IBM

Introduction to IBM Direct-Access Storage Devices and Organization Methods

Student Text

Introduction to IBM Direct-Access Storage Devices and Organization Methods

Student Text

Major Revision, (February 1974)
This is a major revision of the previous edition, GC20-1649-6,
and obsoletes all prior editions.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality. Address comments concerning the contents of this publication to IBM Corporation, DPD Education Development - Publications Services, Education Center, South Road, Poughkeepsie, New York 12602.

Preface

This text discusses the physical characteristics and capacities of the following Direct Access Storage Devices:

2305 Fixed Head Storage
: Available for a System/360 Model 195 or a System/370 Models 145, 155, 158, 165, 168 or 195. This device is also supported by OS/VS.

2314 Disk Storage Facility
: Available for a System/360 Models 30, 40, 50, 65, 67, 75, 195 or a System/370 Models 135, 145, 155, 158, 165, 168 or 195. This device is also supported by OS/VS.

2319 Disk Storage Facility
: Available for a System/360 Models 30, 40, 50, 65, 67, 75, 195 or a System/370 Models 135, 145, 155, 158, 165, 168 or 195. This device is also supported by OS/VS.

3330-Series Disk Storage
: Available for a System/360 Model 195 or a System/370 Models 125, 135, 145, 155, 158, 165, 168, 195, and is supported by OS/VS.

3340 Disk Storage
: Available for a System/370 Models 115, 125, 135, 145, 155, 158, 165, 168, and is supported by DOS/VS and OS/VS2.2.

3850 Mass Storage System (MSS)
: Available for a System/370 Models 145, 155-II, 158, 165-II and 168.

The file organization methods and access methods for these devices are also discussed. The use of direct access storage, basic terminology, and the establishment of controls for a direct access system are other topics addressed by this text.

The intent of this text is to introduce the reader to the Direct Access devices and their Control Units, and the data set organizations supported by OS/VS and DOS/VS. There is no discussion pertaining to the various macro's used by the different access methods. If more detail information is required refer to the references listed in the bibliography of this text.

No attempt at completeness is made. Refer to the publications listed in the Bibliography for additional details.

The following Direct Access Devices are not covered in this text:

2312 and 2318 Disk Storage – composed of a single (2312) or dual (2318) disk storage module for attachment to a 2314 DASF – A Series or to a 2319 A1, A2, or A3 in a System/370 configuration with an Intergrated File Adapter (IFA).

2313 Disk Storage – composed of four disk drive modules. May be attached to a 2314-A1 as a part of 2314 DASF-A Series or to a 2319 Disk Storage A1 in a System/370 configuration with an IFA.

2319 Disk Storage – composed of three disk storage modules for attachment to System/370 models 135 and 145, or in a 2314 DASF-B Series configuration.

The above devices are similar in concept to the 2314. For additional information on a specific device refer to the reference material listed in the Bibliography.

Contents

Introduction 1

Terminology

This chapter presents System/360, System/370, and direct access terminology and concepts that are prerequisite to an understanding of the remainder of the text. It also discusses various ways in which direct access devices can be used.

Direct Access Storage Device (DASD). A direct access storage device (DASD) is one on which each physical record has a discrete location and a unique address. Thus records can be stored on a DASD in such a way that the location of any one record can be determined without extensive searching. Records can be accessed directly as well as serially.

File. The term "file" can mean a physical unit (a DASD, for instance), or an organized collection of related information. In this text, the latter definition usually applies. An inventory file, for example, contains all the data concerning a particular inventory. It may occupy several physical units or part of one physical unit. The Operating System (OS), one of the programming systems available for S/360 and S/370, uses the term "data set" instead of "file" to describe an organized collection of related information.

Record. The term "record" can also mean a physical unit or a logical unit. A logical record may be defined as a collection of data related to a common identifier. An inventory file, for example, would contain a record (logical record) for each part number in the inventory. A physical record consists of one or more logical records. The term "block" is equivalent to the term "physical record". On a DASD, certain "nondata" information required by the control unit of the device is recorded in the same record area as the physical record. This nondata information and the physical record may be referred to as a whole with the term "data record".

Key. Each logical record contains a control field or key that uniquely identifies it. The key of the inventory record, for example, would probably be the part number.

OS/VS data management programs also provide a variety of methods for gaining access to a data set. The methods are based on data set organization and data access techniques.

Data sets can be organized in several ways:

- *SEQUENTIAL:* Records are placed in physical rather than logical sequence. Given one record, the location of the next record is determined by its physical position in the data set. Sequential organization is used for all magnetic-tape devices, and may be selected for direct-access devices. Punched tape, punched cards, and printed output are sequentially organized. Access to records in a sequential file can be made through the use of the Queued Sequential Access Method (QSAM) or the Basic Sequential Access Method (BSAM). (See Chapter 5)

- *INDEXED SEQUENTIAL:* Records are arranged in sequence, according to a key that is a part of every record, on the tracks of a direct-access volume. An index or a set of indexes maintained by the system gives the location of certain principal records. This permits direct as well as sequential access to a record. Access Methods used to access an Indexed Sequential file are the Basic Index Sequential Access Method (BISAM) or the Queued Index Sequential Access Method (QISAM). (See Chapter 7)

- *DIRECT:* The records within the data set, which must be on a direct-access volume, may be organized in any manner you choose. All space allocated to the data set is available for data records. No space is required for indexes. You specify addresses by which records are stored and retrieved directly. Direct data sets are created by using special Basic Sequential Access Method (BSAM) macro's. Records can be accessed by using the Basic Direct Access Method (BDAM). (See Chapter 8)

- *PARTITIONED:* Independent groups of sequentially organized records, called members, are on direct-access storage. Each member has a simple name stored in a directory that is part of the data set and contains the location of the member's starting point. Partitioned data sets are generally used to store programs. As a result, they are often referred to as libraries. (DOS/VS does not support Partitioned organization). (Chapter 6)

- *KEY-SEQUENTIAL:* This type of data organization is used with Virtual Storage Access Method (VSAM). Records are loaded in the data set in key sequence and controlled by an index. Records are retrieved and stored by keyed access or by addressed access, and new records are inserted in the data set in key sequence by means of distributed free space. (See Chapter 9)

- *ENTRY-SEQUENCE:* This is also a data set organization used with Virtual Storage Access Method (VSAM). The records are loaded in the data set in physical sequence without respect to their contents. Records are retrieved and stored by addressed access. New records are added at the end of the data set. (See Chapter 9)

Requests for input/output operations on data sets through macro instructions employ two techniques: the technique for QUEUED ACCESS and the technique for BASIC ACCESS. Each technique is identified according to its treatment of buffering and synchronization of input and output with processing. The combination of an access technique and a given data set organization is called an Access Method. In choosing an access method for a data set, therefore, you must consider not only its organization, but also what you need to specify through macro instructions. Also, you may choose a data organization according to the access techniques and processing capabilities available.

System/370 provides a variety of devices for collecting, storing, and distributing data. Despite the variety, the devices have many common characteristics. The generic term VOLUME is used to refer to a standard unit of auxiliary storage. A volume may be a reel of magnetic tape, a disk pack, or a drum.

This text will address the use of direct-access volumes.

Direct-access volumes are used to store executable programs, including the operating system itself. Direct-access storage is also used for data and for temporary working storage. One direct-access volume may be used for many different data sets, and space on it may be reallocated and reused.

Uses of Direct Access Storage

Online Processing with Direct Access Storage

One requirement for many applications is the ability to process data as it becomes available. The term applied to this type of processing is "online," meaning that input data does not have to be subjected to preliminary editing or sorting before entering the system, whether the input consists of transactions of a single application or transactions of multiple applications.

High-capacity direct access storage devices make the online processing approach feasible. While sorting may still be advantageous before certain processing runs, in many cases the necessity for presorting transactions before processing is eliminated. In addition, the ability to process data online provides solutions to systems problems for which previous solutions were impractical.

As an example of an online systems solution, an automotive parts distributor maintains records for a warehouse inventory of 25,000 items, each of these items identified by a ten-character part number. The distributor wanted to record each transaction affecting each item as it occurred, so that if any one item in inventory was depleted he would immediately receive an out-of-stock notification, thus permitting the inventory to be replaced as soon as possible. His existing data processing system provided these notifications only once a day, because his orders were batched and processed; all transactions affecting inventory were accumulated, sorted into part-number sequence and processed against a master inventory file at the end of each day. The problem was solved with the installation of a direct access storage system. All inventory transactions would be processed online, as they occurred, and the required status notifications would be provided almost immediately.

Although the example refers to multiple types of inventory input transactions which were processed online, it should not be inferred that inline processing is a unique requirement of inventory applications, or that the online concept should be limited to transactions involving a single application. Direct access storage enables the user to maintain up-to-date records for diversified applications and to process nonsequential and intermixed input data for multiple application areas.

Direct Access Storage Inquiry

Data processing installations have always found it desirable to obtain specific information from files in the middle of an operation. Before the development of direct access storage, the ability to request information directly from temporary or permanent storage devices was limited. Procedures were developed but at best they resulted in time-consuming interruptions, and often the information was not completely up to date when received. The special ability of direct access storage systems to process input data of various types for multiple applications inline, along with the ability to immediately update all affected records, makes it possible to request information directly from storage and have the reply displayed in readable form. This is significant because it no longer makes it necessary to disrupt normal processing, nor is there need for a delay between a requirement for informaton and a reply. To illustrate, a large airline operated a number of reservations offices throughout the country and attempted to maintain a record of all flights and passenger reservations on ledger-type cards in a central location. The records were updated and inquiries made by telephone. Replies were often inaccurate and delayed. An analysis of the problem indicated that a direct access storage system would be a solution. Flight-passenger records could be maintained in direct access storage and given the proper communications link from reservation desks to a computer, thus permitting all inquiries to be answered quickly and accurately.

Other examples emphasize the importance of immediate inquiry "What is the balance of account number 133420?" An inventory control question might be: "How many of part number 55632 are there on order?" Manufacturing: "How many subassemblies of part number 16414 are on hand?" And in payroll: "What are the year-to-date earnings of employee number 13862?" Granted that each of these questions could eventually be answered in other data processing approaches, the question is when and how. Normally, it would be at the end of a completed run, which might be too late to be of significant value.

It is necessary, therefore, to consider the impact of immediate inquiry capability on any system, for inquiry may be needed regardless of previous data processing experiences.

The ability to request information directly from a computer and receive an immediate response without involved or complex operational procedures is in itself a justification for direct access storage devices in many applications.

Complex Activity Modification

As the data processing requirements of a business increase, there also tends to be an increased interdependency between applications. Various applications require the same input records, or, for processing, require reference to the same master file records used in other applications. Modification of existing procedures to vary the sequence of file referencing and/or to accommodate additional references is more easily accomplished on direct access storage systems.

In the case of a company with production control, inventory maintenance and budgetary accounting, frequent procedure changes were required when new products were manufactured and when budget revisions were issued. Therefore, the referencing sequence changed and additional references to master file records became necessary. Regardless of the system selected to do the job, the procedures had to be altered when changes occurred. However, a direct access storage system was selected to make changes easier. With it, master file records were always accessible regardless of referencing requirements. In addition, direct access storage units contained both inventory records and budgetary records and each could be referenced as needed. Thus complex activity was handled with a minimum of effort.

In the solution to this problem lies the solution to other data processing problems where multiple, interdependent activities and multiple reference to interrelated records are required.

Direct Access Storage and Low-Activity Data Processing

Many of the applications installed today involve the processing of a limited number of input transactions against very large master files. Although very few master file records are altered or referenced by the input data for a particular run, an entire master file, which is necessarily maintained in sequence, must be searched. As an example, in a representative billing system, 100,000 customer master records are maintained, only 9000 of which are referenced daily. The 9000 records could be collected and sorted into master file customer-number sequence and processed against the file in a single run daily. However, the billing operation requires that bills be completed throughout the day. The data is therefore batch-processed nine times during the day, with the result that 1000 input transactions are processed against the 100,000 master records on each run. Since there is no practical way to skip through a file, every record must be examined by the system in each of the nine runs.

An answer to this billing problem, as well as to many other similar processing problems, lies in the use of direct access devices, which permit the retrieval of a single record. The storing of data records so that the location of any one can be determined without extensive searching is the unique capability of data processing systems using direct access storage efficiently.

High Activity

The use of direct access storage should be considered as a solution to the problems of high-activity applications, that is, those in which a comparatively small number of records are referenced or updated frequently. As an example, in the processing of piecework payroll calculations for a company having 10,000 employees, each employee working on ten or more different jobs each day, each at a specific rate and under a specific guarantee, and each calculation based upon the employee's unique work history, there is a need for continual reference to a comparatively small number of rate tables. In a batch approach, as job completion tickets were received they would be batched by employee, and a master rate file would be searched for all the employee rate tables required to process each employee's job tickets – or, as an alternate, a separate edit run could be made to determine which rate tables would be required. In either case job ticket data would be tagged with a rate table requirement sequence, and all reference to a particular rate table would be completed. When all rate data was extracted, another run would be required to complete the calculation. In a direct access approach there would be access to all rate tables as they were required, without having to batch or to search through the file for each one and without having to go through an involved procedure of repeated sorting and processing to complete the job.

Program Residence

Program steps required for processing can also be stored on direct access storage so that they can be used when required. Doing this offers several advantages:

1. The size of real storage can be reduced because only the optimum number of program steps for processing data need be in real storage at any one time.

