processing program.

Point Routine: The Point routine in module IGG019BL positions the tape at the block for which NOTE was issued. It operates as follows:

- It receives control when a POINT macro instruction is encountered in a processing program.
- A channel program is constructed to read forward or backward one block.
- The channel program is passed for execution the number of times required to position the tape at the desired block.

- The last spacing channel program is followed by a NOP channel program to obtain device–end information for the last spacing operation.
- Control is returned to the processing program, unless a tapemark, load point, or permanent error is encountered in the execution of one of the channel programs. In that case, the routine sets the DCBIFLGS field to indicate a permanent error before returning control to the processing program. (Subsequent attempts by the Read or Write routine to cause scheduling of channel programs for execution results in their not being scheduled. On the next entry into the Check routine, the condition is detected and handled.)

Control Module IGC0002E (SVC 25 — Track Balance, Track Overflow Erase): Module IGC0002E consists of two routines that erase either a part of one track or several tracks. The track balance routine determines the available space by erasing the remainder of the track; the track–overflow erase routine erases tracks at the end of each extent on which there are no data fields for blocks of the data set to which the extent belongs. The routine is used when a block in a data set with track–overflow record format would span extents.

This module is loaded at execution time into supervisor transient area main storage if either Read/Write module IGG019BA or end–of–block module IGG019C2 issues an SVC 25 instruction.

Track Balance Routine

The track balance routine establishes a valid value for the DCBTRBAL field of a DCB opened for output to a direct–access device, when the field value has been invalidated by a preceding READ, POINT, or OPEN macro instruction.

The routine operates as follows:

- It constructs and issues an EXCP macro instruction for a channel program with the Erase command and a count exceeding the track capacity. The erase operation begins following the block just read or on the block pointed to.
- It determines the actual track balance by subtracting the residual count in the channel status word (CSW) from the count used in the channel program and inserts the difference in the DCBTRBAL field of the DCB.
- If standard format is specified and the DCB blocksize is equal to the blocksize saved at Open time, the track balance will be computed arithmetically (rather than taking the approximation from the ERASE operation) and stored in the DCBTRBAL field.

Track–Overflow Erase Routine

The track–overflow erase routine erases the space on a direct–access storage device that lies between the last block to be written into the current extent and the end of that extent. If the track–overflow end–of–block routine IGG019C2 finds that the next segment of a block falls on a track beyond the present extent, that end–of–block routine uses the SVC 25 instruction to pass control and the channel program to this routine.

The routine operates as follows:

- It receives control when it is loaded.
- It substitutes Erase commands for the Write commands in the channel program associated with the present IOB.
- It issues an EXCP macro instruction to cause execution of the channel program and a WAIT macro instruction for its completion.
- It returns control to the track–overflow end–of–block routine, irrespective of any errors in the execution of the channel program.

Control Module IGC0006I (SVC 69 — BSP): Module IGC0006I backspaces the data set one block whether the data set is on a magnetic–tape or direct–access device.

The expansion of the macro instruction BSP includes an SVC 69 instruction which causes the module to be loaded and entered. The module essentially consists of two parts, one for magnetic tape and one for direct–access devices.

For magnetic tape, the module operates as follows:

- It receives control after it is loaded.
- It constructs and issues an EXCP macro instruction for a channel program to backspace one block.
- It constructs and issues an EXCP macro instruction for a NOP channel program to obtain device–end information from the backspace channel program.
- If the backspace channel program executed normally, the module sets register 15 to zero and returns control to the processing program.
- If the channel program executed with an error other than unit exception, the module sets the DCBIFLGS field to indicate a permanent error. The CHECK macro instruction, following the next READ or WRITE macro instruction, causes the Check routine to pass control to the processing program's SYNAD routine.
- If the backspace channel program executed with a unit exception, the module constructs and issues an EXCP macro instruction for a channel program to forward space the tape one block. It next constructs and issues a NOP channel program to obtain device–end information from the forward space channel program. When channel end for the NOP channel program occurs, the module returns control to the processing program with register 15 set to an error code.

For direct–access devices, the module operates as follows:

- It receives control after it is loaded.
- It decreases the DCBFDAD field in the DCB to the preceding block address across tracks, cylinders, or extents.
- It sets the DCBOFLGS field to show that the DCBTRBAL field value is invalid.
- If a valid preceding DCBFDAD value has been established, the module returns control to the processing program with register 15 set to zero.
- If there is no valid preceding DCBFDAD value because the processing program has attempted to backspace beyond the first block, the module returns control to the processing program with register 15 set to an error code.

- If a permanent error is encountered when reading the count fields (to establish the preceding DCBFDAD field value), the DCBIFLGS field value is set to indicate a permanent error. The Check routine, following the next READ or WRITE macro instruction, causes control to pass to the processing program's SYNAD routine.

Basic Partitioned Access Method Routines

A partitioned data set has a directory and members. The directory is read and written using BPAM routines, whereas the members are read and written using BSAM routines. (Refer to the BSAM portion of this publication.) A processing program using BPAM routines for input from the directory is presented with the address of a member in a channel program or in a table; for a processing program using BPAM for output to a directory, the routines determine the address of the member and record that address in the directory.

BPAM Routines

BPAM routines store and retrieve entries in the directory and convert auxiliary storage addresses from relative to absolute. Directory entries are entered and found by constructing channel programs that search the directory for appropriate entry blocks and by locating an equal, or higher, entry within the block. Address converting routines refer to the data extent block (DEB) to determine the address value complementary to the given value.

BPAM routines (see Figure 25) differ from BSAM and QSAM routines in that BPAM routines are not loaded at Open time; the Stow routine is loaded at execution time, all the coding for Find (C option) is a macro expansion, and the Find (D option)/BLDL routine and the converting routines are in resident main storage. Figure 25 shows how these routines gain control.

BPAM Routines	Module Number	Residence	Instruction Passing Control
STOW	IGC0002A	Supervisory Transient Area	SVC 21
STOW (C, D option)	IGG0210A	Supervisory Transient Area	XCTL from IGC0002A
FIND (C option)	(Macro Expansion)	Processing Program Area	FIND (C option)
FIND (D option)	IECPFIND, IECPFND1	Supervisory Resident Area	SVC 18
BLDL	IECPFIND, IECPFND1	Supervisory Resident Area	SVC 18 or BAL IECPBLDL
Convert TTR	IECPFIND, IECPFND1	Supervisory Resident Area	BAL IECPCNVT
Convert MBBCCHRR	IECPFIND, IECPFND1	Supervisory Resident Area	BAL IECPRLTV

Figure 25. BPAM Routines Residence

STOW Modules

STOW Module IGC0002A (SVC 21): Module IGC0002A finds entries in BPAM directory entry blocks and left–justifies directory entries whenever any are inserted or deleted.

The expansion of the STOW macro instruction includes an SVC 21 instruction that causes this module to be loaded and to gain control. The STOW macro instruction is issued in one of two ways:

Explicitly by a processing program using BPAM for output.

Implicitly by a processing program using BSAM, QSAM, or BPAM for output, when issuing a CLOSE macro instruction to a DCB opened for a member of a partitioned data set

The module operates as follows:

- It receives control when it is loaded.
- If either the delete (D) or the change (C) option is specified, the module transfers control to module IGG0210A using in XCTL macro instruction.
- For any option, the module searches the directory for an entry block with a key equal to or higher than the member name, and reads that entry block into the input buffer.
- The module compares the entries in the entry block to the member name in the instruction operand. Entries whose value is lower than that of the member name are moved to the output buffer.
- For entries that equal the member name, the module checks to determine whether the replace (R), the change (C), or the delete (D) option is specified.
- If the REPLACE option is specified, the module moves the new entry from the work area to the output buffer, skips the present entry, and moves the remaining entries to the output buffer. It issues an EXCP macro instruction to write the updated entry block into the directory.
- For entries that are higher than the member name, the module checks to determine whether the add (A) option is specified.
- If the add (A) option is specified, the module moves the new entry from the work area to the output buffer before moving the high entry and those following it. The module then shifts to the right all entries following the added entry by constructing the channel programs necessary alternately to write and read entry blocks. The module writes the full block, moves the remaining entries to the output buffer, reads another entry block, and then completes and writes the output buffer.
- If an add (Not ALIAS) or a replace (Not ALIAS) operation is successfully completed, the module writes an end–of–data set mark (zero–length data block) at the end of the member. The module then stores, for use at the next entry into the Stow module, the relative address of the next block to be written, in the DCBRELAD field of the DCB. The Open routine determines the first relative address for the first entry to this module.
- On completion of all channel programs necessary for the specified option, the routine returns control to either the processing program, or the Close routine.

STOW Module IGG0210A: Module IGG0210A finds entries in BPAM directory entry blocks and left–justifies directory entries whenever any are inserted or deleted.

Module IGG0210A is loaded and receives control through an XCTL macro instruction issued by module IGC0002A if it determines that either the delete or the change option is specified.

The module operates as follows:

- It receives control when it is loaded.
- For any option, the module searches the directory for an entry block with a key equal to or higher than the member name, and reads that entry block into the input buffer.
- The module compares the entries in the entry block to the member name in the instruction operand. Entries whose value is lower than that of the member name are moved to the output buffer.
- If the change option is specified, the module moves the present entry, less the present name, to the new entry work area. To enter the new entry in its proper entry block, the routine continues as though the add option were specified.
- If the delete option is specified, the module skips the present entry and moves the remaining entries to the output buffer. The module now shifts the balance of the entries in the directory to the left by constructing the necessary channel programs. It reads a block, shifts entries into the remaining space of the preceding block, writes the completed entry block, and starts the next block.
- On completion of all channel programs necessary for the specified option, the routine returns control to either the processing program or the Close routine.

FIND (C Option) Macro Expansion

The macro expansion moves the relative address (TTRK) from the BLDL list in main storage to the DCBRELAD field in the requester's DCB. The FIND macro instruciton then does a branch–and–link to the Point routine.

Resident Module IECPFIND

Unless BLDLTAB is specified for the RESIDNT option of the SUPRVSR macro instruction in the system generation program, this module is link–edited at system generation with other modules to make up the resident nucleus. (If BLDLTAB is specified, module IECPFND1 is used.)

The routines in this module gain control through an SVC 18 instruction in a processing program or a BALR instruction in a control program. A FIND (D Option) or BLDL macro instruction expansion generates an SVC 18 instruction which causes control to pass to CSECT IGC018, the entry point for the Find (D Option) and BLDL routines. Control programs may use a BALR instruction and the address found in the communications vector table (CVT) for entry points IECPBLDL, IECPCNVT, and IECPRLTV to pass control to the respective routines.

Find (D Option) Routine — Entry Point and CSECT Name: IGC018 (SVC 18): The Find (D Option) routine finds the relative address of the member named in the macro instruction. It then causes the relative address to be converted into the full device address (FDAD) and to be loaded into the DCBFDAD and IOBSEEK fields. The routine operates as follows:

- It searches the directory for an entry block with a key equal to or higher than the given member name.

- It reads that entry block into main storage and searches the entry block for the matching entry.
- It enters the relative address stated in the entry into the DCBRELAD field in the DCB and issues a BAL instruction to pass control to the Point routine. Control returns to the processing program.

BLDL Routine — Entry Points: IECPBLDL, IGC018 (SVC 18): The BLDL routine completes a BLDL table with the directory entry for each of the members named in the BLDL table. The routine operates as follows:

- It searches the directory for an entry block with a key equal to or higher than the given member name.
- It reads that block into main storage and searches the entry block for the matching entry.
- It moves the entry into the processing program's BLDL table, obtains the next name to be matched, and returns to the beginning of the routine.
- When the BLDL table has been completed, the routine returns control to the processing program.

Convert Relative–to–Full Address Routine — Entry Point: IECPCNVT: Converting routine IECPCNVT accepts, in register 0, a relative address of the form TTR for direct–access devices and presents the corresponding full device address of the form MBBCCHHR at the location shown by register 2.

The routine operates as follows:

- For each extent, the module reduces the amount TT by the number of tracks in the extent. When the balance is negative, the proper extent has been reached.
- It determines the full device address for the specified relative value.

Convert Full–to–Relative Address Routine — Entry Point: IECPRLTV: Converting routine IECPRLTV accepts, from the location shown by register 2, a full device address of the form MBBCCHHR for direct–access devices and presents the corresponding relative address of the form TTR in register 0.

The module totals the number of tracks per extent for the (M – 1) extents. For extent M, it adds the number of tracks entered into the extent.

Resident Module IECPFND1

If BLDLTAB is specified for the RESIDNT parameter of the SUPRVSOR macro instruction when the system is generated, this module is link–edited at system generation with other modules to make up the resident nucleus. (If BLDLTAB is not specified, module IECPFIND is used.) At initial program loading (IPL) time, the nucleus initialization program (NIP) constructs a resident BLDL table from SYS1.LINKLIB directory entries. That table is the one referred to by the Find and BLDL routines in this module.

The routines comprising the module gain control through an SVC 18 instruction in a processing program or a BALR instruction in a control program. A FIND (D Option) or BLDL macro instruction expansion generates an SVC 18 instruction which causes control to pass to CSECT IGC018, the entry point for the Find (D Option) and BLDL routines. Control programs may use a BALR instruction and the address found in the communications vector table (CVT) for entry points IECPBLDL, IECPCNVT, and IECPRLTV to pass control to the respective routines.

Find (D Option) Routine — Entry Point and CSECT Name: IGC018 (SVC 18): The Find (D Option) routine finds the relative address of the member named in the macro instruction. It then causes the relative address to be converted into the full direct–access device address (FDAD) and to be loaded into the DCBFDAD and IOBSEEK fields. The routine operates as follows:

- If SYS1.LINKLIB is the referenced library, it scans the resident BLDL table for an entry that matches the given member name.
- If SYS1.LINKLIB is not the referenced library, or if the name is not in the table, it searches the directory for an entry block with a key equal to or higher than the given member name. It reads that entry block into main storage and searches the entry block for the matching entry.
- If the name is in the table, or after finding the matching entry in an entry block read in, it enters the relative address stated in the entry into the DCBRELAD field in the DCB.
- It issues a BAL instruction to pass control to the Point routine.
- It returns control to the processing program.

BLDL Routine — Entry Points: IECPBLDL, IGC018 (SVC 18): The BLDL routine completes a BLDL table with the directory entry for each of the members named in the BLDL table. The routine operates as follows:

- If SYS1.LINKLIB is the referenced library, it scans the resident BLDL table for an entry that matches the given member name.
- If SYS1.LINKLIB is not the referenced library, or if the name is not in the table, it searches the directory for an entry block with a key equal to or higher than the given member name. It reads that block into main storage and searches the entry block for the matching entry.
- If the name is in the table, or after finding the matching entry in an entry block read in, it moves the entry into the processing program's BLDL table, obtains the next name to be matched, and returns to the beginning of the routine.
- When the BLDL table has been completed, the routine returns control to the processing program.

Convert Relative–to–Full Address Routine — Entry Point: IECPCNVT: Converting routine IECPCNVT accepts, in register 0, relative addresses of the form TTR for direct–access devices and presents the corresponding full device addresses of the form MBBCCHHR at the location shown by register 2.

The routine operates as follows:

- For each extent, the routine reduces the amount TT by the number of tracks in the extent. When the balance is negative, the proper extent has been reached.
- It determines the full device address for the specified relative value.

Convert Full–to–Relative Address Routine — Entry Point: IECPRLTV: Converting routine IECPRLTV accepts, from the location shown by register 2, a full device address of the form MBBCCHHR for direct–access devices and presents the corresponding relative address of the form TTR in register 0.

The routine totals the number of tracks per extent for the (M–1) extents. For extent M, it adds the number of tracks entered into the extent.

Sequential Access Method Executors

Sequential access method executors are routines that receive control from, pass control to, or return control to I/O support routines. For a description of I/O support routines refer to the publication *OS Open/Close/EOV Logic,* GY28–6609. Figure 26 shows the sequence of control between executors and other routines. Executors perform processing unique to an access method when a data control block is being opened or closed, or an end–of–volume condition is being processed. These executors, used for QSAM, BSAM, and BPAM, are of six types:

- Open executor
- Close executor
- SYNAD/EOV executor
- FEOV executor
- EOV/new volume executor
- SETPRT executor

Executors differ from other access method routines in that they are executed from the supervisory transient area. It is the Open executor that loads the access method routines into the processing program area for later use during processing program execution.

The Open executor is entered from the Open routine of I/O support, and returns control to that routine. It constructs the data extent block (DEB), the input/output blocks (IOB), the channel programs, and, if chained channel–program scheduling is used, interruption control blocks (ICB). It selects and loads the access method routines to be used with the data control block (DCB) being opened.

Executor	Number	Receives Control From	Via	Passes Control To
OPEN	See Figures 27, 28, & 29	See Diagram E in Section 5	XCTL (WTG Table)	See Diagram E in Section 5
CLOSE	IGG0201A IGG0201B IGG0201Z IGG0201X IGG0201Y	Close Routine	XCTL (WTG Table)	Close Routine See Figure 30
SYNAD/EOV	IGC0005E IFG0551B	Synchronizing, Check Routines	SVC 55	See Executor Description Figure 31
FEOV	IGC0003A	Processing Program	FEOV Macro Instruction (SVC 31)	EOV Routine Figure 31
EOV/new volume	IFG0551L IFG0551N	EOV Routine	XCTL	See Executor Description Figure 31
SETPRT	IGC0008A IGG08101 IGG08102	Processing Program	SVC81	See Executor Description Figure 32

Figure 26. Sequential Access Method Executors — Control Sequence

The Close executor is entered from the Close routine of I/O support, and returns control to it. The executor handles any pending channel programs and releases the main storage used by the IOBs, ICBs, and channel programs.

The SYNAD/EOV executor is entered when a synchronizing or Check routine finds that a permanent I/O error or end–of–volume (EOV) condition was encountered during the execution of a channel program. The executor passes control to the end–of–volume routine of I/O support, or executes the error options specified by the processing program. The executor provides a work area in main storage for the end–of–volume routine.

The FEOV (force–end–of volume) executor is entered when an FEOV macro instruction is encountered in a processing program. The executor handles any pending channel programs, provides a work area in main storage for the end–of–volume routine, and passes control to the end–of–volume routine of I/O support.

The EOV/new volume executor receives control from the end–of–volume routine of I/O support. The executor causes the I/O supervisor to reschedule any channel programs not executed because of the EOV conditions.

The SETPRT (set printer) executor is entered when a SETPRT macro instruction is issued by a processing program. The executor loads the UCS image in the UCS buffer and prints verification lines if required.

Open Executors

The Open executors are grouped into three stages. Those in the first stage receive control from the Open routine of I/O support. These executors pass control to one of the stage 2 executors, or return control to the Open routine. The stage 2 executors in turn, pass control to the stage 3 executors, or return control to the Open routine. Stage 3 executors return control to the Open routine. Before relinquishing control, each executor specifies the next executor to be called for the data set being opened, and also examines the where–to–go (WTG) table to determine whether other data sets being opened at the same time need its services. For a description of the WTG table, refer to *OS Open/Close/EOV Logic,* GY28–6609.

When a multivolume data set is opened, the direct–access storage devices with the rotational position sensing (RPS) feature incorporate this feature into the channel program only if all of the devices allocated have the record–ready feature.

Diagram E (Section 5) shows the flow of control between the three stages of Open Executors.

Stage 1 Open Executors

Stage 1 Open executors construct data extent blocks (DEBs) and buffer pools. If a printer with the universal character set (UCS) feature and/or a forms control buffer is specified, the executors load the UCS/FCB image specified in the job control statement.

The Open routines determine which executor is required to begin processing of each DCB specified in the Open parameter list. For SAM processing, the entry placed in the WTG table is IGG0191A for an actual data set and IGG0191C for a dummy data set.

The executor for the first entry in the WTG table is loaded into the transient area by the Open routines and control is passed to the executor by means of an XCTL macro instruction.

As each stage 1 executor completes its processing, the name of the next executor (for the DCB being processed) is placed in the WTG table. Then a check is made to determine, for each entry in the Open parameter list, if another DCB requires the use

Access Method Options	**Selections**											
Actual Data Set	X	X	X	X	X			X	X	X	X	
Dummy Data Set							X					
3505 (OMR/RCE) or 3525												X
Direct Access Device		X		X					X		X	
Printer with UCS Feature (1403 or 3211)					X							
Printer with forms control buffer (3211 or 2245)						X						
Buffer Pool Required			X	X						X	X	
User Totaling Specified								X	X	X	X	
Executors												
IGG019AV							AV					
IGG0191A	1A	1A	1A	1A	1A	1A		1A	1A	1A	1A	
IGG0191B	1B	1B	1B	1B	1B	1B		1B	1B	1B	1B	
IGG0191C							1C					
IGG0191I			1I	1I						1I	1I	
IGG0191N		1N		1N					1N	1N	1N	
IGG0191T					1T	1T						
IGG0191U					1U							
IGG0191V					1V							
IGG0191Y								1Y	1Y	1Y	1Y	
IGG0193I			3I	3I						3I	3I	
IGG0196A	6A	6A	6A	6A	6A	6A		6A	6A	6A	6A	
IGG0196B	6B	6B	6B	6B	6B	6B		6B	6B	6B	6B	
IGG0196I	6I	6I	6I	6I	6I	6I		6I	6I	6I	6I	
IGG0197E						7E						
IGG0197F						7F						
IGG0197L												7L
IGG0197M												7M
IGG0197U					7U							

Figure 27. Open Executor Selector — Stage 1

of the executor now in control. If so, the executor is reentered as many times as necessary to process all of the entries in the WTG table requiring this executor. If no other DCBs require this executor, control is passed to the next executor that is specified in the WTG table (starting from the top of the list) for a DCB that has not completed its processing. For a particular DCB, all of the stage 1 executors are executed before control is passed to a stage 2 executor.

Figure 27 lists the access method conditions that cause different stage 1 executors to be selected, loaded, and to receive control after loading. The executors are described in the text that follows. The order of presentation is the same as that shown in Figure 27 under Executors.

In Figure 27, an X in a column represents a condition that must be satisfied for the executor marked in that column. A blank in the upper portion of the table indicates that either the condition is not required for selection or not examined at this time. The table should be used in conjunction with the flow of control information in Diagram E, SAM Flow of Control for Open Executors, Section 5.

Dummy Data Set Module IGG019AV: Dummy data set module IGG019AV operates as follows:

It receives control when a sequential access method macro instruction refers to a dummy data set. For a dummy input data set, the module passes control to the user's EODAD routine; for a dummy output data set, the module returns control to the processing program immediately without scheduling any I/O operation.

Stage 1 Open Executor IGG0191A: Executor IGG0191A receives control from the Open routine unless the DD statement is DUMMY. (If the DD statement is DUMMY, executor IGG0191C receives control from the Open routine.)

The executor operates as follows:

- It tests the OPEN macro option against the DCBMACRF field. It issues an 013 ABEND if any of the conditions listed are found. The conditions are:

  For QSAM:

  that buffer length is not smaller than blocksize if the buffer length is specified

  that the blocksize is not at least 4 bytes larger than logical–record length for variable–length records

  that logical–record length (if specified) is not equal to blocksize for fixed–length unblocked data sets

  For BSAM and QSAM:

  that blocksize is not an even multiple of logical record length for fixed–length blocked data sets

- It performs a test to determine if the blocksize is an integral multiple of the logical record length (LRECL) for QSAM with fixed blocked records or BSAM data sets. If the blocksize is not an integral multiple of LRECL, the blocksize is reduced to the nearest integral multiple of LRECL for SYSOUT data sets on direct access where the blocksize is specified in the DD statement. If the data set is not SYSOUT or on direct access, or if the blocksize is not specified in the DD statement, BSAM data sets are allowed to continue without adjustment to the blocksize; QSAM data sets are abnormally terminated with an ABEND (013).

- If search–direct has been requested (OPTCD=Z in the DCB), the executor evaluates the request and sets the bit in JFCBMASK±6 to X'08' if the request can be honored.
- The executor specifies in the WTG table that module IGG0196I is the next module required for this DCB.
- It searches the WTG table to pass control to another executor.

Stage 1 Open Executor IGG0191B: Executor IGG0191B is loaded after executor IGG0196A, IGG0191N or IGG0191Y has completed processing all entries in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- If the DCB is not a TSO DCB, this routine fills in the DCBDEVT field with the device type and number from the UCB. If unit record equipment is indicated, the routine sets the UR bit in the DCBDEVT field.
- It stores DCBLRECL in the DEB.
- It sets DCBCNTRL to 0.
- If the device type is direct–access storage, the address of the device table is stored in the DCB. From the device table, the key overhead (or 0 if there are no keys) and track balance are stored in the DCB.
- If the JFCB indicates a partitioned data set, the DSCB and the DSORG field of the DCB are checked to be sure they specify partitioned organization. If not, an ABEND is issued.
- If partitioned organization is specified:
  a. For direct–access OUTPUT or OUTIN, the track balance of the last Write, from the DSCB, is stored in the DCBTRBAL.
  b. For direct–access input, the member name from the JFCB is stored in the DEB. The routine then issues a BLDL macro instruction to find the extent.

The executor issues a BALR to the convert routine at CVTPCNVT to convert the TTR to MBBCCHHR and stores it in DCBFDAD.

- If the data set is not partitioned, DCBFDAD is set to DEBBINUM. If a dummy extent is indicated, the DCBFDAD+3 is set to X'FF' to indicate an illegal FDAD.
- If unit record equipment is specified, for input only and Note/Point is requested, DCBCNTRL+1 is set with ID of the dummy routine.
- If LRECL is not specified, DCBLRECL is set equal to DCBBLKSI.
- It then searches the WTG table to pass control to another executor.

Stage 1 Open Executor IGG0191C: Executor IGG0191C operates as follows:

It receives control from the Open routine if the DD statement is DUMMY, and loads module IGG019AV. Dummy data sets require only this executor; if no other data sets are being opened, control returns to the Open routine.

Stage 1 Open Executor IGG0191I: Executor IGG0191I is loaded after IGG0191B, IGG0191T, IGG0191U, or IGG0191V if the Open executor must build buffer pools.

The executor operates as follows:

- It receives control after it is loaded.
- If a buffer pool has already been built, the executor gets main storage for the record area.
- If the values in both the DCBBUFL and DCBBLKSI fields are zero, the executor passes control to the ABEND routine.
- If the value in either the DCBBUFL or DCBBLKSI field is not zero, the executor uses that value to establish the size of the buffer. The value in the field DCBBUFNO determines the number of buffers constructed.
- If logical record interface is required for variable–length spanned records processed in locate mode, the executor adds a length of 32 bytes plus the maximum logical–record length, which is specified in the DCBLRECL field for a record area to the size of main storage required. Four more bytes are added to the buffer control block to store the address of the record area. A flag is set to indicate extended buffer control block.
- It stores the length of the entire record area in the first word of the record area.
- It specifies the Stage 2 executor required for this DCB in the WTG table. It then searches the WTG table to pass control to another executor.
- If the time sharing option (TSO) is specified with BSAM and DCBBUFL and the length of the buffer to terminal line length. When QSAM is specified and DCBBUFL and DCBBLKSI field are zero, it sets the length of the buffer to logic record length. If DCBLRECL field is also zero, it sets the length of the buffer to terminal line length. It transfers control to IGG0196S, (see *OS TSO Control Program Logic,* GY27–7199).

Stage 1 Open Executor IGG0191N: Executor IGG0191N receives control after executor IGG0191A. It supplements executor IGG0191A by building the device–dependent portion of the DEB for direct–access devices.

The user label extent is not inserted in the DEB. This executor specifies either IGG0191B or IGG0191Y as the next entry in the WTG table for processing the DCB.

It then searches the WTG table to pass control to another executor.

Stage 1 Open Executor IGG0191T: Executor IGG0191T is entered after Open executor IGG0196A or IGG0196B if a printer with the UCS feature is specified.

