Concepts underlying the data-management capabilities of OS/360 *are introduced; distinctive features of the access methods, catalog, and relevant system macroinstructions are discussed.*

To illustrate the way in which the control program adapts to actual input/output requirements, a read operation is examined in considerable detail.

The functional structure of OS/360

Part III Data management

by W. A. Clark

The typical computer installation is confronted today with an imposing mass of data and programs. Moreover, with the applicable technologies developing at a rapid pace, the current trend is toward increasing diversity and change in input/output and auxiliary storage devices. Together, these factors dictate that the so-called "input/output" process be viewed in new perspective. Whereas the support provided by a conventional input/output control system is usually limited to data transfer and label processing, the current need is for a data management system that encompasses identification, storage, survey, and retrieval needs—for programs as well as data. Not only should the system employ the capabilities of both direct-access and serial-access devices, but ideally should be able to satisfy a storage or input/output requirement with any storage device that meets the functional specifications of the given requirement.

Our purpose here is to discuss the main structural aspects of OS/360 from the standpoint of data management. In identifying, storing, and retrieving programs and data via OS/360, a programmer normally reckons with device classes rather than specific devices. Because actual devices are not assigned until job-step execution time, a novel degree of device independence is achieved. Moreover, as befits a system intended for a wide range of applications, OS/360

provides for several data organizations and search schemes. Various buffering and transmittal options are provided.

Background

Although the data management services provided by OS/360 are deliberately similar to those provided by predecessor systems, the system breaks with the past in the manner in which it adapts to specific needs.

compiled I/O routines

For mobilizing the input/output routines needed in a given job step, one well-known scheme places these routines into the user's program during the compilation process. No post-assembly program fetching or editing is then required; a complete, executable program results. This scheme has significant disadvantages. It requires that a fairly complete description of device types and intended modes of operation be stated in the source program. Compilation is made more difficult by having to concern itself with details of the input/output function, and compiled programs can be made obsolete by environmental changes that affect the input/output function.

interpretive I/O routines

These disadvantages led the designers of some prior operating systems, for example, IBSYS/IBJOB, to circumvent the inclusion of input/output routines in assembled programs by providing a set of input/output "packages" that could be mobilized at program-loading time.[1] Designed to operate interpretively, these optional packages permitted a source program to be less specific about devices and operating modes; moreover, they permitted change in the input/output environment without program reassembly. On the other hand, interpretive execution tends to reduce the efficiency of packages and limit the feasible degree of system complexity and expandability.

generated I/O routines

Faced with unprecedented diversity in storage devices and potential applications in addition to the complexities of multitask operation, the OS/360 designers have carried the IBSYS/IBJOB philosophy further, but with a number of significant tactical differences. Data-management control facilities are not obtained at program-loading time; instead, they are tailored to current needs during the very course of program execution (wherever the programmer uses an OPEN macroinstruction). The data-access routines are reenterable, and different tasks with similar needs may share the same routines. Because routines do not act interpretively, they can be highly specialized as well as economical of space. A program chooses one of the available access methods and requests input/output operations using appropriate macroinstructions. Device types, buffering techniques, channel affinities, and data attributes are later specified via data-definition statements in the job stream. In fact, the OS/360 job stream permits final specification of nearly every data or processing attribute that does not require re-resolution of main-storage addresses in an assembled program. These attributes include blocking factors, buffering techniques, error checks, number of buffers, and the like.

System definitions

An operating system deals with many different categories of information. Examples from a number of categories are a source program, an assembled program, a set of related subroutines, a message queue, a statistical table, and an accounting file. Each of these examples consists of a collection of related data items. In the OS/360 context, such a collection is known as a *data set*. In the operational sense, a data set is defined by a data-set label that contains a name, boundaries in physical storage, and other parameters descriptive of the data set. The data-set label is normally stored with the data set itself.

volume

A standard unit of auxiliary storage is called a *volume*. Each direct-access volume (disk pack, data cell, drum, or disk area served by one access mechanism) is identified by a volume label. This label always contains a volume serial number; in the case of direct-access devices, it also includes the location of a *volume table of contents* (VTOC) that contains the labels of each data set stored in the corresponding volume. A label to describe the VTOC and an additional label to account for unused space are created. Before being used in the system, each direct-access volume is initialized by a utility program that generates the volume label and, for direct-access devices, constructs the table of contents. This table is designed to hold labels for the data sets to be written on the volume.

Given the volume serial number and data-set name, the control program can obtain enough information from the label to access the data set itself.

A job step can place a data set in direct-access storage via a data definition (DD) statement that requests space, specifies the kind of volume, and gives the data-set name. At job-step initiation, the system allocates space and creates a label for each area requested by a DD statement. Finally, during job-step execution, the label is completed and updated via OPEN and CLOSE macro-instructions.

Each reel of magnetic tape is considered a volume. In view of the serial properties of tape, the method used for identifying volumes and data sets departs somewhat from the method used for direct-access devices. The standard procedure still employs volume labels and data-set labels; but each data-set label exists in two parts: a header label preceding its data set, and a trailer label that follows it. The location of a data set in a tape volume is represented by a sequence number that facilitates tape searching.

Although the system includes a generalized labeling procedure, it permits a user to employ his own tape-label conventions and label-checking routines if so desired. Unlabeled tapes may be used, in which case the responsibility for mounting the right volumes reverts to the operator.

To free the programmer of the need to maintain inventories of his data sets, the system provides a *data-set catalog*. Held in

direct-access storage, this catalog consists of a tree-organized set of indexes. To best serve the needs of individual installations, the organization of the tree structure is left to the user. Each qualifier of a data-set name corresponds to an additional level in the tree. For example, the data set PAYROLL.MASTER.SEGMENT1 is found by searching a master index for PAYROLL, a second-level index for MASTER, and a third-level index for SEGMENT1. Stored with the latter argument are entries that identify the volume containing the data set and the device type; in the case of serial-access devices, a sequence number is also stored.

catalog

A volume containing all or part of the catalog is called a *control volume*. Normally, the operating system resides in a control volume known as the *system residence volume*. The use of a distinctive control volume for a group of related data sets makes it convenient to move the portion of the catalog that is relevant to the group. This is particularly important in planning for the possibility that groups of data sets may be moved from one computer to another.

control volume

A data-set search starts in a system residence volume and continues, level by level, until a volume identification number is obtained. If the required volume is not already mounted, a message is issued to the operator. Then, if the data set is stored in a direct-access device, the search for the data-set location resumes with the volume label of the indicated volume, continues in the volume table of contents, and proceeds from there to the data set's starting location. On the other hand, if the data set is held on a serial-access device, the search continues using a sequence number as an argument.