2. Time between runs is reduced significantly because tapes no longer have to be rewound and set up. Instead, operational setup time can be limited to those functions pertaining to output, such as changing printer output forms.

3. Data can be processed inline, regardless of the type of record referenced or updated. As an example, a company with an inventory control data processing tape system required a total of 35,000 individual computer program steps for the processing of many types of input data. The system selected for the job could contain about 750 program steps in main real storage at one time. A tape program library was considered but the maintenance and continual searching of the library tape was inefficient because runs could not be made in the same sequence as the library tape. A better solution for this problem resulted with the attachment of a direct access storage device to the computer. When an order was entered, it triggered a seek of the order program and a transfer of it to real storage. If there came a time in the processing where the back-order program was required, back-order program steps would overlay the order program in core storage and the back order processed. If a receipt was processed, it would trigger the transfer of the receipt program to real storage and be processed – and so on through the many transactions which affect inventory. All this was done automatically.

Another difficulty that can be resolved by having direct access to program steps is program compaction. (Compacting occurs when the programmer attempts to get as many program steps as possible within a limited number of storage locations.) Although direct access storage does not remove the need for efficiency, the programmer's job is assisted. If his program is not limited by space, he can better spend his time on writing a program that operates efficiently. By setting up his programs as a series of blocks, each with its own specified locations in direct access storage, he also simplifies the task of modifying them. He can organize his programs into sets of expandable subroutines and proceed with the initial layout of the system, confident that all processing planned can be achieved. Large programs can be broken into primary and secondary subroutines with the access and transfer of secondary subroutines when needed and in the sequence required. Only when main storage is exceeded may additional processing time be required for further transfers of subroutines.

Direct Access Storage and Online Systems

"Online" refers to the operation of input/output devices under direct control of the CPU (central processing unit). When this can be accomplished, it eliminates the need for human intervention between input origination and output destination within computer processing. "Online" can be applied to those units under direct control of the CPU and physically located next to it – for example, an online printer. It is also used for teleprocessing units not located next to the CPU but requiring a communications link.

In the airline flight reservation problem the need for inquiry was discussed. Since the reservations offices were remote from the computer, a teleprocessing communications link was necessary. Teleprocessing and direct access equipment therefore were mutually supplemental. Without direct access storage the maintenance of and access to flight records on a computer system would be extremely difficult. Without teleprocessing equipment online, the ability to change records or to inquire regarding information on those records would also be difficult. The lack of either would make a computer system impractical. The reservations office console I/O units were online to make inline processing possible. Any computer system requiring remote I/O units online must be carefully analyzed to determine whether the advantages of direct access storage can also be applied.

Direct Access Storage as Intermediate Storage

When immediate processing of certain I/O types is not required, direct access storage can be used to accumulate the infrequently occurring transactions. For example, in an installation of a manufacturer with serveral salesmen, the sales credits for commission calculation are saved until the end of the week, at which time commission statements are printed. Credit is given to the salesmen at billing time, but credits are accumualted for a weekly run. Rather than calculate the commission for each order at billing time, the required information can be stored as it occurs on a DASD. At the end of the week all of the credit data is processed and statements are printed. This means that all processing of credits can be done at once and that the setup time for printing a special commission statement from the online printer is required only once a week.

In any application where selected input transactions can be accumulated, control totals taken, and total counts of items maintained, it might be advisable to use direct access storage as intermediate storage to gain a time-balanced system. When the accumulated batch is of sufficient size to warrant processing, a signal may be given to the system calling for initiation of processing when the system is temporarily idle; or the system may be programmed to look at a count to determine when the number is of sufficient size for processing.

Output records may be accumulated in the same way. During the course of a day, random transactions may have been processed calling for the generation of output documents which, if produced at that time, involve multiple setups of equipment or the continuous reservation of a magnetic tape drive. For convenience in scheduling the printing operations, records may be retained in a section of the DASD until the information file is large enough to warrant printing, or until some other batch that produces a similar document is run.

In an application that produces several outputs, the intermediate results can be stored on a DASD. For example, when doing a payroll on a system with one printer, all the calculations can be done and the payroll register printed. At the same time, the information required for the checks can be written on a DASD. When all employee records have been processed, the check records can be quickly read back and the checks printed.

Direct Access Storage and the Responsive System

The ability to process input data inline regardless of the diversity of applications and to store both master records and programs makes direct access storage systems uniquely responsive. They can process data randomly, give an immediate response, or, even more appropriately, give these responses on a priority basis.

When a system is called upon to process many applications and the input data is received randomly, it often becomes necessary to schedule processing and establish a priority for processing. The use of direct access storage gives unlimited flexibility in doing this without creating an overpowering burden upon the operators of the system. For example, a general file maintenance run can be interrupted to process an inquiry; upon completion of inquiry processing, the machine can return to its file maintenance run. A payroll job ticket calculation run can be interrupted to do an assembly of a new program or even to test a new program. In other words, a direct access storage system responds to changing priorities and requirements. Rather than always processing data on a first-come, first-served basis, a direct access storage system responds effectively on a controlled first-things-first priority basis.

When a direct access system is selected to fulfill the data processing needs of an installation, it may not obviate the need for sorting records into sequence. Direct access storage devices can be used for very efficient sorting operations.

System/360 System/370 Direct Access Storage Devices 2

Physical Description

Several DASD's are available for System/360 and System/370. The devices differ in physical appearance, capacity, and speed. This chapter discusses these characteristics for each of the devices.

Functionally and logically, however, they are similar in terms of data recording, checking, formatting, and programming (see Chapter 3).

Refer to the preface of this text for device type and Models of S/360 or S/370 for which it is available.

2314 Direct Access Storage Facility (see Figure 2.1). The 2314 uses the 2316 removable disk packs. The packs, when removed from the drive, are enclosed in a protective cover. Each pack consists of 11 disks mounted on a vertical shaft. The disks are 14 inches in diameter and are made of metal with a magnetic oxide coating on both sides. Since the top surface of the top disk and the bottom surface of the bottom disk are not used for recording, each pack contains 20 recording surfaces.

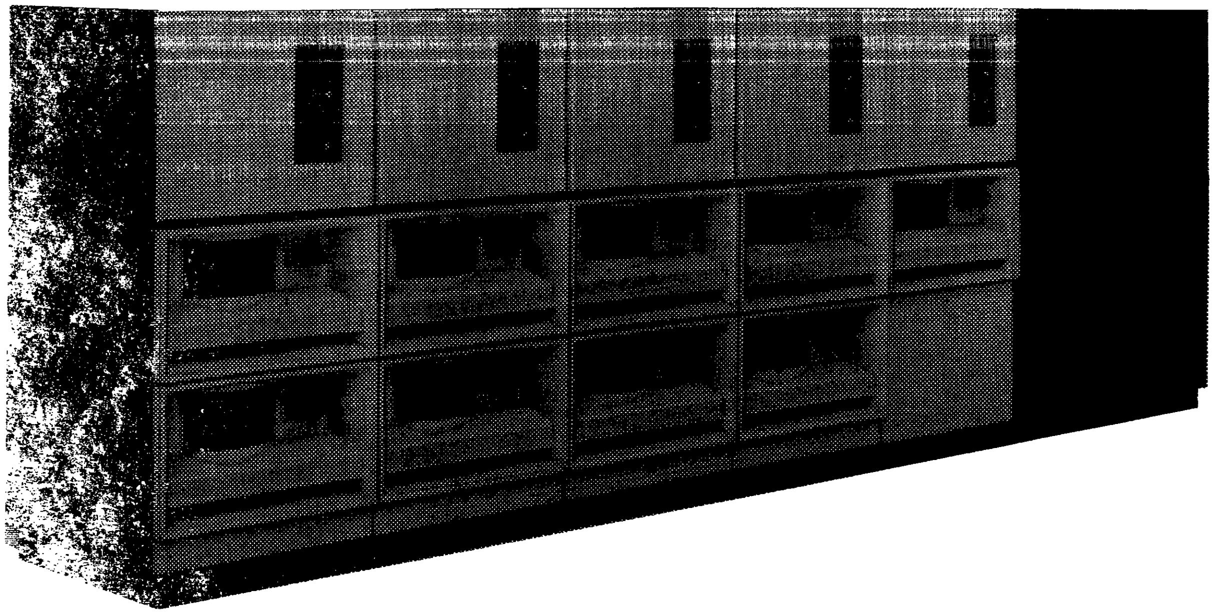


Figure 2.1 2314 Direct Access Storage Facility

3330 Disk Storage. (see Figure 2.2). The 3330 closely follows the design concepts introduced by the IBM 2314. The facility consists of a 3830 Storage Control unit and up to four 3330 Disk Storage Modules, with each module containing two independent disk drives.

Removable logical address plugs permit changing the logical device addresses of the drives within the facility. An additional service address plug is provided with each facility for customer engineer servicing from the CE panel.

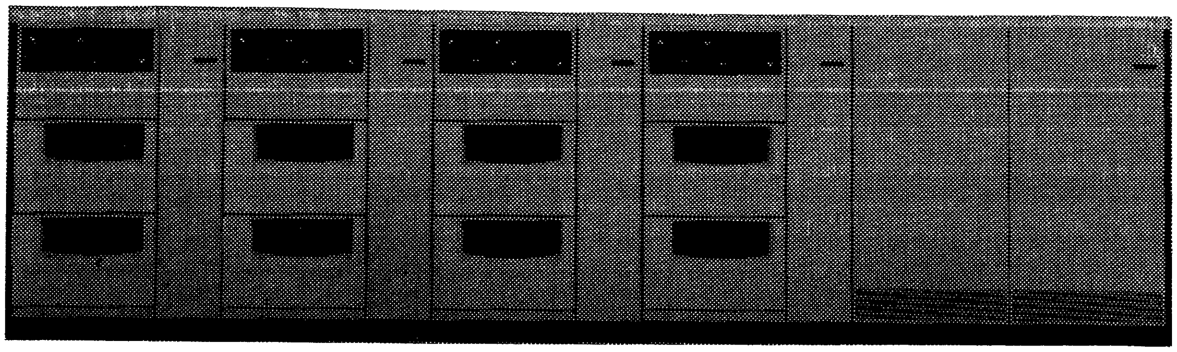


Figure 2.2 3330 Disk Storage Facility

The 3330 uses the IBM 3336 disk pack (see Figure 2.3).

Figure 2.3 3336 Disk Pack

2305 Fixed Head Storage Facility. (see Figure 2.4). The 2305 fixed head storage facility consists of a 2835 Control unit and one or two 2305 fixed head storage modules.

The 2305 fixed head storage module uses six disks permanently mounted in each storage module.

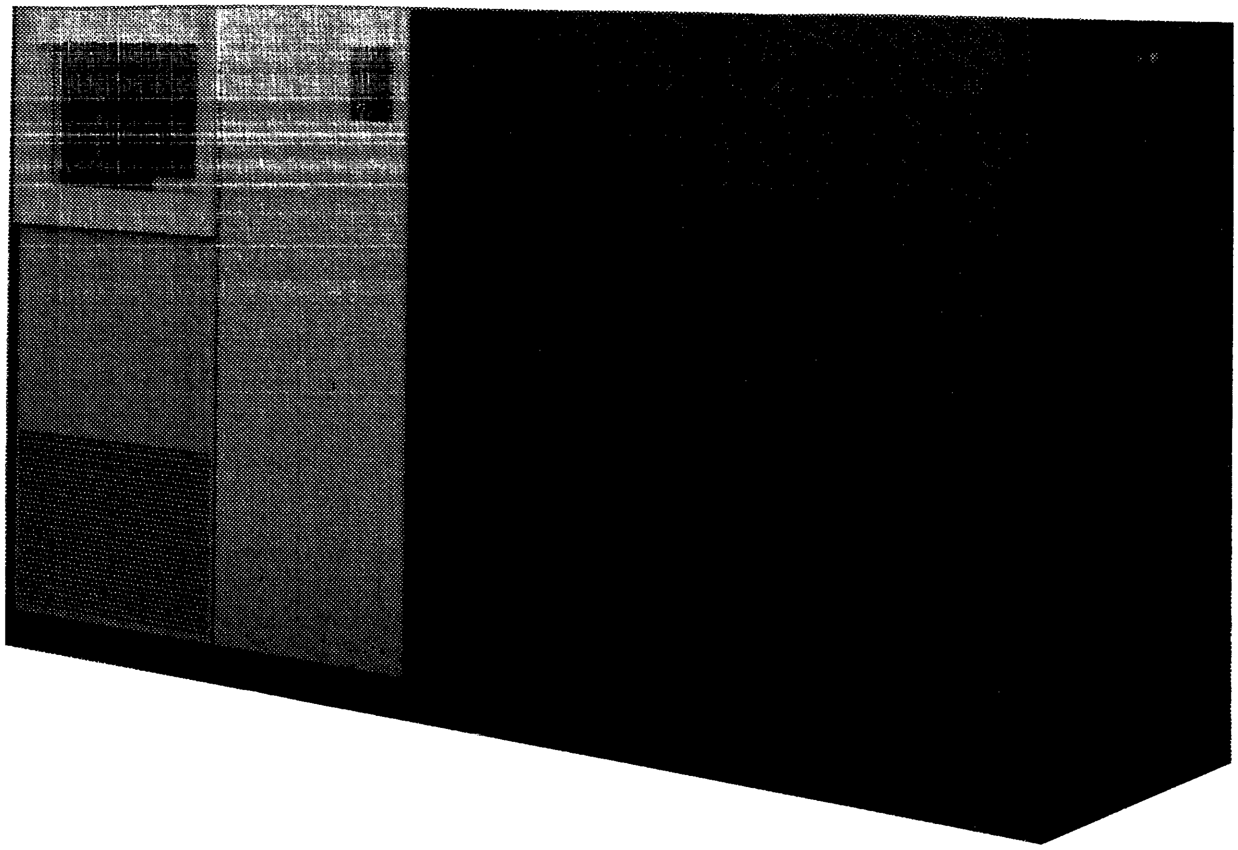


Figure 2.4 2835 Storage Control and 2305 Fixed Head Storage Module

3340 Disk Storage Facility. The 3340, together with the 3348 Data Modules (see Figure 2.5), differs in design from previous DASD devices in that the 3348 Data Module contains the disks, the spindle, the read/write heads, and the access arms. The 3348 Data Module is a sealed cartridge, which reduces exposure to outside contamination.