The executor operates as follows:

- It uses the EXCP macro instruction to execute block data check or reset block data check according to the specification in this DCB.
- It examines the UCB and JFCB to determine whether the FCB image or the UCS image is to be loaded.
- When the specified UCS image has not been loaded by a previous job step, the executor specifies in the WTG table that executor IGG0191U is the next executor required for this DCB.

- When no UCS image is specified in the JFCB and the UCB has no UCS image ID, or the UCS image ID in the UCB is not a default UCS image, the executor requests an operator to specify a UCS image to be used for the DCB. Then it specifies in the WTG table that executor IGG0191U is the next executor required for this DCB.
- When the specified UCS image has been loaded by a previous job step or no UCS image is specified in the JFCB but UCS image ID in the UCB is a default UCS image ID, the currently loaded UCS image is used for this DCB.
- If no FCB image is named in the JFCB and the UCB has no FCB image ID, or the FCB image ID in the UCB is not a default image, the executor issues a console message requesting that an image be specified.
- If there is no UCS activity and the form control buffer must be loaded or FCB verification is requested, IGG0197E is the next executor to receive control.
- If the printer is a 3211 and no FCB or UCS activity is required, control is passed to IGG0197F to obtain 570 bytes for the ERP work area.
- If the printer is a 1403 and no UCS activity is required, the executor searches the WTG table and passes control to IGG0191I, or a stage 2 executor, or the final module of the Open routine.
- It then searches the WTG table to pass control to another executor.

Stage 1 Open Executor IGG0191U: Executor IGG0191U is entered after executor IGG0191T when the specified UCS image is to be loaded from SYS1.IMAGELIB.

The executor operates as follows:

- It requests the operator to mount a chain/train cartridge only if a UCS image is specified in the DD statement.
- It uses the BLDL macro instruction to locate the UCS image in the SYS1.IMAGELIB.
- If the image is not found in the library, the executor requests the operator to specify an alternate UCS image to be used.
- If the printer is a 1403, the executor specifies in the WTG table that IGG0191V is the next executor required for this DCB if the UCS image is to be loaded or verified.
- When the UCS image is neither to be loaded nor to be displayed, it specifies in the WTG table that executor IGG0191I, or a stage 2 executor, or the final module of Open routine of O/C/EOV is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.
- If the printer is a 3211, control is always passed to IGG0191V which determines whether UCS and/or FCB image(s) must be loaded into main storage and/or into the buffer(s).

Stage 1 Open Executor IGG0191V: Executor IGG0191V is entered after executor IGG0191U to load the UCS image into main storage and subsequently into the UCS buffer. It can also issue a message requesting the operator to specify an FCB image.

The executor operates as follows:

- It uses the LOAD macro instruction to retrieve the UCS image from SYS1.IMAGELIB.
- It uses the EXCP macro instruction to load the UCS image into the UCS buffer.
- When all the following conditions are met, it requests the operator to specify which FCB image is to be loaded:

  The printer is a 3211.

  The current FCB image is not a default.

  An FCB image was not specified in the DD statement.

  The UCS image does not have to be verified or loaded. (If retrieval of a UCS image from SYS1.IMAGELIB is required, IGG0191T issues the FCB image request.)
- The executor updates the entry in the WTG table with one of the following:

  IGG0197U if the UCS image must be verified.

  IGG0197E if an FCB load and/or verification is required, and UCS load and/or verification is not required.

  IGG0197F if neither a UCS nor an FCB image has to be loaded into main storage.

  IGG0191I if the printer is a 1403 and the buffer control block is specified.

  IGG0191G if the printer is a 1403 and normal scheduling is specified.

  IGG0191Q if the printer is a 1403 and chained scheduling is specified.
- It then searches the WTG table to pass control to another executor.

Stage 1 Open Executor IGG0191Y: Executor IGG0191Y receives control after executor IGG0196A or executor IGG0191N when the user–totaling option has been specified in the DCB, that is, when bit 6 of DCBOPTCD is 1.

This executor operates as follows:

- It checks to see if the DCB exit list contains an entry for user–totaling. If not, the executor turns bit 6 of DCBOFLGS on (0) so this DCB will not be opened.
- It calculates the size of the area required to save the user's totaling areas and issues a GETMAIN to obtain the space. If there is insufficient free main storage to satisfy the GETMAIN, an ABEND occurs.
- It constructs control blocks for the work area and places the address of the save area in the access method portion of the DEB. Figure 42, Section 4, describes the access method save area)
- It loads the resident save routine IGG019AX and places the ID of the save routine in the DEB and the address in the user–totaling save area.
- It specifies in the WTG table that executor IGG0191B is the next executor required. It then searches the WTG table to determine the next executor to receive control.

Stage 1 Open Executor IGG0193I: This executor receives control from IGG0191I.

The executor specifies which stage 2 executor is specified in the WTG table. The module selector table for stage 2 executors, Figure 28, should be used to determine which stage two executor is required for this DCB.

It then searches the WTG table to determine which executor receives control. The XCTL macro instruction is used to pass control to the next executor.

Stage 1 Open Executor IGG0196A: Executor IGG0196A receives control from and supplements IGG0196I.

- The executor specifies in the WTG table which module is the next one required for this DCB, as follows:

  For direct access — executor IGG0191N.

  If the device type is a printer with UCS feature and EXCP is specified — executor IGG0191T.

  If the device type is other than a printer with UCS feature and EXCP is specified — the final module of the Open routine, IGG0190S.

  If the device type is tape and the user–totaling facility is specified — executor IGG0191Y.

  If the device type is other than a printer with UCS feature or direct access, BSAM or QSAM is specified, and the user–totaling facility is not specified — executor IGG0191B.

- It then searches the WTG table to pass control to another executor.

Stage 1 Open Executor IGG0196B: Executor IGG0196B receives control from and supplements IGG0191B.

The executor operates as follows:

- DCBDIND1 is set if exchange buffering is indicated.
- For QSAM, DCBBUFNO is set to two (three for 2520 or 2540) if not previously specified.
- Executor issues ABEND macro instruction (calls problem determination module) if the buffer length is less than blocksize or if data set is for a printer and something other than output (only) is specified.
- Determines the next executor to receive control. The module selector table for stage 2 executors, Figure 28, should be used to determine which stage 2 executors are required. Some additional considerations for selecting the next executor are:

  a. For a time sharing option (TSO) task, control is transferred to IGG0196S (see *OS TSO Control Program Logic,* GY27–7199), unless buffers are wanted. The Open routine then transfers control to IGG0191I.

  b. A test is made to determine if either the 3505 (without OMR or RCE) or 3525 is being used, just prior to the XCTL subroutine. If either device is being used, control is passed to module IGG0197L; otherwise, normal processing continues.

Stage 1 Open Executor IGG0196I: Executor IGG0196I receives control from and supplements IGG0191A.

The executor operates as follows:

- It computes the main–storage requirement for the DEB and obtains the space. The space does not include the user label extent, as it is reflected in the first extent field of a format–1 DSCB for a physical sequential or direct data set. If no primary extent has been requested for an output data set, as shown by the contents of the DS1NOEPV field of the DSCB, the executor sets the DCBCIND1 field to show a volume–full condition.
- It specifies in the WTG table that executor IGG0196A is the next executor required for this DCB.
- It then searches the WTG table to pass control to another executor.

Stage 1 Open Executor IGG0197E: Executor IGG0197E locates the FCB image and, if required, loads the FCB buffer. It is entered from IGG0191T, IGG0191V, or IGG0197U.

The executor operates as follows:

- It checks the DCB exit list to see whether the specified FCB image is defined in the problem program.
- It uses the BLDL macro instruction to locate the FCB image in SYS1.IMAGELIB if the image was not defined in the problem program.
- If the image is not found in the library, the executor requests the operator to specify an alternate FCB image.
- The FCB image is loaded into the FCB buffer, if required.
- It specifies in the WTG table that IGG0197F is the next executor required for this DCB.
- It then searches the WTG table to pass control to another executor.

Stage 1 Open Executor IGG0197F: Executor IGG0197F prints a verification of the FCB image, issues an "align forms and verify" message to the operator and obtains 570 bytes for an ERP work area. It is entered from IGG0191T, IGG0191V, IGG0197E, or IGG0197U.

The executor operates as follows:

- It checks to see wither an align–forms–only or a verify–only switch is set.
- If VERIFY is specified, a verification message is sent to the printer.
- If VERIFY or ALIGN is specified, the operator is instructed to align the forms.
- It obtains 570 bytes of main storage for an ERP work area.

The executor specifies in the WTG table the next module required for this DCB, as follows:

IGG0191I if the buffer control block is specified.

IGG0191G if normal scheduling is specified.

IGG01910 if chained scheduling is specified.

The last load of Open if the DCB is EXCP.

It then searches the WTG table to pass control to another executor.

Stage 1 Open Executor IGG0197L: Executor IGG0197L receives control from IGG0196B whenever the 3505 or 3525 is specified.

The executor operates as follows:

- It initiates registers with the addresses of the DCB, UCB, ECB, and CVT.
- A test is made to determine if either OMR or RCE is being used.
- If OMR is specified, a test is made to determine if the device is a 3525. If the device is a 3525, control is transferred to IGG0197M.
- If either OMR or RCE is specified, the format descriptor record is loaded and decoded.
- After the Read–only has been executed and the format card has been translated, an OMR or RCE CCW is constructed and executed (writes the format of the device).
- It specifies in the WTG table that IGG0197M is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 1 Open Executor IGG0197M: IGG0197M receives control from IGG0197L.

The executor operates as follows:

- If an OMR or RCE format card is invalid, or if an invalid device is specified for OMR, this module issues a WTP message and an ABEND (004) with a return code of 05.
- If no invalid condition exists, the executor specifies in the WTG table the next module required for this DCB, as follows:

  IGG0191I if QSAM is specified and no buffer pool control block exists.

  IGG0197N if either BSAM or QSAM is specified and the user has specified a buffer–pool control block.

  IGG0191I if BSAM is specified and the user has specified a buffer number but not a buffer buffer–pool control block.

- It then searches the WTG table to pass control to another executor.

Stage 1 Open Executor IGG0197U: Executor IGG0197U is entered from IGG0191V to verify a UCS image. It can also issue a message requesting the operator to specify an FCB image.

The executor operates as follows:

- It prints a message verifying a UCS image load.
- It asks the operator to specify the FCB image to load, only if all of the following conditions exist:

  The printer is a 3211.

  The current FCB image is not a default.

  An FCB image was not specified in the DD statement.

  Verification of the UCS image was requested.

- The executor specifies in the WTG table the next module required for this DCB, as follows:

  IGG0197E if an FCB load and/or VERIFY is required.

  IGG0197F if the DD statement requests an FCB image which is already in the buffer and VERIFY is not specified. It is also called when FCB parameters are not in the DD statement and the buffer contains a default image.

  IGG0191I if the printer is a 1403 and the buffer control block is specified.

  IGG0191G if the printer is a 1403 and normal scheduling is specified.

  IGG0191Q if the printer is a 1403 and chained scheduling is specified.

- It then searches the WTG table to pass control to another executor.

Stage 2 Open Executors

A stage 2 Open executor establishes device–oriented information for the processing described by a DCB, and completes device–oriented control blocks or fields. One of the stage 2 executors receives control for each DCB being opened; the WTG table identifies the executor required for each DCB. On conclusion of an executor's processing it enters in the WTG table the identification of the stage 3 executor required. Figure 28 lists the access conditions that cause the different stage 2 executors to be loaded and to receive control.

The device–oriented processing performed by a stage 2 executor primarily consists of the construction of input/output blocks (IOB), their associated channel programs, and the identification of the end–of–block routine required for the processing described by the DCB. For chained channel–program scheduling, executors also construct interruption control blocks (ICB).

Figure 28 lists the access conditions that cause the different stage 2 executors to be loaded and to receive control. The executors are described in the text that follows and are in the same sequence as the list in Figure 28 under Executors.

In this figure an X in a column represents a condition that must be met for the executor to be selected. A No in a column indicates that the condition must not be specified for the executor to be selected. A blank in the upper portion of the table indicates that either the condition is not required for selection or not examined at this time. The table should be used in conjunction with the flow of control information in Diagram E, SAM Flow of Control for Open Executors, Section 5.

Stage 2 Open Executor IGG0191D: Executor IGG0191D receives control after executor IGG0196B or IGG0191I if the Open parameter list specifies:

Input or Output

and the DCB specifies:

Direct–access storage device

BSAM or QSAM and simple buffering

However update, track overflow, and chained channel–program scheduling are not specified. It may also receive control after executors IGG0191E, IGG0191F, or IGG0191K.

Access Method Options	Selections																							
BSAM or	X	X			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
QSAM	X	X	X	X	X		X		X		X	X	X	X	X		X		X	X	X	X	X	X
Input or	X	X	X						X			X								X	X	X	X	
Output	X			X													X				X		X	X	
Inout, Outin						X		X									X		X					
Update	No	No			No	No	No						X	X						X				No
Unit Record or					X										X						X	X	X	X
Magnetic Tape or			X	X	X	X									X	X								X
Paper Tape					X																			X
Direct-Access Storage	X	X	X	X				X	X			X	X	X			X	X	X				X	
Write-Load (Create-BDAM)										X	X													
Simple Buffering	X	X			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
Exchange Buffering			X	X																				
Track Overflow	No	No	No	No	No	No	X				X			X									X	No
Chained Scheduling	No	No			No	No			X						X	X	X	X						No
Search Direct		X																		X			X	
3505																					X			
3525																						X		
OMR or																								
RCE or																								
Print only and Associated Files																								
Executors																								
IGG0191D	1D																							
IGG0191E			1E																					
IGG0191F				1F																				
IGG0191G					1G	1G																		
IGG0191H							1H																	
IGG0191J								1J																
IGG0191K									1K															
IGG0191L										1L														
IGG0191M											1M													
IGG0191O												1O												
IGG0191P													1P	1P										
IGG0191Q															1Q									
IGG0191R																1R								
IGG0191S							1S																	
IGG0191W																	1W							
IGG0191X																		1X						
IGG0191Z																				1Z				
IGG0196J			6J																					
IGG0196K																								6K
IGG0196L								6L																
IGG0196P													6P	6P										
IGG0197N																					7N	7N	7N	7N
IGG0197P																							7P	7P
IGG0197Q																							7Q	7Q
IGG0199K																								9K
IGG0199L										9L														
IGG0199O		9O																						

Figure 28. Open Executor Selector — Stage 2

The executor operates as follows:

- If input is specified with search direct (OPTCD=Z), control is passed to IGG0199O where the channel program is constructed.
- If input is specified without search–direct, control is passed to IGG0191O where the channel program is constructed.
- For output data sets, the executor constructs IOBs and writes channel programs. The address of the first IOB is placed in the DCB. See Appendix B for the format of the channel program constructed by this executor.

A test of the non–rotational position sensing (RPS) indicator bit is made to see whether the channel programs utilize the RPS feature. If the bit is on (1), standard channel programs are built. If, however, the bit is not on, additional main storage is acquired to employ the RPS feature's two CCWs (Set–sector and Read–sector) in the channel programs. The two commands are incorporated where appropriate.

- If variable–length records are specified, IGG01915 is the next executor required for this DCB. Otherwise IGG01910 is specified in the WTG table as the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191E: Executor IGG0191E receives control after executor IGG0196B or IGG0193I if the Open parameter list specifies:

Input

and the DCB specifies:

Exchange buffering

Magnetic–tape or direct–access storage

(but not track overflow). The executor is loaded and gains control when its identification in the WTG table is found by another executor.

After the module determines that the executor has been entered for direct access, a test of the rotational position sensing (RPS) indicator is made to determine whether the RPS feature is to be utilized. If it is to be utilized, additional space for the channel program is acquired.

The executor operates as follows:

- If the operating mode is move, or the record format is variable–length or format–D blocked, or the record format is variable–length or format–D and the operating mode is substitute, simple buffering is substituted for exchange buffering. Therefore, it identifies in the WTG table executor IGG0191D if the device type is direct–access storage, or executor IGG0191G if the device–type is unit record, as the executor required next for this DCB. It then searches the WTG table to pass control to another executor.
- If exchange buffering can be supported, the module then calculates the amount of main storage needed to build the IOBs and issues a GETMAIN for this area. It then sets this area of main storage to zeros and puts the ID of executor IGG0196J into the WTG table as the next executor for this DCB.
- It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191F: Executor IGG0191F receives control after executor IGG0191I if the Open parameter list specifies:

Output

and the DCB specifies:

Exchange buffering

Magnetic–tape or direct–access storage

(but not track overflow). The executor is loaded and gains control when its identification in the WTG table is found by another executor.

After the module determines that the executor has been entered for direct access, a test of the rotational position sensing (RPS) indicator is made to determine whether the RPS feature is to be utilized. If it is to be utilized, additional space for the channel program is acquired.

The executor operates as follows:

- If the operating mode is move, or the record format is variable–length or format–D blocked, or the record format is variable–length or format–D and the operating mode is substitute, simple buffering is substituted for exchange buffering. Therefore, it identifies in the WTG table executor IGG0191D if the device type is direct–access storage, or IGG0191G if the device type is unit record or magnetic tape, as the executor required next for this DCB. It then searches the WTG table to pass control to another executor.
- If search–direct has been requested (OPTCD=Z in the DCB), the executor creates the desired channel programs.
- It identifies the end–of–block routine to be used in the processing specified by the DCB, obtains space for and constructs IOBs and channel programs, and links them.
- It specifies in the WTG table that executor IGG01914 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191G: Executor IGG0191G receives control after executors IGG0196B, IGG0193I, IGG0191T, IGG0191U, IGG0191V under normal conditions or from executors IGG0191E (exchange buffering not supported), IGG0191F, IGG0191R, IGG0191Q (chained scheduling not supported), under abnormal conditions if:

The DCB specifies BSAM or QSAM and either unit record, magnetic tape, or paper tape.

The Open macro parameter is INOUT or OUTIN and the DCB specifies magnetic tape.

The executor operates as follows:

- It computes the amount of main storage required for the IOBs, issues a GETMAIN macro instruction from subpool 250 and then sets the main storage for the IOBs to zeros.

- It then tests to see if the device type for this data set is unit record. If so, IGG0196K is specified in the WTG table for this DCB and the check for other DCBs that need this executor is made.
- If the device is not unit record, processing continues in this module. It constructs the channel programs in the IOBs and fills in the other fields of the IOBs. It stores the address of the first IOB in the DCB and sets the first IOB bit in the first IOB. If there is only one IOB for this data set, it sets the IOB unrelated flag.

The executor specifies in the WTG table the next executor required for this DCB. If the DCB specifies exchange buffering, the next executor is IGG01914. If the DCB specifies paper tape, the next executor is IGG01912. If the DCB specifies variable–length record format, the next executor is IGG01915. For the remaining access conditions that cause this executor to be used, the next executor is IGG01910. The executor then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191H: Stage 2 Open executor IGG0191H receives control after executor IGG0191S, if the DCB specifies:

Track overflow

(but not update). If both track overflow and update are specified, executor IGG0191P receives control.

The module checks the non–rotational position sensing (RPS) indicator and, if it is off, inserts the RPS CCWs. When RPS channel programs are built for variable record format, the SILI bit is turned on in the Read–data CCW, thereby eliminating length checking.

The executor operates as follows:

- It receives control from executor IGG0191S.
- It identifies the end–of–block routine and the direct access Note/Point routine to be used in the processing specified by this DCB.
- It specifies in the WTG table that executor IGG01913 (for IGG01916, if the DCB specifies variable–length record format) is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191J: Executor IGG0191J receives control after executor IGG0196B or IGG0193I if the Open parameter list specifies:

INOUT or OUTIN

and the DCB specifies:

Direct–access storage

The executor operates as follows:

- It calculates the amount of main storage needed to build the IOBs for the data set and then issues a GETMAIN for the required space. It then sets the area to zeros.
- In calculating the main storage area needed for the IOBs, the executor tests for non–rotational position sensing. If the indicator is off, additional space to implement the RPS CCWs in the channel programs is acquired.

- The executor then begins constructing the channel programs and filling in fields in the IOBs. It constructs the search ID equal, the TIC, the READ, and if RPS is specified, the Set–sector commands. It also includes a portion for write–check if specified.
- The executor specifies in the WTG table that executor IGG0196L is the next executor needed for this DCB and then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191K: Executor IGG0191K receives control after executor IGG0196B or IGG0193I if the DCB specifies:

Chained channel–program scheduling

Direct–access storage

It is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- If the NOTE/POINT macro instruction is used, the executor identifies direct access Note/Point module IGG019BK to be loaded for use with this DCB.
- It identifies the end–of–block routine to be loaded and used for the processing described by this DCB.
- It sets the ID number in DCBCNTRL field.
- It obtains space for and constructs one IOB, the required number of ICBs (that is, one ICB per channel program or buffer) and their associated channel programs, and then links them.
- When the module is entered for direct access, a test of the non–rotational–position–sensing (RPS) indicator (bit 2 of JFCBMASK+6) is made. If this bit is not on, additional main storage is acquired by GETMAIN to incorporate the RPS CCWs. An additional doubleword is also acquired for the sector values. When the channel programs are built, the new CCWs are inserted.
- It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191L: Executor IGG0191L receives control after executor IGG0196B or IGG0191I if the DCB specifies:

Create–BDAM (Write–Load)

The executor constructs IOBs and enters the address of the first IOB into the DCB. Then it loads the Create–BDAM Write, Check, and Channel End appendages and inserts their addresses into the DCB.

With the rotational position sensing (RPS) feature, more main storage is needed for the record–ready channel programs. This executor computes the extra bytes needed for the record–ready channel programs and issue a GETMAIN. The sector bytes are placed at the end of all the IOBs and channel programs. The last doubleword of the GETMAIN area is used for sector manipulation. The first byte is used by Set–sector, the second is used by Read Sector. The third byte is used as a byte of zero on which to issue a Set–sector command in order to position at the beginning of the track.

If track overflow is specified, the routine specifies that executor IGG0191M is the next executor required for this DCB. Otherwise, the routine specifies IGG0199L as the next executor required. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191M: Stage 2 Open executor IGG0191M constructs channel programs to write track–overflow blocks using BSAM for a data set to be later processed by BDAM. Executor IGG0191L identifies it in the WTG table as its successor executor if the DCB specifies:

Create–BDAM (Write–Load)

Track overflow

It is loaded and gains control when another executor finds its identification in the WTG table.

With the rotational position sensing (RPS) feature, more main storage is needed for the record–ready channel programs. This executor computes the extra bytes needed for the record–ready channel programs and issues a GETMAIN. The sector bytes are placed at the end of all the IOBs and channel programs. The last doubleword of the GETMAIN area is used for sector manipulation. The first byte is used by Set–sector, the second is used by Read Sector. The third byte is used as a byte of zero on which to issue a Set–sector command in order to position at the beginning of the track.

The executor operates as follows:

- It receives control after it is loaded.
- If the extents are smaller than the blocks, it passes control to the ABEND routine.
- It constructs channel programs to write the number of segments required by the size of the block.
- It specifies in the WTG table that Open executor processing is completed for this DCB. It then searches the WTG table to pass control to another executor. If the WTG table has no other entries, the executor returns control to the Open routine.

Stage 2 Open Executor IGG0191O: Executor IGG0191O receives control from IGG0191D if the Open parameter list specifies:

Input

The executor constructs Read channel programs for the IOBs constructed in IGG0191D.

The module tests the non–rotational position sensing (RPS) indicator. If the indicator is not on, IGG0191O inserts the RPS CCWs, where appropriate, in the channel program.

The Read channel program is modified for offset Read (that is, for reading a BDAM data set with VS record format and keys using BSAM READ macro instructions.)

The executor specifies in the WTG table that executor IGG01917 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191P: Stage 2 Open executor IGG0191P receives control after executors IGG0196B or IGG0193I if the Open parameter list specifies:

Update

(whether or not track overflow is also specified). It is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- It identifies module IGG019CC as the end–of–block routine to be loaded for use with the DCB.
- If the NOTE/POINT macro instruction is specified, it identifies module IGG019BK as the NOTE/POINT routine to be loaded for use with this DCB.
- If record–ready, executor IGG0191Z is specified in the WTG table. If non–record–ready, executor IGG0196P is specified in the WTG table. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191Q: Executor IGG0191Q gains control after executors IGG0196B, IGG0191T, IGG0191U, IGG0191V, or IGG0191I if the DCB specifies:

Chained channel–program scheduling

Unit record, magnetic tape

It is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- If the DCB specifies the CNTRL macro instruction, this executor identifies executor IGG0191G in the WTG table as the next executor to receive control for this DCB. It then searches the WTG table to pass control to another executor.
- If the NOTE/POINT macro instruction is specified and the device is magnetic tape, it identifies module IGG019BL to be loaded for use with the DCB.
- If the NOTE/POINT macro instruction is specified, and the device is unit record, it identifies dummy data set module IGG019AV to be loaded and used in place of Note/Point.
- It identifies the end–of–block routine to be loaded and used for the processing described by this DCB.
- It obtains space for and constructs one IOB, the required number of ICBs (one per buffer or channel program) and channel programs appropriate to the device, and links them.
- It specifies in the WTG table that executor IGG01913 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191R: Open executor IGG0191R receives control after executors IGG0196B or IGG0191I if the Open parameter list specifies:

INOUT, or OUTIN

and the DCB specifies:

Chained channel–program scheduling

The executor is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- If the device is direct–access storage, it identifies Note/Point module IGG019BK to be loaded for use with the DCB.
- If the device is magnetic tape, it identifies Note/Point module IGG019BL to be loaded for use with the DCB.
- It identifies the end–of–block routine to be loaded for use with the DCB.
- It obtains space for and constructs one IOB, the required number of ICBs (one per buffer or channel program) and channel programs for direct–access storage or magnetic tape, and links them.
- It specifies in the WTG table that executor IGG01913 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191S: Stage 2 Open executor IGG0191S receives control after executor IGG0196B or IGG0191I if the DCB specifies:

Track overflow

(but not update). The executor is loaded and gains control when another executor finds its identification in the WTG table.

The module obtains and clears the space for IOBs and channel programs for the maximum number of segments possible. It also tests the non–rotational position sensing (RPS) indicator bit and, if it is off, increases the space acquired to facilitate implementation of the two RPS CCWs. If the non–RPS indicator is on, space for standard channel programs is acquired.

The executor operates as follows:

- It identifies the end–of–block routine and the direct–access NOTE/POINT routine to be used in processing specified by this DCB.
- It obtains space for and constructs IOBs and channel programs for the maximum number of segments possible. It links the channel programs to the IOBs and the IOBs to one another.
- If search–direct has been requested (OPTCD=Z in the DCB), the executor specifies IGG0199K as the next executor required for this DCB.
- It specifies in the WTG table that executor IGG0191H is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191W: Executor IGG0191W receives control after executor IGG0191B or IGG0191I if the DCB specifies:

Chained channel–program scheduling

Direct–access storage

Output

It is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- It identifies the end–of–block routine to be loaded and used for the processing described by this DCB.
- It obtains space for and constructs one IOB, the required number of ICBs (that is, one ICB per channel program or buffer) and their associated channel programs, and then links them. If the non–rotational position sensing (RPS) bit is off, the space for the IOB and ICB is increased to incorporate the RPS CCWs and the space is inserted.
- It specifies in the WTG table that executor IGG01913 is the next executor required for the DCB. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191X: Executor IGG0191X receives control after executors IGG0191B or IGG0191I if the Open parameter list specifies:

INOUT or OUTIN

and the DCB specifies:

Chained scheduling

Direct–access storage

The executor is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- It identifies the end–of–block routine to be loaded for use with the DCB.
- It obtains space for and constructs one IOB, the required number of ICBs (one per buffer or channel program) and channel programs for direct–access storage and links them. If the rotational position sensing (RPS) indicator is off, the space acquired for the IOB is incremented to incorporate the RPS CCWs, which will then be inserted in the channel program.
- It specifies in the WTG table that executor IGG01913 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0191Z: Executor IGG0191Z receives control after executor IGG0191P, if the Open parameter list specifies:

Update

Record–ready channel programs are to be generated

The executor operates as follows:

- It constructs IOBs and channel programs to empty and refill each buffer.
- For QSAM, the executor links the channel programs so that a buffer may be either refilled only (by executing only the second half of the channel program) or emptied and refilled (by executing the channel program from the beginning).
- It specifies in the WTG table that executor IGG01912 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0196J: This executor receives control after executor IGG0191E has obtained the necessary main storage to construct the IOBs.