To simplify DD (data definition) statements for recurrent updating jobs, data sets related by name and cataloging sequence can be identified as a *generation group*. In applications that regularly use the n most prior generations of a group to produce a new generation, the new generation may be named (and later referred to) relative to the most recent generation. Thus, the DD statement need not be changed from run to run. When the index for the generation group is established, the programmer specifies n. As each new generation is cataloged, the oldest generation is deleted from the catalog. Provision is also made for the special case in which n varies systematically, starting at 1 and increasing by 1 until it reaches a user-specified number N, at which time it starts over at 1.

generation group

To safeguard sensitive data, any data set may be flagged in its label as "protected." This protection flag is tested as a consequence of the OPEN instruction; if the flag is on, a correct *password* must be entered from the console. The data set name and appended password serve as an argument for searching a control table. The OPEN routine is not permitted to continue unless a matching entry is found in the table.

password

Because the control table has its own security flag and *master password*, it can be reached only by the control program and those programmers privileged to know the master password.

record and block

In discussing the internal structure and disposition of a data set, it is necessary to distinguish between the *record*, an application-defined entity, and the *block*, which has hardware-defined boundaries and is governed by operational considerations. Let b denote block length (in bytes) and B a maximum block length. Although OS/360 requires that B be specified for each given data set, conventions permit three block-format categories: unspecified, variable, and fixed. The first category requires that $b \leq B$ for all blocks. The second is similar to the first, except that each b is stored in a count field at the beginning of its block. The third category dictates that all blocks be of length B.

A fixed or variable block may contain multiple records. A fixed block contains records of fixed size. In the variable block, records may vary in size, and each record is preceded by a field that records its size. For storage devices that employ interblock gaps, it is well known that record blocking can increase effective data rates, conserve storage, and reduce the needed number of input/output operations in processing a data set. For data sets of unspecified block format, the system makes no distinction between block and record; any applicable blocking and deblocking must be done by the user's program. The unspecified format is intended for use with peripheral equipment, such as transmission devices, address label printers, and the like.

buffer

A *buffer* is a main storage area allocated to the input/output function. The portion of a buffer that holds one record is called a *buffer segment.* A group of buffers in an area of storage formatted by the system is called a *buffer pool*; a data set associated with a buffer pool is assigned buffers from the pool. Unless a programmer assigns a buffer pool to a data set, OS/360 does so; unless buffer size is specified by the programmer, OS/360 sets the size to B.

In processing records from magnetic tape, it is customary to read and process records from one or more data sets, and to create one or more new data sets. A number of buffering considerations come into play. It may suffice to process a record within a buffer; it may be preferable to move the input record to a work area and the updated record from the work area into an output buffer; other possibilities may suggest themselves. Moreover, in processing records from direct-access storage, the same data set may be accessed for input and output.

transmittal modes

To allow flexibility in buffer usage, the OS/360 record-transfer routines invoked by the GET and PUT macroinstructions permit three *transmittal modes.* In the "move" mode, each record is moved from an input buffer to a work area and finally to an output buffer. In the "locate" mode, a record is never actually moved, but a pointer to the record's buffer segment is made available to the application program. In the "substitute" mode, which also uses pointers, the application program provides a work area equal in size to a record, and the buffer segment and work area effectively change roles.

To supplement the transmittal modes in special cases, three

methods of buffer allocation are defined. *Simple buffering,* the most general method, allocates one or more buffers to each data set. *Exchange buffering,* used with fixed-length records, utilizes data-chaining facilities to effect record gather and record scatter operations. Buffer segments from an input data set are exchanged with buffer segments of an output data set or work area. Not only can each buffer segment be treated, in turn, as an input area, work area, and output area, but chaining allows noncontiguous segments to simulate a block. Exchange buffering is particularly useful in updating sequential files, merging, and array manipulation.

buffer allocation

Chained-segment buffering is designed for messages of variable size. Segments are established dynamically, with chaining being used to relate physically separate segments. This method is designed to circumvent the need for a static allocation of space to a remote terminal: of the many terminals that can be present in a system, only a fraction are ordinarily in use at a given time.

Access principles

To fall within the OS/360 data-management framework, a data set must belong to one of five organizational categories. As will be seen, the classification is based mainly on search considerations.

Data sets consisting of records held in serial-access storage media (such as magnetic tapes, paper tapes, card decks, or printed listings) are said to possess *sequential* organization. If so desired, a data set held in a direct-access device may also be organized sequentially.

data-set categories

Three of the five categories apply solely to direct-access devices. The *indexed sequential* organization stores records in sequence on a key (record-contained identifier). Because the system maintains an index table that contains the locations of selected records in the sequence, records can be accessed randomly as well as sequentially. A *direct* organization is similar, but dispenses with the index table and leaves record addressing entirely up to the programmer. A *partitioned* organization divides a sequentially organized data set into *members;* member names and locations are held in a directory for the data set. A member consists simply of one or more blocks. Included primarily for data sets consisting of programs or subroutines, this organization is suitable for any data set of randomly retrieved sequences of blocks.

A *telecommunications* organization is provided for queues of messages received from or enroute to remote on-line terminals. Provision is made for forming message queues and for retrieving messages from queues. Queues may be held in direct-access storage as well as in main storage.

A broad distinction is made between two classes of data-management languages. Designed for programming simplicity, the *queued access* languages apply only to organizations with sequential properties. The programmer typically uses the macro-

language categories

Table 1

Organization	*Language category* *Queued*	*Basic*
Sequential	QSAM	BSAM
Indexed Sequential	QISAM	BISAM
Direct		BDAM
Partitioned		BPAM
Telecommunication	QTAM	BTAM

instructions GET and PUT to retrieve and store records, and buffers are managed automatically by the system. On the other hand, the *basic access* languages provide for automatic device control, but not for automatic buffering and blocking. Typically, the READ and WRITE macroinstructions are used to retrieve and store blocks of data. Because the programmer retains control over device-dependent operations (such as card reader or punch-stacker selection, tape backspacing, and the like), he may use any desired searching, buffering, or blocking methods.

access methods

Of the ten possible combinations of data-set and language categories, eight are recognized by the system as *access methods*. These eight methods bear the mnemonic names given in Table 1: QSAM denotes "queued sequential access method," and so on. For each access method, a vocabulary of suitable macroinstructions is provided.

To employ a given access method, a programmer resorts to the vocabulary of macroinstructions provided for that method. Vocabularies for six of the methods are summarized in Table 2. Although six macroinstructions are common to all of these methods, the parameters to be specified in a macroinstruction may vary from method to method. If so desired for specialized applications, a programmer can circumvent the system-supported access methods and employ the *execute channel program* (EXCP) macroinstruction in fashioning his own access method. In this case, he must prepare his own channel program (sequence of channel command words).

A few words on each vocabulary element of Table 2 help to clarify access principles. At assembly time, the DCB macroinstruction reserves space for a *data control block* and fills in control block fields that designate the desired access method, name a relevant DD statement, and select some of the possible options. The application programmer is expected to provide symbolic addresses of any applicable supplementary routines, as for example, special label-processing routines.