The data module concept offers the following advantages:

- Drive capacity can be changed by changing the data modules.
- Preventive maintenance of the heads, disks, and spindle is eliminated by reducing the exposure to outside contamination.
- Reliability is improved by dedicated read/write heads. Each head reads only the data it previously wrote.

Another feature of the data module is defect skipping. Defect skipping allows data to be written before and after a surface defect. Thus, all of the track can be used except for that portion that has the defect. This also eliminates the access time that was formerly required to move the read/write heads to an alternate track.

The 3340 configuration includes combinations of these disk storage modules (see Figure 2.6).

> 3340-A2 (control unit and 2 drives)
> 3340-B1 (1 drive)
> 3340-B2 (2 drives)

All 3340 subsystems must have one 3340-A2 module.

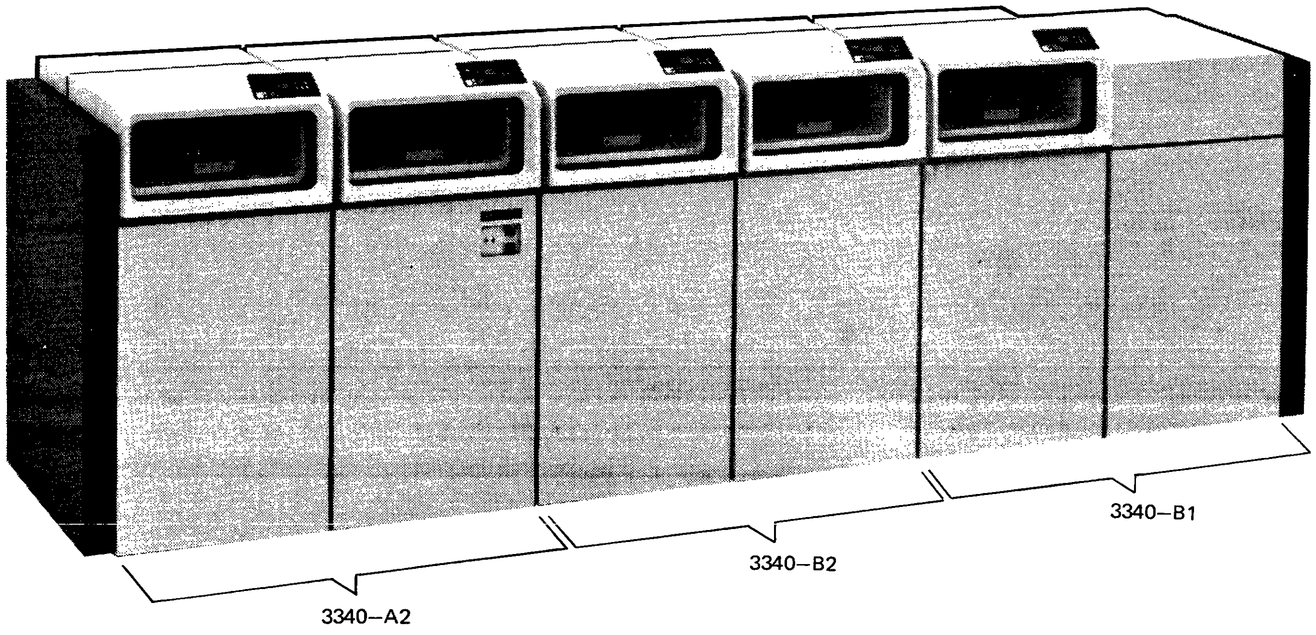


Figure 2.5 3348 Data Module

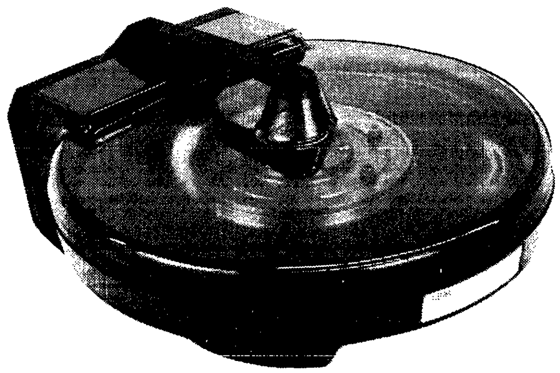


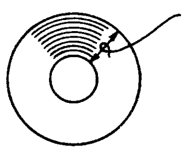
Figure 2.6 3340 Disk Storage Facility

Recording of Data

The recording surface of each device is divided into many tracks. A track is defined as a circumference of the recording surface. The tracks are concentric, not a spiral like a phonograph record.

Data is recorded serially bit by bit, eight bits per byte, along a track. The parity bit associated with each byte in core storage is not recorded (for the way in which data transfer is checked, see Chapter 3). On the 2301, data is recorded in parallel groups of four bits. Actually, each addressable track of the 2301 consists of four physical tracks.

The number of tracks per recording surface and the capacity of a track for each device are as shown in Figure 2.7. Each track has some "nondata" information recorded on it (again see Chapter 3). The capacity given is the maximum number of data bytes that can be recorded on a track. Where alternate tracks are shown, these are reserved for use in case of damage to the recording surfaces. For the drum devices, "spare" tracks are provided for this purpose.



2314 Storage Facility:	200 tracks per surface (plus 3 alternates); 7294 bytes per track.
3330 Disk Drive:	404 tracks per surface (plus 7 alternates); 13,030 bytes per track.
2305 Fixed Head Facility:	

Mod. 1	384 addressable tracks;
	14,576 bytes per track (R0 no key);
	14,136 bytes per track (R1 no key).

Mod. 2	768 addressable tracks;
	14,866 bytes per track (R0 no key);
	14,660 bytes per track (R1 no key).

3340 Disk Drive:	
with 3348-35 module	348 tracks per surface (plus 1 alternate); 8,368 bytes per track 100,416 bytes per cylinder 34,944,768 bytes per data module
with 3348-70 module	696 tracks per surface (plus 2 alternates); 8,368 bytes per track 100,416 bytes per cylinder 69,889,536 bytes per data module

Figure 2.7 DASD tracks

Access Mechanisms

Each device has some type of access mechanism whereby data is transferred to and from the device. The mechanisms are different for each device, but each mechanism contains a number of read/write heads that transfer data as the recording surfaces rotate past them. Only one head can be transferring data (either reading or writing) at a time.

2314 Disk Storage Facility (see Figure 2.8). The access mechanism consists of a group of access arms that move together as a unit. This comb type access mechanism can move horizontally to the different positions on the disk, thus giving access to all the tracks. Each arm has two read/write heads. There are ten arms giving a total of twenty heads – one for each recording surface.

3330 Storage Facility. Each drive of the 3330 has a comb type mechanism like the 2314. The 3330 having 19 tracks per cylinder has 19 read/write heads – one for each recording surface.

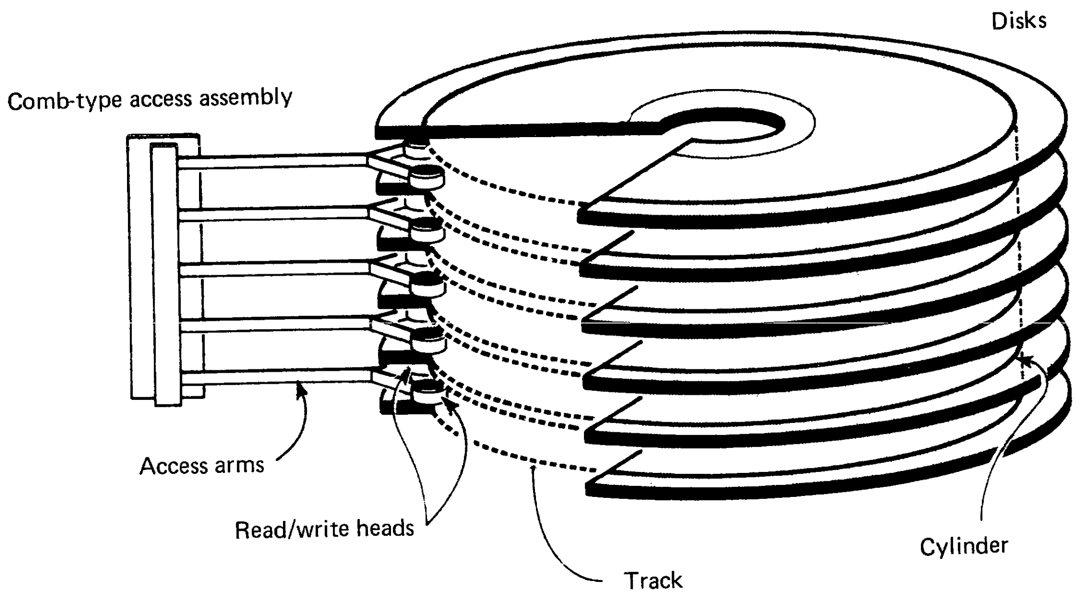


Figure 2.8 Comb type access mechanism

2305 Model 1. An addressable recording track occupies a 180-degree arc on a disk surface (see Figure 2.9). It consists of two logical track segments, one on the top surface of a disk and the other directly below it on the lower surface of the same disk. Two recording elements (R/W heads) are paired to access each addressable track in parallel. Data is recorded serially by bit but parallel by byte. All odd bytes are recorded on the upper segment and all even bytes are recorded on the lower segment. Half a rotation is required to record a full logical track of data and the average latency is one quarter of a revolution. There are 384 tracks each with a capacity of 14,576 bytes, giving a module a capacity of over 5M bytes.

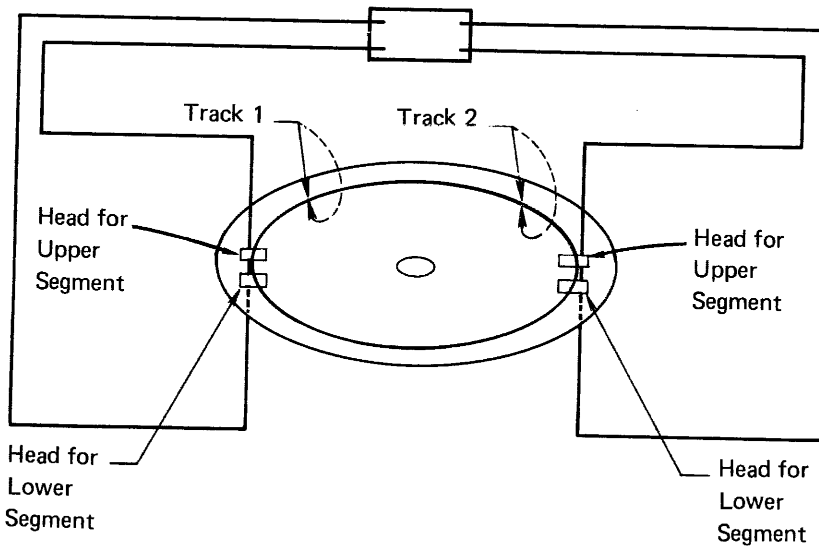


Figure 2.9 2305 Model 1

2305 Model 2. Four non-removable access mechanisms are positioned around the disks (see Figure 2.10). Each access mechanism accesses one quarter of the tracks on each surface. Data is recorded serially by bit on each track. There are 768 addressable tracks. Only having one head per track allows twice as many recording tracks to be used per surface compared with the Model 1. Each track has a normal capacity of 14,866 bytes, giving a module capacity of over 11M bytes.