The executor operates as follows:

- If search–direct has been requested (OPTCD=Z in the DCB), the executor creates the desired channel programs.
- It identifies the end–of–block routine to be used in the processing specified by the DCB, obtains space for and constructs IOBs and channel programs, and links them.
- If the device is direct–access storage, the executor copies the starting Seek address from the DCB into the IOB.
- It specifies in the WTG table that executor IGG01914 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0196K: Executor IGG0196K receives control if executor IGG0191G determines that the device type is unit record.

It performs the same functions (except for GETMAIN) as executor IGG0191G.

The executor specifies in the WTG table the next executor required for this DCB. If the DCB specifies exchange buffering, the next executor is IGG01914. If the DCB specifies variable–length record format, the next executor is IGG01915. For the remaining access conditions that cause this executor to be used, the next executor is IGG01910.

The executor then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0196L: The executor receives control from executor IGG0191J to finish building the IOBs assembled in IGG0191J.

The executor operates as follows:

- Starting at the end of the last CCW constructed by IGG0191J, it completes the building of the channel programs. Appendix B (Section 6), BSAM/QSAM Channel Programs, shows the channel program constructed by this executor and executor IGG0191J.

- The executor specifies in the WTG table that executor IGG01910 (or IGG01915, if the DCB specifies variable–length record format, is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0196P: Executor IGG0196P receives control after executor IGG0191P, if the OPEN parameter list specifies:

Update

Non–record–ready channel programs are to be generated

The executor operates as follows:

- It constructs IOBs and channel programs to empty and refill each buffer.
- For QSAM, the executor links the channel programs so that a buffer may be either refilled only (by executing only the second half of the channel program) or emptied and refilled (by executing the channel program from the beginning).
- With the rotational position sensing (RPS) feature, more main storage is needed for record–ready channel programs. The executor computes the extra bytes needed for these channel programs and issues a GETMAIN. The sectors are placed at the end of all the IOBs and channel programs. When the main storage requirement for all IOBs and channel programs has been computed, 2 bytes for the Read segment of the channel program and 1 byte for the Write segment of each channel program are added to the byte count and then rounded up to a multiple of 8. The first two bytes following all the IOBs are used by the Read segments. The first byte is used by Set–sector, and the second is used by Read–sector. A byte from which to use Set–sector is needed by the Write segment of each channel program. When a record is read, the sector value for the search for that record must be stored before the next channel program is executed before it will be overlaid by the next sector value.
- If record area is present (which indicates that the record format is variable–length spanned), it specifies in the WTG table that executor IGG01915 is the next executor required for this DCB. Otherwise, it specifies executor IGG01912. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0197N: Executor IGG0197N receives control from IGG0193I whenever the 3505 or 3525 is specified, or from IGG0197M whenever the same devices are specified and a buffer pool is not needed.

The executor operates as follows:

- It makes a test to determine if the FUNC parameter is being used.
- If the FUNC parameter is not being used, and if the file is for Read only (without OMR or RCE) or Punch only, IGG0191G is specified in the WTG table as the next executor required for this DCB.
- If the FUNC parameter specifies print only or associated files, IGG0197P is specified in the WTG table as the next executor required for this DCB.
- If a specified parameter combination is found to be invalid, a message to the programmer (WTP) is issued along with a subsequent ABEND (004).

- If the FUNC parameter is not being used, but the file is a Read only with OMR or RCE, IGG0197P is specified in the WTG table as the next executor required for this DCB.
- Once the validity of the FUNC parameter is established, the DCBMACRF field is tested to determine if the CNTRL is valid for an input data set. If it is not valid, a WTP message and an ABEND macro (004) with a return code of 02 are issued.
- If the CNTRL specification is valid, a test is made to determine if the associated DCBs specify the same access methods.
- If the access methods are not the same, a message is written to the programmer along with a subsequent ABEND (004).
- It specifies in the WTG table that IGG0197P or IGG0191G is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0197P: IGG0197P receives control from IGG0197N if neither Read only (without OMR or RCE) nor Punch only is specified for the 3505 or 3525.

The executor operates as follows:

- It builds the IOB and CCWs and appends a work area to the IOB, according to the type of data set that is specified.
- It specifies in the WTG table that IGG0197Q is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0197Q: IGG0197Q receives control from IGG0197P.

The executor operates as follows:

- A test is made to determine if data protection image (DPI) is specified in the FUNC parameter.
- If DPI is specified, SVC 105 is issued. This places the address of IMAGELIB in register one.
- Both a BLDL and a LOAD macro are issued so that the DPI image can be built and the image address can be loaded in register zero.
- The address is saved for the image deletion (after the image has been copied into IOB+64) by the DELETE macro.
- If DPI is not specified, tests are made to determine which EOB and/or control module ID is to be entered in the DCB. (The same tests are made if DPI is specified.)
- It specifies in the WTG table that IGG01910 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0199K: Executor

IGG0199K receives control after executor IGG0191S.

The executor operates as follows:

- It obtains space for and constructs the IOB and channel programs for direct–access devices with the search–direct feature (OPTCD=Z).

- Sets all known flags and completes related fields in DCB.
- It specifies in the WTG table that IGG01916 is the next executor required for the DCB if variable-length records are specified. Otherwise, IGG01913 is specified in the WTG table.
- It then searches the WTG table to pass control to another executor.

Stage 2 Open Executor IGG0199L: Executor Executor IGG0199L receives control after executor IGG0191L if the DCB specifies:

Create–BDAM (Write–Load)

The executor constructs channel programs. When the DCB specifies RECFM=VS and BFTEK=R, the routine constructs a segment work area for spanned record processing and creates an IRB for the asynchronous exit routine, which executes writing of the successive segments. It then searches the WTG table to pass control to another executor. If the WTG table has no other entries, the executor returns control to the Open routine.

With the rotational position sensing (RPS) feature, more main storage is needed for the record–ready channel programs. This executor computes the extra bytes needed for the record–ready channel programs and issues a GETMAIN. The sector bytes are placed at the end of all the IOBs and channel programs. The last doubleword of the GETMAIN area is used for sector manipulation. The first byte is used by Set–sector, the second is used by Read–sector. The third byte is used as a byte of zero on which to issue a Set–sector command in order to position at the beginning of the track.

If search–direct has been requested (OPTCD=Z), the executor creates the desired channel programs.

Note: A user may provide a segment work area by setting a bit in the DCBMACRF field and placing the address of that area in the DCBEOB field. In this case, this routine will not construct the segment work area.

Stage 2 Open Executor IGG0199O: Executor IGG0199O receives control from executor IGG01991D if the Open parameter list specifies:

Input

and the DCB specifies:

OPTCD = Z (search–direct)

The executor operates as follows:

- The executor constructs the IOBs and channel programs required when search direct is specified. The format of the channel programs constructed by this executor are shown in Appendix B under "BSAM/QSAM Channel Programs."
- If format–F or –U records are specified, IGG01910 required for this DCB. Otherwise, (for format–V) executor IGG01915 is specified in the WTG table as the next executor required for this DCB.
- It then searches the WTG table to pass control to another executor.

Stage 3 Open Executors

A stage 3 executor identifies and loads the modules needed to perform the processing described by the DCB. If QSAM is used, and an input data set is to be processed, a second stage 3 executor also primes the buffers.

The stage 3 Open executors are altered to load in the fixed standard end of extent modules and the format–U channel end module when the rotational position sensing (RPS) feature is used.

Figure 29 lists the access conditions that cause the different stage 3 executors to be loaded and to gain control. The executors are described in the text that follows in a sequence identical to the list under "Executors" in Figure 29.

In this table an X in a column represents a condition that must be satisfied before the executor is selected. A blank in the upper portion of the table indicates that either the condition is not required for selection or not examined at this time. The table should be used in conjunction with the flow of control information in Diagram E, SAM Flow of Control for Open Executors, in Section 5.

Stage 3 Open Executor IGG01910: Executor IGG01910 receives control after executor IGG0191D, IGG0199O or IGG0191J. It also receives control after executor IGG0191G unless the DCB specifies paper tape.

This executor operates as follows:

- It receives control after it is loaded.
- It identifies, loads, and puts the address into the DCB of:

  A Get or Put routine

  A synchronizing routine
- It enters into the DEBSUBID field of the DEB the identification of each routine loaded.
- It specifies executor IGG01917 in the WTG table as the next executor to receive control for this DCB.

Stage 3 Open Executor IGG01911: Executor IGG01911 is entered from executors IGG01917, IGG01918, IGG01919, IGG01990, IGG01991, and IGG01992 if the DCB specifies:

Get or Put

This executor operates as follows:

- It completes any remaining DCB fields.
- It completes the IOBs.
- For input it issues a BALR instruction to pass control to the end–of–block routine identified by a stage 2 executor and loaded by one of the other stage 3 executors. The end–of–block routine issues an EXCP macro instruction to prime the buffers.
- For output it sets a flag, which is used to identify the first entry, into the Put routine.
- It searches the WTG table to pass control to another executor. If the WTG table has no other entries, the executor returns control to the Open routine.

Access Method Options	Selection										
Paper Tape	X										
Update		X									
Chained Scheduling			X								
Exchange Buffering				X							
Track Overflow					X					X	
None of the preceding						X					
Input							X				
QSAM							X				
Variable–length Record Format								X	X		
Spanned Records											X
Executors											
IGG01910						10					
IGG01911							11				
IGG01912	12	12									
IGG01913			13		13						
IGG01914				14							
IGG01915								15			15
IGG01916									16		
IGG01917						17					
IGG01918	18	18									
IGG01919			19		19						19
IGG01926			26		26						26
IGG01990				90							
IGG01991								91			
IGG01992									92		
IGG01993								93			
IGG01994											94

Figure 29. Open Executor Selector — Stage 3

Stage 3 Open Executor IGG01912: Executor IGG01912 is entered after executor IGG0191P and also from executor IGG0191G if the Open parameter is:

Update

or if the DCB specifies:

Paper tape

The executor operates as follows:

- It identifies and loads the device–independent routines.
- If rotational position sensing (RPS) channel programs are constructed and the record format is fixed, the format–F channel–end appendage will always be loaded. For RPS with format–U without track overflow, a format–U channel–end appendage is loaded.
- It loads the device–dependent routines.
- It enters the addresses of the routines into the DCB, and the address of the paper tape appendage into the appendage vector table.
- It specifies executor IGG01918 in the WTG table as the executor to receive control next for this DCB.

Stage 3 Open Executor IGG01913: Executor IGG01913 receives control after executors IGG0191H, IGG0191K, IGG0191Q, and IGG0191R if the DCB specifies:

Chained channel–program scheduling, or track overflow.

It is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- If QSAM is specified, it identifies, loads, and places the address into the DCB of:

  A Get or a Put routine

  A synchronizing routine
- If BSAM is specified, it identifies, loads, and places the address into the DCB of:

  A Read or Write routine

  A Check routine
- It specifies in the WTG table that Open executor IGG01919 is to receive control next for this DCB.

Stage 3 Open Executor IGG01914: Executor IGG01914 receives control after executors IGG0191E, IGG0191F, and IGG0191G, if the DCB specifies:

Exchange buffering.

It is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- It specifies in the WTG table that executor IGG01910 is required for this DCB if the access conditions specified are:

  Output and locate, or

  Input and move, or

  Input, locate, and variable–length, or

  format–D.

- It then searches the WTG table to pass control to another executor.
- It identifies, loads, and puts the address into the DCB of:

  A Get or Put routine

  A synchronizing routine

  and specifies executor IGG01990 in the WTG table as the executor to receive control next for this DCB.
- It searches the WTG table to pass control to another executor.

Stage 3 Open Executor IGG01915: Executor IGG01915 receives control after executors IGG0191D, IGG01910, IGG0191G, IGG0196K, and IGG0199O, if the DCB specifies:

Variable–length record format

Executor IGG01915 receives control from executor IGG0196P or IGG0191Z, if the DCB specifies:

Variable–length spanned record format

The executor operates as follows:

- If QSAM is specified, the executor identifies and loads a Get or Put routine and a synchronizing routine.
- If BSAM is specified, the executor identifies and loads a Read or Write routine, a Check routine, and a routine to service the NOTE/POINT macro instruction if it is specified.
- It places the identifiers (IDs) of the routine loaded into the DEB subroutine ID field and the addresses of the routines into the DCB.
- For a 3211 printer:

  An abnormal–end appendage is loaded and its address is placed in the appendage vector table.

  An asynchronous error routine is loaded. The IRB used for scheduling this routine is built and the IRB address placed in the DEB.
- It specifies in the WTG table that executor IGG01991 is the next executor required for this DCB.
- It searches the WTG table to determine to which executor it should pass control.

Stage 3 Open Executor IGG01916: Executor IGG01916 receives control after executors IGG0191H, IGG0191K, IGG0191Q, and IGG0191R if the DCB specifies:

Variable–length record format

Track overflow

The executor operates as follows:

- If QSAM is specified, the executor identifies and loads a Get or Put routine and a synchronizing routine.

- If BSAM is specified, the executor identifies and loads a Read or Write routine, a Check routine, and a routine to service the NOTE/POINT macro instruction if it is specified.
- It places the IDs of the routine, loaded into the DEB subroutine ID field and the addresses of the routines into the DCB.
- It specifies in the WTG table that executor IGG01992 is the next executor required for this DCB.
- It searches the WTG table to determine to which executor it should pass control.

Stage 3 Open Executor IGG01917: Executor IGG01917 is entered after executor IGG01910 or IGG0191O. It is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- It identifies and loads all the appendages required and places their addresses into the appendage vector table.
- If rotational position sensing (RPS) channel programs are constructed and the record format is fixed, the format–F channel–end appendage is always loaded. It loads in the fixed standard end–of–extent module IGG019C4 where the fixed standard record format is used.
- For PRS with format–U without track–overflow, a format–U channel–end appendage is loaded.
- It loads the end–of–block routine identified by a stage 2 executor and places its address into the DCB.
- If search–direct has been requested (OPTCD=Z in the DCB), the executor loads in the necessary appendages.
- It enters into the DEBSUBID field of the DEB the identification of each routine loaded.
- If QSAM is used, the executor specifies in the WTG table that executor IGG01911 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.
- If BSAM is used, the executor specifies in the WTG table that Open executor processing is completed for this DCB. It then searches the WTG table to pass control to another executor. If the WTG table has no other entries, the executor returns control to the Open routine.

Stage 3 Open Executor IGG01918: Executor IGG01918 is entered after executor IGG01912. It is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- It loads the end–of–block routine identified by a stage 2 executor and places its address into the DCB.

- It enters into the DEBSUBID field of the DEB the identification of each routine loaded.
- If QSAM is used, the executor specifies in the WTG table that executor IGG01911 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.
- If BSAM is used, the executor specifies in the WTG table that Open executor processing is completed for this DCB. It then searches the WTG table to pass control to another executor. If the WTG table has no other entries, the executor returns control to the Open routine.

Stage 3 Open Executor IGG01919: Executor IGG01919 is entered after IGG01913. It is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- It identifies and loads all the appendages required and places their addresses into the appendage vector table.
- If rotational position sensing (RPS) channel programs are constructed and the record format is fixed, the format–F channel–end appendage is always loaded. For RPS with format–U without track overflow, a format–U channel–end appendage is loaded.
- If search–direct has been requested (OPTCD=Z in the DCB), the executor loads in the necessary appendages.
- It enters into the DEBSUBID field of the DEB the identification of each routine loaded. If the WTG table has no other entries, the executor returns control to the Open routine.
- It specifies in the WTG table that executor IGG01926 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Stage 3 Open Executor IGG01926: Executor IGG01926 is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- It loads the end–of–block routine identified by a stage 2 executor and places its address into the DCB.
- It enters into the DEBSUBID field of the DEB the identification of each routine loaded.
- If QSAM is used, the executor specifies in the WTG table that executor IGG01911 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.
- If BSAM is used, the executor specifies in the WTG table that Open executor processing is completed for this DCB. It then searches the WTG table to pass control to another executor. If the WTG table has no other entries, the executor returns control to the Open routine.

Stage 3 Open Executor IGG01990: Executor IGG01990 is entered after executor IGG01914 if the DCB specifies:

Exchange buffering

It is loaded and receives control when another executor finds its identification in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- It identifies and loads all of the appendages required and places their addresses into the appendage vector table.
- If rotational position sensing (RPS) channel programs are constructed and the record format is fixed, the format–F channel–end appendage will always be loaded. It loads in the fixed standard end–of–extent module IGG019C4 where the fixed standard record format is used.
- For RPS with format–U without track overflow, a format–U channel–end appendage is loaded.
- It loads the end–of–block routine identified by a stage 2 executor and places its address into the DCB.
- If search–direct has been requested (OPTCD=Z in the DCB), the executor loads in the necessary appendages.
- It enters into the DEBSUBID field of the DEB the identification of each routine loaded.
- If QSAM is used, the executor specifies in the WTG table that executor IGG01911 is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.
- If BSAM is used, the executor specifies in the WTG table that Open executor processing is completed for this DCB. It then searches the WTG table to pass control to another executor. If the WTG table has no other entries, the executor returns control to the Open routine. Executor IGG01991 receives control after, and is a continuation of, executor IGG01915. It completes the loading of subroutines for a DCB which specifies:

  Variable–length or record format–D

Stage 3 Open Executor IGG01991: The executor operates as follows:

- It identifies and loads all the appendages required and places their addresses into the appendage vector table.
- If rotational position sensing (RPS) channel programs are constructed and the record format is variable, the format–V channel–end appendage is always loaded. It loads in the variable standard end–of–extent module, IGG019C4, where the variable record format is used.
- For RPS with format–U without track overflow, a format–U channel–end appendage is loaded.

- It loads the end–of–block routine identified by the stage 2 executor and places its address into the DCB.
- If search direct has been requested (OPTCD=Z in the DCB), the executor loads in the necessary appendages.
- It enters the IDs of the routines loaded into the DEB subroutine ID field.
- The executor specifies in the WTG table that IGG01993 is the next executor required for this DCB. It then searches the WTG table to determine the next executor to receive control.

Stage 3 Open Executor IGG01992: Executor Executor IGG01992 receives control after, and is a continuation of, executor IGG01916. The executor loads subroutines for a DCB which specifies:

Variable–length record format

Track overflow

The executor operates as follows:

- It identifies and loads all the appendages required and places their addresses into the appendage vector table.
- If rotational position sensing (RPS) channel programs are constructed and the record format is variable, the format–V channel end appendage is always loaded. It loads in the variable standard end–of–extent module, IGG019C4, where the variable–record format is used.
- For RPS with format–U without track overflow, a format U channel end appendage is loaded.
- If search–direct has been requested (OPTCD=Z in the DCB), the executor loads in the necessary appendages.
- It enters the IDs of the routines loaded into the DEB subroutine ID field.
- It specifies in the WTG table that executor IGG01994 is the next executor required for this DCB. It then searches the WTG table to determine the next executor to receive control. If there are no other entries in the WTG table, the executor returns control to the Open routine.

Stage 3 Open Executor IGG01993: Executor IGG01993 is a continuation of executor IGG01991. It completes the process of loading subroutines for a DCB that specifies:

Variable–length record format

The executor operates as follows:

- It identifies and loads all of the appendages required and places their addresses into the appendage vector table.
- If rotational position sensing (RPS) channel programs are constructed and the record format is variable, the format–V channel–end appendage is always loaded. It loads in the variable standard end–of–extent module, IGG019C4, where the variable–record format is used.
- It loads the end–of–block routine identified by the stage 2 executor and places its address into the DCB.

- It enters the IDs of the routines loaded into the DEB subroutine ID field.
- If QSAM is used, the executor specifies in the WTG table that executor IGG01911 is the next executor required for this DCB. It then searches the WTG table to determine the next executor to receive control.
- If BSAM is used, the executor specifies that Open executor processing is completed for this DCB. It then searches the WTG table to determine the next executor to receive control. If there are no other entries in the WTG table, the executor returns control to the Open routine.
- If search–direct has been requested (OPTCD=Z in the DCB), the executor loads in the necessary appendages.

Stage 3 Open Executor IGG01994: Executor IGG01994 is loaded and receives control when another executor finds its identification in the WTG table. It completes the process of loading subroutines for a DCB that specifies:

Variable–length record format

Track overflow

The executor operates as follows:

- It loads the end–of–block routine identified by the stage 2 executor and places its address into the DCB.
- It enters the IDs of the routines loaded into the DEB subroutine ID field.
- If QSAM is used, the executor specifies in the WTG table that executor IGG01911 is the next executor required for this DCB. It then searches the WTG table to determine the next executor to receive control.
- If BSAM is used, the executor specifies that Open executor processing is completed for this DCB. It then searches the WTG table to determine the next executor to receive control. If there are no other entries in the WTG table, the executor returns control to the Open routine.

Close Executors

There are seven Close executors. IGG0201A or IGG0201Z receives control if one of the sequential access methods is used. Control goes to IGG0201A if the device type is tape or unit record. Executor IGG0201X is an extension of IGG0201A. If the device type is direct–access storage, control is passed to IGG0201Z. Executor IGG0201B receives control after executors IGG0201A or IGG0201Z if QSAM was used with an output data set and a channel program encountered an error condition while one of the other Close executors had CPU control. Executor IGG0201P receives control from IGG0201A whenever the 3525 or the 3505 is specified. Executor IGG0201R is an extension of IGG0201P.

Control returns to the Close routine of I/O support when Close executor processing is completed.

Figure 30 shows the conditions that cause the Close executors to gain control.

Access Method Options	Selection					
Tape or unit record	X	X				
Direct–access storage			X	X		
Permanent error or end–of–volume condition when using QSAM for output		X		X		X
3505 (OMR/RCE) or 3525					X	X
Executors						
IGG0201A	1A	1A			1A	1A
IGG0201B		1B		1B		1B
IGG0201P					1P	1P
IGG0201R					1R	1R
IGG0201X	1X	1X				
IGG0201Y			1Y	1Y		
IGG0201Z			1Z	1Z		

Figure 30. Close Executor Selector

Close Executor IGG0201A: Executor IGG0201A receives control from the Close routine of I/O support if the DCBDSORG field specifies a value of PS and if the device type is tape or unit record.

The executor operates as follows:

- It receives control after it is loaded.
- If the Open parameter is output and the DCB specifies a Put operation, the executor issues a TRUNC and a PUT macro instruction to cause scheduling of the last buffer. On return of control, the executor awaits execution of the last channel program.
- If all channel programs were executed without encountering either an end–of–volume condition or a permanent error, the executor continues processing.
- If any of the preceding channel programs encountered either a permanent error or an end–of–volume condition, the executor specifies in the WTG table that executor IGG0201B is required for this DCB. Depending on the remaining entries in the WTG table, it then either processes another DCB, or passes control to executor IGG0201B.
- If the 3525 or the 3505 with either OMR or RCE is specified, the executor specifies in the WTG table that executor IGG0201P is required for this DCB.
- If neither Output nor PUT is specified, the executor issues a PURGE macro instruction for any pending channel programs. Note that when processing under BSAM, the Check routine ensures execution of all channel programs.
- It then searches the WTG table to pass control to another executor.

Close Executor IGG0201B (Error Processing): Executor IGG0201B receives control after either executor IGG0201A or IGG0201Z if one of the latter finds that a channel program for an output data set using QSAM encountered a permanent error or an end–of–volume condition. It is loaded and receives control when its identification is found in the WTG table.

The executor operates as follows:

- It receives control after it is loaded.
- It determines whether a channel program encountered a permanent error or an end–of–volume condition.
- If a channel program encountered a permanent error, the executor performs its remaining processing. Any buffers not written out are not processed.
- If a channel program encountered an end–of–volume condition, the executor finds the IOB associated with that channel program and places its address into the DCBIOBA field. It then passes control to the output synchronizing routine for normal processing of the end–of–volume condition. When control returns, the executor performs its remaining processing, unless one of the channel programs encountered a permanent error or another end–of–volume condition. In either of those cases, it resumes processing as it did when it first received control.
- If Output and either a DCBDSORG field value of PP, or WRITE or PUT with a DD statement of the form (MEMBERNAME) are specified, the executor issues a STOW macro instruction. On completion of the Stow routine, the executor tests for I/O errors and for logical errors, such as insufficient space in the directory. For either type of error, the executor issues an ABEND macro instruction.
- The executor specifies in the WTG table that Close executor processing is completed for this DCB. Depending on the remaining entries in the WTG table, the executor either processes another DCB or returns control to the executor IGG0201Y.

Close Executor IGG0201P: This module receives control from IGG0201A whenever:

The 3525 is specified or the 3505 is specified with either OMR or RCE.

The module operates as follows:

- It turns on the Close flag.
- Tests are made to determine if either OMR or RCE is being used with the 3505.
- If either is being used, the module issues a Feed and Stacker–select command (with the OMR/RCE flag bit off) to return the device to normal punched mode.

- If either an associated data set or PRINT is being used with the 3525, the following apply:

File Type	Feed Caused by Close of
Print	Print File
Read/Print	Print File*
Read/Punch/Print	Print File**
Read/Punch	Punch File**
Punch/Print	Print File
Punch/Interpret	Punch File
Read	Read File
Punch	Punch File

*A feed is executed if an end–of–file is caused by the hardware; a feed is not executed if it is caused by a data delimiter card.

**Punching or printing delimiter cards is not allowed for these file types since the Close routine always issues a feed command.

- If a channel program for an output (QSAM) data set encountered a permanent error, IGG0201B is specified in the WTG table as the next executor required for this DCB. Otherwise, executor IGG0201R is specified in the WTG table.

  It then searches the WTG table to pass control to another executor.

Close Executor IGG0201R: This module receives control from IGG00201P.

The module operates as follows:

- It frees buffer space from the buffer pool.
- It also frees IOB and ICB space.
- It clears BSAM and QSAM vectors in the DCB.
- It specifies in the WTG table that executor IGG0201B is the next executor required for this DCB. It then searches the WTG table to pass control to another executor.

Close Executor IGG0201X: Executor IGG0201X is a continuation of executor IGG0201A and receives control from that executor.

The executor operates as follows:

- If QSAM and simple buffering are specified, the executor returns the buffers associated with the DCB to the buffer control block pointed to by the address in the DCBBUFCB field.
- The executor computes the amount of space occupied by the channel programs, IOBs (and ICBs, if chained scheduling is used), and returns that space to the supervisor by using a FREEMAIN macro instruction.
- The executor specifies in the WTG table that Close executor processing is completed for this DCB. Depending on the remaining entries in the WTG table, it then either processes another DCB, passes control to executor IGG0201B, or returns control to the Close routines.

Close Executor IGG0201Y: Executor IGG0201Y receives control from executor IGG0201Z.

The executor operates as follows:

- After it is loaded, it receives control.
- When record–ready channel programs are constructed, a GETMAIN macro instruction is issued for more bytes. In the Close routine, when the IOB and channel program areas are freed, the number of additional bytes is computed and added to the byte count before issuing the FREEMAIN macro instruction.
- It purges the segment work area for a DCB that specifies BFTEK=R, RECFM=VS, and MACRF=WL.
- It frees the record area obtained by an Open operation when a DCB specifies BFTEK=A, spanned record, and QSAM locate mode.

Close Executor IGG0201Z: Executor IGG0201Z receives control from the Close routine of O/C/EOV if the DCBDSORG field specifies a value of PS or PO and if device type is direct–access storage.