The programmer issues an OPEN macroinstruction for each data control block. At execution time, OPEN supplies information not declared in the DCB macroinstruction, selects access routines and establishes linkages, issues volume mounting messages to the operator, verifies labels, allocates buffer pools, and positions

Table 2 Access-method vocabularies

Macro-instruction	QSAM	QISAM	BSAM	BPAM	BISAM	BDAM	*Macroinstruction function in brief*
DCB	•	•	•	•	•	•	Generate a data control block
OPEN	•	•	•	•	•	•	Open a data control block
CLOSE	•	•	•	•	•	•	Close a data control block
BUILD	•	•	•	•	•	•	Structure named area as a buffer pool
GETPOOL	•	•	•	•	•	•	Allocate space to and format buffer pool
FREEPOOL	•	•	•	•	•	•	Liberate buffer-pool space
GET	•	•					Obtain a record from an input data set
PUT	•	•					Include a record in an output data set
PUTX	•	•					Include an input record in an output data set
RELSE	•	•					Force end of input block
TRUNC	•						Force end of output block
FEOV	•		•				Force end of volume
CNTRL	•		•				Control reader or printer operation
PRTOV	•		•				Test for printer carriage overflow
SETL		•					Set lower limit for scan
ESETL		•					Postpone fetching during scan
CHECK			•	•			Wait for I/O completion and verify proper operation
NOTE			•	•			Note where a block is read or written
POINT			•	•			Point to a designated block
FIND				•			Obtain the address of a named member
BLDL				•			Build a special directory in main store
STOW				•			Update the directory
RELEX					•	•	Release exclusive control of a block
FREEDBUF					•	•	Free dynamically obtained buffer
GETBUF			•	•	•	•	Assign a buffer from the pool
FREEBUF			•	•	•	•	Return a buffer to the pool
WAIT			•	•	•	•	Wait for I/O completion
READ			•	•	•	•	Read a block
WRITE			•	•	•	•	Write a block

volumes. The programmer may free a data control block and return associated buffers to the pool by the CLOSE macroinstruction; if he omits CLOSE, the system performs the corresponding functions at task termination.

The programmer can request the system to allocate and format a buffer pool at execution time by issuing a GETPOOL macroinstruction, which specifies the address of the data control block, the buffer length, and the desired number of buffers. When a pool area is no longer needed, it can be returned to the system by FREEPOOL.

Where the programmer's knowledge permits him to allocate space more wisely than the control program, he may choose to designate the area to be set aside for a buffer pool. The area may,

for example, supplant a subroutine that is no longer needed. By issuing a BUILD macroinstruction, he can request the system to employ the reserved area as a buffer pool, the details being similar to GETPOOL. With subsequent BUILD's, moreover, he can restructure the area again and again.

QSAM

QSAM corresponds closely to the schemes most favored in previous input/output systems. QSAM yields a great deal of service to the programmer for a minimum investment in programming effort. Retrieval is afforded by GET, which supplies one record to the program; disposition of an output record is afforded by PUT or PUTX. PUT transfers a record from a work area or buffer to a data set; PUTX transfers a record from one data set to another. In consequence, PUT involves one data control block, whereas PUTX involves two.

To aid the programmer in creating short blocks and in disposing of a block before all records therein have been processed, two macroinstructions permit intervention in buffer control. RELSE requests the system to release the remaining buffer segments in an input buffer, i.e., to view the buffer as empty. Analogously, TRUNC asks the system to view an output buffer as full, and to go on to another buffer.

FEOV requests the system to force an end-of-volume status for a designated data set, and thereupon to undertake the normal volume-switching procedure. CNTRL provides for specialized card-reader, printer, or tape control functions.

QISAM

The QISAM scheme is closely akin to QSAM, but the macroinstructions provide the additional functions required of indexed sequential data sets and direct-access devices. Records are arranged in logical sequence on the key, a field that is part of each record. Record keys are related to physical addresses by indexes. For a record with a given data key, a *cylinder index* yields cylinder address, and a *track index* yields track-within-cylinder address. To facilitate in-channel searches, the key of the last record in each block is placed in a hardware-defined control field.

In the initial creation of a data set, PUT's are used in the "load" mode to store records and generate indexes. Successive GET's in the "scan" mode retrieve records sequentially; SETL (set lower limit) may be issued to designate the first record obtained. Unless a SETL is issued, retrieval starts from the first record of the data set. In scan mode, PUTX may follow a GET to return an updated record to the data set. ESETL (end of scan) halts any anticipatory buffering on the part of the system until issuance of a subsequent GET.

BISAM

BISAM applies to the same sequential data organization as QISAM, but selective reading and writing is permitted through the READ and WRITE macroinstructions. Using BISAM, new records can be inserted without destroying sequence. If an insertion does not fit into the intended track, the system moves one or more records from the track to an overflow area and then reflects this overflow status in the appropriate indexes. (The existence of over-

flows does not alter the ability of QISAM to scan records in logical sequence.)

To permit other operations to be synchronized with BISAM input/output operations, a WAIT macroinstruction supplements READ and WRITE. (Because WAIT serves a general function in synchronizing tasks, it is discussed in Part II.)

In a multitask environment, it is possible that one task may want to use or update a record while the record is being updated by another task. To forestall confusion in the order that updating operations are accomplished, READ can request exclusive control of the record during updating. For a record being updated in place, WRITE releases exclusive control. If the record is not updated in place, the RELEX macroinstruction can be used to release control.

Because record insertions may lead to overflows, and overflows tend to reduce input/output performance, the system is designed to provide statistics that can help a programmer in determining when data-set reorganization is desirable. Held in the data control block are the number of unused tracks in an independent overflow area and, optionally, the number of full cylinder areas, as well as the number of accesses to overflow records not appearing at the head of overflow chains. Reorganization can be accomplished via the QISAM load mode, using the existing data set as input.

As implied by the above discussion, QISAM and BISAM complement one another and may be used together where the user needs to access a data set randomly as well as sequentially. For the sake of convenience, a data control block for an indexed sequential data set can be opened jointly for QISAM and BISAM.

BSAM

BSAM assumes a sequentially organized data set and deals with blocks rather than records. A block is called into a specified buffer by READ. Unless program execution is deliberately suspended during the retrieval period by a CHECK macroinstruction, the program may continue during reading. Similarly, after an output operation is initiated, CHECK can be used to postpone further processing until the operation is completed.[2] Following a READ or WRITE, the macroinstruction NOTE saves the applicable block address in a standard register; subsequently, the preserved address may be helpful in logically repositioning the volume by POINT.

BDAM

Of the access methods for direct-access devices, BDAM offers the greatest variety of access possibilities. Using WRITE and READ, the programmer can store or retrieve a block from a data set by specifying a track address and block number. Optionally, he may specify a number relative to the data set itself, either (1) a relative track number at which a search should start for a given key or (2) a relative block number. The relative numbers, which help to isolate application programs from device peculiarities, are converted to actual track addresses and block numbers by the system. GETBUF and FREEBUF are the means by which buffers can be explicitly requested and released. A dynamic buffer option, requested in the DCB macroinstruction, enables the programmer to obtain automatic buffer management (BUILD and GETPOOL are

not used in conjunction with the option). The FREEDBUF macroinstruction permits release of a buffer under the dynamic option.