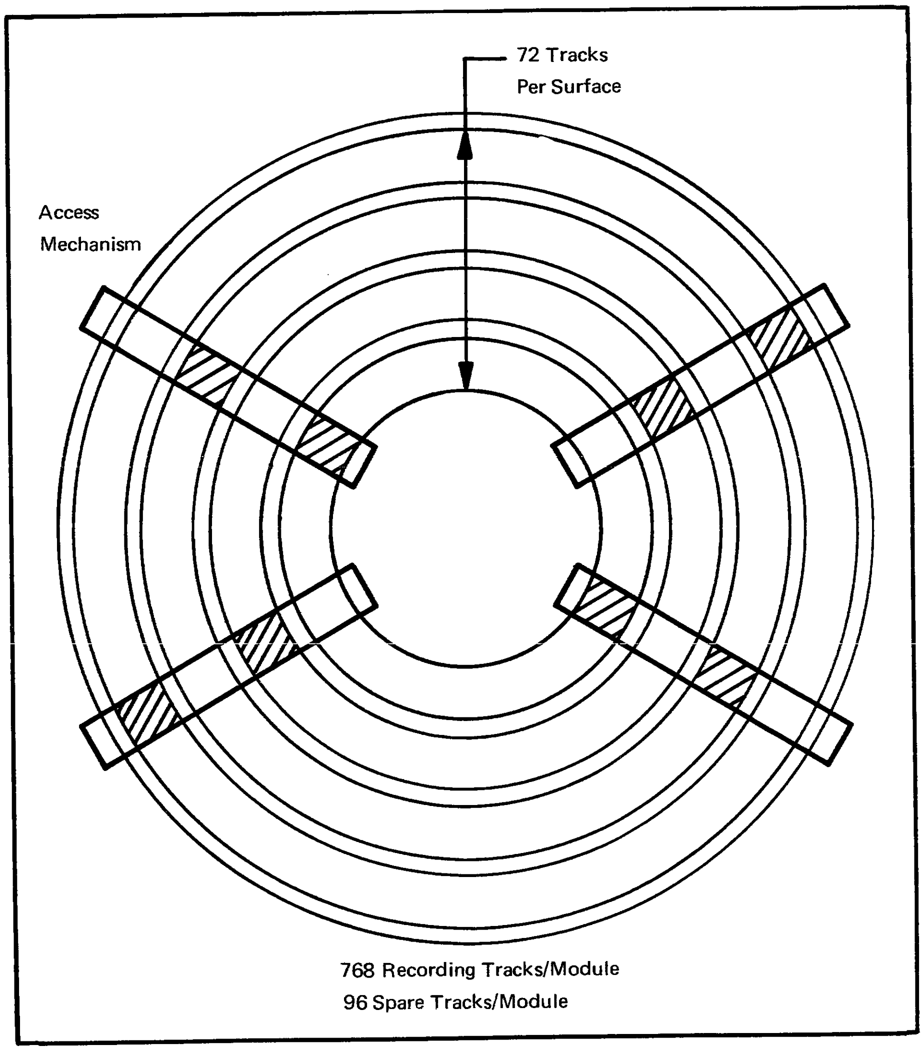


Figure 2.10 2305 Model 2

3340 Disk Drive. The 3340 uses the 3348 disk module which has the access mechanism contained in the sealed cartridge. The 3348 having 12 tracks per cylinder has 12 read/write heads – one for each recording surface. Because these data modules are removable, they permit unlimited offline storage capacity.

Cylinder Concept and Capacities

A cylinder of data is the amount that is accessible with one positioning of the access mechanism. This is an important concept, since movement of the access mechanism represents a significant portion of the time required to access and transfer data. A large amount of data can be stored in a single cylinder, thus minimizing the movements of the access mechanism. Using the 2314 as an example, physically the pack consists of twenty separate horizontal recording surfaces, while from an access point of view it consists of 203 separate vertical cylinders of twenty tracks each (see Figure 2.11).

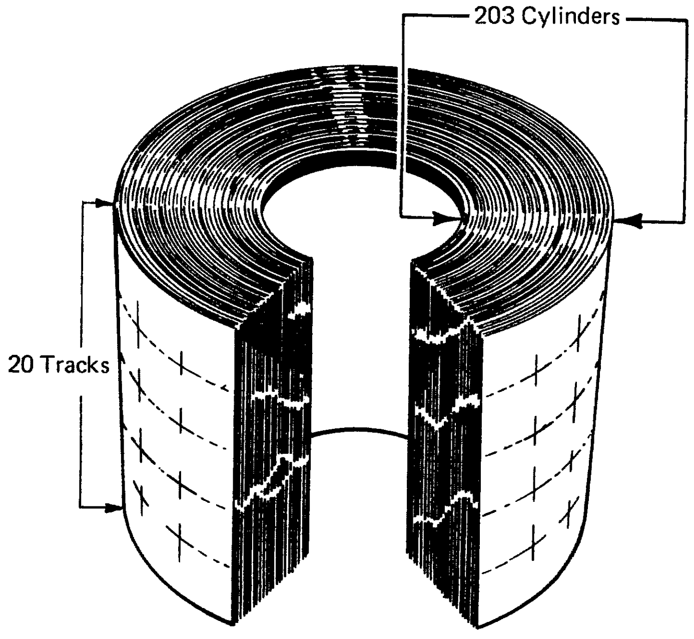


Figure 2.11   2314 cylinders

The capacities given below do not include the surfaces or tracks reserved as alternates or spares and assume the use of part of each track for information required by the IBM operating systems.

2314 Storage Facility. Each pack has 200 cylinders (plus three alternates), which is equal to the number of positions to which the access mechanism can move. Each cylinder has 20 tracks, which is equal to the number of recording surfaces. A cylinder has a maximum capacity of 145,880 data bytes (7294 bytes per track, 20 tracks per cylinder). A pack has a maximum capacity of 29.17 million bytes. A 2314 Model A1 (eight drives) has a maximum of 233.4 million bytes available to the system at one time. A 2314 Model A2 (five drives) has an on-line capacity of 145.880 million bytes.

3330 Storage Facility. Each pack has 411 cylinders (including 7 alternates), which is equal to the number of positons to which the access mechanism can be moved. Each cylinder has 19 tracks, which is equal to the number of recording surfaces. A cylinder has a maximum capacity of 247,570 data bytes (13,030 bytes per track, 19 tracks per cylinder). A pack has a maximum capacity of 100M bytes.

2305 Fixed Head Storage Module. The storage module is a fixed head disk drive with each addressable track having it's own fixed read/write element.

The 2305 Model 1 has 864 R/W elements, 768 are positioned to address 384 recording tracks (two elements per data track handle data in parallel (see Figure 2.9). 96 elements are positioned to address the 48 spare tracks. These spare tracks can be physically connected by the customer engineer to replace a faulty original track. The module has a capacity in bytes (full track records; no key) of 5,428,224.

The 2305 Model 2, like the Model 1, also has 864 R/W elements, but since it records data serially by bytes, it has 768 elements positioned to address 768 recording tracks, and 96 elements for the spare tracks. Module capacity of the Model 2 is 11,258,880 bytes (full track records, no key).

3340 Disk Drives. Drives of the 3340 series use the 3348 sealed data module. Drive capacities may be changed by selecting one of the following options:

- 3348-35 has 348 cylinders with 1 alternate. There are 12 tracks per cylinder. The byte capacity of a track is 8,368 bytes giving the data module a capacity of 34,944,768 bytes.

- 3348-70 has 696 cylinders with 2 alternate tracks available. As with the 3348-35, this data module also has 12 tracks per cylinder but has a capacity of 69,889,536 bytes.

Timing

The time required to access and transfer data consists of four parts: access motion, head selection, rotational delay, and data transfer.

Access Motion Time. This is the time required to position the access mechanism at the cylinder containing the specified record. If the mechanism is already at the correct cylinder, there is no need to move it, so access time is zero. In the following discussion of each device, the figure given is the minimum access time if the mechanism does move.

- *2314 Storage Facility:* As shown in Figure 2.12, acceleration of the mechanism is a factor, but the access motion time is essentially a function of the number of cylinders moved. For a movement of one cylinder, the minimum time is 25 ms, the maximum is 130 ms with an average of 60 ms.

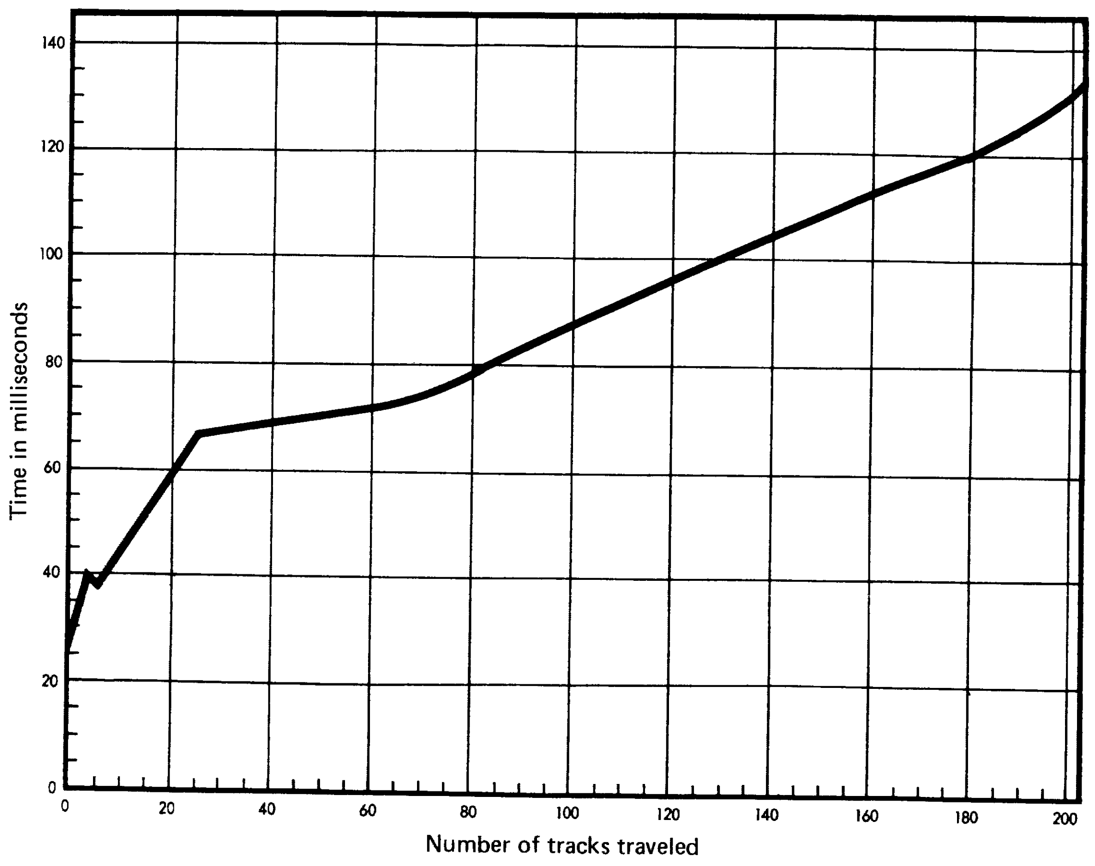


Figure 2.12 2314 access time

- *3330 Disk:* For a movement of one cylinder, the minimum time is 10 milliseconds; the maximum is 55 ms; the average over the entire pack is 30 ms.

- *2305 Fixed Head:* None, since the access mechanism does not move.

- *3340 Storage Facility:* For a movement of one cylinder, the minimum time is 10 milliseconds; the maximum is 50 ms; the average over the entire pack is 25 ms.

Head Selection. Electronic switching is required to select the correct read/write head of the mechanism. The time is negligible in all cases.

Rotational Delay. This is the time required for the correct data to rotate to the read/write head so that the data transfer can begin. It can range from zero to a full rotation (revolution). Half a rotation (average rotational delay) is generally used for timing purposes. The full rotation and average rotational delay for each device are:

	Full	*Average*
2314 storage facility	25 ms	12.5 ms
3330	16.7 ms	8.4 ms
2305-I	10 ms Rotation 5.1 Access	2.5 ms
2305-II	10 ms Rotation 10.25 Access	5.0 ms
3340 disk drive	20.25	10.12

Data Transfer. The time required to transfer data between the device and core storage is a function of rotation speed and the density at which the data is recorded.

	KB*	Milliseconds per byte
2314 storage facility	312 KB	0.0032051
3330 disk drive	806 KB	0.0012407
2305 Model I	3000 KB	0.0003333
Model 2	1500 KB	0.0006666
3340 disk drive	885 KB	0.0011300

* *Thousands of bytes per second*

Summary of Timing. In timing a job, the direct access portion consists of access motion time plus rotational delay plus data transfer. An average of half a rotation is generally used for rotational delay. Complete timing for a job requires, of course, the consideration of additional factors such as problem program processing time, access method processing time, and control program time. In this text, only direct access device timing is discussed.

DASD Control Units 3

This chapter discusses the ways in which the devices are alike. They all attach to a control unit which in turn attaches to the CPU via a channel. It is the control unit that determines the functional and logical characteristics of the devices.

The 2314 Storage Facility has a self contained control unit. Modular growth of the facility is provided by attaching various combinations of 2312, 2313, and 2318 disk storage units to the 2314 storage control.

A 2844 Auxiliary Storage Control unit can be attached to the 2314 as an integral unit. The IBM 2844 is a second control unit (in addition to the basic 2314 control) for the disk modules in the 2314. It attaches to the 2314 facility as an integral unit. Any on-line 2314 disk module can be controlled through either the 2844 or the basic 2314 control. Switching of any module to operate with either control unit is effected by programming.

This second control unit, when attached to two System/360 Model 30's or 40's or to a Model 50 and up (where a second channel is available), potentially doubles 2314 throughput and significantly increases system throughput. A 2314/2844 complex is ideally suited for very high activity applications and for real-time and teleprocessing applications that require maximum disk storage availability.

The 2314/2844 complex provides for:

1. Simultaneous operation (such as for reading and writing operations) of any two 2314 on-line disk modules with two selector channels.

2. Systems availability to the 2314 modules if either control unit (i.e., the basic 2314 or the 2844) should require preventive or unscheduled maintenance.

The 3830 control unit is used for the 3330 Disk Drive and controls up to four 3330s.

The control unit for the 2305 Fixed Head Storage Facility is the 2835. The 2835 control unit will handle 1 or 2 fixed head storage modules.

This chapter also discusses the record formats permitted when using IBM programming systems.