The executor operates as follows:

- It receives control after it is loaded.
- If the Open parameter is Output and the DCB specifies a Put operation, the executor issues a TRUNC and a PUT macro instruction to cause scheduling of the last buffer. On return of control, the executor awaits execution of the last channel program.
- If all channel programs were executed without encountering either an end–of–volume condition or a permanent error, the executor continues processing.
- If any of the preceding channel programs encountered either a permanent error or an end–of–volume condition, the executor specifies in the WTG table that executor IGG0201B is required for this DCB. Depending on the remaining entries in the WTG table, it then either processes another DCB, or passes control to executor IGG0201B.
- If neither Output nor a Put operation is specified, the executor issues a PURGE macro instruction for any pending channel programs. Note that, when processing under BSAM, the Check routine ensures execution of all channel programs.
- If Output and either a DCBDSORG field value of PO, or WRITE or PUT with a DD statement of the form (MEMBERNAME) are specified, the executor issues a STOW macro instruction. On completion of the Stow routine, the executor tests for I/O errors and for logical errors, such as insufficient space in the directory. For either type of error, the executor issues an ABEND macro instruction.
- If QSAM and simple buffering are specified, the executor returns the buffers associated with the DCB to the buffer control block pointed to by the address in the field DCBBUFCB.
- The executor computes the amount of space occupied by the channel programs and IOBs (and ICBs, if chained scheduling is used), and returns that space to the supervisor by using a FREEMAIN macro instruction.
- The executor specifies in the WTG table that Close executor processing is completed for this DCB. Depending on the remaining entries in the WTG table, it then either processes another DCB, passes control to executor IGG0201B, or returns control to the Close routines.

SYNAD/FEOV/EOV Executors

The executors listed in Figure 31 do error processing and rescheduling of channel programs as required by BSAM and QSAM routines.

Type	Module Number	SVC	Function
SYNAD/EOV	IGG0005E	55	Performs error processing for EOV condition.
SYNAD/Diagnostic	IFG0551B		Performs error processing for permanent errors.
FEOV	IGC0003A	31	Discontinues reading and writing on present volume.
EOV/New Volume	IFG0551L		Schedules channel program to be executed with new volume.
EOV/New Volume	IFG0551N		Performs rescheduling of channel programs when chained scheduling is used.

Figure 31. SAM EOV, FEOV, and Error Processing Executors.

SYNAD/EOV Executor IGC0005E (SVC 55): Executor IGC0005E performs error–condition processing. If a synchronizing and error routine in QSAM, or a Check routine in BSAM, finds that the execution of a channel program encountered either a permanent error or an end–of–volume (EOV) condition, the routine issues an SVC 55 instruction. (The update synchronizing–and–error–processing routine passes control to this executor only for an end–of–volume condition; the paper–tape–synchronizing–and–error–processing routine never passes control to this executor.) An SVC 55 instruction causes this executor to be loaded and to receive control.

Control passes to and from this executor along three paths, depending on whether control was received due to an EOV condition, because of a permanent error condition and there is a SYNAD routine present, or because of a permanent error condition and there is no SYNAD routine present.

The flow of control under these three conditions for QSAM is shown in Diagram H (Section 5). For BSAM, the same conditions are shown in Diagram I.

For an EOV condition, the executor operates as follows:

- It obtains a work area.
- It passes control to the end–of–volume routine of O/C/EOV. If that routine finds a new volume, it eventually passes control to EOV/new volume executor. After processing, the executor returns control to the synchronizing–and–error–processing routine or to the Check routine.

If there is no SYNAD routine present, the executor operates as follows for a permanent error condition:

- For QSAM, the executor passes control to the SYNAD/diagnostic executor, IFG0551B.
- For BSAM, the executor passes control to the ABEND routine.

If there is a SYNAD routine present, the executor operates as follows for a permanent error condition:

- For QSAM, the executor passes control to the SYNAD/diagnostic executor (IFG0551B) after setting error indicators for the SYNAD routine. After error processing, the user's SYNAD routine may return control to the synchronizing routine. The synchronizing routine issues a second SVC 55 instruction to pass control to this executor.
- For BSAM, the executor returns control to the Check routine. The Check routine passes control to the user's SYNAD routine. A return of control from the SYNAD routine to the Check routine in BSAM is interpreted as an Accept–error option. The Check routine issues a second SVC 55 instruction to pass control to this executor again.
- For BSAM, the executor implements the Accept–error option and returns control to the check routine. For chained channel–program scheduling or purged IOBs, control passes to the SYNAD/diagnostic executor, IFG0551B.

SYNAD/Diagnostic Executor IFG0551B: Executor IFG0551B performs error processing for the SYNAD/EOV executor (IGC0005E) when a permanent error is detected.

For QSAM, if there is a SYNAD routine specified, IFG0551B returns control to the error synchronizing routine which passes control to the user's SYNAD routine.

For QSAM, if there is no SYNAD routine or if there is a SYNAD routine and it returns to the error synchronizing routine, IFG0551B implements the error options specified in the DCBEROPT field in the DCB. It returns control to the synchronizing routine for the Skip or Accept option if the error is one the user can accept.

For BSAM, if the SYNAD routine returns to the Check routine (this implies Accept) and for chained–channel–program scheduling and purged IOB's, the executor reschedules the channel programs.

The executor implements error options in the following manner:

- For the Terminate–error option, the executor passes control to the ABEND routine.
- For the Accept–error option, the executor issues EXCP macro instructions to reschedule all channel programs except the one executed with an error. If the device is a printer, all channel programs are rescheduled.
- For the Skip–error option, the executor issues EXCP macro instructions to reschedule all channel programs, including the one executed with an error.

In QSAM the error options Accept and Skip are implemented only when a data error is present; a control error results in the Terminate option. A control error is differented from a data error by the presence of 1 bits in any of the following:

- Bits 42–47 of the Channel Status Word (CSW)
- For direct–access devices, all bits in sense byte 1 or all bits except bit 4 (data check) in sense byte 0. Sense bytes 0 and 1 are found in the IOB at IOBSENS0 and IOBSENS1.

FEOV Executor IGC0003A (SVC 31): Executor IGC0003A causes reading or writing to be discontinued for the balance of the present volume and permits the processing program to start reading or writing a new volume. The FEOV (force–end–of–volume) macro expansion includes an SVC 31 instruction that causes this executor to be loaded and to gain control.

For an input data set, processed under QSAM or BSAM, the executor operates as follows:

- It receives control when the processing program uses an FEOV macro instruction.
- It obtains a work area by means of a GETMAIN macro instruction.
- It prevents the execution of any pending channel programs by means of the PURGE macro instruction.
- It passes control and the address of the work area to the end–of–volume routine of I/O support by means of an XCTL macro instruction.

For an output data set processed under BSAM, the executor operates as follows:

- It receives control when the processing program uses an FEOV macro instruction.
- It obtains a work area by means of a GETMAIN macro instruction.
- It passes control, and the address of the work area, to the end–of–volume routine of I/O support by means of an XCTL macro instruction.

For an output data set processed under QSAM, the operation of the executor and the resultant flow of control depends on the operating mode and how certain channel programs execute.

Operation for Output Under QSAM

The operation of the FEOV executor for an output data set processed under the queued sequential access method (QSAM) depends on the operating mode and the execution of certain channel programs.

In the move operating mode, the execution of all channel programs is tested by the FEOV executor. It awaits the execution of the channel program for the present buffer and causes processing of any error conditions.

In the locate operating mode, the execution of the channel program for the next buffer in the chain is tested by the output synchronizing routine. This test occurs immediately after the end–of–block routine has caused the channel program for the present buffer to be scheduled for execution. The execution of the channel programs for all the following buffers, including the one just scheduled, is tested by the FEOV executor after the last channel program has executed.

When a QSAM routine tests the execution of a channel program, one of three conditions may be established, with the stated results:

- The channel program executed normally: Normal processing continues.
- The channel program is not yet executed: The testing routine awaits completion of the channel program.
- The channel program executed with an error condition: The testing routine passes control to the SYNAD/EOV executor (IGC0005E), by means of an SVC 55 instruction in synchronizing routine IGG019AR. The executor distinguishes between permanent error conditions and end–of–volume conditions. For a description of the error–processing operations initiated by the SYNAD/EOV executor, refer to "Sequential Access Method Executors."

The FEOV executor substitutes its own SYNAD routine (contained within module IGC0003A) for that of the processing program's. That SYNAD routine releases the work area normally obtained by the executor and issues an ABEND macro instruction.

The operation of the FEOV executor, and the resultant flow of control between it and other control program routines, differs for each of eight conditions.

Diagram J (Section 5) illustrates the flow of control between the executor and related routines for each of the eight conditions described in the following paragraphs.

Condition 1: An output data set is processed under QSAM in the move mode, and all channel programs execute normally.

The executor operates as follows:

- It issues a TRUNC macro instruction to pass control to the Put routine. (The Put routine passes control to the end–of–block routine, which causes the channel program for the present buffer to be scheduled for execution. Control returns to the Put routine, which returns control to this executor.)
- It awaits execution of the channel program for the present buffer.
- It tests the execution of the channel program and finds that it executed normally.
- It passes control to the end–of–volume routine of I/O support. (That routine passes control to the EOV/new volume executor, which returns control to the processing program.)

Condition 2: An output data set is processed under QSAM in the move mode, and a permanent error condition is encountered in the execution of a channel program.

The executor operates as follows:

- It issues a TRUNC macro instruction to pass control to the Put routine. (The Put routine passes control to the end–of–block routine, which causes the channel program for the present buffer to be scheduled for execution. Control returns to the Put routine, which returns control to this executor.)
- It awaits execution of the channel program and finds that it encountered an error condition in its execution. It passes control to the synchronizing routine. (That routine finds the same error condition and passes control to the SYNAD/EOV executor (IGC0005E) by means of an SVC 55 instruction. The SYNAD/EOV executor finds that the error condition is a permanent error condition and returns control to the synchronizing routine, which returns control to the FEOV executor.)
- It issues an ABEND macro instruction.

Condition 3: An output data set is processed under QSAM in the move mode, and an end–of–volume condition is encountered in the execution of a channel program.

The executor operates as follows:

- It issues a TRUNC macro instruction to pass control to the Put routine. (The Put routine passes control to the end–of–block routine, which causes the channel program for the present buffer to be scheduled for execution. Control returns to the Put routine, which returns control to the FEOV executor.)
- It awaits execution of the channel program and finds that it encountered an error condition in its execution.
- It passes control to the synchronizing routine. (The routine finds the same error condition and passes control to the SYNAD/EOV executor (IGC0005E) by means of an SVC 55 instruction. The SYNAD/EOV executor finds that the error condition is an end–of–volume condition and passes control to the EOV routine of I/O support. That routine passes control to the EOV/new volume

executor, which returns control to the synchronizing routine. The synchronizing routine now returns control to the FEOV executor.)

- It passes control to the end–of–volume routine of I/O support. That routine passes control to the EOV/new volume executor again, which now returns control to the processing program.

Condition 4: An output data set is processed under QSAM in the locate mode, and all channel programs execute normally.

The executor operates as follows:

- It issues a TRUNC and a PUT macro instruction to pass control to the Put routine. (The Put routine passes control to the end–of–block routine, which causes the channel program for the present buffer to be scheduled for execution. The Put routine then passes control to the synchronizing routine to obtain the next buffer. That routine finds that the channel program for the next buffer executed normally and returns control to the Put routine. The Put routine returns control to the FEOV executor.)
- It awaits execution of the last channel program and finds that the channel program executed normally.
- It passes control to the end–of–volume routine of O/C/EOV. (That routine passes control to the EOV/new volume executor, which returns control to the processing program.)

Condition 5: An output data set is processed under QSAM in the locate mode, and the execution of the channel program for the next buffer in the chain encountered a permanent error.

The FEOV executor operates as follows:

- It issues a TRUNC and a PUT macro instruction to pass control to the Put routine. (The Put routine passes control to the end–of–block routine, which causes the channel program for the present buffer to be scheduled for execution. The Put routine then passes control to the synchronizing routine to obtain the next buffer. The synchronizing routine finds that the channel program executed with an error condition and passes control to the SYNAD/EOV executor (IGC0005E), by means of an SVC 55 instruction. The SYNAD/EOV executor finds that the error condition is a permanent error condition, and returns control to the synchronizing routine. The synchronizing routine now returns control to the FEOV executor.)
- It issues an ABEND macro instruction.

Condition 6: An output data set is processed under QSAM in the locate mode, and the execution of the channel program for any buffer other than the buffer specified in condition 5 encounters a permanent error.

The executor operates as follows:

- It issues a TRUNC and a PUT macro instruction to pass control to the Put routine. (The Put routine passes control to the end–of–block routine, which causes the channel program for the present buffer to be scheduled for execution. The Put routine then passes control to the synchronizing routine, which returns control to the Put routine. The Put routine returns control to the executor.)

- It awaits execution of the channel program for the last buffer and finds that the channel program executed with an error condition.
- It passes control to the synchronizing routine. (The routine finds the same error condition and passes control to the SYNAD/EOV executor (IGC0005E), by means of an SVC 55 macro instruction. The SYNAD/EOV executor finds that the error condition is a permanent error condition and returns control to the synchronizing routine, which returns control to the FEOV executor.)
- It issues an ABEND macro instruction.

Condition 7: An output data set is processed under QSAM in the locate mode, and the execution of the channel program for the next buffer in the chain encountered an end–of–volume condition.

The executor operates as follows:

- It issues a TRUNC and a PUT macro instruction to pass control to the Put routine. (The Put routine passes control to the end–of–block routine, which causes the channel program for the present buffer to be scheduled for execution. The Put routine then passes control to the synchronizing routine to obtain the next buffer. The synchronizing routine finds that the channel program executed with an error condition, and passes control to the SYNAD/EOV executor (IGC0005E), by means of an SVC 55 instruction. The SYNAD/EOV executor finds that the error condition is an EOV condition, and passes control to the EOV routine of I/O support. That routine passes control to the EOV/new volume executor, which passes control to the synchronizing routine. The synchronizing routine returns control to the Put routine, which now returns control to the FEOV executor.)
- It passes control and the work area to the end–of–volume routine of O/C/EOV. (That routine passes control to the EOV/new volume executor again, which now returns control to the processing program.)

Condition 8: An output data set is processed under QSAM in the locate mode, and the channel program for any buffer other than the one specified in condition 7 encounters an end–of–volume condition.

The executor operates as follows:

- It passes control to the Put routine. (The Put routine passes control to the end–of–block routine, which causes the channel program for the present buffer to be scheduled for execution. The Put routine then passes control to the synchronizing routine, which returns control to the Put routine. The Put routine returns control to the FEOV executor.)
- It awaits execution of the channel program for the present buffer, and then finds that the channel program executed with an error condition.
- It passes control to the synchronizing routine. (The routine finds the same error condition and passes control to the SYNAD/EOV executor (IGC0005E) by means of an SVC 55 instruction. The SYNAD/EOV executor finds that the error condition is an condition and passes control to the end–of–volume routine of O/C/EOV. That routine passes control to the EOV/new volume executor, which passes control to the synchronizing routine, which returns control to the FEOV executor.)
- It passes control and the work area, to the end–of–volume routine of O/C/EOV. (That routine passes control to the EOV/new volume executor again, which now returns control to the processing program.)

Note: An EOV condition is found during the implementation of an FEOV macro instruction in conditions 3, 7, and 8. The subsequent processing results in three volumes: Two volumes containing all the blocks scheduled for output by the FEOV macro instruction and prior PUT macro instructions, and a third volume available for writing new blocks.

EOV/New Volume Executor IFG0551L (Alias — IGG0551A): Executor IFG0551L schedules, for execution with the new volume, any channel programs not executed with the old volume. The end–of–volume routine of O/C/EOV issues an XCTL macro instruction to pass control to this executor after the routine has caused the mounting of the next volume of the input data set; for an output data set, the routine passes control to this executor after the routine has mounted a new volume, or acquired additional space on the current volume.

The executor operates as follows:

- It receives control when the next volume is available or when more space is available on the same volume.
- If the user has requested an end–of–volume exit by a DCB exit list entry code '06', the user's exit routine receives control by way of an SVC 12 instruction after switching to a new volume. When an end–of–volume exit is taken for an output data set and user–totaling has been specified, the address of the user–totaling image area is in register 0. Upon regaining control from the user's exit routine, executor IFG0551L restores the pointers to the data extent block (DEB), the request block (RB), the task control block (TCB), the task input/output table (TIOT), and the unit control block (UCB). This restoration is necessary because a restart occurring in the user's end–of–volume exit routine can change the pointers. The end–of–volume exit and the checkpoint/restart are discussed in more detail in *OS Data Management Services Guide,* GC26–3746.
- It resets all indications of the end–of–volume condition in the DCB. If chained scheduling is specified, the executor issues an XCTL macro instruction to pass control to IFG0551N.
- If the device type is direct access, the executor inserts the new full device address (FDAD) into the DCB and the IOB.
- It issues BALR instructions to pass pending channel programs to the end–of–block routine to have them scheduled for execution. If the DCB specifies MACRF=WL (Write in the load mode) for creating a BDAM data set, control passes to the Create–BDAM Write routine.
- It issues a FREEMAIN macro instruction for the work area obtained for the end–of–volume routine.
- It returns control to the routine that passed control to the end–of–volume routine via the SVC 55 instruction. For a normal end–of–volume condition found by a synchronizing or Check routine, control returns to the synchronizing or Check routine. For a forced end–of–volume condition established by an FEOV macro instruction in the processing program, control returns to the processing program. For an end–of–volume condition arising during the FEOV executor, control returns to the FEOV executor.

EOV/New Volume Executor IFG0551N (Alias — IGG0551B): Executor IFG0551N performs the rescheduling functions of IFG0551L for those data sets being processed that use chained scheduling. Executor IFG0551L transfers control to this module by an XCTL macro instruction when it determines that chained scheduling is specified.

The executor operates as follows:

- It receives control when chained scheduling is specified.
- If device type is direct access, the executor inserts the new full device address (FDAD) into the DCB and the IOB.
- It issues BALR instructions to pass pending channel programs to the end–of–block routine to have them scheduled for execution.
- It issues a FREEMAIN macro instruction for the work area obtained for the end–of–volume routine.
- It returns control to the routine that passed control to the end–of–volume routine by means of the SVC 55 instruction.

SETPRT Executors

There are six executors associated with the SETPRT macro instruction. The executor IGC0008A receives control when the SVC 81 instruction is issued. Executor IGG08101 receives control after executor IGC0008A if a specified UCS image is to be loaded from the SYS1.IMAGELIB. The executor IGG08102 receives control after executor IGG08101 to load the UCS image into the UCS buffer and to print verification lines if required. The executors, IGG08103 and IGG08104, respectively, locate the FCB image and load it into the forms control buffer. Executor IGG08104 also verifies the load and/or allows forms alignment.

Executor IGC0010E receives control when the SVC 105 instruction is issued. Figure 32 shows the conditions that cause the executors to gain control.

Access Method Conditions	Selection						
SETPRT macro	X						
Retrieve UCS image from SYS1.IMAGELIB		X					
Load UCS image			X				
Locate the FCB image in calling program or retrieve it from SYS1.IMAGELIB				X			
Load FCB image					X		
Skeleton DCB and DEB needed for IMAGELIB						X	
Executors							
IGC0008A	8A						
IGG08101		01					
IGG08102			02				
IGG08103				03			
IGG08104					04		
IGC0010E						0E	

Figure 32. SETPRT Executor Selector

SETPRT Executor IGC0008A: The macro instruction SETPRT (set–printer) expands into an SVC 81 instruction that causes this executor to be loaded and to gain control. Executor IGC0008A determines whether the specified UCS image is to be loaded from the SYS1.IMAGELIB after processing all outstanding output requests for the DCB.

The executor operates as follows:

- When BSAM is specified for the DCB, the executor purges all the outstanding output requests.
- When QSAM is specified for the DCB, the executor causes all outstanding output requests to quiesce.
- It uses the EXCP macro instruction to execute block data check or reset block data check according to the specification in the SETPRT macro instruction.
- It uses the GETMAIN macro instruction to obtain a work area.
- When the specified UCS image has been loaded by a previous job step, it uses the FREEMAIN macro instruction to free the work area and passes control back to the user's program.
- When the specified UCS image has not been loaded by a previous job step, it passes control to executor IGG08101.
- If an FCB load is required but a UCS load is not, executor IGG08103 is called.

SETPRT Executor IGG08101: Executor IGG08101 is entered from executor IGC0008A when the specified UCS image is to be loaded from the SYS1.IMAGELIB.

The executor operates as follows:

- It uses the BLDL macro instruction to locate the UCS image in the SYS1.IMAGELIB.
- If the UCS image is not in the library, the executor requests the operator to specify an alternate UCS image to be loaded.
- When the UCS image is in the library, the executor uses the LOAD macro instruction to retrieve the UCS image from the library.
- The executor passes control to executor IGG08102 if the retrieved UCS image is to be loaded into the UCS buffer or is to be displayed for visual verification.
- Control is passed to IGG08103 when a UCS image is not to be loaded or displayed and an FCB image is specified. If an FCB image is not specified, the executor returns control to the user's program.

SETPRT Executor IGG08102: Executor IGG08102 is entered from executor IGG08101 to load the UCS image into the UCS buffer and to print verification lines if required.

The executor operates as follows:

- It uses the EXCP macro instruction to load the UCS image into the UCS buffer.
- When verification of the image is required, the executor uses the EXCP macro instruction to print the UCS image for verification.
- It updates the UCB and uses the DELETE macro instruction to release the UCS image. If the FCB is not to be loaded, a FREEMAIN macro instruction releases the work area and control is returned to the user's program.

If an FCB image is specified, executor IGG08103 is called; otherwise, control is given to the user's program.

SETPRT Executor IGG08103: Executor IGG08103 locates the FCB image in the problem program's DCB exit list or in SYS1.IMAGELIB and loads it into main storage. It is entered from IGC0008A, IGG08101, or IGG08102.

The executor operates as follows:

- It checks the DCB exit list to see whether the specified FCB image is defined in the problem program.
- It uses the BLDL macro instruction to locate the FCB image in SYS1.IMAGELIB.
- If the image is not found in the library, the executor requests the operator to specify an alternate FCB image.
- If the FCB image is specified in the problem program or successfully loaded into main storage from SYS1.IMAGELIB, executor IGG08104 is called to load the image into the forms control buffer and/or print a verification.
- If no further FCB activity is required, control is returned to the user's program.

SETPRT Executor IGG08104: Executor IGG08104 loads the FCB image into the forms control buffer and verifies the load and/or allow forms alignment. It is entered from IGC0008A, IGG08102, or IGG08103.

The executor operates as follows:

- It checks to see whether an align–forms–only switch or a verify–only switch is set.
- If neither switch is set, the forms control buffer is loaded.
- If VERIFY is specified, a verification message is printed.
- If VERIFY or ALIGN is specified, the operator is instructed to align the forms.
- The executor always exits to the problem program.

IMAGELIB Executor IGC0010E: Executor IGC0010E builds a skeleton DCB and DEB for the SYS1.IMAGELIB data set or deletes the DCB and DEB for the SYS1.IMAGELIB data set, depending on the parameter passed to it in register 1. The executor is entered from the SVC 105 instruction.

The executor operates as follows:

- It makes a test to determine whether the control blocks for IMAGELIB need to be built or deleted. If register 1 contains 0s, a DCB and DEB are built.
- It uses a GETMAIN macro instruction to obtain a work area and then uses a LOCATE macro instruction to determine where the IMAGELIB volume is residing.
- It takes the address of the UCB table from the CVT and searches for the corresponding UCB.
- It uses the OBTAIN macro instruction to read in the format–1 DSCB and uses the information found in the format–1 DSCB and the UCB address to construct a skeleton DCB and DEB for the IMAGELIB volume.

- If register 1 contains an address when the executor tests to determine whether the control blocks for the IMAGELIB volume need to be built or deleted, the DCB and DEB for IMAGELIB are to be deleted.
- It uses the FREEMAIN macro instruction to delete the control blocks.
- It returns control to the calling routine through the SVC 3 instruction.

Buffer–Pool Management

Buffer–pool management routines form main storage space into buffers, and they return buffers that are no longer needed. Figure 33 lists the buffer–pool management routines.

Type	Module Name	Function
GETPOOL	IECQBFG1	This routine obtains main storage and forms a buffer pool.
BUILD	IECBBFB1	This routine forms a buffer pool in main storage supplied by the processing program.
GETBUF	(Macro Expansion)	This routine provides buffers from the buffer chain.
FREEBUF	(Macro Expansion)	This routine returns buffers to the buffer pool.
FREEPOOL	(Macro Expansion)	This routine returns main storage previously used for a buffer pool.
BUILDRCD	IGG019BO	This routine allows a pointer to a record area to be incorporated in a buffer pool in main storage supplied by the processing program.

Figure 33. Buffer–Pool Management Routines

GETPOOL Module IECQBFG1

Module IECQBFG1 obtains main–storage space and forms it into buffers. It is loaded at execution time by a LINK macro instruction.

The module operates as follows:

- It rounds the buffer length to the next higher doubleword multiple if the specified length is not such a multiple.
- It determines buffer alignment from the DCBBFALN field value in the DCB.
- It computes the number of bytes required and issues a GETMAIN macro instruction.
- It constructs a buffer–pool control block in the first eight bytes of storage obtained.
- If doubleword (not fullword) alignment is specified in the DCBBFALN field in the DCB, the module starts the first buffer at the byte immediately following the BUFCB.

- If fullword (not doubleword) alignment is specified in the DCBBFALN field, the module skips one word after the buffer–pool control block before starting the first buffer.
- It chains the first buffer to the buffer–pool control block and determines the start of the next buffer by adding the rounded buffer length value to the address of the first buffer. The module chains the next buffer to the preceding buffer and continues until all the buffers have been chained.
- It returns control to the processing program.

Figure 34 illustrates the buffer–pool control block (BUFCB) that describes the buffer pool. Figure 35 illustrates the buffer–pool structures formed by the GETPOOL module.

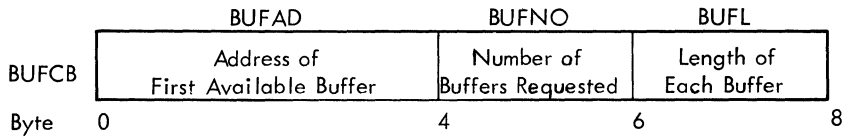


Figure 34. Buffer–Pool Control Block

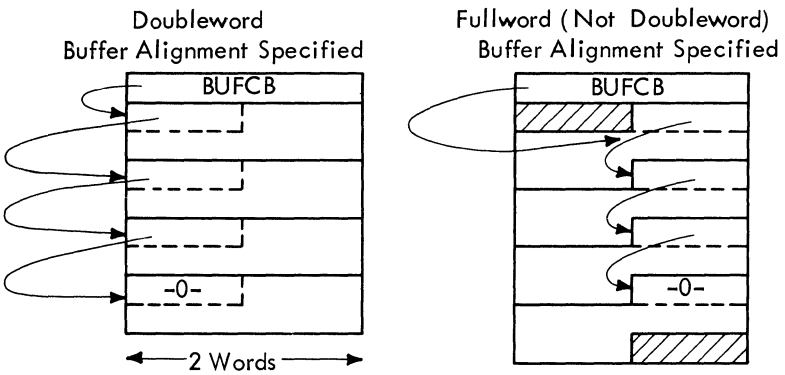


Figure 35. GETPOOL Buffer–Pool Structures

Build Module IECBBFB1

Module IECBBFB1 forms main storage space supplied by the processing program into buffers. It is loaded at execution time by a LINK macro instruction.

The module operates as follows:

- It rounds the buffer length to the next higher fullword multiple if the specified length is not such a multiple.
- It constructs a buffer–pool control block in the first 8 bytes of the main–storage space provided by the processing program.