BPAM

BPAM is designed for storing and retrieving members of a partitioned data set held on a direct-access device. Associated with the data set is a *directory* that relates member name to track address. To prepare for access, a FIND performs the directory search. A located member can be retrieved using one or more READ's, as required by the number of blocks in the member. New members can be placed by one or more WRITE's, followed by a STOW that enters the member's name and location in the directory. CHECK again serves to synchronize the program with data-transmission operations.

A summary of the main characteristics of the eight access-methods appear in Table 3.

Control elements and system operation

data control block

With general definitions and access methods in mind, we turn to the internal structure of OS/360 as it pertains to data management.

Associated with each data set of a problem program is a data control block (DCB), which must be opened before any data transfer takes place. However, some data sets, e.g., the catalog data set, are opened automatically by the control program, and may be indirectly referred to or used in a problem program without additional opening or closing. Data-access macroinstructions, such as GET and PUT, logically refer to a data set, but actual reference is always via a data control block.

The data control block is generated and partially filled when the DCB macroinstruction is encountered at compilation time. The routine called at execution time by OPEN completes the data control block with information gained principally from a job-stream DD statement or cataloged procedure. For input data sets, a final source of such information is the data-set label. In the case of an output data set where the label has yet to be created, the final source can be the label of another data set or another DD statement.

In addition to completing the data control block, the OPEN routine ensures that needed access routines are loaded and address relations are completed. The routine prepares buffer areas and generates channel command word lists; it initializes data sets by reading or writing labels and performs a number of other housekeeping operations.

The selection of access routines is governed by choices in data organization, buffering technique, access language, input/output unit characteristics, and other factors. The selection is relayed to the supervisor, which allocates main storage space and performs the loading.

In operation, some access routines are treated as part of the user's program and are entered directly rather than through a supervisor-call interruption. These routines block and deblock

Table 3 Access-method summary

Characteristic	QSAM	QTAM	QISAM	BSAM	BTAM	BPAM	BISAM	BDAM
Data set organization	Sequential or member of partitioned	Telecom	Indexed sequential	Sequential	Telecom	Partitioned	Indexed Sequential	Direct
Basic element of data set	Record	Message or message segment	Record	Record	Message	Member	Record	Record
Basic concern of access method	Record	Message	Record	Block	Block	Block	Block	Block
Primary input and output macroinstructions	GET PUT PUTX	GET PUT RETRIEVE	Scan SETL Scan GET Load PUT Scan PUTX	READ WRITE	READ WRITE	FIND READ WRITE STOW	READ WRITE	READ WRITE
Buffer pool acquisition	BUILD GETPOOL Automatic	BUFFER	BUILD GETPOOL Automatic	BUILD GETPOOL Automatic	BUILD GETPOOL Automatic	BUILD GETPOOL Automatic	BUILD GETPOOL Automatic	BUILD GETPOOL Automatic
Buffer management for a data set	Automatic (simple or exchange)	Automatic (chained-segment)	Automatic (simple)	GETBUF FREEBUF	GETBUF FREEBUF	GETBUF FREEBUF	Dynamic FREEDBUF GETBUF FREEBUF	Dynamic FREEDBUF GETBUF FREEBUF
Transmittal mode	Move Locate Substitute	Move	Move Locate					
Synchronization	Automatic	Automatic	Automatic	CHECK WAIT	WAIT	CHECK WAIT	WAIT	WAIT
Record/block format*	F, V record	U message	F, V record	F, V record	U block	F, V, U block	F, V record	F, V, U block
Special provisions for data-set search		Sequence number; relative addressing	Cylinder & track indexes			Directory of members	Cylinder & track indexes	Can use relative record or track number

* F, V, and U denote fixed, variable, and unspecified lengths.

records, control the buffers, and call the input/output supervisor when a request for data input or output is needed. Other routines, treated as part of the I/O supervisor and therefore executed in the privileged mode, perform error checks, prepare user-oriented completion codes, post interruptions, and bridge discontinuities in the storage areas assigned to a data set.

I/O supervisor

The input/output supervisor performs all actual device control (as it must if contending programs are not to conflict in device usage); it accepts input/output requests, queues the requests if necessary, and issues instructions when a path to the desired input/output unit becomes available. The I/O supervisor also ensures that input/output requests do not exceed the storage areas allocated to a data set. The completion of each input/output operation is posted and, where necessary, standard input/output error-recovery procedures are performed. EXCP, the execute channel program macroinstruction, is employed in all communication between access routines and the input/output supervisor.

To portray the mechanics of data management, let us consider one job step and the data-management operations that support a READ macroinstruction for a cataloged data set in the BSAM context.