Control Unit Functions

File Commands

The control unit interprets and executes the file commands obtained from the CPU via the channel. It is these commands that control the operation of the devices. They are discussed in more detail later in this chapter.

Status Information

The control unit furnishes status information to the CPU. Examples are (1) transfer of data has been completed, (2) the end of the data file has been sensed, and (3) an error has been detected.

Data Transfer

The control unit provides a path for data between the CPU and the devices, and translates the data between the CPU and the devices.

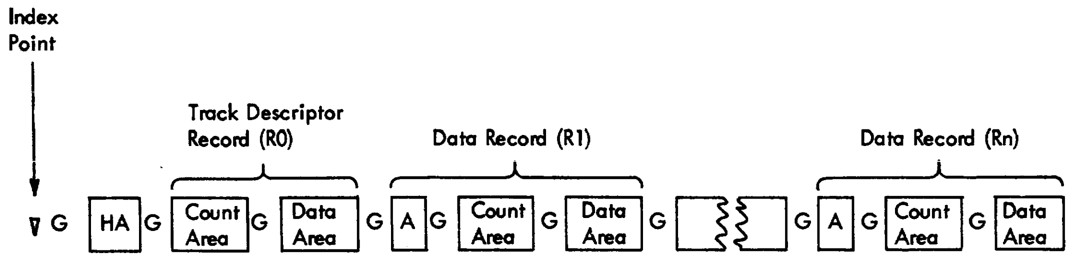
Checking

The control unit checks the validity of data transfer. As data is written (transferred from the CPU to a device), the control unit removes the parity bit from each byte. It then computes two Cyclic Check bytes, which are written at the end of each area. The two Cyclic Check bytes are coded to represent the data in the associated area. As data is read (transferred from a device to the CPU), all areas read are inspected by the control unit. Cyclic Check bytes are recalculated for each area and compared with those retrieved from storage. As the control unit transmits data to the CPU, Cyclic Check bytes are removed and parity bits are restored as needed to maintain odd parity.

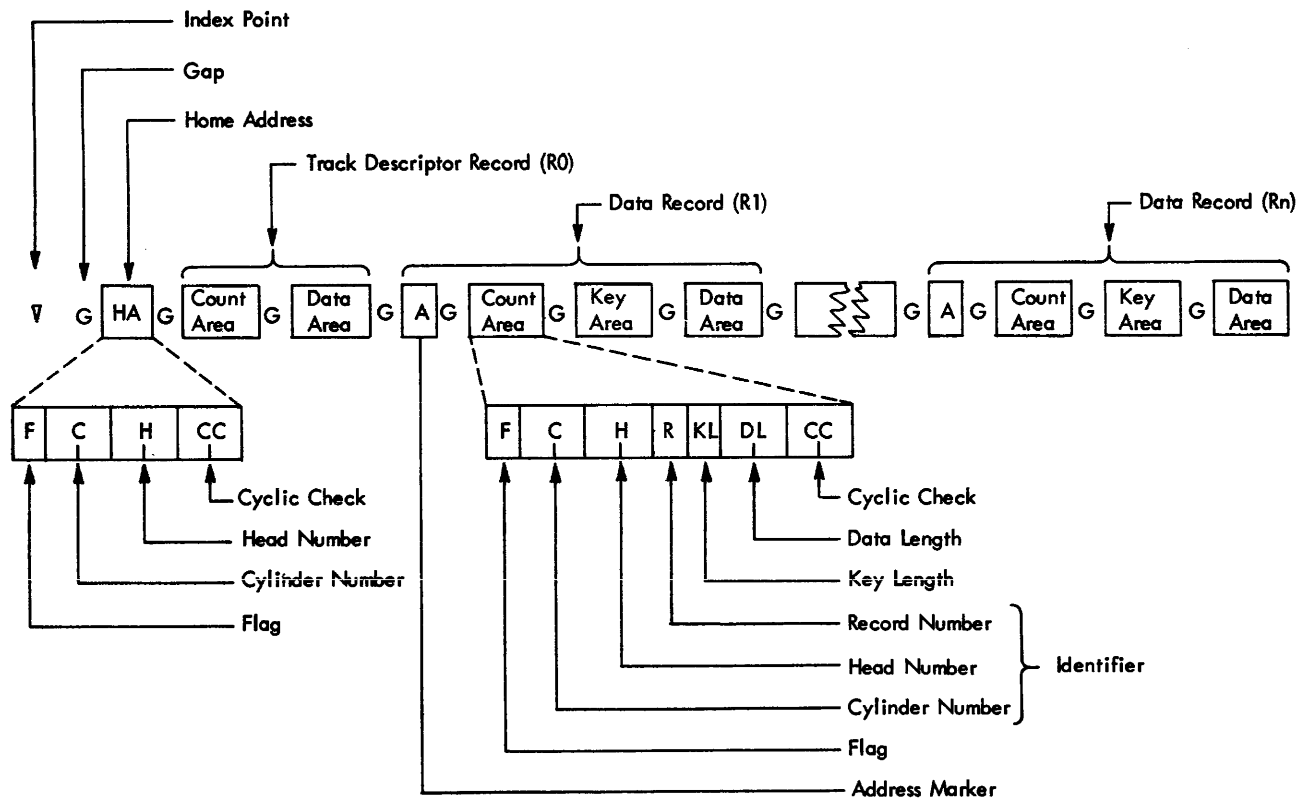
There are two advantages to this method of checking. It detects more errors than can be checked with a parity check. It also saves storage space on the devices; checking requires 16 bits per data area rather than one bit per byte.

Track Format

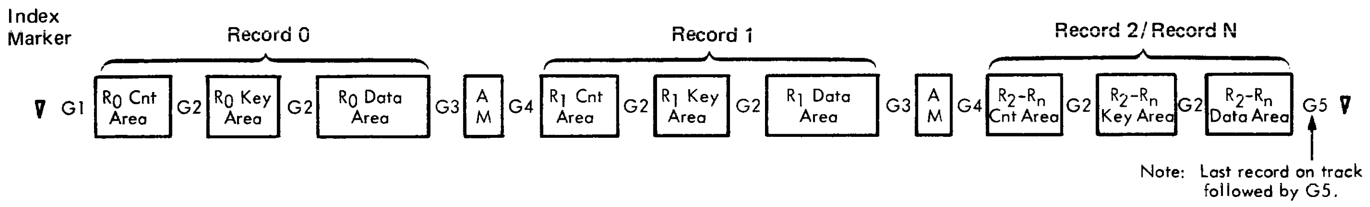
Information is recorded on all devices in a format which is prescribed by the control unit and which is identical for all devices. Each track contains certain "nondata" information (such as the address of the track, the address of each record, the length of each record, and gaps between area) as well as data information (see Figure 3.1).



A. Count-data format



B. Count-key data format



C. 2305 Record Format

Figure 3.1 Track formats

Index Point For each device, there is one Index Point to indicate the physical beginning of each track.

Home Address On each track, there is one Home Address to define the physical location of the track (the track address) and the condition of the track. As shown in Figure 3.1B, it is a seven-byte area consisting of:

- Flag – one byte indicating the condition of the track (operative or defective) and the use of the track (primary or alternate).
- Cylinder Number – two bytes indicating the cylinder in which the track is located.
- Head Number – two bytes indicating the read/write head that services this track. The combination of cylinder and head numbers indicates the address of the track.
- Cyclic Check – two bytes used for error detection, as already described. Special Home Address commands are used to read or write home addresses. Normally, this function is performed only by utility programs.

Gaps Gaps separate the different areas on the track. Certain equipment functions take place as the gap is rotating past the read/write head. The length of the gap varies with the device, the location of the gap, and the length of the preceding area. For instance, the gap that follows the index point is a different length from the gap that follows the home address, and the length of the gap that follows a record depends on the length of that record.

Track Descriptor Record (R0) This record, sometimes referred to as R0, is the first record after the Home Address and is also illustrated in Figure 3.1. IBM programming systems use R0 to store various information about the track. Details about its contents and use are discussed later.

Data Record Formats One or more user data records follow record R0 on the track. The first part of each data record is an Address Marker, a two-byte area which is supplied by the control unit as the record is written and which enables the control unit when reading records to locate the beginning of the record. As shown in Figure 3.1, there are two possible data record formats (Count-Data and Count-Key-Data), one of which may be chosen for a particular file.

Count-Data Format Records of this format (see Figure 3.1a), consist of an Address Marker, a Count Area and a Data Area. Records formatted in this way are said to be formatted without keys.

The count area is an eleven-byte field which identifies the record (in terms of cylinder number, head number, and record number) and indicates the record's format (Count-Key-Data or Count-Data) and length. The fields within the Count Area are as follows:

- Flag – a byte containing the same information as the Home Address flag byte and some additional information used by the control unit.
- Identifier (ID) – a collective term used to refer to the cylinder number, head number, and record number fields as a whole.

  Cylinder and Head numbers – four bytes normally containing the same information as the corresponding bytes in the Home address.

  Record Number – one byte containing a record number (in binary notation) ranging from 1 to 255. The first user data record is record 1 (R1), the second is record 2 (R2), etc.
- Key Length – a one-byte field always containing 0 for a record of the Count-Data format.
- Data Length – two bytes specifying the number of bytes in the Data Area of the record excluding the Cyclic Check. It is in binary notation, so it can range from 0 - to a theoretical maximum of 65,535. A data length of 0 indicates the end of a logical file.
- Cyclic Check – two bytes used for error detection, as already described.

Count-Key-Data Format

Records of this format (see Figure 3.1b), consist of an Address Marker, a Count Area, a Key Area, and a Data Area. Records formatted in this way are said to be formatted with keys. The Key Area, which can range from 1 to 255 bytes, contains the key (part number, man number, account number, etc.) that identifies the following Data Area. In most cases records will be formatted with keys so that they can be quickly located.

The major difference between the two formats is that the Count-Key-Data format contains a Key Area while the Count-Data format does not. The existence of a Key Area causes one other difference between the two formats. The Key Length field of the Count Area in the Count-Data format is always zero, but in the Count-Key-Data format it specifies (in binary notation) the length of the Key Area and therefore contains a number from 1 to 255.

The 2305 does not record a home address area between the index point and R0 (see Figure 3.1c). However, for compatibility with other similar devices, it does accept and emulate Write Home Address, Read Home Address, and Search Home Address commands.

Track Descriptor Record (R0)

The Track Descriptor Record, as mentioned earlier, follows the Home Address and is used by IBM programming systems to store information about the track. The programming systems require that it contain a Count Area and a Data Area and no Key Area. The Count Area is the same as described for data records except that record number is always 0 (hence its name R0), Key Length is always 0, and Data Length is always 8. The Data Area is therefore eight bytes long plus two bytes for the Cyclic Check.

Figure 3.2 shows that the track Descriptor Record serves another purpose in addition to its use by programming systems. In case a track on a non-drum device becomes defective, R0's Count Area provides a cross reference between the original primary track and the alternate track to which data has been moved by containing the cylinder number and head number (track address) of the alternate track, instead of (as is normal) the track address of the original primary track. On drum devices, the address of an alternate track is changed by the customer engineer to the address of the original primary track.

If IBM programming systems are not used, the data area of R0 may contain user's data. If this choice is made, the restrictions noted above that Key Length equal 0 and Data Length equal 8 do not apply. This choice, however, is not recommended, since the use of IBM programming systems greatly simplifies the user's programming.

Track	Home Address			R0 Count Area		
	F *	C	H	F*	C	H
Primary	2	2	8	2	200	1
Alternate	1	200	1	1	2	8

* A 2 in the flag byte indicates that this is a defective primary track; a 1 indicates that this is an operative alternate track.

Figure 3.2 Cross-referencing between original track and alternate track via RO

Record Formats

When using IBM programming systems, logical records may be in one of five formats, as shown in Figure 3.3. The same five formats shown are permissible without Key Areas. In all cases, if the records are formatted with keys, all records in the file must have Key Areas and all of the Key Areas must be the same length.