- It starts the first buffer at the byte immediately following the buffer–pool control block.
- It chains the first buffer to the buffer pool control block and determines the start of the next buffer by adding the rounded buffer–length value to the address of the first buffer. The module chains the next buffer to the preceding buffer, and continues until all the buffers are chained.
- It returns control to the processing program.

Figure 36 lists for each possible combination of space alignment and buffer length parity the illustration that shows the structure of the resulting buffer chain or pool. Figure 34 illustrates the buffer pool control block (BUFCB), Figure 37 illustrates the various buffer alignments that the Build module forms.

GETBUF Macro Expansion

The purpose of this coding is to provide the next buffer from the buffer pool. The macro expansion produces inline code that presents the address of the next buffer to the processing program and updates the buffer-pool control block to point at the following buffer.

FREEBUF Macro Expansion

The purpose of this coding is to return a buffer to the buffer chain. The macro expansion produces inline code that stores the address presently in the buffer–pool control block in the first word of the buffer being returned, and then stores the address of that buffer in the buffer–pool control block.

Alignment of first byte of space passed in BUILD macro instruction	Parity of number of words in buffer length after rounding up length parameter of BUILD macro instruction	Buffer pool structure
Doubleword	Even	A
	Odd	B
Fullword (Not doubleword)	Even	C
	Odd	D

Figure 36. Build Buffer Structuring Table

FREEPOOL Macro Expansion

The purpose of this coding is to return the space previously allotted to the buffer chain to available main storage. The macro expansion produces inline code that computes the total number of bytes to be returned, issues a FREEMAIN macro instruction, and sets the DCBBUFCB field in the DCB to show that no buffer pool is associated with that DCB.

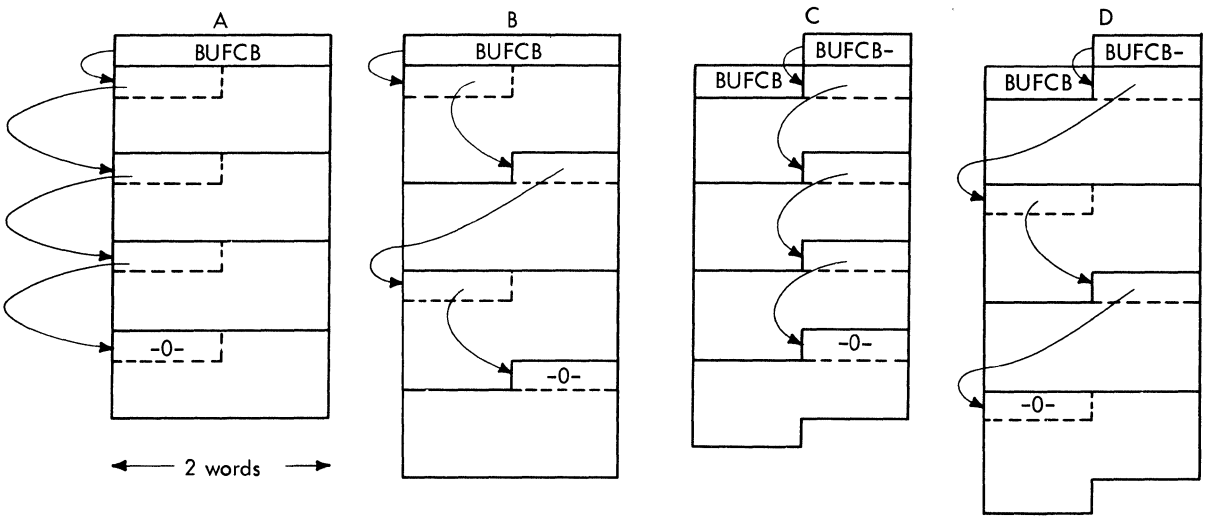


Figure 37. Build Buffer Pool Structure

BUILDRCD Routine IGG019B0

This routine forms main–storage space supplied by the processing program into buffers and links the buffer pool to a record area also supplied by the processing program. It is loaded at execution time by a LINK macro instruction.

The module operates as follows:

- It rounds the buffer length to the next higher fullword multiple if the specified length is not such a multiple.
- It constructs a buffer–pool control block (see Figure 38) in the first twelve bytes of the main–storage space provided by the processing program.
- It turns on the high–order bit of the BUFLG byte of the buffer–pool control block to indicate that a record area address is present.
- It clears the control field (32 bytes) of the record area.
- It stores the record area length in the record area (see Figure 39) provided by the processing program.
- It chains the first buffer to the buffer–pool control block and determines the start of the next buffer by adding the rounded buffer length value to the address of the first buffer. The next buffer is chained to the preceding buffer until all buffers are built.
- It returns control to the processing program.

Figure 38 illustrates the buffer–pool control block (BUFCB) that describes the buffer pool when logical record interface is required for variable–length spanned records processed in the locate mode.

Figure 39 illustrates the record area used to assemble and segment a spanned record. This record area is either acquired dynamically by data management at Open time, when the DCB specifies RECFM–VS/VBS, MACRF=GL/PL, and BFTEK=A, or provided by the problem program by means of a BUILDRCD macro instruction.

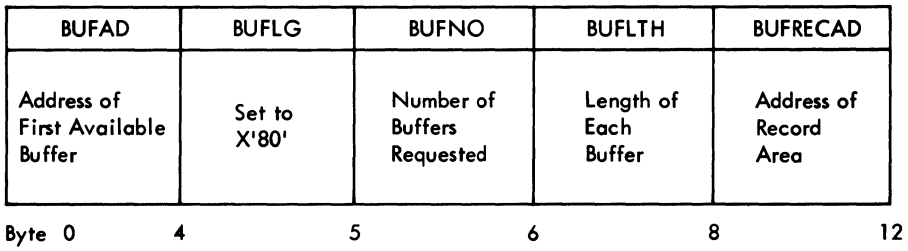


Figure 38. Buffer Pool Control Block

BUFAD: 4 bytes, contains the address of the first available buffer in the pool.

BUFLG: 1 byte, set to X'80' when a record area address is present in the buffer control block.

BUFNO: 1 byte, contains the number of buffers requested.

BUFLTH: 2 bytes, contains the length rounded to the nearest fullword of each buffer requested.

BUFRECAD: 4 bytes, contains the starting address of the record area.

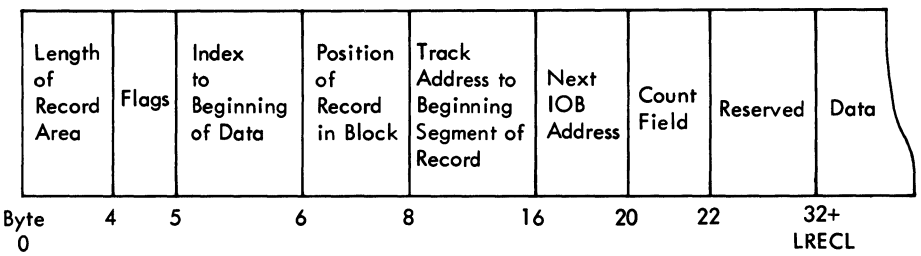


Figure 39. Record Area

A description of the fields contained in the record area follows:

- Length of record area

  This 4–byte field contains the length of the entire record area (data field + 24 bytes). The length may be determined by the LRECL of the DCB macro at Open time plus 8 bytes for alignment or specified in the length of the record area parameter of the BUILDRCD macro instruction, in which case the BUILDRCD routine places the length of the record area in this field. The second high–order bit of the first byte of this field is set to X'40' by the COBOL processor to indicate special processing of variable–length spanned records. If this bit is set,

all records (spanned or nonspanned) are presented to the processing program in the record area.

- Flags

  This 1–byte field is used for internal data management control flags.

- Index to beginning of data

  This 1–byte field contains the index value to the beginning of the data (record descriptor word) in the data field.

- Position of record in block

  This 2–byte field contains the relative position of the beginning segment of a record within the block.

- Track address of beginning segment of record

  This 8–byte field is used to save the track address of that block which has a beginning segment of a record that is being processed. The low–order three bytes of this field are used to save the record address of the block that will have the beginning segment of a record if a spanned record is to be written.

- Next IOB address

  This 4–byte field is used to save the next IOB address if a spanned record is to be written.

- Count field

  This 2–byte field is used to accumulate the number of bytes of data moved while segmenting.

- Reserved — Not used

- Data

  The assembled logical record is located in this field. The maximum length of this field is either determined by the LRECL field of the DCB macro at Open time plus 8 bytes for alignment or equal to 24 bytes less than the length of the record area parameter of the BUILDRCD macro instruction.

SECTION 3: DIRECTORY

Directory

Module Name	Module Type	CSECT Name	SVC Entry	Logic Manual Reference	Module Description (Page)
IECBBFB1	Build Module	IECBBFB1		Figure 33	178
IECPFIND	Resident Module	IGC01B	18	Figure 25	126
		IECPBLDL			127
		IECPCNVT			127
		IECPRLTV			127
IECPFND1	Resident Module	IEC018	18	Figure 25	127
		IECPCNVT			128
		IECPRLTV			128
IECQBFG1	GETPOOL Module	IECQBFG1		Figure 33	177
IFG0551B	SYNAD/Diagnostic Executor	IFG0551B		Figure 31	168
IFG0551L	EOV/New Volume Executor	IFG0551L		Figure 31	173
IFG0551N	EOV/New Volume Executor	IFG0551N		Figure 31	174
IGC0002A	STOW Module	IGC0002A	21	Figure 25	124
IGC0002E	Control Module	IGC0002E	25	Figure 23	116
IGC0003A	FEOV Executor	IGC0003A	31	Figure 31	168
IGC0005E	SYNAD/EOV Executor	IGC0055	55	Figure 31	167
IGC0006I	Control Module	IGC0006I	69	Figure 23	123
IGC0008A	SETPRT Executor	IGC0008A	81	Figure 32	175
IGC0010E	IMAGELIB Executor	IGC0010E	105	Figure 32	176
IGG0010C	Translate Routine	IGG0010C	103	See Appendix A Section 6	216
IGG019AA	Get Module	IGG019AA		Figure 1	5
IGG019AB	Get Module	IGG019AB		Figure 1	6
IGG019AC	Get Module	IGG019AC		Figure 1	7
IGG019AD	Get Module	IGG019AD		Figure 1	8
IGG019AE	Get Module	IGG019AE		Figure 6	25
IGG019AF	Synchronizing Module	IGG019AF		Figure 15	73
IGG019AG	Get Module	IGG019AG		Figure 1	9
IGG019AI	Put Module	IGG019AI		Figure 7	32
IGG019AJ	Put Module	IGG019AJ		Figure 7	33
IGG019AK	Put Module	IGG019AK		Figure 7	34
IGG019AL	Put Module	IGG019AL		Figure 7	35
IGG019AM	Get Module	IGG019AM		Figure 1	9
IGG019AN	Get Module	IGG019AN		Figure 1	10
IGG019AQ	Synchronizing Module	IGG019AQ		Figure 15	74
IGG019AR	Synchronizing Module	IGG019AR		Figure 15	75
IGG019AT	Get Module	IGG019AT		Figure 1	12
IGG019AV	Open Executor	IGG019AV		Figure 27	132
IGG019AW	Appendage	IGG019AW		Figure 17	81
IGG019AX	Save Module	IGG019AX		Figure 14	68
IGG019BA	Read/Write Module	IGG019BA		Figure 20	103
IGG019BB	Check Module	IGG019BB		Figure 21	111

Module Name	Module Type	CSECT Name	SVC Entry	Logic Manual Reference	Module Description (Page)
IGG019BC	Control Module	IGG019BC		Figure 22	117
IGG019BD	Control Module	IGG019BD		Figure 22	118
IGG019BE	Control Module	IGG019BE		Figure 22	119
IGG019BF	Read Module	IGG019BF		Figure 20	105
IGG019BG	Check Module	IGG019BG		Figure 21	113
IGG019BH	Read/Write Module	IGG019BH		Figure 20	106
IGG019BI	Check Module	IGG019BI		Figure 21	113
IGG019BK	Control Module	IGG019BK		Figure 22	120
IGG019BL	Control Module	IGG019BL		Figure 22	121
IGG019BM	Appendage	IGG019BM		Figure 17	83
IGG019BN	Get Update Module	IGG019BN		Figure 6	27
IGG019BO	Get Module	IGG019BO		Figure 1	13
IGG019BP	Put Module	IGG019BP		Figure 7	36
IGG019BQ	Synchronizing Module	IGG019BQ		Figure 15	75
IGG019BR	Write Module	IGG019BR		Figure 20	106
IGG019BS	Check Module	IGG019BS		Figure 21	114
IGG019BT	Channel–End Appendage	IGG019BT		Figure 17	88
IGG019BU	Read Module	IGG019BU		Figure 20	107
IGG019BV	Channel–End Appendage	IGG019BV		Figure 17	88
IGG019B0	BUILDRCD Routine	IGG019B0		Figure 33	180
IGG019CA	Control Module	IGG019CA		Figure 18	101
IGG019CB	Control Module	IGG019CB		Figure 18	102
IGG019CC	EOB Module	IGG019CC		Figure 9	48
IGG019CD	EOB Module	IGG019CD		Figure 9	49
IGG019CE	EOB Module	IGG019CE		Figure 9	50
IGG019CF	EOB Module	IGG019CF		Figure 9	51
IGG019CG	SIO Appendage	IGG019CG		Figure 17	85
IGG019CH	End–of–Extent Appendage	IGG019CH		Figure 17	84
IGG019CI	Channel–End Appendage	IGG019CI		Figure 17	89
IGG019CJ	Channel–End Appendage	IGG019CJ		Figure 17	90
IGG019CK	Channel–End Appendage	IGG019CK		Figure 17	91
IGG019CL	SIO Appendage	IGG019CL		Figure 17	86
IGG019CM	Code Conversion Module	IGG019CM		See Appendix A Section 6	215
IGG019CN	Code Conversion Module	IGG019CN		See Appendix A Section 6	215
IGG019CO	Code Conversion	IGG019CO		See Appendix A Section 6	215
IGG019CP	Code Conversion Module	IGG019CP		See Appendix A Section 6	215
IGG019CQ	Code Conversion Module	IGG019CQ		See Appendix A Section 6	216
IGG019CR	Code Conversion Module	IGG019CR		See Appendix A Section 6	216
IGG019CS	Channel–End Appendage	IGG019CS		Figure 17	91
IGG019CT	EOB Module	IGG019CT		Figure 9	52
IGG019CU	Appendage	IGG019CU		Figure 17	96
IGG019CV	EOV Module	IGG019CV		Figure 11	59
IGG019CW	EOB Module	IGG019CW		Figure 11	61

Module Name	Module Type	CSECT Name	SVC Entry	Logic Manual Reference	Module Description (Page)
IGG019CX	EOB Module	IGG019CX		Figure 11	63
IGG019CY	EOB Module	IGG019CY		Figure 11	63
IGG019CZ	End–of–Extent Appendage	IGG019CZ		Figure 17	84
IGG019C1	Asynchronous Error Processing Module	IGG019C1		Figure 16	77
IGG019C2	EOB Module	IGG019C2		Figure 14	69
IGG019C3	Appendage	IGG019C3		Figure 17	98
IGG019C4	Appendage	IGG019CA		Figure 17	85
IGG019C6	Appendage	IGG019C6		Figure 17	92
IGG019DA	Write Module	IGG019DA		Figure 20	108
IGG019DB	Write Module	IGG019DB		Figure 20	110
IGG019DC	Check Module	IGG019DC		Figure 21	114
IGG019DD	Write Module	IGG019DD		Figure 20	110
IGG019EA	Get Module	IGG019EA		Figure 3	19
IGG019EB	Get Module	IGG019EB		Figure 3	20
IGG019EC	Get Module	IGG019EC		Figure 3	21
IGG019ED	Get Module	IGG019ED		Figure 3	21
IGG019EE	Put Module	IGG019EE		Figure 8	42
IGG019EI	Appendage	IGG019EI		Figure 17	92
IGG019EJ	Appendage	IGG019EJ		Figure 17	93
IGG019EF	Put Module	IGG019EF		Figure 8	44
IGG019FA	Control Module	IGG019FA		Figure 18	102
IGG019FB	Get Module	IGG019FB		Figure 1	14
IGG019FD	Get Module	IGG019FD		Figure 1	15
IGG019FF	Get Module	IGG019FF		Figure 1	17
IGG019FG	Put Module	IGG019FG		Figure 7	38
IGG019FJ	Put Module	IGG019FJ		Figure 7	39
IGG019FK	EOB Module	IGG019FK		Figure 9	53
IGG019FL	Put Module	IGG019FL		Figure 7	40
IGG019FN	SIO Appendage	IGG019FN		Figure 17	86
IGG019FP	Channel–End Appendage	IGG019FP		Figure 17	95
IGG019FQ	EOB Module	IGG019FQ		Figure 9	53
IGG019FR	Appendage	IGG019FR		Figure 17	82
IGG019FS	Asynchronous Error–Processing Module	IGG019FS		Figure 16	78
IGG019FU	EOB Module	IGG019FU		Figure 9	54
IGG019TC	EOB Module	IGG019TC		Figure 9	55
IGG019TD	EOB Module	IGG019TD		Figure 9	56
IGG019TV	EOB Module	IGG019TV		Figure 11	64
IGG019TW	EOB Module	IGG019TW		Figure 11	66
IGG019T2	EOB Module	IGG019T2		Figure 14	69
IGG019T4	TSO Put Routine	IGG019T4		See description of IGG0201A Figure 30	163
IGG0191A	Open Executor	IGG0191A		Figure 27	132
IGG0191B	Open Executor	IGG0191B		Figure 27	133
IGG0191C	Open Executor	IGG0191C		Figure 27	133
IGG0191D	Open Executor	IGG0191D		Figure 28	140
IGG0191E	Open Executor	IGG0191E		Figure 28	142
IGG0191F	Open Executor	IGG0191F		Figure 28	143

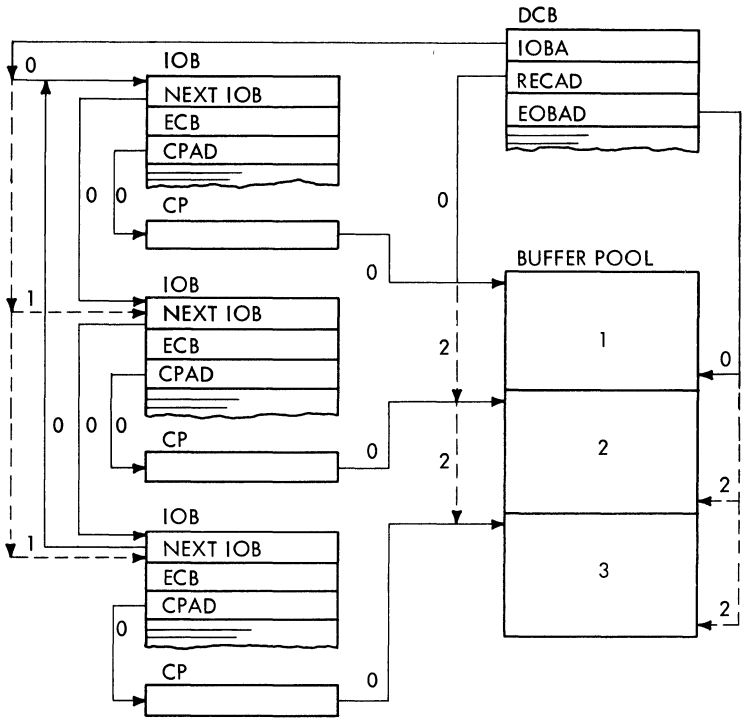
Module Name	Module Type	CSECT Name	SVC Entry	Logic Manual Reference	Module Description (Page)
IGG0191G	Open Executor	IGG0191G		Figure 28	143
IGG0191H	Open Executor	IGG0191H		Figure 28	144
IGG0191I	Open Executor	IGG0191I		Figure 27	134
IGG0191J	Open Executor	IGG0191J		Figure 28	144
IGG0191K	Open Executor	IGG0191K		Figure 28	145
IGG0191L	Open Executor	IGG0191L		Figure 28	145
IGG0191M	Open Executor	IGG0191L		Figure 28	146
IGG0191N	Open Executor	IGG0191N		Figure 27	134
IGG0191O	Open Executor	IGG0191O		Figure 28	146
IGG0191P	Open Executor	IGG0191P		Figure 28	147
IGG0191Q	Open Executor	IGG0191Q		Figure 28	147
IGG0191R	Open Executor	IGG0191R		Figure 28	148
IGG0191S	Open Executor	IGG0191S		Figure 28	148
IGG0191T	Open Executor	IGG0191T		Figure 27	134
IGG0191U	Open Executor	IGG0191U		Figure 27	135
IGG0191V	Open Executor	IGG0191V		Figure 27	135
IGG0191W	Open Executor	IGG0191W		Figure 28	149
IGG0191X	Open Executor	IGG0191X		Figure 28	149
IGG0191Y	Open Executor	IGG0191Y		Figure 27	136
IGG0191Z	Open Executor	IGG0191Z		Figure 28	150
IGG01910	Open Executor	IGG01910		Figure 29	154
IGG01911	Open Executor	IGG01911		Figure 29	154
IGG01912	Open Executor	IGG01912		Figure 29	155
IGG01913	Open Executor	IGG01913		Figure 29	156
IGG01914	Open Executor	IGG01914		Figure 29	156
IGG01915	Open Executor	IGG01915		Figure 29	157
IGG01916	Open Executor	IGG01916		Figure 29	157
IGG01917	Open Executor	IGG01917		Figure 29	158
IGG01918	Open Executor	IGG01918		Figure 29	158
IGG01919	Open Executor	IGG01919		Figure 29	159
IGG01926	Open Executor	IGG01926		Figure 29	159
IGG0193I	Open Executor	IGG0193I		Figure 27	137
IGG0196A	Open Executor	IGG0196A		Figure 27	137
IGG0196B	Open Executor	IGG0196B		Figure 27	137
IGG0196I	Open Executor	IGG0196I		Figure 27	138
IGG0196J	Open Executor	IGG0196J		Figure 28	150
IGG0196K	Open Executor	IGG0196K		Figure 28	150
IGG0196L	Open Executor	IGG0196L		Figure 28	150
IGG0196P	Open Executor	IGG0196P		Figure 28	151
IGG0196S	TSO Module	IGG0196S		See IGG0191I, Figure 27	
IGG0197E	Open Executor	IGG0197E		Figure 27	138
IGG0197F	Open Executor	IGG0197F		Figure 27	138
IGG0197L	Open Executor	IGG0197L		Figure 27	139
IGG0197M	Open Executor	IGG0197M		Figure 27	139
IGG0197N	Open Executor	IGG0197N		Figure 28	151
IGG0197P	Open Executor	IGG0197P		Figure 28	152
IGG0197Q	Open Executor	IGG0197Q		Figure 28	152
IGG0197U	Open Executor	IGG0197U		Figure 27	139

Module Name	Module Type	CSECT Name	SVC Entry	Logic Manual Reference	Module Description (Page)
IGG0199K	Open Executor	IGG0199K		Figure 28	152
IGG0199L	Open Executor	IGG0199L		Figure 28	153
IGG0199O	Open Executor	IGG0199O		Figure 28	153
IGG01990	Open Executor	IGG01990		Figure 29	160
IGG01991	Open Executor	IGG01991		Figure 29	160
IGG01992	Open Executor	IGG01992		Figure 29	161
IGG01993	Open Executor	IGG01993		Figure 29	161
IGG01994	Open Executor	IGG01994		Figure 29	162
IGG0201A	Close Executor	IGG0201A		Figure 30	163
IGG0201B	Close Executor	IGG0201B		Figure 30	164
IGG0201P	Close Executor	IGG0201P		Figure 30	164
IGG0201R	Close Executor	IGG0201R		Figure 30	165
IGG0201X	Close Executor	IGG0201X		Figure 30	165
IGG0201Y	Close Executor	IGG0201Y		Figure 30	166
IGG0201Z	Close Executor	IGG0201Z		Figure 30	166
IGG0210A	STOW Module	IGG0210A		Figure 25	125
IGG0551A	Alias for IFG0551L			Figure 31	173
IGG0551B	Alias for IFG0551N			Figure 31	174
IGG08101	SETPRT Executor	IGG08101		Figure 32	175
IGG08102	SETPRT Executor	IGG08102		Figure 32	175
IGG08103	SETPRT Executor	IGG08103		Figure 32	176
IGG08104	SETPRT Executor	IGG08104		Figure 32	176

SECTION 4: DATA AREAS

QSAM Control Blocks

Figure 40 shows the control blocks used in QSAM. Through the data control block (DCB), the QSAM routines associate the data being processed with the processing program. Fields in the DCB point to the start of a buffer, the end of a buffer, and an input/output block (IOB). These fields are updated as successive channel programs are executed. Each IOB points at the next IOB and at a channel program (CP), and carries an event control block (ECB) that the I/O supervisor posts after the channel program has been executed.



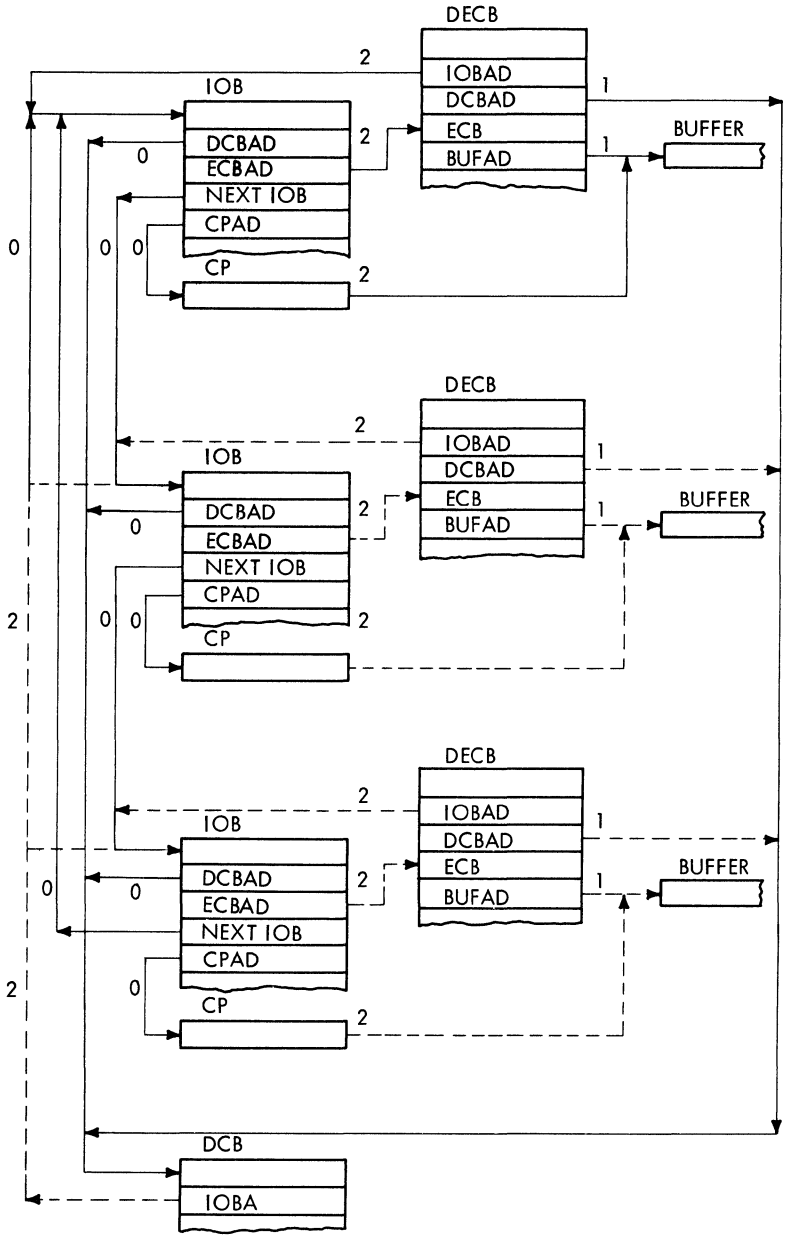
Legend:
Address Values:
0 Entered by the OPEN executor.
1 Updated by the synchronizing routine.
2 Updated by the GET or PUT routine.
--- Successive Address Values

Figure 40. QSAM Control Blocks

BSAM Control Blocks

Figure 41 shows the control blocks used in BSAM and their stages of completion. Stage 0 shows the state of the control blocks before any READ or WRITE macro instruction. Stage 1 shows the effect of the READ or WRITE macro instruction, that is, the values supplied by the processing program in the data event control block (DECB). Finally, stage 2 shows the effect of the Read or Write routine having tied together these control blocks.