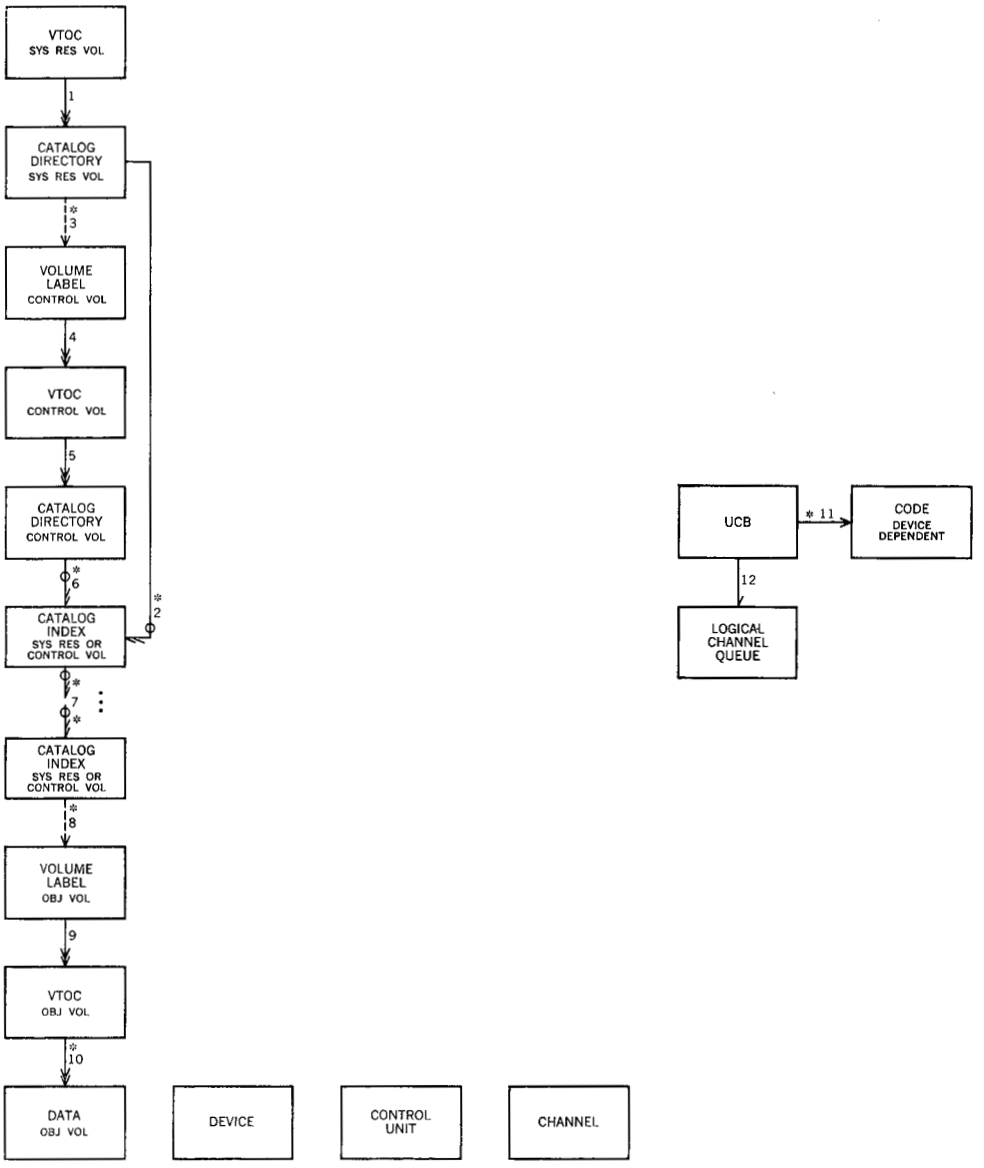
To begin with, we observe the state of the system just before the job is introduced; of interest at this point are the devices, control blocks, programs, and catalog elements that exist prior to job entry. Next to be considered are the data-management activities involved in DD-statement processing, and in establishment by the job scheduler of a task for the given job step. Third, we consider the activities governed by the OPEN macroinstruction; these activities tailor the system to the requirements of the job step. Finally, operation of the READ macroinstruction is considered, with special attention to the use of the EXCP macroinstruction. Essential to the four stages of the discussion are four cumulative displays. Frequent reference to numbered points within the figures is made by means of parenthetical superscripts in the text. The description refers more often to the objects generated and manipulated by the system than to the functional programs that implement the system.

catalog organization

The basic aspects of catalog implementation become apparent when we consider the manner in which the system finds a volume containing a cataloged data set. Recall that each direct-access volume contains a volume label that locates its VTOC (volume table of contents) and that the VTOC contains a data-set label for each volume-contained data set. Identified by data-set name, the data-set label holds attributes (such as record length) and specifies the location in the volume of the data set.

Search for a data set begins (see Figure 1) in the VTOC of the system residence volume, where a data-set label identifying the portion of the catalog in this volume[1] appears. This part of the catalog is itself organized as a partitioned data set whose directory is the highest level (most significant) index of the catalog. For

Figure 1 Control elements: before job entry



LEGEND

- MAIN STORAGE ADDRESS
- RELATIVE MAIN STORAGE ADDRESS
- DASD TRACK OR BLOCK ADDRESS
- RELATIVE DASD TRACK OR BLOCK ADDRESS
- VOLUME IDENTIFICATION NUMBER
- INTERRUPTION
- DATA SET NAME
- DATA DEFINITION STATEMENT NAME
- OBJECT DATA FLOW
- CONTROL DATA FLOW
- O INDICATES THAT ITEM POINTED TO IS ONE OF A CHAIN OF SIMILAR ITEMS
- * INDICATES THAT THE SOURCE OF A POINTER IS A TABLE WHICH IDENTIFIES SIMILAR ITEMS

data sets cataloged on the system residence volume, entries in this directory contain the addresses of lower-level indexes;[2] for data sets cataloged on other control volumes,[3] directory entries contain the appropriate volume identification numbers.

Assume for the moment that the search is for a data set cataloged on control volume V and that V is not the system residence volume. In this case, the volume label of V contains the location of V's VTOC.[4] (Volume label and VTOC are recorded separately to allow for device peculiarities.) One of the data-set labels in this VTOC identifies the part of the catalog on V;[5] just as in the case of the residence volume, this part is organized as a partitioned data set. Inasmuch as the directory of this partitioned data set is the subset of the highest-level index governing that part of the catalog recorded on V, directory entries contain the addresses of the next-level indexes on V.[6] It should be added that all index levels needed to catalog a data set appear on a single control volume; the part of the catalog on any given control volume is known to other control volumes, because the directory entries of the given control volume appear in the directories of the others.

Each index level below the directory[7] is used to resolve one qualification in the name of a data set. For example, were the name of a data set A.B.C, a directory entry A would locate an index containing an entry B, which in turn would locate an index containing the entry C. This last entry identifies the volume[8] that holds the data set named A.B.C.[3]