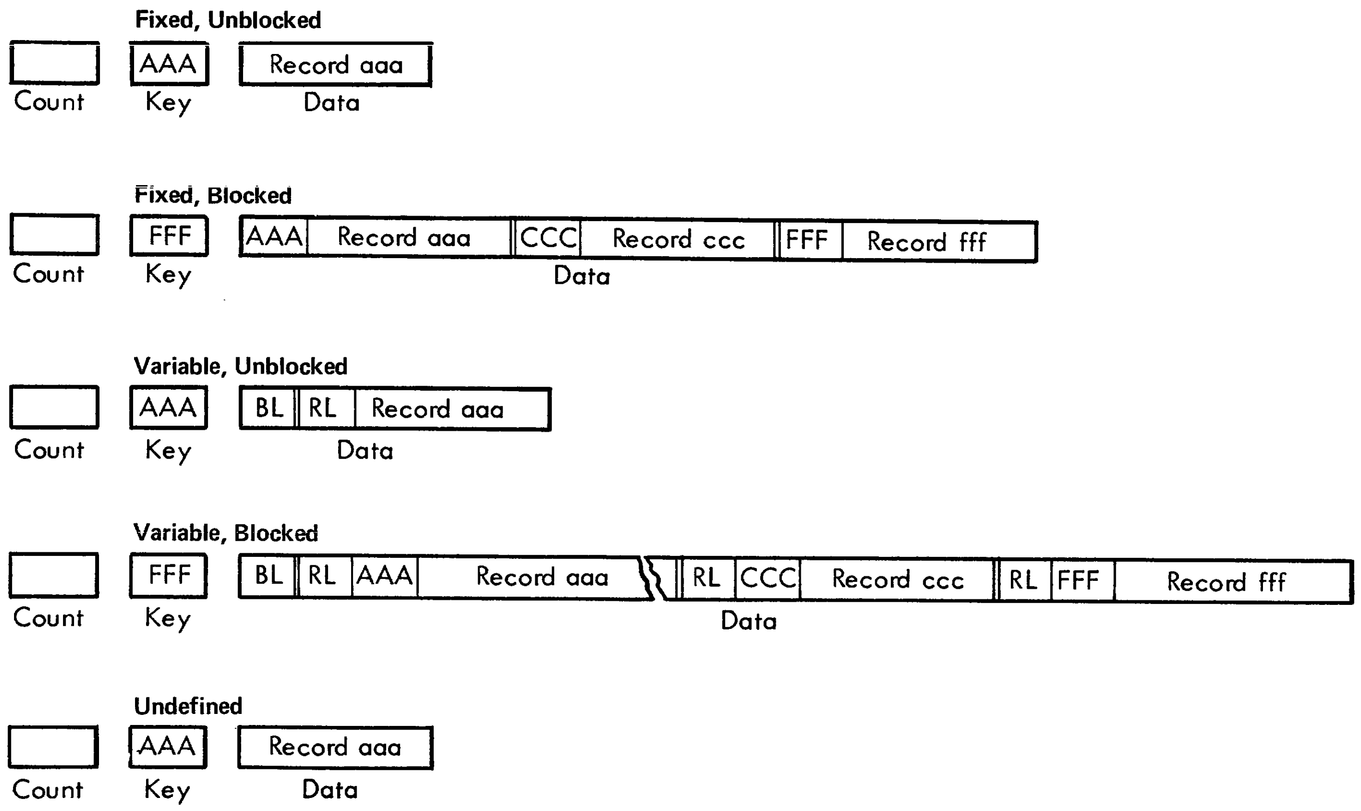


Figure 3.3 Record formats

Fixed,Unblocked

All records in the file are the same length. Each Data Area contains one logical record. If the records are formatted with keys as shown, the key is usually not repeated in the Data Area. In some cases, the key may appear in both areas, as discussed in Chapters 5-8.

Fixed,Blocked

All records in the file are the same length. Each Data Area contains a block of more than one logical record. All blocks are the same length except for a possible short block at the end of the file. The Key Area usually contains the key of the highest record in the block. The key is also a field in each logical record, so that records can be identified during processing.

Variable, Unblocked The records in the file are of varying lengths. Each Data Area contains one logical record and the special fields shown. BL (block length) indicates the number of bytes in the block including itself. RL (record length) indicates the number of bytes in the record including itself.

Variable, Blocked The records in the file are of varying lengths. Each Data Area contains a block of logical records. BL and RL have the same significance as for Variable, Unblocked.

Undefined This format is provided to permit the handling of records that do not conform to the other formats. An example is variable-length records that do not contain the BL and RL fields.

Reasons for Blocking Records The primary reason for blocking records is to pack direct access storage more efficiently. With blocked records, there is an Address Marker, Count Area, Key Area, and gaps for each block of records rather than for each logical record.

Another reason for blocking is that it may save time. If records are processed consecutively, there is only one rotational delay before reading or writing a block of records. If records are not processed consecutively, however, blocking may be a disadvantage, since it takes longer to transfer the entire block rather than the single record to be processed.

Track Capacity

In Chapter 2 the capacity of a track was expressed in terms of the maximum number of data bytes. This maximum may be achieved when there is one physical data record (block) per track formatted without a key. As the track is divided into multiple data records, the additional Address Markers, Count Areas and gaps reduce the number of bytes available for data. This section discusses track capacity from the more realistic standpoint of how many physical data records of a given length will fit on a track.

In the tables and formulas presented in Figures 3.4 and 3.5, the capacities are based on the Track Descriptor Record being used as specified by IBM programming systems rather than for user's data.

In most cases, the table shown in Figure 3.4 can be used to look up the number of given-length records per track. Note that the table is divided into two parts, since the capacity varies depending on whether records are formatted with or without keys. Examples using the table:

- Device is the 3330, records are unblocked and formatted without keys, and data length is 200 bytes. There will be 28 records per track.

Note: Use formula in Fig. 3.5 to calculate physical record size.

- Device is the 3330, records are unblocked and formatted with keys, data length is 200 bytes, and key length is 8 bytes. In using the right-hand side of the table, the number to look up is data length plus key length. There will be 13 records per track.

Note: Use formula in Fig. 3.5 to calculate physical record size.

In some cases, the table in Figure 3.4 cannot be used and the number of records per track for a given record design must be calculated using the formulas shown in Figure 3.5. The formulas are different for each device because the gap lengths required by each device are different. The formulas in Figure 3.5 indicate the number of bytes required for each data record other than the last one on the track, as well as the number of bytes required for the last data record on the track. These two categories are further divided into data records formatted with keys and data records formatted without keys. In the formulas, KL = Key Area Length (not including the Cyclic Check), and DL = Data Area length (not including the Cyclic Check).

Maximum Bytes per Physical Record Formatted without keys				Physical Records per Track	Maximum Bytes per Physical Record Formatted with Keys			
2314	3330	2305 Mod.1	2305 Mod.2		2314	3330	2305 Mod.1	2305 Mod.2
7294	13030	14136	14660	1	7249	12974	13934	14569
3521	6447	6852	7231	2	3476	6391	6650	7140
2298	4253	4222	4754	3	2254	4197	4222	4663
1693	3156	3210	3516	4	1649	3100	3008	3425
1332	2498	2480	2773	5	1288	2442	2278	2682
1092	2059	1996	2278	6	1049	2003	1794	2187
921	1745	1648	1924	7	878	1689	1446	1833
793	1510	1388	1659	8	750	1454	1186	1568
694	1327	1186	1452	9	650	1271	984	1361
615	1181	1024	1287	10	571	1125	822	1196
550	1061	892	1152	11	506	1005	690	1061
496	962	782	1040	12	452	906	580	949
450	877	688	944	13	407	821	486	853
411	805	608	863	14	368	749	406	772
377	742	538	792	15	333	686	336	701
347	687	478	730	16	304	631	276	639
321	639	424	676	17	277	583	222	585
298	596	376	627	18	254	540	174	536
276	557	334	584	19	233	501	132	493
258	523	296	544	20	215	467	94	453
241	491	260	509	21	198	435	58	418
226	463	230	477	22	183	407		386
211	437	200	448	23	168	381		357
199	413	174	421	24	156	357		330
187	391	150	396	25	144	335		305
176	371	128	373	26	133	315		282
166	352	106	352	27	123	296		261
157	335	88	332	28	114	279		241
148	318	70	314	29	105	262		223
139	303	52	297	30	96	247		206

Figure 3.4 Track capacity table

		BYTES REQUIRED BY PHYSICAL DATA RECORDS		
Device	Track Capacity (in bytes)	Data Records (except for last)	Last Data Record	

2314	7294	[534(KL+DL)/512] *+C+101	KL+DL+C	C= 0 when KL=0
				C=45 when KL≠0

2305-1 2305-2	14136 13934	(with key) (no key) 632+KL+DL 430+DL 289+KL+DL 198+DL	***	

3330	13030	135+C+KL+DL	***	C= 0 when KL=0
				C=56 when KL≠0

3340	8,368	C+KL+DL	***	C=167 when KL=0
				C=242 when KL≠0

* Truncate any fraction

*** Last record calculated as any other record on the track

KL = Key Length

DL = Data Length

Figure 3.5 Track capacity formulas

- The *track capacity* figure is the number of bytes left for data records after subracting the bytes required for the Home Address, the Track Descriptor Record (R0 is used by programming systems), and the *Address Marker, Count Area, Cyclic Check and gaps for one data record.*

- The formula for the number of bytes required for the *last data record* represents only Data Area length (and Key Area length if formatted with keys). The number of bytes required for the fixed portion of the last record and the gaps has already been subtracted from the track capacity figure.

- The formula for the number of bytes required for *each data record except the last* includes the bytes required for the Address Marker, Count Area, Cyclic Check, and fixed gaps for a record of this type. The 2314, for instance, requires 146 bytes for this information. This formula sometimes includes a fixed factor to account for the allowable deviation in the position of the record. The 2314 formula is an example of this.

- The formulas for *data records with keys* differ from those for data records without keys in that they include the length of the Key Area itself (represented by KL) and a fixed factor which accounts for the Cyclic Check and gap that follow the Key Area. The fixed factor for the 2314 is 45.

The formulas can be combined in the following way to determine the number of data records per track:

data records per track =

$$1 + \left(\frac{\text{capacity per track} - \text{bytes required for last data record}}{\text{bytes required for each data record except the last}} \right)$$

The formulas in Figure 3.5 are used rather than the table in Figure 3.4 if the data records are shorter than those shown in the table. In an example where the records to be recorded are unblocked and formatted with keys, the key length is 6, the data length is 50, and the device to be used is the 2314, how many records can be placed on each track? The solution is as follows:

Bytes for each data record except the last =

$$\frac{534\,(6+50)}{512} + 45 + 101 = 204$$

Bytes for the last data record = $45 + 6 + 50 = 101$

Records per track = $1 + \frac{7294-101}{204} = 1 + 36 = 37$

Note: The remainder is dropped in both division calculations.

File Commands

Although the IBM programming systems relieve the user of the need to program I/O operations at the command level, a familiarity with the commands is helpful in understanding the various access methods. The commands, which are interpreted and executed by the control unit, are the same for all direct access devices and fall into the four groups discussed as follows.

Control Commands

The Seek command positions the access mechanism at the specified cylinder and/or selects the specified read/write head. Once the specified address has been transferred from main storage to the control unit, the channel is not busy during a Seek. (There are several other control commands not pertinent to this discussion.)

Search Commands

The search commands cause a comparison between data from main storage and the specified area (ID, Home Address, or Key) on the device. The search may be restricted to one track or it may continue on successively higher tracks. The search terminates when the specified condition has been satisfied or when the end of the search occurs. A single-track search is ended when the end of the track is reached. A multiple-track search may be extended to the end of the drum. The search does not itself cause any transfer of data; it is normally followed by a read or a write command which performs the data transfer. The channel is busy during a search operation. The search commands are:

- Search Home Address Equal
- Search Identifier Equal. This causes a search of the five-byte Identifier (cylinder-head-record number) portion of the Count Areas. This and the other search identifier commands start the search with the ID of the record following the next Address Marker or Index Point.
- Search Identifier High. The condition searched for is an Identifier on the device higher than the search argument in real storage.
- Search Identifier High or Equal
- Search Key Equal. This causes a search of the Key Areas. This and the other search key commands start the search with the Key Area of the record following the next Address Marker.
- Search Key High
- Search Key High or Equal
- Search Key and Data Equal. This command, like the next two, requires that the control unit has the File Scan feature. It causes a search of all or part of the Key and Data Areas. The search argument in core storage has all 1-bits in the bytes that are not to be compared.
- Search Key and Data High
- Search Key and Data High or Equal

When a search is restricted to one track and followed by a read or write command to transfer the data, and the search condition is satisfied, the search-read sequence or search-write sequence takes place during one rotation. One can, in this case, think of the search as taking place during rotational delay time. If the search ends without the condition being satisfied (that is, if a Search Key Equal for one track was programmed but no equal key was found), one rotation should be estimated as the direct access time for that search.

Read Commands

The read commands cause the specified area to be transferred to real storage and checked. The Cyclic Check bytes of the area are not transferred to real storage. The channel is busy during a read operation. The read commands are:

- Read Home Address. This transfers five bytes of the Home Address (all except the Cyclic Check).
- Read Track Descriptor Record. Both the Count Area and the Data Area of R0 are transferred.
- Read Count. Eight bytes of the Count Area (all except the flag byte and Cyclic Check) following the next Address Marker encountered are transferred.
- Read Count, Key and Data. The entire record (except gaps and Cyclic Checks) following the next Address Marker encountered is transferred.
- Read Data. This command is normally chained from a search command. The Data Area transferred is that of the record which satisfied the search condition. *Both the search and the read take place on the same revolution.* If not chained from a search command, the Data Area following the next Address Marker encountered is transferred.
- Read Key and Data. The same comments as for Read Data apply, except that both the Key Area and Data Area are transferred.

Write Commands

The write commands cause data to be transferred from core storage to the specified area on the device. During the transfer, the control unit generates and adds the Cyclic Check bytes to each area. The channel is busy during a write operation.

Three of the write commands are used to initialize tracks or records. After a chain of these commands has been completed, the remaining portion of the track is erased. These format write commands are:

- Write Home Address.

- Write Track Descriptor Record. The first eight bytes transferred become the Count Area (the flag byte is generated by the control unit). The remaining data becomes the Key Area and Data Area as specified by the Key Length and Data Length fields of the Count Area.

- Write Count, Key and Data. This is the same as Write Track Descriptor Record, except that an Address Marker is generated by the control unit and written in front of the Count Area.

The other two write commands are used to add records to a previously formatted track or to update records. They must be chained from a search equal command. These data write commands are:

- Write Data. This must be chained from a Search Identifier Equal or a Search Key Equal. As with Read Data, both the search and the data transfer take place on the same revolution.

- Write Key and Data. This must be chained from a Search Identifier Equal. Again, both the search and the write take place on the same revolution.