Before any READ or WRITE macro instruction, the data control block (DCB) points to the first input/output block (IOB). This IOB points back to the DCB, to the next IOB, and to the channel program (CP). The READ or WRITE macro instruction identifies the DCB and the buffer to be read into or written out. Finally, the Read or Write routine connects the DECB with the current IOB, inserts the address of the ECB (which is located in the DECB) into the IOB, and points the channel program to the buffer. Successive macro instructions cause updating of the IOB address in the DCB and insert address values in the next DECB, IOB, and channel program.

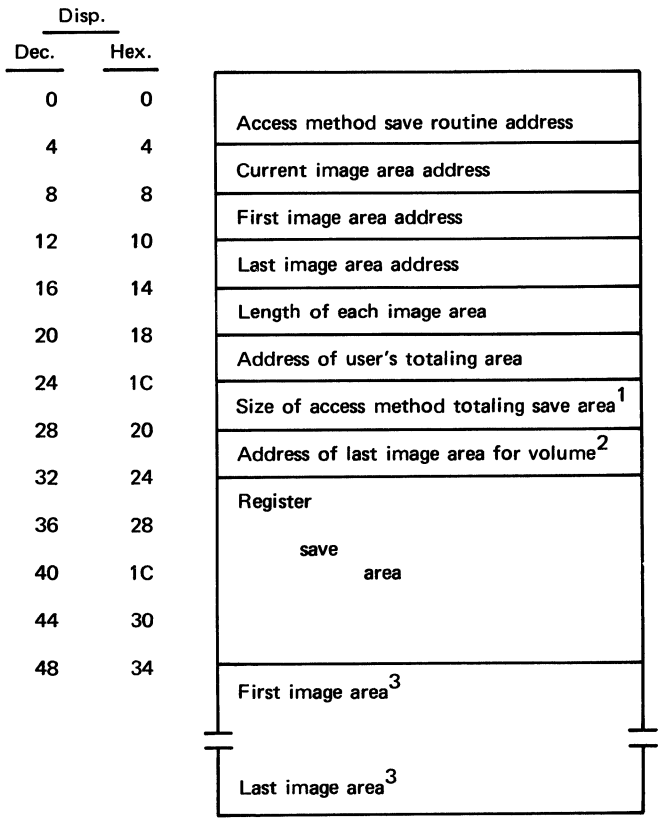


Legend:
Address Values
0 Entered by the OPEN Executor.
1 Provided by the processing program.
2 Completed by the READ or WRITE routine.
--- Successive Address Values.

Figure 41. BSAM Control Blocks

Access Method Save Area for User Totaling

The access method save area for user totaling is pointed to by the address in bytes 5–7 in the EXCP access method, BSAM, or QSAM–dependent section of the DEB.



[1]The size of this save area includes the space used by image areas.

[2]This field is adjusted by the End-of-Volume routine so that it points to the image area containing the user's total for the last record written on the volume.

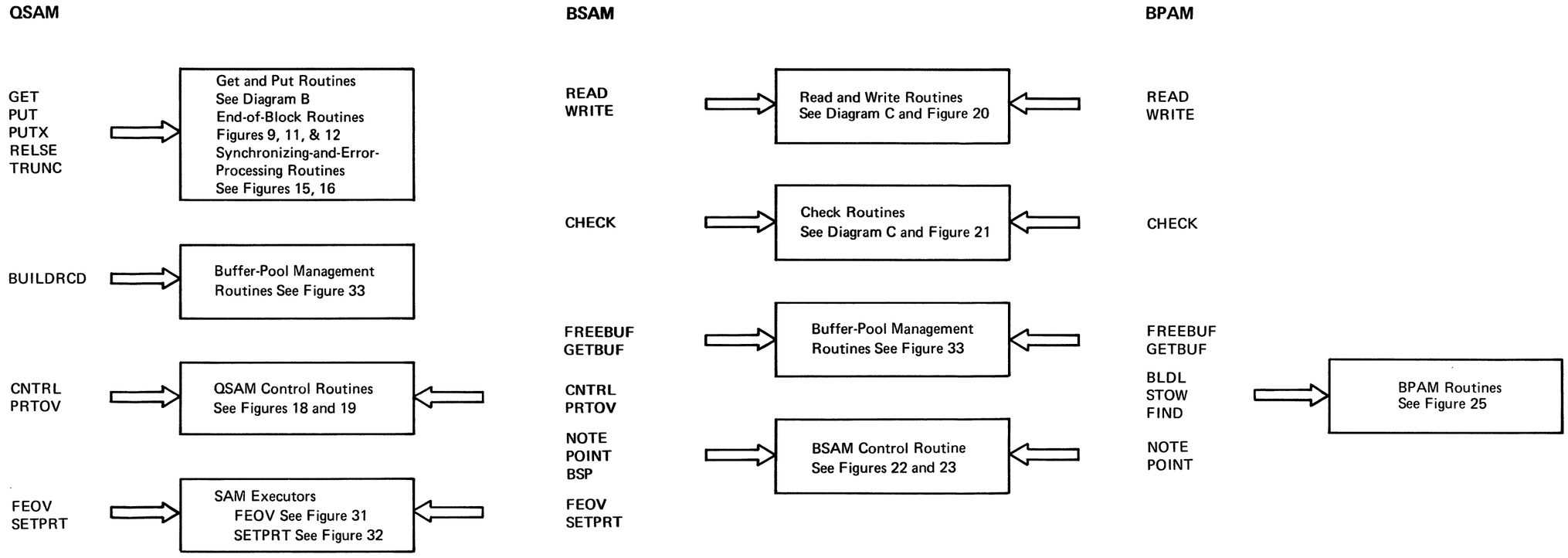
[3]The image areas are all the same size, that is, the length of the user's totaling area rounded to the nearest halfword.

Figure 42. Access Save Area for User Totaling

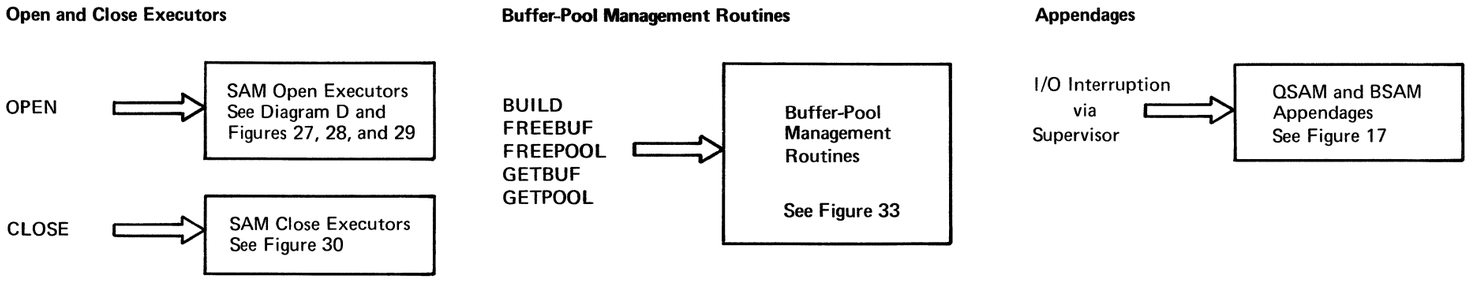
SECTION 5: PROGRAM ORGANIZATION AND FLOW OF CONTROL

Sequential Access Method - Overview

DIAGRAM A

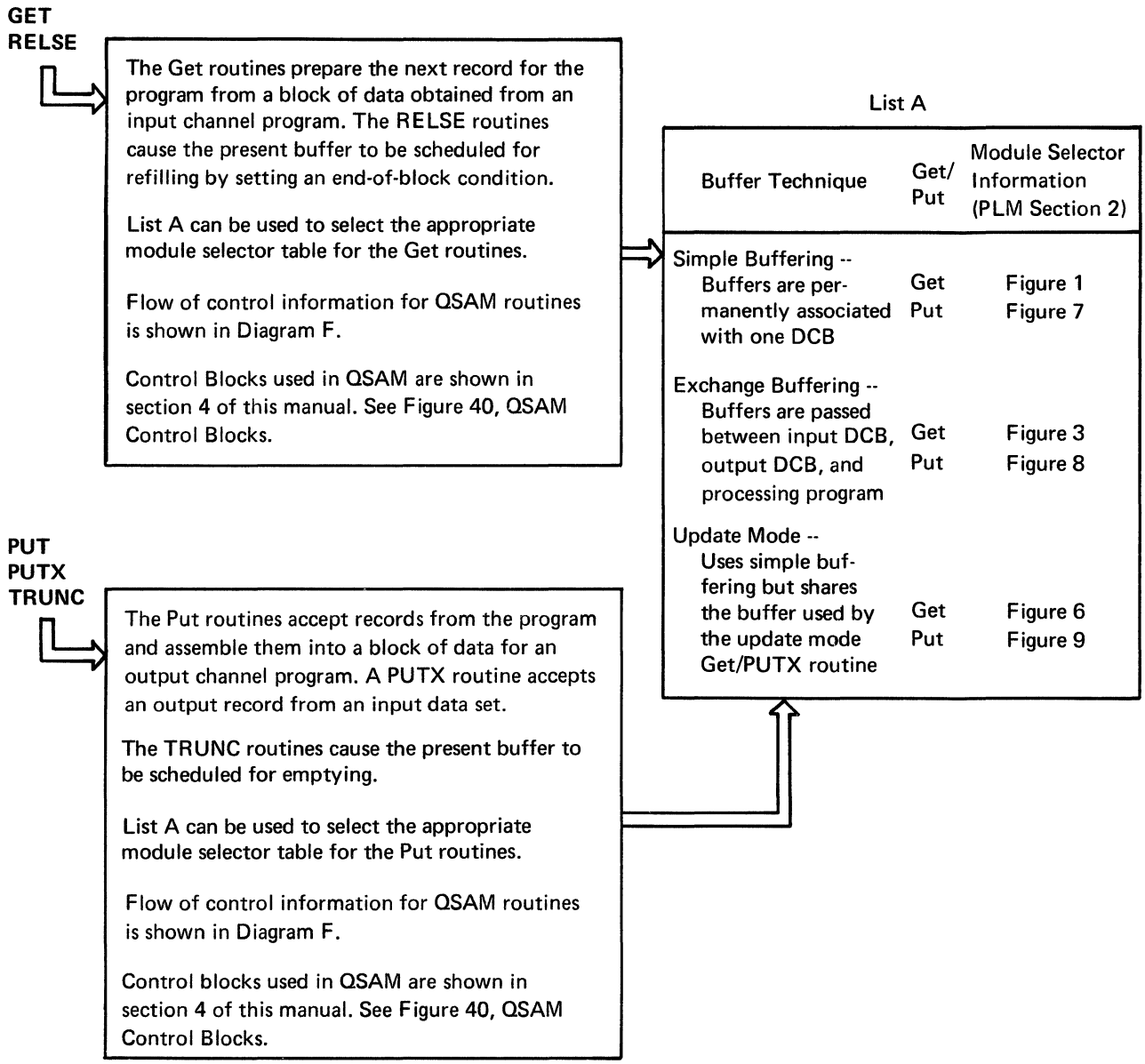


COMMON ACCESS METHOD ROUTINES



QSAM Get and Put Routines

DIAGRAM B



READ/WRITE ⟹

A Read or Write routine completes some of the entries in the channel program from parameters in the data event control block (DECB).

The Read/Write modules are listed in Figure 20 (section 2 of manual)

For flow of control information for BSAM/BPAM routines, see Figure 25 in section 2 and Diagram G in this section.

Control blocks used in BSAM are shown in section 4. See Figure 41, BSAM Control Blocks.

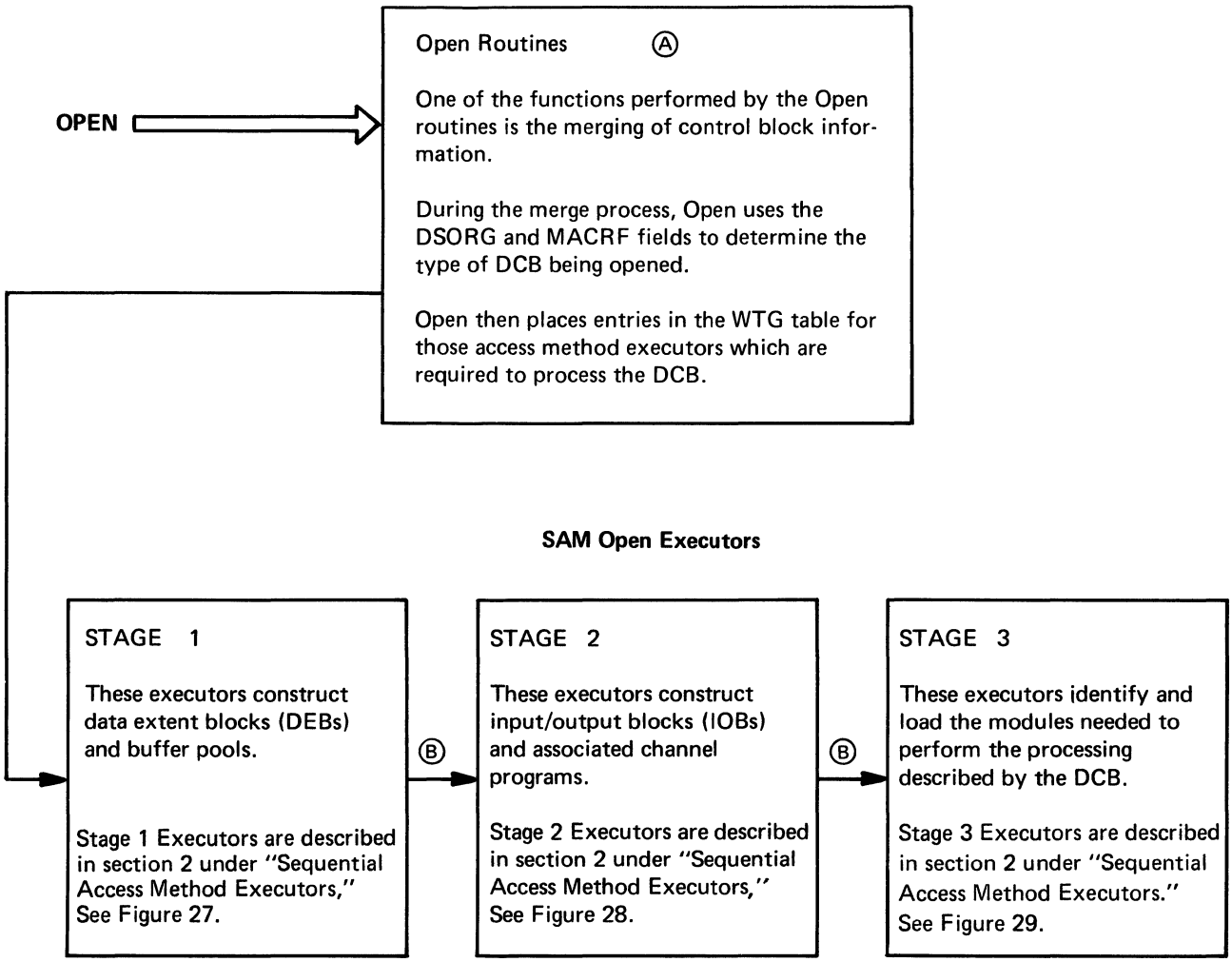
CHECK ⟹

The DECB is examined by a Check routine to determine the status of the channel program.

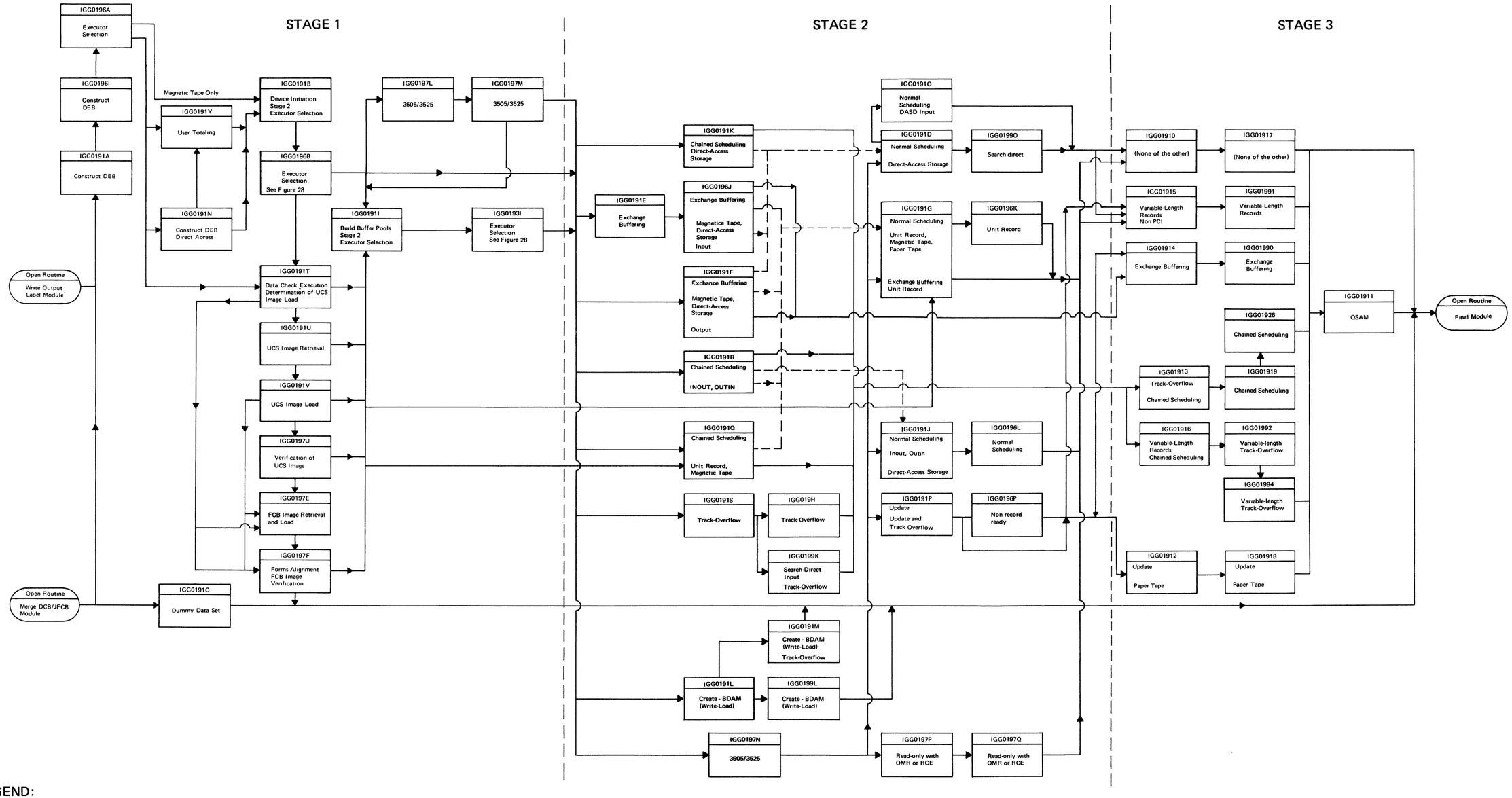
The Check modules are listed in Figure 21 (section 2 of manual)

For flow of control information for BSAM/BPAM routines, see Figure 25 in section 2 and Diagram G in this section.

Control blocks used in BSAM are shown in section 4. See Figure 41, BSAM Control Blocks.



Ⓐ	Open routines are described in Open/Close/End-of-Volume Logic, GY28-6609. For information on the WTG and XCTL tables, see the "Access Method Determination" section of the manual.
Ⓑ	Diagram E shows the flow of control between the three stages of Open Executors.



LEGEND:

——— Normal Program Flow

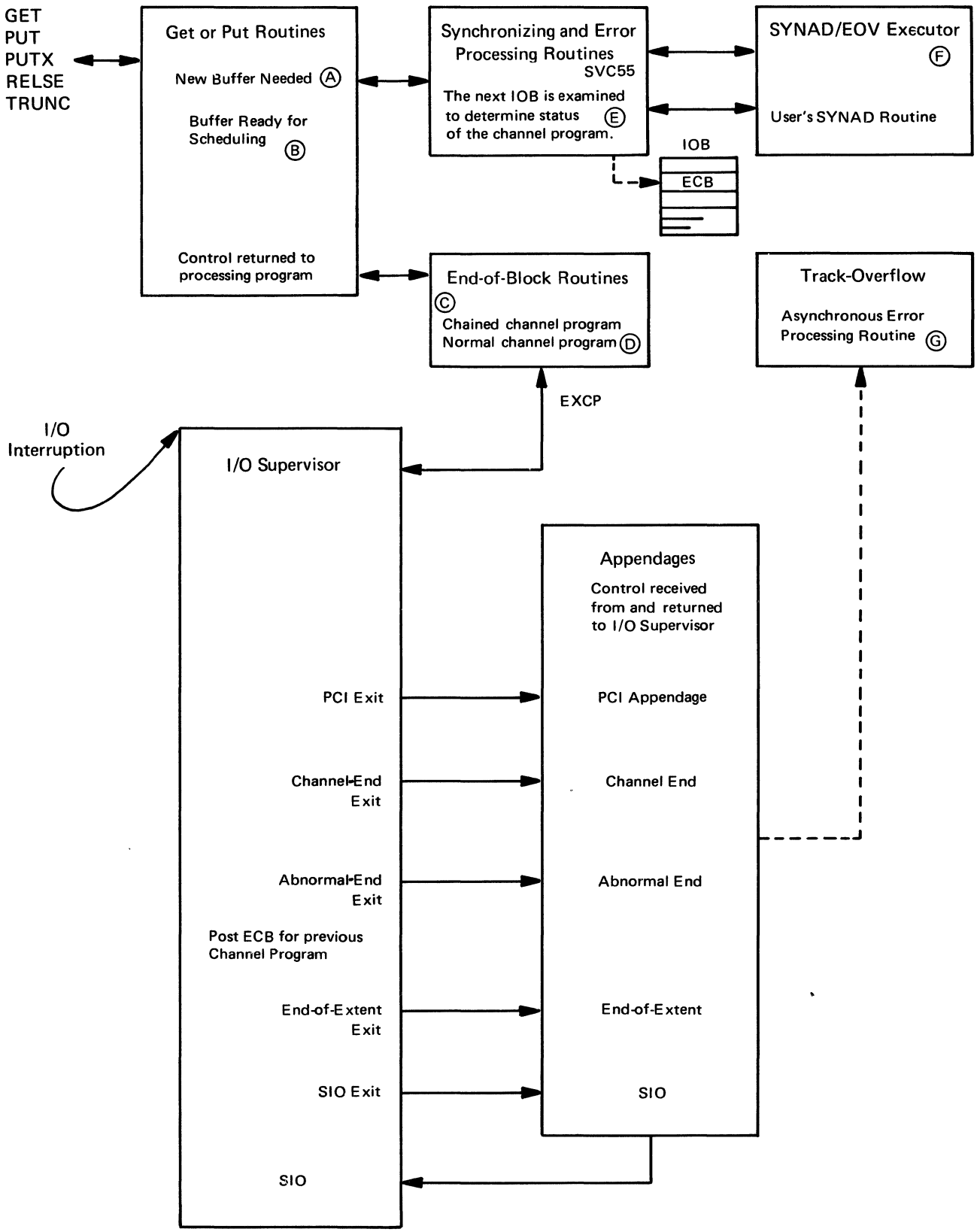
– – – – Abnormal Program Flow

QSAM Flow of Control

Notes for Diagram F

Ⓐ	A synchronizing-and-error-processing routine receives control when another full input buffer is needed or if a new empty output buffer is needed.
Ⓑ	An end-of-block routine receives control when an input buffer is empty or an output buffer is full.
Ⓒ	The end-of-block routine attempts to add the present channel program to the last one in the chain of scheduled channel programs. If successful, control returns to the processing program. If unsuccessful, control is passed to the I/O supervisor by an EXCP instruction.
Ⓓ	For normal channel-program scheduling, the routine passes control to the I/O supervisor by an EXCP instruction to cause scheduling of the buffer.
Ⓔ	Depending on the status of the execution, a synchronizing routine may retain control (using the WAIT macro instruction), return control to the Get or Put routine, or pass control to the user's SYNAD routine or to the SYNAD/EOV executor. Control is passed to the SYNAD/EOV executor by using an SVC 55 instruction in the event that an end-of-volume or a permanent error condition is detected. Refer to Figure 40, "QSAM Control Blocks" (section 4) for a diagram of the relationship of the IOBs to the other QSAM control blocks.
Ⓕ	The flow of control for the SYNAD/EOV executor IGC0005E is described in Diagram H, QSAM Flow of Control for SYNAD/EOV Executor.
Ⓖ	This executor receives control by being scheduled for execution by the track-overflow abnormal-end appendage IGG019C3. Control is passed to the processing program through the supervisor.