unit control block

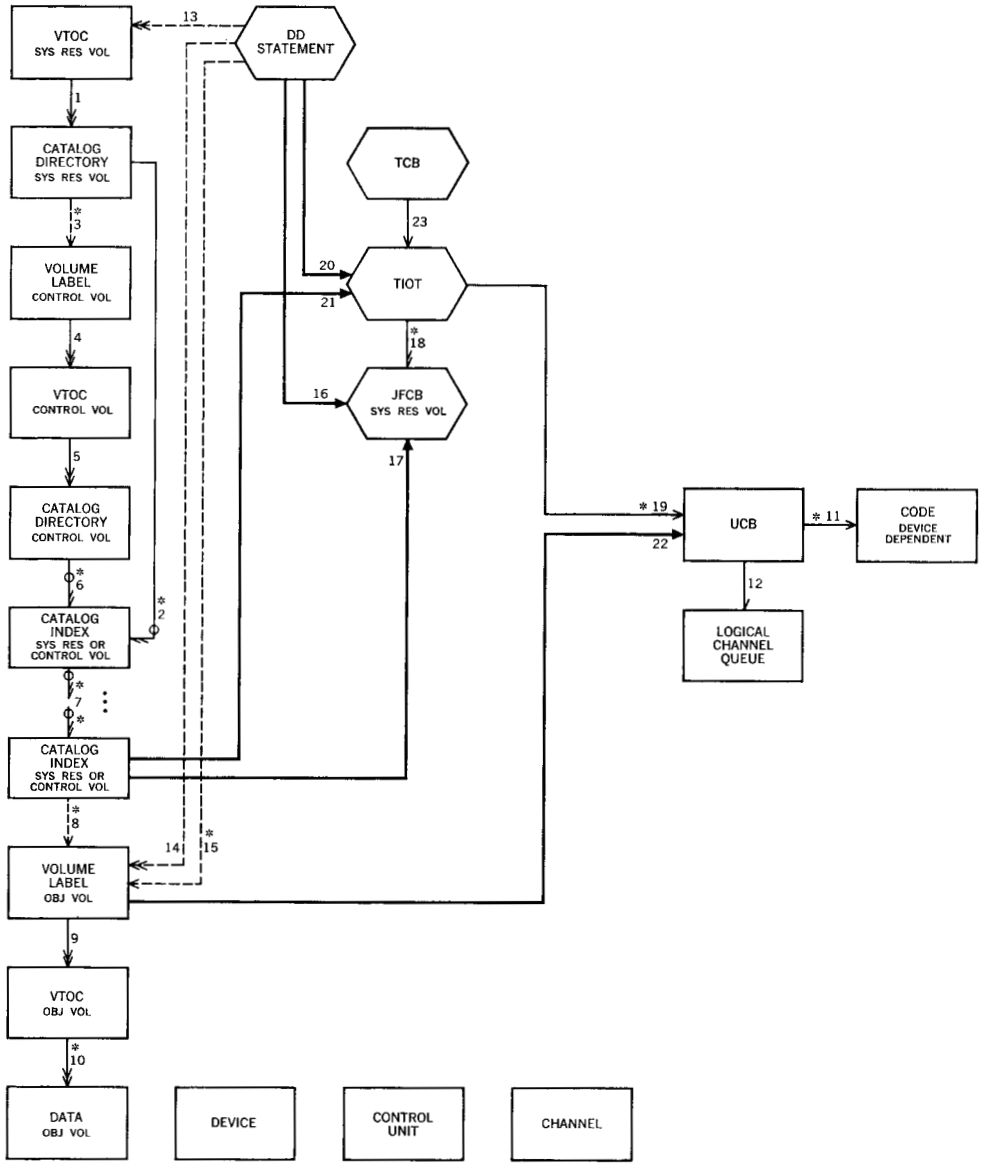
During the system generation process, one *unit control block* (UCB) is created for each I/O device attached to the system (each tape drive, disk drive, drum, card reader/punch, etc). Each UCB contains device-status information, the relevant device address or addresses, the locations of the input/output supervisor subroutines[11] that treat device peculiarities (such as start-I/O, queue-manipulation, and error routines), and the location of the *logical channel queue*[12] used with the device.[4]

DD-statement processing

The principal purpose of the DD statement (Figure 2) is to supply the (variable) name of a data set to be located via the catalog,[13] and to relate the data set to the (constant) name of the DD statement. However, a great amount of additional information may be supplied if the user desires. This information may include: the device type together with a list of volume identification numbers which serve to locate the data set without recourse to the catalog;[14,15] label information used to create new labels; attributes that determine the nature of the data set created or processed; and processing options that modify the operation of the program. After being encoded by the job scheduler, most of this information is included in a *job file control block* (JFCB)[16] that is used in lieu of the original DD statement.

As was suggested above, a data set can be located either by an explicit list of volume identification numbers and an indication of the device type (if this information is given on the DD statement), or by data-set name alone. In the latter case, a list of volume

Figure 2 Control elements: job scheduling—hexagonal blocks denote elements of first concern at time job is scheduled



LEGEND

- MAIN STORAGE ADDRESS
- RELATIVE MAIN STORAGE ADDRESS
- DASD TRACK OR BLOCK ADDRESS
- RELATIVE DASD TRACK OR BLOCK ADDRESS
- VOLUME IDENTIFICATION NUMBER
- INTERRUPTION
- DATA SET NAME
- DATA DEFINITION STATEMENT NAME
- OBJECT DATA FLOW
- CONTROL DATA FLOW
- O INDICATES THAT ITEM POINTED TO IS ONE OF A CHAIN OF SIMILAR ITEMS
- * INDICATES THAT THE SOURCE OF A POINTER IS A TABLE WHICH IDENTIFIES SIMILAR ITEMS

identification numbers is extracted from the catalog and placed in the JFCB.[17]

Prior to establishing a task for the job step, the job scheduler assigns devices to the step. To represent this assignment, the job scheduler constructs a *task input/output table* (TIOT). An entry is made in this table for each DD statement supplied by the user; each entry relates a DD-statement name to the location of the corresponding JFCB[18] and the unit or units assigned to the data set.[19] The assignment of a specific device derives from the specification of device type supplied through the DD statement[20] or the catalog,[21] together with a table of available units maintained by the job scheduler.

The job scheduler then assures that all volumes initially required by the step are mounted. As each volume is mounted, its volume label is read; the volume identification number and the location of its VTOC are placed in the corresponding UCB for future reference.[22] Finally, the job scheduler "attaches" a task for the step. In the process, the supervisor constructs a *task control block* (TCB). The TCB is used by the supervisor as an area in which to store the general registers and program status word of a task at a point of interruption; it contains the address of the TIOT.[23]

OPEN

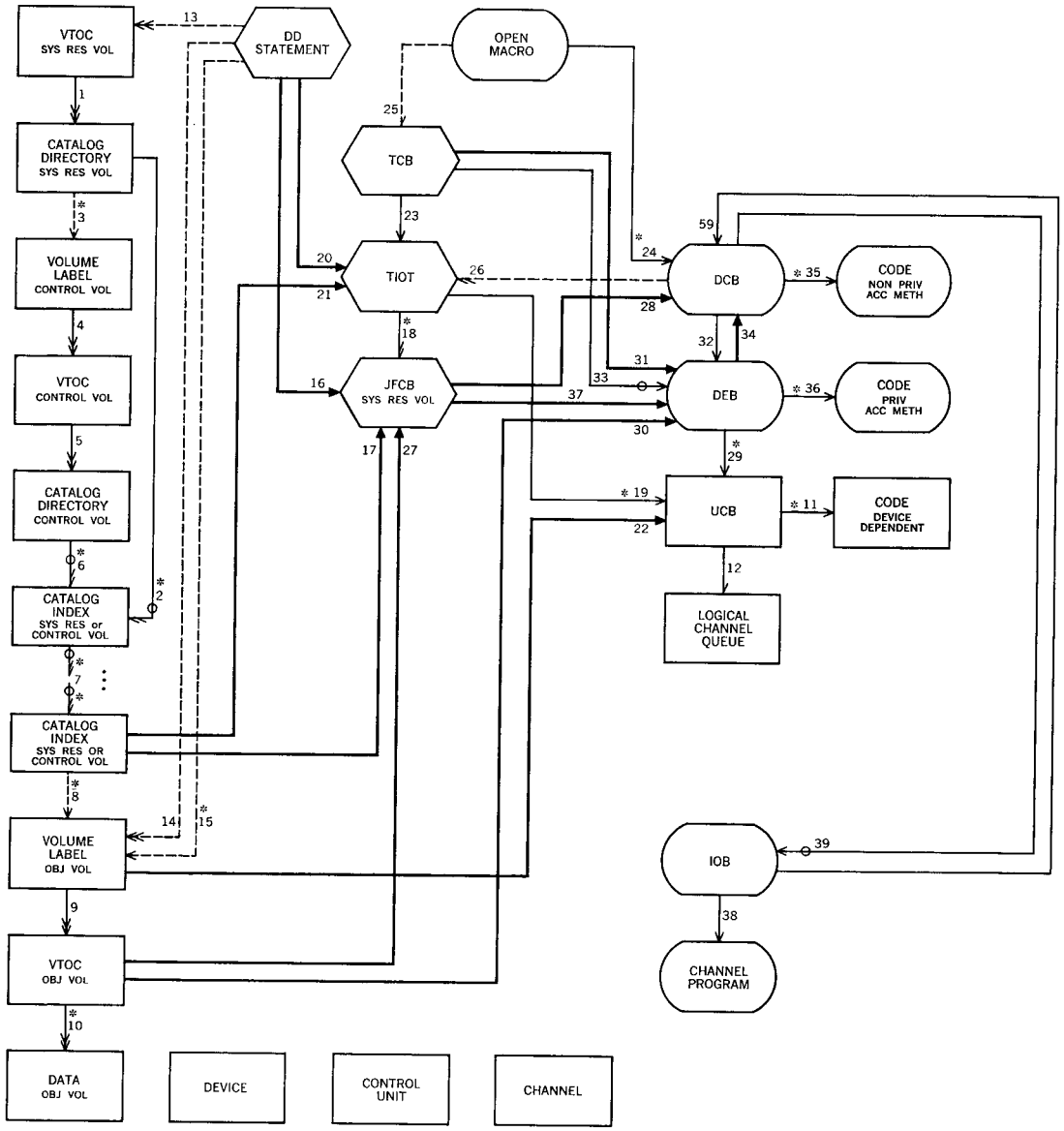
Execution of the OPEN macroinstruction (Figure 3) identifies one or more data control blocks (DCB's) to be initialized:[24] since an SVC interruption results, the TCB of the calling task[25] is also identified. The name of the DD statement, contained in the DCB, is used to locate the entry in the TIOT corresponding to the data set to be processed.[23,26] The related JFCB is then retrieved.[18]