Verification of Write Operations

As already discussed under "Checking", the parity check verifies that data transfer between the CPU and the control unit is correct, and the Cyclic Check verifies that data transfer from the device to the control unit on a read operation is correct. The Cyclic Check does not verify that data transfer from the control unit to the device is correct. It only establishes a check for subsequent reads.

As data is transferred from the channel to disk storage (write operation), the storage control removes the parity bit associated with each byte. It then computes the error correction code bytes, which are written after each recorded area. The correction code bytes, coded to represent the data in the recorded area, are used for both error detection and correction.

As data is transferred from disk storage to the channel (read operation), each area is inspected by the storage control and the error correction code bytes are recalculated for each area.

If a correctable data error is detected in the home address, count, or key areas, the storage control internally executes the error correction function through the use of command retry. If an uncorrectable data error, or a correctable data error in a data area, is detected, the correction function is determined by the system error recovery procedures.

The correction code bytes are removed and proper parity is generated by the storage control before the data is transferred to the channel.

Data Integrity

Unless corrected immediately, soft write errors cause hard read errors. Therefore, where data integrity is required, verification should be incorporated within the program. Thus, in the event of soft errors, the record can be rewritten and verified before the original data is destroyed.

Either of two verification methods may be used: full readback check or correction code check.

Full Readback Check: All of the data just written is read back into real storage and compared, byte-for-byte, with the original information.

Correction Code Check: A read operation is performed with the skip bit on. This method causes the storage control to check the validity of the record using the error correction code bytes.

Control Unit Features

The features discussed below are standard for some control units and available as a special feature on others. The support of each feature by programming systems and the estimated date at which the support will be available should be verified with the local IBM representative. To review, the control units are:

- 2314 – self-contained control unit of the disk storage facility.
- 2835 – Controls the 2305 fixed head storage facility.
- 3340 – has a self contained control unit as part of the 3340-A2 module.
- 3830 – Controls the 3330 series disk drives.

File Scan

As already discussed, this feature permits the use of the Search Key and Data Equal, High, and High and Equal commands, which in turn permit a search of all or part of *both* the Key and Data Areas of records. It is standard on the 2314 and available as a special feature on the 2841. When installed on a 2841, this feature is effective for any 2311s, and 2321s attached to that 2841.

Record Overflow

This feature permits a record to overflow from one track to the next. It is useful in achieving a greater data packing efficiency and in formatting records that exceed the capacity of a track. The cylinder boundary is the factor that limits the size of a record (DOS/VS does not support track overflow).

Each segment of an overflow record (the portion written on one track) has a Count Area. The Data Length field specifies the length of that segment only. For all segments except the last, a bit in the flag byte indicates that the record is an overflow record. If the records are formatted with keys, there is normally just one Key Area associated with the first segment. On read or write operations, all segments of the overflow records are transferred on successive revolutions. The record overflow feature is standard on the 2314.

2844 Auxiliary Control Unit

A 2844 may be attached to the basic 2314 facility to serve as a second control unit. When so attached, any online 2314 disk module can be controlled through either the 2844 or the basic 2314 control. Switching of any module to operate with either control unit is effected by programming.

The 2844 and the basic 2314 control can be attached to the same or to two different selector channels. If two channels are used, the 2844 attaches to only one, and the basic 2314 control attaches only to the other. Each control can communicate with only one channel except when the Two Channel Switch feature is used.

The 2314/2844 complex provides for:

1. Simultaneous operation of any two 2314 online disk modules with two selector channels.
2. Availability of the 2314 modules to the system if either control (that is, the basic 2314 or the 2844) should require preventive or unscheduled maintenance.

The Two Channel Switch feature can be installed on either the 2314 or the 2844 or both to permit operations with any 2314 module to be initiated through any of the four (maximum) channels to which the 2314/2844 complex is attached.

Concepts of DASD Switching

DASD subsystem configurations may include features to provide switching capabilities to:

- Allow sharing of devices between systems
- Enhance subsystem availability
- Enable potential performance improvement in some configurations

The standard I/O interface for S/360 and S/370 requires a device to be attached to a CPU by means of a channel and control unit.

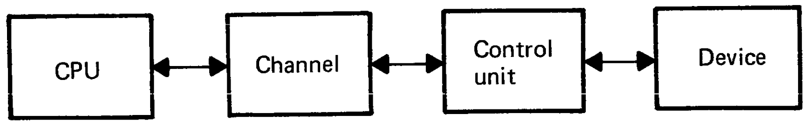


Figure 3.6 Standard device attachment

Figure 3.6 shows a static, hardwired attachment. It allows the CPU to access its attached device, always using the one and only path by which the device can be reached. The device has only one hardware address, to which it responds when selected by the CPU program to perform an I/O operation. If any link in the interface is inoperative, the device is inaccessible.

For many reasons, it is sometimes desirable to have a device dynamically accessible by more than one path, either control unit and/or channel and/or CPU. The hardware mechanism to provide this is a switch, set under either program control or manually to complete an electronic path from CPU to channel to control unit to device. Depending on the DASD device to be switched and the result to be achieved, this hardware switch may be at the control unit level and/or at the "string" level, applying to a group of two to eight DASD drives. The switchable device may be reached by more than one path and, therefore, will have more than one hardware address.

Channel Switching

When the switch is at the control unit, channel switching is the result. If the control unit is a 3830 Model 2, it may be switched between two or four channels, only two of which may be on the same CPU. Thus, channel switching is generally termed two or four channel switching. If the control unit is an Integrated Storage Control, two channel switching is available to channels on the same or different CPUs. Channel switching is not available for the Integrated File Adapter.

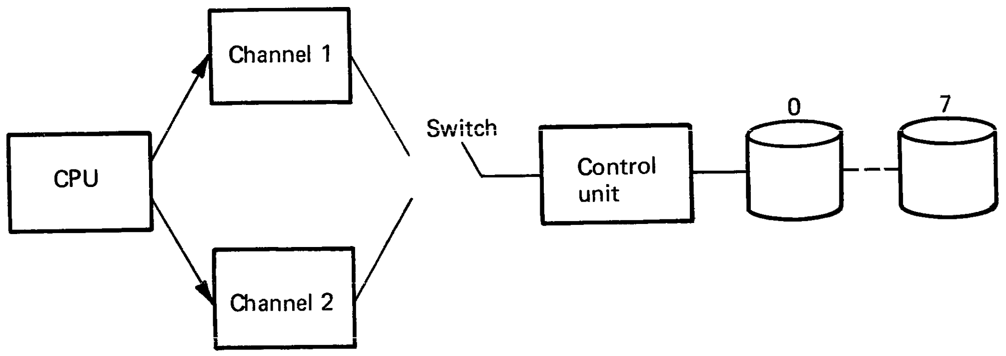


Figure 3.7 Channel switching

In Figure 3.7, the switch can be set to complete a path from the CPU through channel 1 *or* through channel *2* to one of the devices, but not both at the same time. When the switch is "open" or in a neutral status as shown in Figure 3.7, no path is completed and no data is being transmitted. If the CPU program selects the device by address '140' (channel 1, control unit 4, device 0), the switch will be set to channel 1, and normal I/O operations can take place via that path as if it were hardwired. The switch will return to its open position when the channel is no longer needed for the I/O operation. It then becomes available to be set to the first channel attempting to access one of the attached devices. Note: Operations at the device level not requiring channel or control unit (Seeks and Set Sectors), once initiated, will continue regardless of the setting of the switch.

String Switching

On certain DASD, a switch may be installed at the string level, applying to a group of two to eight DASD drives. Termed "string switch", this allows the devices in the string to be switched between two control units on the same or different CPUs.

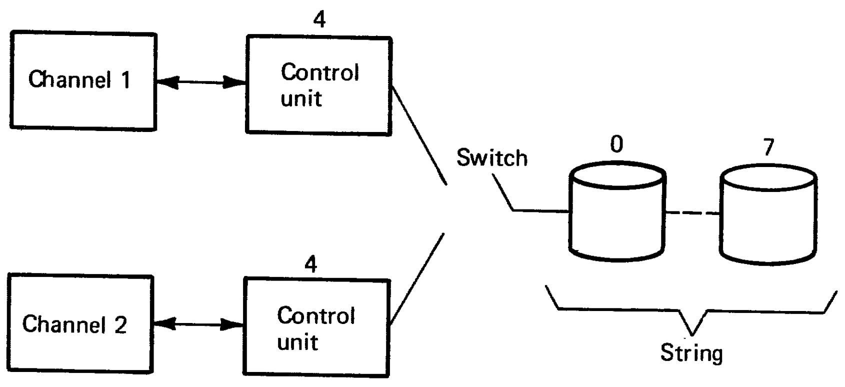


Figure 3.8 String switch

The string switch (Figure 3.8) is conceptually the same as the two or four channel switch. The first path (channel and control unit) to select one of the devices in the string will get the switch set to its interface, completing the path necessary for I/O operations. The switch is at the head of the string, and allows for one data path into the string of devices from one control unit at a time.

With additional strings of DASD added to the configuration, each with the feature of string switching, a path can be completed from each control unit to a different string. This allows two data transfer operations to occur simultaneously.

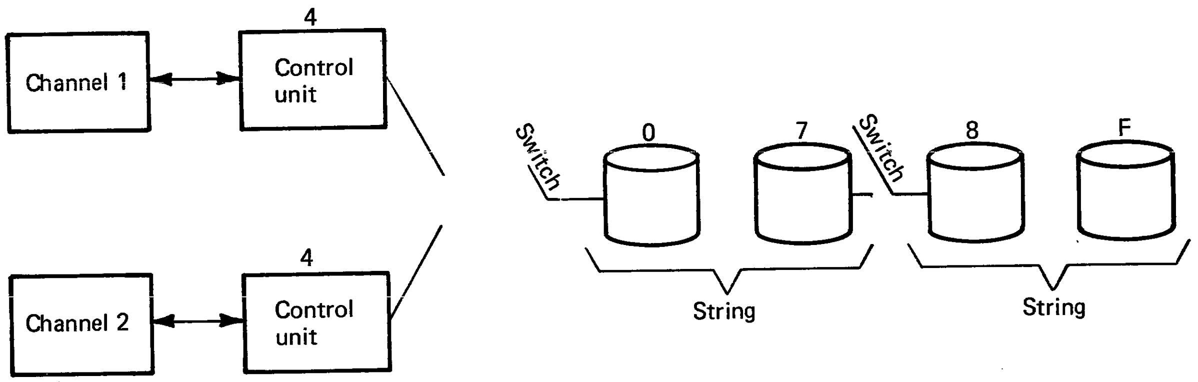


Figure 3.9 *String switch–two strings*

In Figure 3.9, if device 140 is selected and is transferring data, one of the devices in the range 8-F can be simultaneously selected through the control unit on channel 2 to transmit data. However, if a device in the range 0-7 is selected through channel 2 while channel 1 has the switch to access that string, no connection can be made and the I/O operation through channel 2 will be terminated with a "device busy" interrupt.

Combinations of Switching

The two types of dynamic switching, channel and string, may be used in conjunction to provide additional flexibility in configuring a system.

In Figure 3.10, two switches must be set to complete a path to one of the devices: one at the control unit level, and one at the head of the string. The first channel to select a device will have the switches set to complete the required path for I/O operations. As an example, suppose channel 2 selects device '245'. No other channel will be able to access a device in the range 0-7 until the switches return to the neutral position. However, either channel 3 or 4 could access a device in the second string (address 8-F) and get an I/O path completed. Channel 1 could not, as its control unit path to get to a device is being used by channel 2.

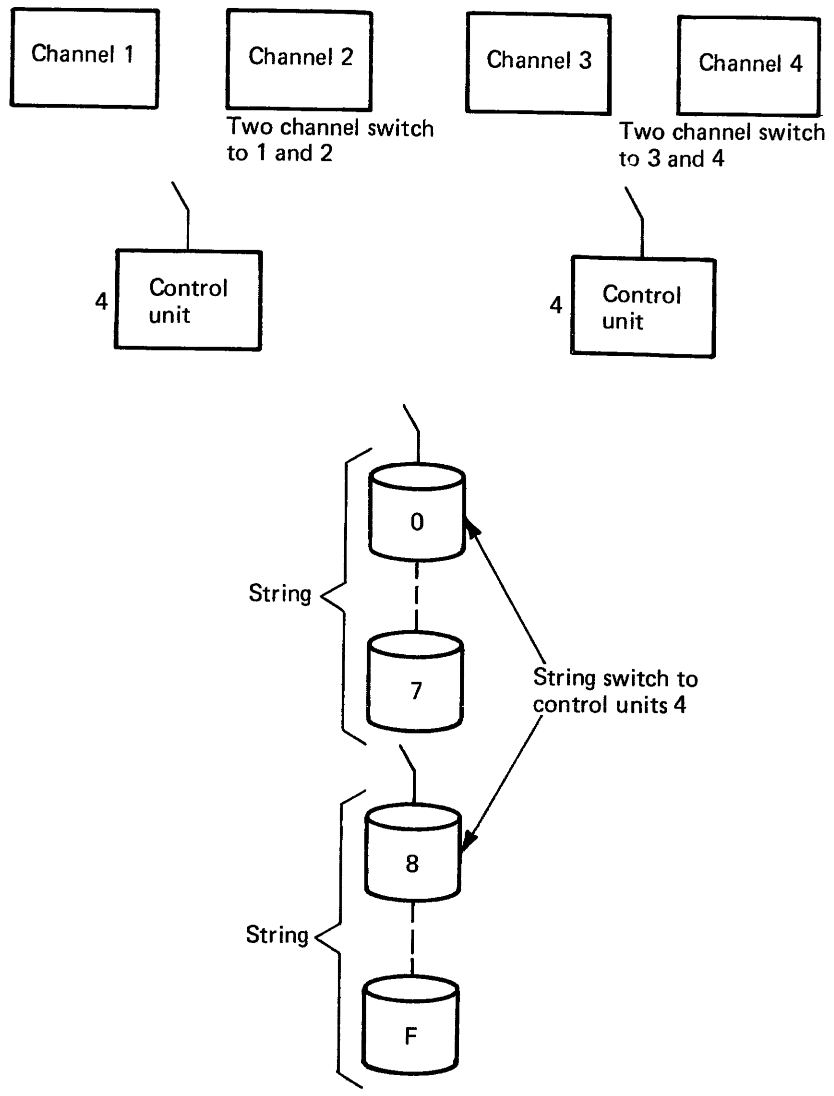

Channel 1
Channel 2
Channel 3
Channel 4
Two channel switch to 1 and 2
Two channel switch to 3 and 4
4
Control unit
4
Control unit
0
7
String
String switch to control units 4
8
F
String

Figure 3.10 Two channel switch plus string switch

With switching carried to the maximum on current hardware, one direct access device may be accessible by eight different CPUs. A device may be string-switched between two control units; each control unit (3830 Model 2 only) may have a four channel switch going to channels on four different CPUs. Figure 3.11 shows this configuration.