DIAGRAM F

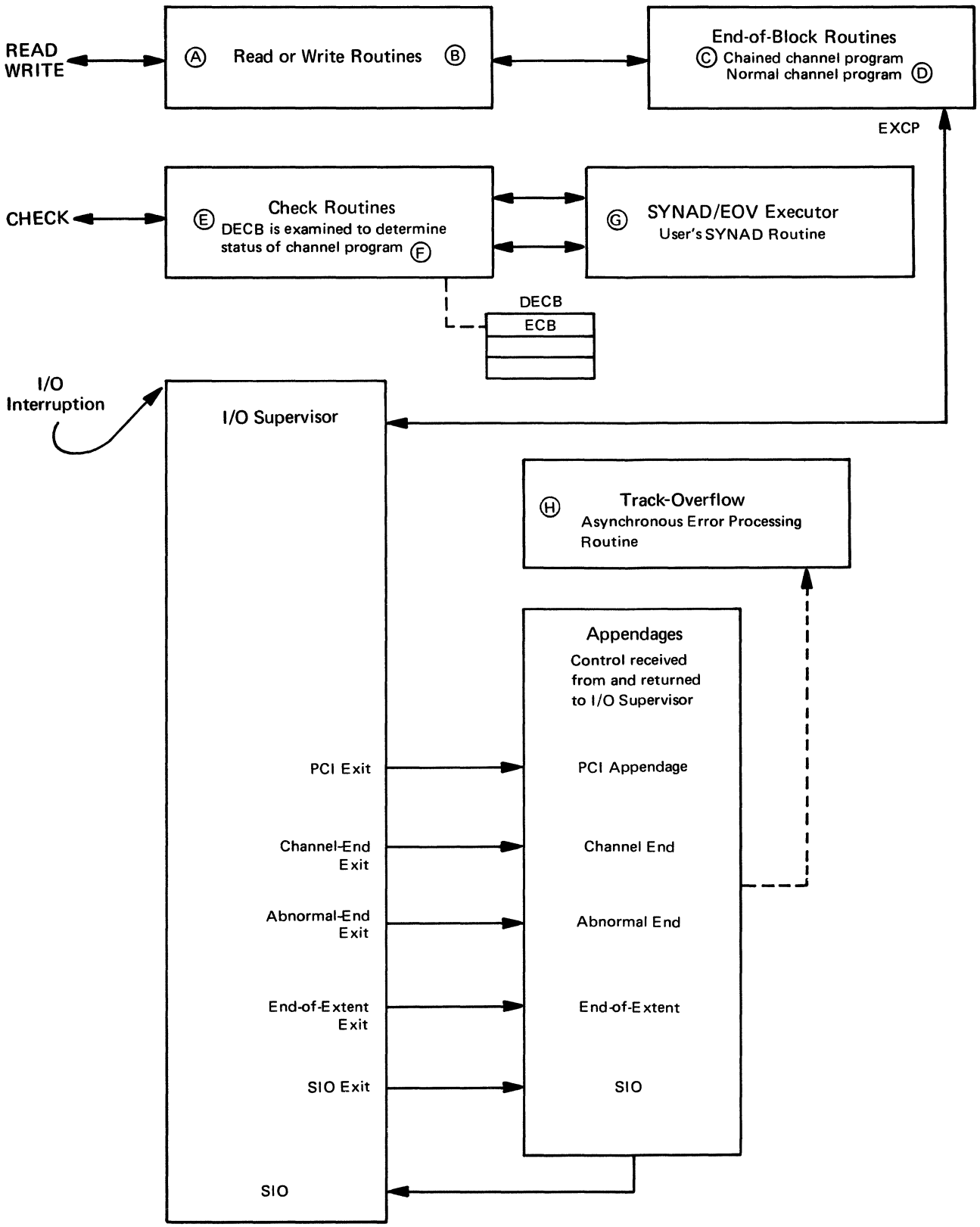

GET
PUT
PUTX
RELSE
TRUNC
Get or Put Routines
New Buffer Needed Ⓐ
Buffer Ready for Scheduling Ⓑ
Control returned to processing program
Synchronizing and Error Processing Routines
SVC55
The next IOB is examined to determine status of the channel program. Ⓔ
IOB
ECB
SYNAD/EOV Executor Ⓕ
User's SYNAD Routine
End-of-Block Routines Ⓒ
Chained channel program
Normal channel program Ⓓ
Track-Overflow
Asynchronous Error Processing Routine Ⓖ
EXCP
I/O Interruption
I/O Supervisor
Appendages
Control received from and returned to I/O Supervisor
PCI Exit
PCI Appendage
Channel End Exit
Channel End
Abnormal End Exit
Abnormal End
Post ECB for previous Channel Program
End-of-Extent Exit
End-of-Extent
SIO Exit
SIO
SIO

BSAM/BPAM Flow of Control

Notes for Diagram G

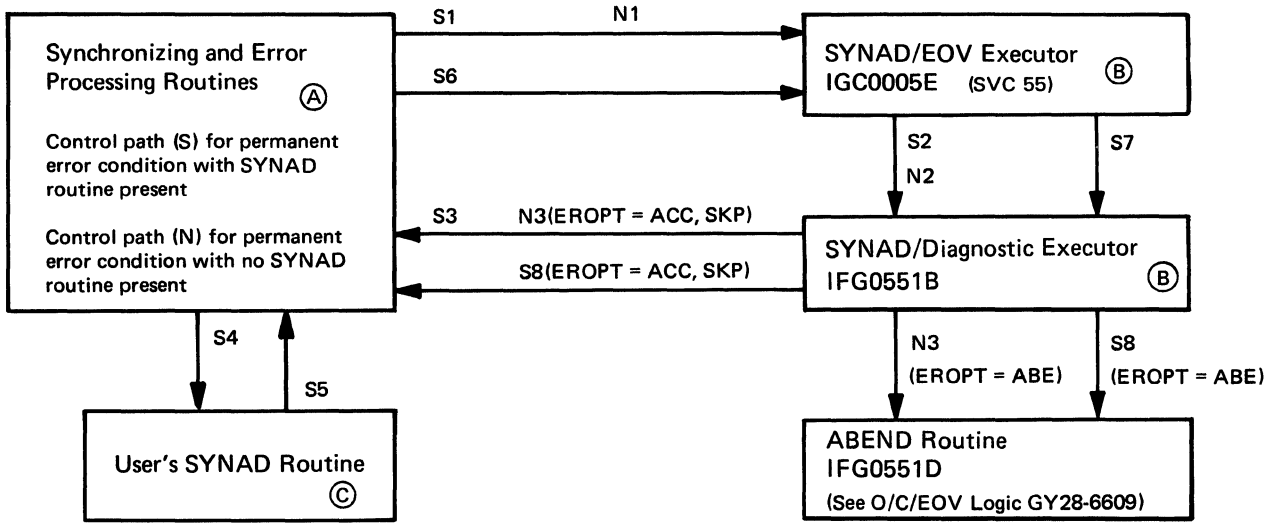
Ⓐ	A Read or a Write routine receives control after a READ or a WRITE macro instruction is issued by a processing program.
Ⓑ	A Read or Write routine partially completes a channel program using parameters from the data event control block (DECB), and passes the DECB, together with the Input/ Output block (IOB), to an end-of-block routine.
Ⓒ	The end-of-block routine attempts to add the present channel program to the last one in the chain of scheduled channel programs. If successful, control returns to the processing program. If unsuccessful, control is passed to the I/O supervisor by an EXCP instruction. These routines are described in the QSAM portion of the manual. See Figure 11.
Ⓓ	For normal channel program scheduling, the routine passes control to the I/O supervisor by an EXCP instruction to cause scheduling of the buffer. The end-of-block routines are described in the QSAM portion of the manual. See Figure 9.
Ⓔ	A Check routine receives control from the processing program via a CHECK macro instruction.
Ⓕ	A Check routine returns control to the processing program if the channel program executes normally (without errors). See Figure 41, BSAM Control Blocks (section 4) for a diagram of the relationship of the DECB to the other BSAM control blocks. Control is passed to the SYNAD/EOV executor for a permanent I/O error condition or for an end-of-volume condition. See Diagram I, (section 5) for further information about the flow of control between the Check routines and the SYNAD/EOV executor.
Ⓖ	The flow of control between the Check routine, SYNAD/EOV executor (SVC 55) and the user's SYNAD routine is described in Diagram I, BSAM Flow of Control for SYNAD/EOV executor.
Ⓗ	This executor receives control by being scheduled for execution by the track-overflow abnormal-end appendage IGG019C3. Control is passed to the processing program through the supervisor.

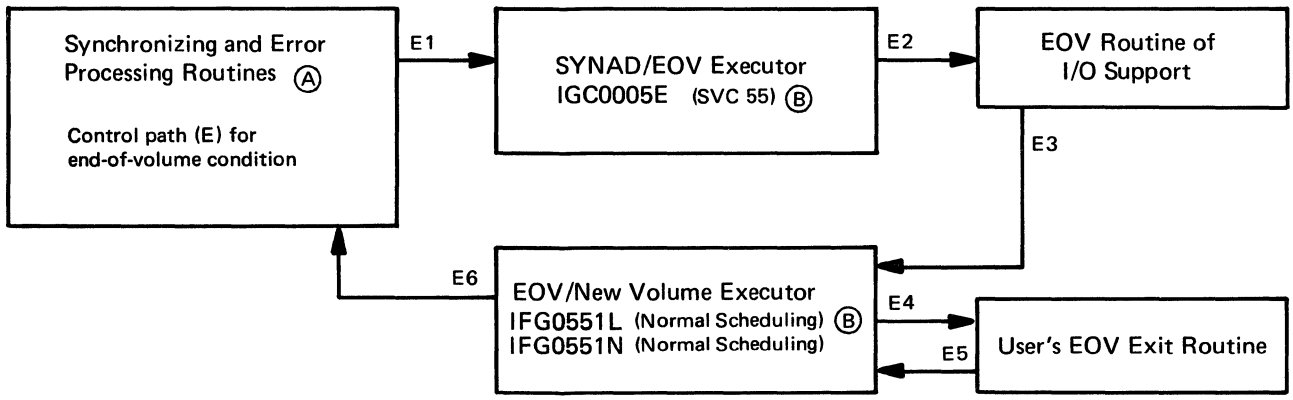
DIAGRAM G


READ
WRITE
Ⓐ Read or Write Routines Ⓑ
End-of-Block Routines
Ⓒ Chained channel program
Normal channel program Ⓓ
EXCP
CHECK
Ⓔ Check Routines
DECB is examined to determine status of channel program Ⓕ
Ⓖ SYNAD/EOV Executor
User's SYNAD Routine
DECB
ECB
I/O
Interruption
I/O Supervisor
Ⓗ Track-Overflow
Asynchronous Error Processing Routine
Appendages
Control received from and returned to I/O Supervisor
PCI Exit
PCI Appendage
Channel-End Exit
Channel End
Abnormal-End Exit
Abnormal End
End-of-Extent Exit
End-of-Extent
SIO Exit
SIO
SIO

QSAM Flow of Control for SYNAD/EOV Executor

DIAGRAM H

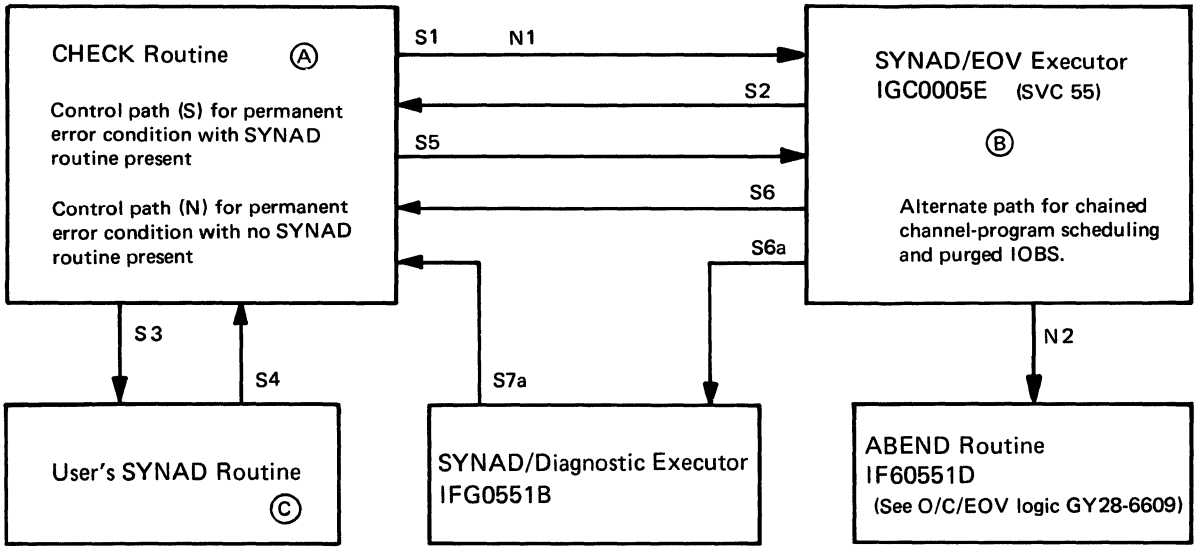


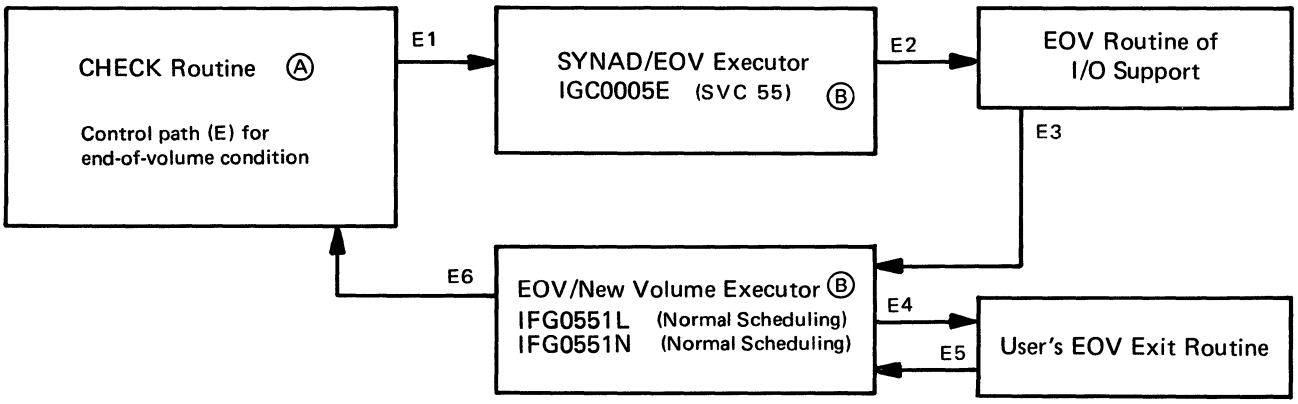


Ⓐ	Descriptive information on these routines is located in the QSAM portion of section 2 under "Synchronizing and Error Processing Routines." See module selector information in Figures 15 and 16.
Ⓑ	Executors IGC0005E, IFG0551L, IFG0551N, and IFG0551B are described in section 2. See Figure 31.
Ⓒ	The user's SYNAD routine is described in the *OS Data Management Services Guide,* GC26-3746.

BSAM Flow of Control for SYNAD/EOV Executor

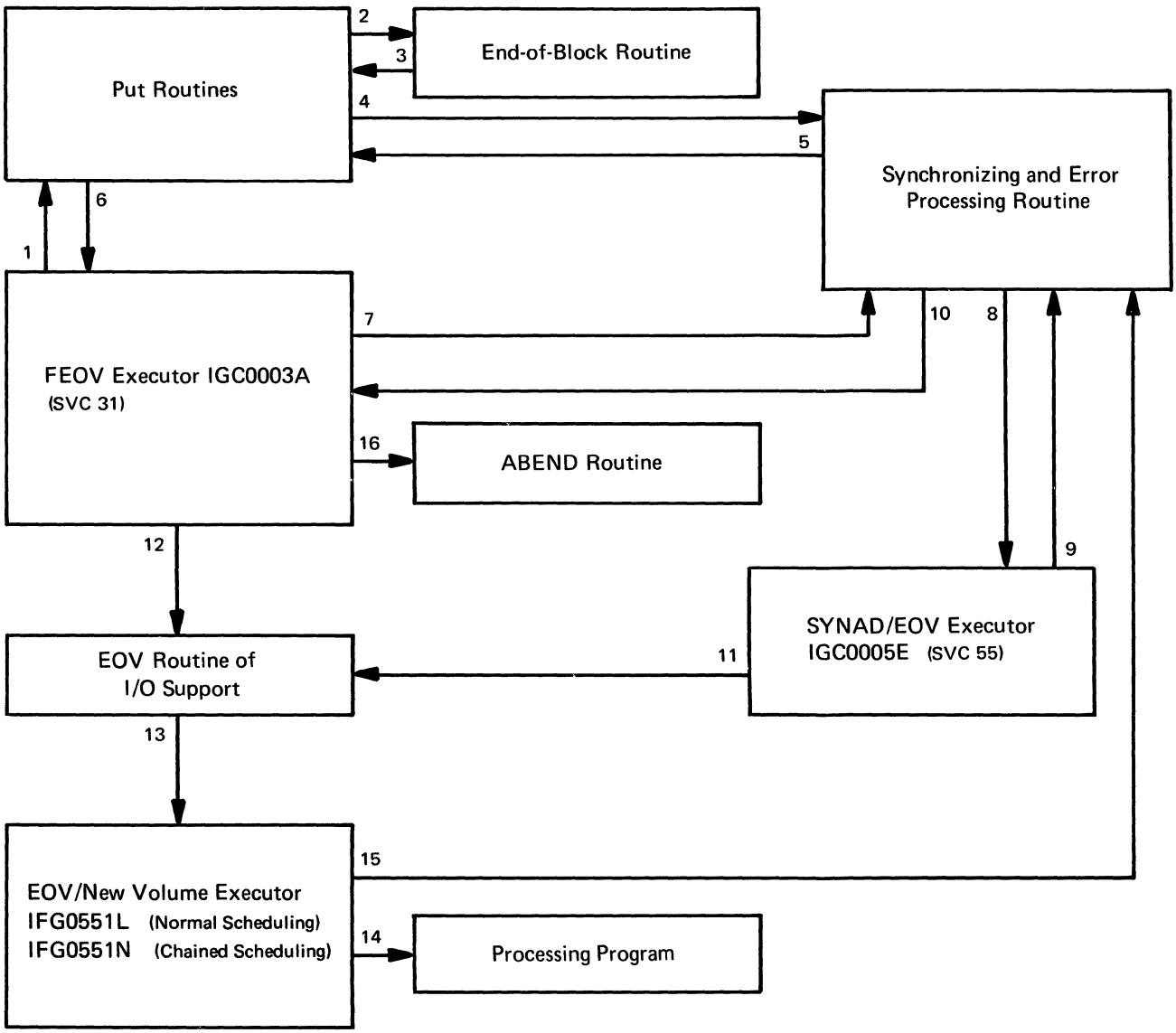
DIAGRAM I





Ⓐ	Descriptive information on the Check routines is located in section 2 under "Basic Sequential Access Method Routines." See Figure 21.
Ⓑ	Executors IFG0551L and IFG00551N are described in section 2. See Figure 31.
Ⓒ	The user's SYNAD routine is described in the *OS Data Management Services Guide,* GC26-3746.

QSAM Operation of FEOV Executor for output data set (IGG0003A) DIAGRAM J



Condition*	Sequence of Control
1	1,2,3,6,12,13,14
2	1,2,3,6,7,8,9,10,16
3	1,2,3,6,7,8,11,13,15,10,12,13,14
4	1,2,3,4,5,6,12,13,14
5	1,2,3,4,8,9,10,16
6	1,2,3,4,5,6,7,8,9,10,16
7	1,2,3,4,8,11,13,15,10,12,13,14
8	1,2,3,4,5,6,7,8,11,13,15,10,12,13,14

*These conditions are described in section 2 under "FEOV Executor IGC0003A," see Figure 31.

SECTION 6: APPENDIXES

A. Code Conversion Routines
Modules IGG019CM, IGG019CN, IGG019CO, IGG019CP, IGG019CQ, IGG019CR, and IGG0010C.

B. BSAM/QSAM Channel Programs
Modules IGG0191D, IGG0191E & IGG0196J, IGG0191F, IGG0191H, IGG0191J & IGG0196L, IGG0191K, IGG0191O, and IGG0199O.

C. Update Channel Programs
Module IGG0191P

D. Chained Scheduling Channel Programs
Modules IGG0191K and IGG0191R.

E. BSAM (BDAM Create) Channel Programs
Modules IGG0191M, IGG0199L, and IGG0199M

F. ABEND Codes Cross–Reference Table

Appendix A: Code Conversion Routines

Get routine IGG019AT (paper tape) and Write routine IGG019BF (paper tape) use the tables in the following modules to convert characters read from paper tape to EBCDIC characters.

Code Conversion Module IGG019CM

This module is loaded by the Open executor if the DCB specifies paper tape, and code conversion for teletype transmission code.

The module consists of three tables:

- A validity–checking and special functions table
- A lowercase character translation table
- An uppercase character translation table

Code Conversion Module IGG019CN

This module is loaded by the Open executor if the DCB specifies paper tape, and code conversion for ASCII paper tape code.

The module consists of two tables:

- A validity–checking and special functions table
- A character translation table

Code Conversion Module IGG019CO

This module is loaded by the Open executor if the DCB specifies paper tape and code conversion for Burroughs paper tape code.

The module consists of two tables:

- A validity–checking and special functions table
- A character translation table

Code Conversion Module IGG019CP

This module is loaded by the Open executor if the DCB specifies paper tape and type and code conversion for Friden paper tape code.

The module consists of three tables:

- A validity–checking and special functions table
- A lowercase character translation table
- An uppercase character translation table

Code Conversion Module IGG019CQ

This module is loaded by the Open executor if the DCB specifies paper tape and code conversion for IBM PTTC/8 code.

The module consists of three tables:

- A validity–checking and special functions table
- A lowercase character translation table
- An uppercase character translation table

Code Conversion Module IGG019CR

This module is loaded by the Open executor if the DCB specifies paper tape and code conversion for NCR paper tape code.

The module consists of three tables:

- A validity–checking and special functions table
- A lowercase character translation table
- An uppercase character translation table

Translate Routine IGG0010C

This routine is a type 2 SVC routine if made resident at system generation or a type 3 SVC routine if it is transient that translates data from EBCDIC to ASCII on output and ASCII to EBCDIC on input. It is entered when an XLATE macro instruction (SVC 103) is issued.

The routine must be given the following parameters:

- Register 0 = Length of area to be translated
- Register 1

  Byte 0 — bit 0 = 0: Translate from ASCII to EBCDIC

  bit 0 = 1: Translate from EBCDIC to ASCII

  bits 1–7: Not used

  Byte 1–3: Address of area that contains record to be translated.

Appendix B: BSAM/QSAM Channel Programs

CHANNEL PROGRAM FOR OUTPUT, SIMPLE BUFFERING (IGG0191D)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	WRT CKD	CCW8[1]	DS	8
CCW4		DCBBUFCB[2]	C[4]S	DCBBLKSI[2]
	(WRITE CHECK)			
CCW5	SCH ID EQ	CCW8	C	5
CCW6	TIC	*-8		
CCW7	READ KD		K	
CCW8	CCHHRKDD[1]			
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	WRT CKD	CCW11[3]	DS	8
CCW5		DCB BUFCB[2]	DS	DCBBLKSI[2]
CCW6	RD SECTOR	SECTOR2	C[4]	1
	(WRITE CHECK)			
CCW7	SET SECTOR	SECTOR2	C	1
CCW8	SCH ID EQ	CCW11	C	5
CCW9	TIC	*-8		
CCW10	RD KD		K	
CCW11	CCHHRKDD[3]			

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip

1. If there is no write check, CCHHRKDD will be in CCW5.
2. For QSAM only.
3. If there is no write check, CCHHRKDD will be in CCW7.
4. For write-check only.

CHANNEL PROGRAM FOR INPUT, EXCHANGE BUFFERING, STANDARD FORMAT-F (IGG0191E and IGG0196J)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	M/T RD DATA	BUFFER	D	LRECL[1]
.				
.				
.				
CCW3	M/T RD DATA	BUFFER+ (N-1)DCBPRECL		LRECL
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	M/T RD DATA	BUFFER	D	LRECL[1]
.				
.				
.				
CCW4	M/T RD DATA	BUFFER+ (N-1)DCBPRECL	C	LRECL
CCW5	M/T RD CNT		CK	8
CCW6	RD SECTOR	SECTOR2		1

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip

1. Number of CCWs = Blocking factor.

CHANNEL PROGRAM FOR EXCHANGE BUFFERING, NOT STANDARD FORMAT-F, INPUT (IGG0191E and IGG0196J)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	TIC	*+8		
CCW4[3]	M/T RD CNT	NEXT IOBSEEK+3[1]	CS	5
CCW5	M/T RD DATA	BUFFER	DS	LRECL[2]
	.			
	.			
	.			
CCW5	M/T RD DATA	BUFFER+ (N-1)DCBPRECL	S	LRECL
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	TIC	*+8		
CCW5[4]	M/T RD CNT	CCW8	C	8
CCW6	M/T RD DATA	BUFFER	DS	LRECL
.	.			
.	.			
.	.			
CCW6	M/T RD DATA	BUFFER+ (N-1)DCBPRECL	S	LRECL
CCW7	RD SECTOR	SECTOR2		1
CCW8	CCHHRKDD			

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip

1. If there is only one IOB, store address of DCBFDAD+3.
2. Number of CCWs = Blocking factor.
3. If there is a search-direct, CCW4 and CCW5 are interchanged.
4. If there is a search-direct, CCW5 and CCW6 are interchanged.

CHANNEL PROGRAM FOR OUTPUT, EXCHANGE BUFFERING (IGG0191F)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	WRT CKD	CCW8	DS	8
CCW4[1]	01	BUFFER	DS	LRECL
.				
.				
CCW4	01	BUFFER+ (N-1)LRECL	C[2]S	LRECL
	(WRITE CHECK)			
CCW5	SCH ID EQ	CCW8	C	5
CCW6	TIC	*-8		
CCW7	M/T RD DATA		K	BUFL
CCW8	CCHHRKDD[2]			
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	WRT CKD	CCW11	DS	8
CCW5[1]	01	BUFFER	DS	LRECL
.				
.				
.				
CCW5	01	BUFFER+ (N-1)LRECL	CS	LRECL
CCW6	RD SECTOR	SECTOR2	C[3]	1
	(WRITE CHECK)			
CCW7	SET SECTOR	SECTOR2	C	1
CCW8	SCH ID EQ	CCW11	C	5
CCW9	TIC	*-8		
CCW10	M/T RD DATA		K	BUFL
CCW11	CCHHRKDD[3]			

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip

1. The number of CCWs is proportionate to the blocking factor.
2. If there is no write-check, CCHHRKDD will be in CCW5, and the command chain bit in CCW4 will be off.
3. If there is no write-check, CCHHRKDD will be in CCW7, and the command chain bit in CCW6 will be off.

CHANGE PROGRAM FOR TRACK-OVERFLOW, INPUT, NON-FORMAT-U (IGG0191H)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	RD DATA		CSK	7
CCW4	RD CNT	DCBFDAD+3[1]	CS	5
CCW5	RD DATA[2]	BUFAD[3]	S[4]	
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	RD DATA		CSK	7
CCW5	RD CNT	DCBFDAD+3[1]	C	5
CCW6	RD DATA	BUFAD[3]	CS	
CCW7	RD SECTOR	SECTOR2		1

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip

1. If there is more than one IOB, then address is IOBSEEK+3 of next IOB.
2. If keys are specified, READ KD.
3. For QSAM only.
4. SILI bit is on for format-V records.

Note: Format-U with track-overflow is not supported by RPS.

CHANNEL PROGRAM FOR OUTPUT, TRACK-OVERFLOW, NON-FORMAT-U (IGG0191H)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8	TIC	ADDR CCW7
CCW3	WRT S CKD [1]	A COUNT AT END OF C. P.	D	8
CCW4	null	DATA ADDR	C [2]	KEY+SEGMENT DATA LNTH [3]
CCW5 [4]	M/T SCH ID EQ [5]	SRCH ADDR AT END OF C. P.	C	5
CCW6	TIC	*-8	0	SRCH ADDR IN CCW5
CCW3				
CCW4				
	(WRITE CHECK)			
CCW7	SEEK CYL	IOBSEEK+1	C	6
CCW8	SCH ID EQ	ADDR OF FIRST SEG COUNT FLD	C	5
CCW9	TIC	*-8		
CCW10	RD KD	0	K	

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip

1. Op code is WRT CKD for the last record segment.
2. Chaining is turned off in this CCW of the last segment if no validity checking is specified.
3. The key is written on the first record segment only.
4. CCWs 5 and 6 are required if more than one segment is to be written. Each segment but the first is written by CCWs 5, 6, 3, 4, in that order.
5. CCW5 op code and address may be overlaid with the parasitic TIC from CCW2 if not all the channel program segments are needed, but validity-checking is specified.

CHANNEL PROGRAM FOR TRACK-OVERFLOW, OUTPUT, NON-FORMAT-U (IGG0191H)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8	TIC	ADDR CCW8
CCW4	WRT S CKD [1]	COUNT AT END OF C. P.	D	8
CCW5	null	DATA ADDR	C	KEY+SEG DATA LNTH [3]
CCW6 [4]	M/T SCH ID EQ [5]	SRCH ADDR AT END OF C. P.	C	5
CCW7	TIC	*-8	0	ADDR OF SRCH ARG IN CCW5
CCW4				
CCW5				
CCW8	RD SECTOR (WRITE CHECK)	SECTOR2	C [2]	1
CCW9	SEEK CYL	IOBSEEK+1	C	6
CCW10	SET SECTOR	SECTOR2	C	1
CCW11	SCH ID EQ	ADDR FRST SEG CNT	C	5
CCW12	TIC	*-8		
CCW13	RD KD	0	K	

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip

1. The op code is WRT CKD for the last record segment.
2. Chaining is turned off in this CCW of the last segment if no validity checking is specified.
3. The key is written on the first record segment only.
4. CCWs 6 and 7 are required if more than one segment is to be written. Each segment but the first is written by CCWs 6, 7, 4, and 5. The last segment is followed by the parasitic TIC to CCW8 if not all the channel program segments are used.
5. CCW6 op code and address may be overlaid with the parasitic TIC from CCW3 if not all the channel program segments are needed.