After assuring that the required volumes are mounted,[19] the open subroutines read the data-set label(s) and place in the JFCB all data-set attributes that were not specified (or overridden) by the DD statement.[27] At this point, the DCB and JFCB comprise a complete specification of the attributes of the data set and the access method to be used. Next, data-set attributes and processing options not specified by the DCB macroinstruction are passed from the JFCB to the DCB.[28]

The system then constructs a *data extent block* (DEB), logically a protected extension of the DCB. This block contains a description of the *extent* (devices and track boundaries) of the data set,[29,30] flags which indicate the set of channel commands that may be used with the data set,[37] and a priority indicator.[31] The DEB is normally located via the DCB;[32] but in order to purge a failing task or close the DCB upon task termination, it may be located via the TCB.[33] If the data set is to be retrieved sequentially, the address of the first block of the data set is moved to the DCB.[34]

Next, the access-method routines are selected and loaded. The addresses of these routines are placed in the DCB.[35] If privileged interrupt-handling or error routines are required, they are loaded and their addresses recorded in the DEB.[36] Finally, the channel programs which will later be used to access the data set are generated. For each channel program, an *input/output block* (IOB) is

Figure 3 Control elements: OPEN macroinstruction—oblate blocks denote elements of first concern during execution of OPEN macroinstruction



LEGEND

- MAIN STORAGE ADDRESS
- RELATIVE MAIN STORAGE ADDRESS
- DASD TRACK OR BLOCK ADDRESS
- RELATIVE DASD TRACK OR BLOCK ADDRESS
- VOLUME IDENTIFICATION NUMBER
- INTERRUPTION
- DATA SET NAME
- DATA DEFINITION STATEMENT NAME
- OBJECT DATA FLOW
- CONTROL DATA FLOW
- O INDICATES THAT ITEM POINTED TO IS ONE OF A CHAIN OF SIMILAR ITEMS
- * INDICATES THAT THE SOURCE OF A POINTER IS A TABLE WHICH IDENTIFIES SIMILAR ITEMS

created.[38] The IOB is the major interface between the problem program (or the access-method routines) and the I/O supervisor. It contains flags that govern the channel program, the location of the DCB,[59] the location of an event control block used with the channel program, the location of the channel program itself, the "seek address," and an area into which the I/O supervisor can move the channel status word at the completion of the channel program. IOB's are linked in a chain originating at the DCB.[39]

READ

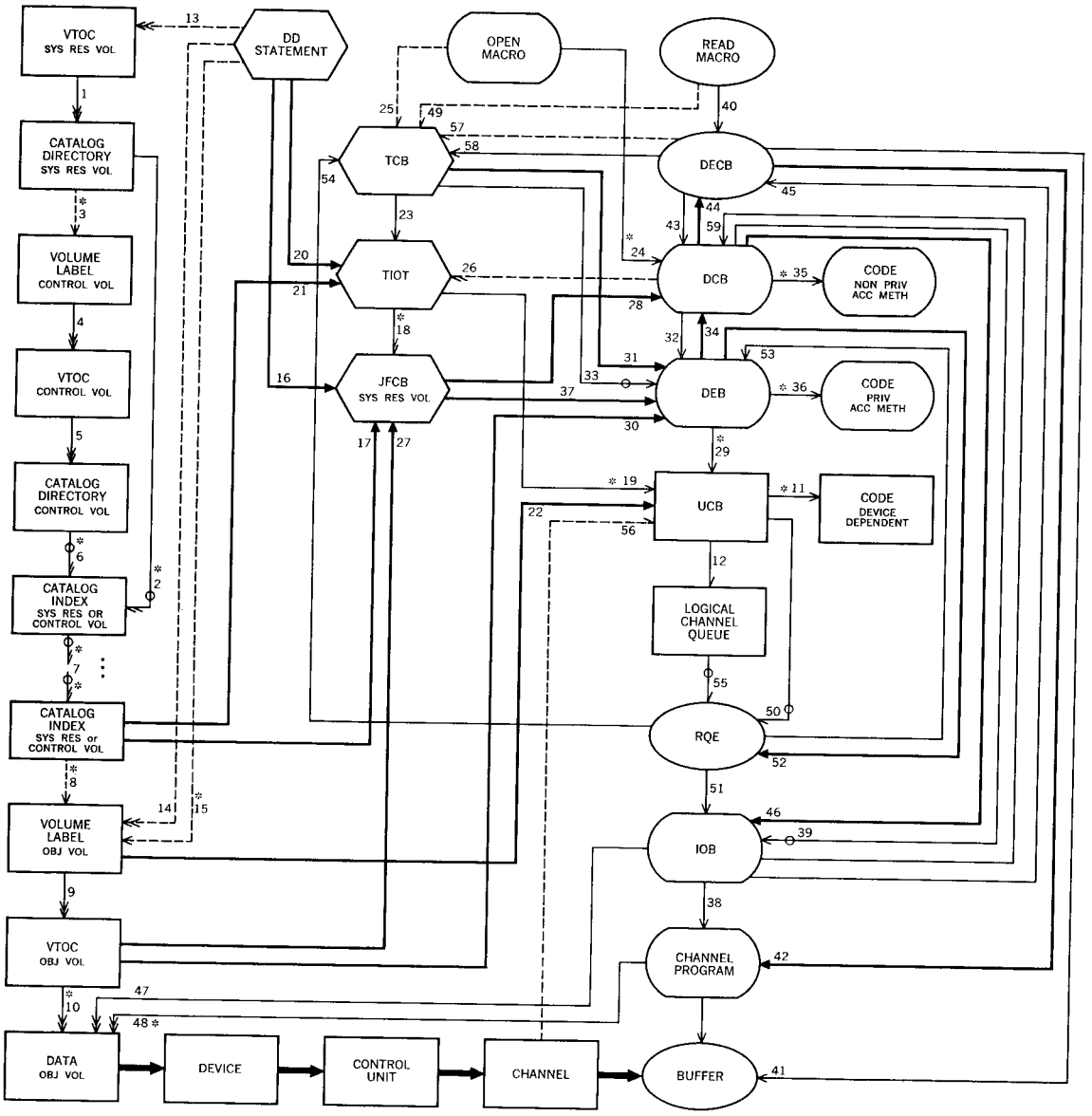
The READ macroinstruction (see Figure 4) identifies a parameter list, called the *data event control block* (DECB),[40] that is prepared either by the user or the READ macroinstruction. This block contains the address of a buffer,[41] the length of the block to be read (or the length of the buffer), the address of the DCB associated with the data set,[43] an event control block, and the like. Buffer address and block or buffer length are obtained from the DCB if not supplied by the user.[44] Using an address previously placed in the DCB,[35] the READ macroinstruction branches to an access-method routine that assigns an IOB and a channel program to the DECB. Subsequently, the routine modifies the channel program to reflect the block length and the location of the buffer;[42] it then records the address of the DECB in the IOB.[45] In addition, the routine computes the track and block addresses of the next block and updates the IOB and channel program using the results.[42,46,47,48] The access method routine then issues the EXCP macroinstruction.