CHANNEL PROGRAM FOR INOUT, OUTIN (IGG0191J and IGG0196L)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	M/T RD DATA		CSK	1
CCW4	M/T RD CNT	NEXT IOBSEEK+3[1]	CS	5
CCW5	M/T RD KD[2]		S[3]	
CCW6	SCH ID EQ	IOBSEEK+3	C	5
CCW7	TIC	*-8		
CCW8	WRT CKD	CCW13	DS	8
CCW9	00 (WRITE CHECK)		S	
CCW10	SCH ID EQ	CCW13	C	5
CCW11	TIC	*-8		
CCW12	M/T RD KD		SK	
CCW13	additional search argument			

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING AND WITH SEARCH-DIRECT				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	TIC	*+8		
CCW4	M/T RD KD[2]		CS	
CCW5	M/T RD CNT		C	
CCW6	00			
CCW7	00			
CCW8	SCH ID EQ		C	
CCW9	TIC	*-8		
CCW10	WRT CKD			
CCW11	00			
CCW12	00			
CCW13	SCH ID EQ	CCW13	C	5
CCW14	TIC	*-8		
CCW15	M/T RD KD		SK	

FLAGS

D = Data Chain
C = Common Chain
S = SILI
K = Skip

1. If there is only one IOB, the address is the DCBFDAD+3.
2. Read both key and data only if key is specified.
3. SILI bit is on for format-U and format-V records.

CHANNEL PROGRAM FOR INOUT, OUTIN (IGG0191J and IGG0196L)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	M/T RD DATA		CSK	1
CCW5	M/T RD CNT	CCW19	C	8
CCW6	M/T RD KD[1]		CS	
CCW7	RD SECTOR	SECTOR2		1
CCW8	SET SECTOR	SECTOR1	C	1
CCW9	SCH ID EQ	IOBSEEK+3	C	5
CCW10	TIC	*-8		
CCW11	WRT CKD	CCW18	DS	8
CCW12	00		CS	
CCW13	RD SECTOR (WRITE CHECK)	SECTOR2	C	1
CCW14	SET SECTOR	SECTOR2	C	1
CCW15	SCH ID EQ	CCW18	C	5
CCW16	TIC	*-8		
CCW17	M/T RD KD		SK	
CCW18	additional search argument			
CCW19	additional search argument			

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITH ROTATIONAL POSITION SENSING AND WITH SEARCH-DIRECT				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	TIC	* +8		
CCW5	M/T RD KD[1]		CS	
CCW6	M/T RD CNT	CCW19	C	8
CCW7	RD SECTOR	SECTOR2		1
CCW8	SET SECTOR	SECTOR1	C	1
CCW9	SCH ID EQ	IOBSEEK+3	C	5
CCW10	TIC	*-8		
CCW11	WRT CKD	CCW18	DS	8
CCW12	00		CS	
CCW13	RD SECTOR (WRITE CHECK)	SECTOR2	C	1
CCW14	SET SECTOR	SECTOR2	C	1
CCW15	SCH ID EQ	CCW18	C	5
CCW16	TIC	*-8		
CCW17	M/T RD KD		SK	
CCW18	additional search argument			
CCW19	additional search argument			

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip

1. Read both key and data only if key is specified.

CHANNEL PROGRAM FOR PCI, DIRECT-ACCESS OUTPUT WITH WRITE-CHECK (IGG0191K)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	M/T SCH ID EQ	ICBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	WRT CKD	CCW9	DSP	8
CCW4	00	BUFFER	CS	BUF LENGTH
CCW5	SCH ID EQ	CCW9	C	5
CCW6	TIC	*-8		
CCW7	M/T RD DATA[1]		CK	1
CCW8	NOP	CCW1 or 3 in NEXT ICB	S	1
CCW9	CCHHRKDD			
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR = 0	C	1
CCW2	READ HA		CK	5
CCW3	M/T SCH ID EQ	ICBSEEK+3	C	5
CCW4	TIC	*-8		
CCW5	WRT CKD	CCW13	DSP	8
CCW6	00	BUFFER	CS	BUF LENGTH
CCW7	READ SECTOR	SECTOR 2	C	1
CCW8	SET SECTOR	SECTOR 2	C	1
CCW9	SCH ID EQ	CCW13	C	5
CCW10	TIC	*-8		
CCW11	M/T READ DATA[1]		CK	1
CCW12	NOP	CCW 1 or 5 in NEXT ICB	S	1
CCW13	CCHHRKDD			

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip
P = PCI

1. If keys are specified, READ KD.

CHANNEL PROGRAM FOR STANDARD FORMAT-F INPUT, SIMPLE BUFFERING (IGG0191O)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	READ KD [1]	DCBBUFCB [2]		DCBBLKSI [2]
WITH ROTATIONAL POSITION SENSING				
CCW1	SET S	SECTOR 1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	READ KD [1]	DCBBUFCB [2]	C	DCBBLKSI [2]
CCW5	M/T READ CNT		CSK	1
CCW6	READ S	SECTOR 2		1

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip

1. If key is not specified, read data only.
2. For QSAM only.

CHANNEL PROGRAM FOR NOT-STANDARD FORMAT-F SIMPLE BUFFERING, INPUT (IGG0191O)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	TIC	*+8	CSK	
CCW4	RD COUNT	NEXT IOBSEEK+3[2]	CS	5
CCW5	RD KD[3]	DCB BUFCB[1]	S	DCBBLKSI[1]
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	TIC	*+8	CSK	
CCW5	RD COUNT	CCW8	C	8
CCW6	RD KD[3]	DCBBUFCB[1]	CS	DCBBLKSI[1]
CCW7	RD SECTOR	SECTOR2		
CCW8	CCHHRKDD			

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip

1. For QSAM only.
2. If there is one IOB, then address is DCBFDAD+3.
3. If the data set has no keys, then the op code is RD D.

CHANNEL PROGRAM FOR NOT-STANDARD FORMAT-F, SIMPLE BUFFERING, INPUT, SEARCH-DIRECT (IGG0199O)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQU	IOBSEEK+3	C	5
CCW2	TIC	*–8		
CCW3	TIC[1]	*+8[1]	C	8
CCW4	M/T RD D[2,3]		C	D_L
CCW5	M/T CNT	*+8		
CCW6	CCHHR$K_L D_L D_L$			
CCW7	CCHHR$K_L D_L D_L$			
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	CS	1
CCW2	SCH ID EQU	IOBSEEK+3	C	5
CCW3	TIC	*–8		
CCW4	TIC[1]	*+8[1]	C	
CCW5	M/T RD D[2,3]	BUFFER	CS	D_L
CCW6	M/T RD CNT	*+16	C	8
CCW7	READ SECTOR	SECTOR2		1
CCW8	CCHHR$K_L D_L D_L$			
CCW9	CCHHR$K_L D_L D_L$[1]			

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip
P = PCI

1. For a search on record 0, this CCW is changed to an M/T RD CNT and its address field is CCW7.
2. If keys are present, the op code is RD KD and count is $K_L + D_L$.
3. For exchange buffering, there will be as many RD D CCWs as the blocking factor.

Appendix C: Update Channel Programs

CHANNEL PROGRAM FOR DIRECT ACCESS, UPDATE, NO TRACK OVERFLOW (IGG0191P)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	WRT DATA[1] (WRITE CHECK)[8]	BUFFER[2]	CS[3]	DCBBLKSI[2]
CCW4	SEEK CYL	IOBSEEK+1	C	6
CCW5	SCH ID EQ	IOBSEEK+3	C	5
CCW6	TIC	*-8		
CCW7	RD DATA[4]		C[5]S[3]K	DCBBLKSI[2]
CCW8	SEEK CYL[6]	CCW 14+1	C	6
CCW9	SCH ID EQ	CCW 14+3	C	5
CCW10	TIC	*-8		
CCW11	TIC	*+8		
CCW12	RD COUNT	NEXT CCW 14[7]	CS	5
CCW13	RD DATA[4]	BUFFER[2]	S[3]	DCBBLKSI[2]
CCW14	MBBCCHHR			

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip

1. If key is specified, WRT KD.
2. For QSAM only.
3. SILI bit is on for format-U or format-V records.
4. If key is specified, READ KD.
5. Command chain off for BSAM.
6. CCW8 is present for QSAM only.
7. If there is only one IOB, the address field is DCBFDAD+3.
8. CCWs 4-7 are present for write-check.

CHANNEL PROGRAM FOR DIRECT-ACCESS UPDATE, NO TRACK-OVERFLOW (IGG0191P)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR A	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	WRT DATA[1] (WRITE CHECK)[8]	BUFFER	C[7]S	DCBBLKSI[2]
CCW5	SEEK CYL	IOBSEEK+1	C	6
CCW6	SET SECTOR	SECTOR A	C	1
CCW7	SCH ID EQ	IOBSEEK+3	C	5
CCW8	TIC	*-8		
CCW9	RD DATA[3]		C[4]SK	DCBBLKSI[2]
CCW10	SEEK CYL[5]	CCW18+1	C	6
CCW11	SET SECTOR	SECTOR1	C	1
CCW12	SCH ID EQ	CCW18+3	C	5
CCW13	TIC	*-8		
CCW14	TIC	*+8		
CCW15	RD COUNT	CCW19[6]	C	8
CCW16	RD DATA	BUFFER[2]	CS	DCBBLKSI[2]
CCW17	RD SECTOR	SECTOR2		1
CCW18	MBBCCHHR			
CCW19	CCHHRKDD			

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip

1. If key is specified, WRT KD.
2. For QSAM only.
3. If key is specified, READ KD.
4. Command chain is off for BSAM.
5. CCW10 is present for QSAM only.
6. If there is only one IOB, the address field is DCBFDAD+3.
7. Command chain is off for BSAM if write-check is not specified.
8. CCWs 5-9 are present for write-check.

CHANNEL PROGRAM FOR DIRECT-ACCESS UPDATE, TRACK-OVERFLOW (IGG0191P)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	WRT DATA[1] (WRITE CHECK)[8]	BUFFER[2]	CS[3]	DCBBLKSI[2]
CCW4	SEEK CYL	IOBSEEK+1	C	6
CCW5	SCH ID EQ	IOBSEEK+3	C	5
CCW6	TIC	*-8		
CCW7	RD DATA[4]		C[5]S[3]K	DCBBLKSI[2]
CCW8	SEEK CYL[6]	CCW14+1	C	6
CCW9	SCH ID EQ	CCW14+3	C	5
CCW10	TIC	*-8		
CCW11	RD DATA		CSK	50 (non-zero length)
CCW12	RD COUNT	NEXT CCW14[7]	CS	5
CCW13	RD DATA[4]	BUFFER[2]	S[3]	DCBBLKSI[2]
CCW14	MBBCCHHR			

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip

1. If key is specified, WRT KD.
2. For QSAM only.
3. SILI bit is on for format-U and format-V records.
4. If key is specified, READ KD.
5. Command chain is off for BSAM.
6. CCW8 is present for QSAM only.
7. If there is only one IOB, the address field is DCBFDAD+3.
8. CCWs 4-7 are present for write-check.

CHANNEL PROGRAM FOR DIRECT-ACCESS UPDATE, TRACK-OVERFLOW (IGG0191P)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR A	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	WRT DATA[1] (WRITE CHECK)[7]	BUFFER	C[6]S	DCBBLKSI[2]
CCW5	SEEK CYL	IOBSEEK+1	C	6
CCW6	SET SECTOR	SECTOR A	C	1
CCW7	SCH ID EQ	IOBSEEK+3	C	5
CCW8	TIC	*-8		
CCW9	RD DATA[3]		C[4]SK	DCBBLKSI[2]
CCW10	SEEK CYL[5]	CCW18+1	C	6
CCW11	SET SECTOR	SECTOR1	C	1
CCW12	SCH ID EQ	CCW18+3	C	5
CCW13	TIC	*-8		
CCW14	RD DATA		CSK	50 (nonzero length)
CCW15	RD COUNT	NEXT CCW18	CS	5
CCW16	RD DATA	BUFFER[2]	CS	DCBBLKSI[2]
CCW17	RD SECTOR	SECTOR2		1
CCW18	MBBCCHHR			

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip

1. If key is specified, WRT KD.
2. For QSAM only.
3. If key is specified, READ KD.
4. Command chain is off for BSAM.
5. CCW10 is present for QSAM only.
6. Command chain is off for BSAM if write-check is not specified.
7. CCWs 5-9 are present for write-check.

Appendix D: Chained Scheduling Channel Programs

CHANNEL PROGRAM FOR MAIN IOB, PCI (IGG0191K)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	TIC	ICB CH PGM [1]		
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	TIC	ICB CH PGM [1]		

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip
P = PCI

1. TIC goes to the first ICB channel program in the chain.

CHANNEL PROGRAM FOR PCI, INPUT, DIRECT ACCESS (IGG0191K)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	M/T RD COUNT	NEXT ICBSEEK+3	CSP	5
CCW2	M/T RD DATA[1]	BUFFER	CS[2]	BUF LENGTH
CCW3	NOP	CCW1 IN NEXT ICB	S	1
WITH ROTATIONAL POSITION SENSING				
CCW1	M/T RD COUNT	NEXT ICBSEEK+3	CP	8
CCW2	M/T RD DATA[1]	BUFFER	CS	BUF LENGTH
CCW3	TIC (HEX'88')	*+8[3]	S	
CCW4	RD SECTOR	SECTOR 2		1

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip
P = PCI

1. If keys are specified, the command code is READ KD.
2. SILI bit is on for format-U and format-V records.
3. Appendage may change to CCW1 in the next ICB.

CHANNEL PROGRAM FOR PCI, DIRECT ACCESS, OUTPUT, NO WRITE-CHECK (IGG0191K)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	M/T SCH ID EQ	ICBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	WRT CKD	CCW6	DSP	8
CCW4	00	BUFFER	CS	BUF LENGTH
CCW5	NOP	CCW1 or 3 in NEXT ICB	S	1
CCW6	CCHHRKDD			
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR = 0	C	1
CCW2	M/T SCH ID EQ	ICBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	WRT CKD	CCW8	DSP	8
CCW5	00	BUFFER	CS	BUF LENGTH
CCW6	TIC (X'88')	* +8[1]	S	
CCW7	RD SECTOR	SECTOR 2		1
CCW8	CCHHRKDD			

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip
P = PCI

1. Appendage may change to CCW1 or 4 in the next ICB.

CHANNEL PROGRAM FOR PCI, INOUT, OUTIN, ICB, WITHOUT WRITE-CHECK (IGG0191R)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITH RECORD READY				
CCW1	M/T RD CNT	NEXT ICBSEEK+3	CSP	5
CCW2	M/T RD DATA		CS [1]	
CCW3	TIC (X'88')	* +8 [2]	S	
CCW4	READ SECTOR	SECTOR2		1
CCW5	M/T SCH ID EQ	ICBSEEK+3	CS	5
CCW6	TIC	*-8		
CCW7	WRT CKD	CCW11	DSP	8
CCW8			CS	
CCW9	TIC ('88')	* +8 [2]	S	
CCW10	READ SECTOR	SECTOR2		1
CCW11	CCHHRKDD			

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip
P = PCI

1. SILI bit is on for format-U and format-V records.
2. Appendage may change to the next ICB CCW.

CHANNEL PROGRAM FOR PCI, INOUT, OUTIN, ICB WITH WRITE-CHECK (IGG0191R)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITH ROTATIONAL POSITION SENSING				
CCW1	M/T RD CNT	NEXT ICBSEEK+3	CSP	5
CCW2	M/T RD DATA		CS	
CCW3	TIC ('88')	*+8[1]	S	
CCW4	READ SECTOR	SECTOR2		1
CCW5	M/T SCH ID EQ	ICBSEEK+3	CS	5
CCW6	TIC	*-8		
CCW7	WRT CKD	CCW15	DSP	8
CCW8			CS	
CCW9	READ SECTOR	SECTOR2	C	1
CCW10	SET SECTOR	SECTOR2	C	1
CCW11	SCH ID EQ	CCW15	CS	5
CCW12	TIC	*-8		
CCW13	M/T RD DATA		CSK	
CCW14	NOP	NEXT ICB CCW7	S	1
CCW15	CCHHRKDD			

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip
P = PCI

1. Appendage may change to next ICB CCW.

CHANNEL PROGRAM FOR PCI, INOUT, OUTIN, ICB (IGG0191R)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	M/T RD CNT	NEXT ICBSEEK+3	CSP	5
CCW2	M/T RD DATA		CS[1]	1
CCW3	NOP	NEXT ICB CCW2	S	
CCW4	M/T SCH ID EQ	ICBSEEK+3	CS	5
CCW5	TIC	*-8		
CCW6	WRT CKD	CCW12	DSP	8
CCW7			CS	
	(WRITE CHECK[2])			
CCW8	SCH ID EQ	CCW12	CS	5
CCW9	TIC	*-8		
CCW10	M/T RD DATA		CSK	
CCW11	NOP	NEXT ICB CCW6	S	1
CCW12	CCHHRKDD			

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip
P = PCI

1. SILI bit is on for format-U and format-V records.
2. If there is no request for a write-check, CCWs 8-10 are omitted and CCW7 has no command chaining.

CHANNEL PROGRAM FOR PCI, MAIN JOB, INOUT, OUTIN (IGG0191R)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	TIC	ICB CH PGM [1]		
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	TIC	ICB CH PGM [1]		

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip
P = PCI

1. TIC goes to the first ICB channel program in the chain.

Appendix E: BSAM (BDAM Create) Channel Programs

CHANNEL PROGRAM FOR ERASE CCWs FOR BSAM LOAD MODE, TRACK-OVERFLOW (IGG0191M)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1[1]	SCH ID EQ	CCW10	C	5
CCW2	TIC	*-8		
CCW3	WRT DATA	CCW10	CS	7
CCW4	SCH ID EQ	CCW10	C	5
CCW5	TIC	*-8		
CCW6	RD DATA		CSK	8
CCW7	ERASE	CCW7	CS	8
CCW8	M/T RD HA		C	5
CCW9	TIC	CCW1		
CCW10	R0 ADDR = CCHH0000			
WITH ROTATIONAL POSITION SENSING				
CCW1[1]	SET SECTOR	SECTOR = 0	C	1
CCW2	SRCH ID EQ	CCW13	C	5
CCW3	TIC	*-8		
CCW4	WRT DATA	CCW13	CS	7
CCW5	SET SECTOR	SECTOR = 0	C	1
CCW6	SCH ID EQ	CCW13	C	5
CCW7	TIC	*-8		
CCW8	RD DATA		CSK	8
CCW9	ERASE	CCW9	CS	8
CCW10	SET SECTOR	SECTOR = 0	C	1
CCW11	M/T RD HA		C	5
CCW12	TIC	CCW1		
CCW13	R0 ADDR CCHH0000			

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip

1. Address of CCW1 is stored in DCBEOBW

CHANNEL PROGRAM FOR BSAM LOAD MODE, TRACK-OVERFLOW (IGG0191M)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	WRT CKD	IOBDNRCF (1)	D	8
CCW4				
CCW5[1]	TIC/NOP[2]	CCW20	CS	CCW20
CCW6	SCH ID EQ	IOBR0CNT (1)	C	5
CCW7	TIC	*-8		
CCW8	WRT DATA	IOBR0DAT (1)	C	8
CCW9[1]	READ R0		SK	16
CCW10[3]	M/T SCH ID EQ	IOBR0CNT (2)	C	5
CCW11	TIC	*-8		
CCW12	WRT CKD	IOBDNRCF (2)	D	8
CCW13				
CCW14[1]	TIC/NOP[2]	CCW19	CS	CCW19
CCW15	SCH ID EQ	IOBR0CNT (2)	C	5
CCW16	TIC	*-8		
CCW17	WRT DATA	IOBR0DAT (2)	C	8
CCW18[1]	READ R0 (WRITE CHECK)		SK	16
CCW19	SEEK CYL	IOBSEEK+1	C	6
CCW20	SCH ID EQ	IOBDNRCF (1)	C	5
CCW21	TIC	*-8		
CCW22	RD KD		SK	K_L+D_L

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip

1. CCWs 5, 9, 14, and 18 are omitted if verify is not specified.
2. The TIC/NOP at CCW5 and CCW14 is set to NOP if R0 is to be written on this track.
3. CCWs 10-18 are repeated as many times as are needed to write all segments.

CHANNEL PROGRAM FOR CREATE BDAM (IGG0199L)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITHOUT ROTATIONAL POSITION SENSING				
CCW1	SCH ID EQ	IOBSEEK+3	C	5
CCW2	TIC	*-8		
CCW3	WRT CKD	IOBDNRCF	D	8
CCW4	WRT CKD (WRITE CHECK)[1]		C	K_1+BLKSIZE
CCW5	SCH ID EQ	IOBDNRCF	C	5
CCW6	TIC	*-8		
CCW7	RD KD		CSK	256
CCW8	SCH ID EQ	IOBR0CNT	C	5
CCW9	TIC	*-8		
CCW10	WRT DATA	IOBR0DAT	C	8
CCW11	SCH ID EQ[2]	IOBR0CNT	C	5
CCW12	TIC	*-8		
CCW13	RD DATA		CSK	1
CCW14	ERASE[3]	*	S	8

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip

1. CCWs 5-7 are omitted if write-check is not specified.
2. CCWs 11-13 are always generated for format-U and format-V records, or if write check is specified.
3. CCW14 is generated for format-U and format-V records only.

CHANNEL PROGRAM FOR CREATE BDAM (IGG0199L)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	WRT CKD	IOBDNRCF	D	8
CCW5	WRT CKD		C	K_L+BLKSIZE
CCW6	RD SECTOR (WRITE CHECK)[1]	SECTOR2	C	1
CCW7	SET SECTOR	SECTOR2	C	1
CCW8	SCH ID EQ	IOBDNRCF	C	5
CCW9	TIC	*-8		
CCW10	RD KD		CSK	256
CCW11	SET SECTOR	SECTOR = 0	C	1
CCW12	SCH ID EQ	IOBROCNT	C	5
CCW13	TIC	*-8		
CCW14	WRT DATA	IOBRODAT	C	8
CCW15	SET SECTOR[3]	SECTOR2	C	1
CCW16	SCH ID EQ	IOBROCNT	C	5
CCW17	TIC	*-8		
CCW18	RD DATA		CSK	1
CCW19	ERASE[2]	*	S	8

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip

1. CCWs 7-10 are omitted if write-check is not specified.
2. CCW19 is generated for format-U and format-V records only.
3. CCWs 15-18 are always generated for format-V and format-U records or if write-check is specified.

CHANNEL PROGRAM FOR BSAM LOAD MODE, TRACK OVERFLOW (IGG0199M)

CCW #	COMMAND CODE	ADDRESS	FLAGS	COUNT
WITH ROTATIONAL POSITION SENSING				
CCW1	SET SECTOR	SECTOR1	C	1
CCW2	SCH ID EQ	IOBSEEK+3	C	5
CCW3	TIC	*-8		
CCW4	WRT CKD	IOBDNRCF (1)	D	8
CCW5	00			
CCW6[1]	TIC/NOP[2]	CCW24	CS	CCW24
CCW7	SET SECTOR	SECTOR=0	C	1
CCW8	SCH ID EQ	IOBR0CNT (1)	C	5
CCW9	TIC	*-8		
CCW10	WRT DATA	IOBR0DAT (1)	C	8
CCW11	SET SECTOR	SECTOR=0	C	1
CCW12	READ R0		SK	16
CCW13	M/T SCH ID EQ	IOBR0CNT (2)	C	5
CCW14	TIC	*-8		
CCW15	WRT CKD	IOBDNRCF (2)	D	8
CCW16				
CCW17[1]	TIC/NOP[2]	CCW24	CS	CCW24
CCW18	SET SECTOR	SECTOR=0	C	1
CCW19	SCH ID EQ	IOBR0CNT (2)	C	5
CCW20	TIC	*-8		
CCW21	WRT DATA	IOBR0DAT (2)	C	8
CCW22	SET SECTOR	SECTOR=0	C	1
CCW23	READ R0		SK	16
CCW24	RD SECTOR	SECTOR2	C	1
CCW25	SEEK CYL	IOBSEEK+1	C	6
CCW26	SET SECTOR	SECTOR1	C	1
CCW27	SCH ID EQ	IOBDNRCF (1)	C	5
CCW28	TIC	*-8		
CCW29	RD KD		SK	K_L+D_L

FLAGS

D = Data Chain
C = Command Chain
S = SILI
K = Skip

1. CCWs 11, 12, 22, 23, and 25-29 are omitted if verify is not specified.
2. The TIC/NOP at CCW6 and CCW17 is set to NOP if R0 is to be written on this track.

Appendix F: ABEND Codes and Cross–Reference Table

ABEND Code	Return Code	Module	Logic Manual Reference
003	01	IGG019CC	Figure 9
	02	IGG019CE	Figure 9
		IGG019CF	Figure 9
		IGG019FK	Figure 9
	03	IGG019FA	Figure 18
		IGG019FQ	Figure 9
004	01	IGG0197N	Figure 28
	02	IGG0197N	Figure 28
	03	IGG0197N	Figure 28
	04	IGG0197N	Figure 28
	05	IGG0197M	Figure 27
	06	IGG0197Q	Figure 28
013	10	IGG0191C	Figure 27
	14	IGG0191B	Figure 27
	18	IGG0191B	Figure 27
	1C	IGG0191B	Figure 27
	20	IGG0191A	Figure 27
	24	IGG0191A	Figure 27
	28	IGG0191A	Figure 27
	2C	IGG0196J	Figure 28
	30	IGG0191F	Figure 28
	34	IGG0191I	Figure 27
	38	IGG0191H	Figure 28
	3C	IGG0191D	Figure 28
	40	IGG01910	Figure 29
	44	IGG0191K	Figure 28
	48	IGG01911	Figure 29
	4C	IGG0196B	Figure 27
	50	IGG0196B	Figure 27
	54	IGG0196A	Figure 27
	58	IGG01912	Figure 29
	5C	IGG01915	Figure 29
		IGG01916	Figure 29
	60	IGG0191A	Figure 27
B13	04	IGG0191U	Figure 27
	08	IGG0191U	Figure 27
	0C	IGG0191V	Figure 27
	10	IGG0197U	Figure 27
	14	IGG0197U	Figure 27
	18	IGG0197F	Figure 27
	1C	IGG0197F	Figure 27

ABEND Code	Return Code	Module	Logic Manual Reference
B14	04	IGG0201B	Figure 30
	08	IGG0201B	Figure 30
		IGG0201Z	Figure 30
	0C	IGG0201B	Figure 30
		IGG0201Z	Figure 30
	10	IGG0201B	Figure 30
		IGG0201Z	Figure 30

INDEX

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

C

D

E

G

I

L

M

N

O

P

Q

R

S

T

U

W

X

READER'S COMMENT FORM

OS SAM Logic

Order Number GY28-6604-5

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

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

Reply requested

Yes ☐

No ☐

Name _____

Job Title _____

Address _____

_____ Zip _____

No postage necessary if mailed in the U S A

Order Number GY28-6604-5

YOUR COMMENTS, PLEASE . . .

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

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

fold fold

FIRST CLASS
PERMIT NO. 2078
SAN JOSE, CALIF.

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

POSTAGE WILL BE PAID BY . . .

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

Attention: Programming Publications, Dept. D78

fold fold

IBM

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

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

Order Number GY28-6604-5

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

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