EXCP

The EXCP macroinstruction causes an SVC interruption[49] that calls the I/O supervisor and passes to it the addresses of the IOB and, indirectly, the DCB.[59] Using the DCB, the address of the DEB is obtained and verified.[32] Next, assuming that other requests for the device are pending, the IOB is placed in a seek queue to await the availability of the access mechanism. Queues maintained by the IOS take the form of chains of *request queue elements* (RQE's) which identify the IOB's in queues.[51] An RQE contains a priority byte obtained from the DEB,[52] the address of the DEB,[53] and the address of the TCB of the requesting task[54] (used to purge the system of the IOB's upon task termination). Seek queues originate from UCB's,[50] and are (optionally) maintained in ascending sequence by cylinder address to reduce average seek time.

When, as a result of the completion of other requests, the access mechanism becomes available to the current IOB, a seek operation is initiated using the track address in the IOB. Just prior to this, the track address is verified (using the contents of the DEB) to ensure that the seek address lies within the extent of the data set. Assuming that the seek operation was not immediately completed, seek commands to other devices are issued; the channel is then used for other operations if possible. At the completion of the relevant seek operation,[56] the RQE is removed from the top of the seek queue and placed in the appropriate logical channel queue[55] in priority sequence. For the performance of all of these functions,

Figure 4 Control elements: READ and EXCP macroinstructions—elliptical blocks denote elements of first concern during execution of READ or EXCP macroinstruction



LEGEND

- MAIN STORAGE ADDRESS
- RELATIVE MAIN STORAGE ADDRESS
- DASD TRACK OR BLOCK ADDRESS
- RELATIVE DASD TRACK OR BLOCK ADDRESS
- VOLUME IDENTIFICATION NUMBER
- INTERRUPTION
- DATA SET NAME
- DATA DEFINITION STATEMENT NAME
- OBJECT DATA FLOW
- CONTROL DATA FLOW
- o INDICATES THAT ITEM POINTED TO IS ONE OF A CHAIN OF SIMILAR ITEMS
- * INDICATES THAT THE SOURCE OF A POINTER IS A TABLE WHICH IDENTIFIES SIMILAR ITEMS

device-dependent routines addressed by the UCB[11] are executed by the I/O supervisor.

When the IOB reaches the top of the logical channel queue and a related channel is free, the channel program associated with the IOB is logically prefixed with a short supervisory channel program and the result executed. The control unit is initialized by the supervisory channel program to inhibit the channel program from executing commands that might destroy information outside of the extent of the data set, leave the channel and control unit unused for significant periods, or attempt to write in a data set that is to be used in a read-only manner.[5] When the channel program finishes,[56] its completion is posted in the event control block within the DECB.[45]

At any time after issuing a READ macroinstruction, the program may issue a WAIT or CHECK macroinstruction which refers to the same DECB as the READ macroinstruction. Either of these macroinstructions suspends the task[57,58] until the READ operation has been completed, i.e., until the I/O supervisor posts the completion of the operation in the DECB.

Although the foregoing discussion applies specifically to the READ macroinstruction in the BSAM context and to the use of a direct-access device, the first three displays (Figures 1, 2, and 3) are applicable to other operations as well. In fact, the discussion introduces most of the control elements that bear on data-management operations in any context.

Summary

The design of OS/360 assures that data sets of all kinds can be systematically identified, stored, retrieved, and surveyed. Versatility is served by a variety of techniques for structuring data sets, catalogs, buffers, and data transfers. In the interest of operational adaptability, the system tailors itself to actual needs on a dynamic basis. For programming efficiency, source programs may be device-independent to a novel degree.

CITED REFERENCE AND FOOTNOTES

1. A. S. Noble, Jr., "Design of an integrated programming and operating system, Part I, system considerations and the monitor," *IBM Systems Journal* **2**, 153–161 (June 1963).
2. Although the CHECK macroinstruction includes the effect of the WAIT macroinstruction, the latter may also be used prior to CHECK.
3. Ordinarily, the results of a catalog search include the device type, the identification number of the desired volume, and label verification information. If the data set is a generation of a generation group (a case not considered in the main discussion), the results are the location of an index of generations and an archetype data-set label.
4. Generally, "logical channel" and physical channel are indistinguishable. The logical channel is taken to be the set of physical channels by which a device is accessible. All devices (independent of their type) that share exactly the same set of physical channels are associated with the same logical channel queue. For example, a set of tape drives attached to physical channels 1 and 2 would share a logical channel distinct from that of a printer attached only to physical channel 1.

5. In general, the control unit is initialized to inhibit seek operations that move the access mechanism. More stringent restrictions are placed on channel programs that actually refer to cylinders shared by two or more data sets. This is not to say that inter-cylinder seek operations are disallowed; rather, the I/O supervisor verifies that these operations refer to areas within the extent of the data set. During inter-cylinder seek operations, the channel and control unit are freed for other uses.