If the channels and/or control units to which a device may be switched are on the same CPU, the programming support is called Optional Channel support. If the channels and/or control units are on different CPUs, the support is called Shared DASD. Both forms of support are applicable if the device is reachable via more than one channel on one CPU, and also by more than one CPU.

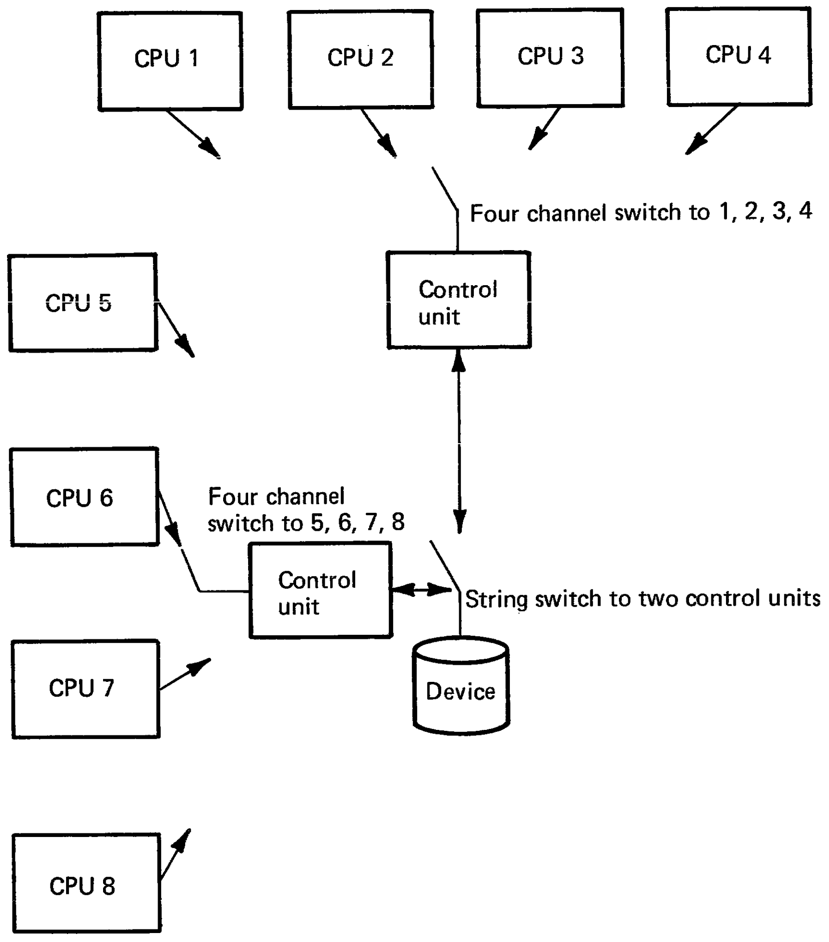

CPU 1
CPU 2
CPU 3
CPU 4
Four channel switch to 1, 2, 3, 4
CPU 5
Control unit
CPU 6
Four channel switch to 5, 6, 7, 8
Control unit
String switch to two control units
CPU 7
Device
CPU 8

Figure 3.11 *Maximum switching configuration*

Hardware Switching

Two Channel Switch

The two channel switch allows a single control unit and its attached devices to be accessed by two different channels on the same or different CPUs. At any point in time, only one of the channels can be transmitting data to or from a CPU and one of the attached "switched" devices. However, Seeks and Set Sector commands previously initiated by either or both of the channels may be simultaneously active on any of the other devices.

The switch has three settings, one for each of the two channels involved in the switching and a neutral position. When the switch is in the neutral position, no data path is completed, and no data is being transmitted to or from either of the channels involved. The switch will be set to the channel position which first selects a device on the switched control unit (i.e., starts an I/O operation to a device).

Once the switch is set to the channel doing the I/O, no path can be completed between the switched control unit and the other channel until the switch returns to the neutral position.

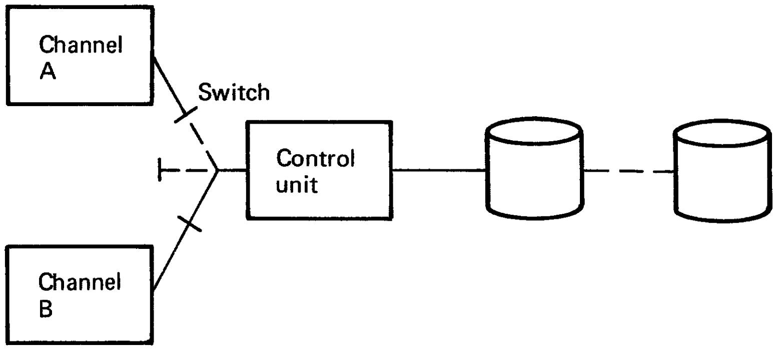


Figure 3.12 Two channel switch

In Figure 3.12, while channel B has the switch, channel A cannot access any device on the control unit (either the same one channel B is using or a different one).

Four Channel Switch (Two Channel Switch Additional)

The four channel switch functions identically to the two channel switch except that the switch has five positions - four denoting the channel interfaces by which it may be selected and a neutral position. Any two (and only two) of the interfaces may connect to channels on the same CPU. The four channel switch is available only on the 3830 Model 2 Storage Control Unit and cannot be installed when 3340s with the Fixed Head feature are attached.

String Switching

The string switch provides switching capability at a lower level than the two or four channel switch. The term "string" applies to a group of two to eight drives headed by either a 3333 Disk Storage and Control or 3340 Model A2 Direct Access Storage Facility. With the feature of string switching added to the 3333 or 3340 Model A2, any of the attached drives may be accessed by either of two control units on the same or different CPUs. The control units may be any two of the following:

- 3830 Model 2
- Integrated Storage Control on S/370 Models 145, 158, 168
- Integrated File Adapter on S/370 Model 135

At any point in time, only one data path between one of the control units and the string will be completed by means of the programmable string switch. However, previously initiated Seeks and Set Sector commands may be simultaneously active on the devices in the string from either control unit.

The standard configuration allows two strings to be attached to a control unit. Without the string switch feature, only one string at a time may be transferring to or from the single control unit. If the strings are attached to another control unit via the string switch feature, devices in both strings may be simultaneously transferring data to or from the different control units.

The string switch is functionally similar to the two channel switch, having three positions: one for each control unit interface and a neutral setting. The first control unit to select a device in the string gets the switch set to its interface. Should the other control unit attempt to Start I/O to a device in the same string, its request will be terminated with Device Busy. It will receive a Device End for the appropriate device when the switch returns to neutral.

Operator Action

The operator panel on a control unit with the two or four channel switch feature has one toggle switch for each channel interface that can access the control unit (see Figure 3.13).

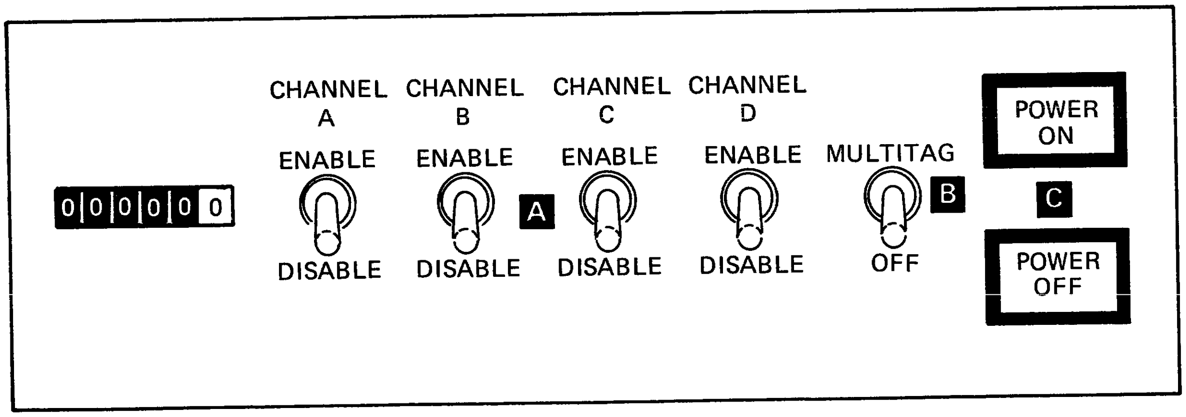


Figure 3.13 Operator Panel - 3830 Model 2

The toggle switch must be set to the "enable" position before the control unit is available to the designated channel represented by the toggle switch. If the switch is in the "disable" position for a channel, and the channel attempts to start I/O through the control unit, the Start I/O will be rejected with a condition "control unit not operative" to the CPU program, and makes *all* drives attached to the control unit unavailable to the disabled channel interface.

The multitag toggle switch on the operator panel is also used with the two or four channel switch feature. It controls the handling of device end status when a drive attached to the control unit goes from the not ready to the ready state. This occurs when a disk pack or data module is mounted on the drive.

When the toggle switch is set to multitag, *all* channel interfaces not disabled will receive the device end status. If a device is reserved to a channel and the switch is set to multitag, only the reserving channel is given the device end status resulting from a disk pack or data module mount. With the toggle switch set to off, the device end status will be presented only once, to the first channel that accepts it.

The switch should *always* be set to multitag when multiple CPUs are sharing the control unit, so all CPUs are aware of volume changes on the drive.

When string switching is utilized, an external switch on the 3333 or 3340 Model A2 may be set by the operator to dedicate the string to one control unit or the other to which it may attach. If a control unit attempts to access a drive in a string disabled by the switch, its Start I/O will be rejected with a condition "path not operative".

There is no external multitag toggle switch for the 3333 or 3340 Model A2 for the string switch. In effect, it is built into the device and always set to the multitag position.

Optional Channel Path

Optional Channel Path is the term used to describe the capability of accessing a direct access device through more than one channel on the same CPU. This can occur when either:

1. A two channel switch attaches the device's control unit (ISC) to two channels on the same CPU.
2. A four channel switch attaches a device's control unit to two channels on the same CPU.
3. String switching is utilized in the 3333 or 3340 Model A2, and the "control units" to which they are switched are on two channels of the same CPU. The "control units" may be any two of the following: 3830 Model 2, ISC, or IFA.

The number of optional paths a device may have depends on hardware and software.

Reasons for Optional Channel Path

- Availability

  Depending upon the type of switching used, the impact of a channel or control unit outage is greatly reduced when the device may still be accessed through an alternate path.

- Performance

  There *may* be an increase in CPU/channel overlap causing improved performance, as the additional paths could cause the channel load to be leveled. This, of course, would depend on the total configuration, system activity, and the nature of the workload.

Shared DASD

"Shared DASD" is the term used to describe the capability of accessing direct access devices from two or more computing systems. The accessing of a particular device is not concurrent from the computing systems sharing it, but sequential, the sequence being determined by both hardware and software.

Sharing of DASD devices can occur when:

- A two channel switch attaches the device's control unit (ISC) to two channels on different CPUs.
- A four channel switch attaches the device's control unit to channels which are on different CPUs.
- String switching is utilized in the 3333 or 3340 Model A2, and the control units to which they are switched are on channels of two different computing systems. The control units may be any two of the following: 3830 Model 2, ISC, or IFA.

The term "Shared DASD" is sometimes confused with the other levels of sharing data available within a multiprogramming environment:

- A data set may be shared between tasks of one job in one computing system.
- A data set may be shared between two or more jobs in the same computing system.
- A data set may be shared between jobs executing in *different* computing systems. This is the situation to which the term *"Shared DASD"* is applied in this document and in all other IBM literature. The sharing requires the use of some kind of hardware switching, and two or more operating systems are involved.

Shared DASD - General Operation

With the Shared DASD option, an I/O operation may be started to a shared device from any of the CPUs able to access the device by means of the switch. Each sharing computer vies for the programmable switch to gain access into the pool of devices governed by the switch. The first requesting CPU gets the switch set to its interface so it may perform I/O operations to one of the shared devices. When the switch returns to the neutral position, any other CPU, or the same one, may select a shared device and have the switch set to its interface.

It is important to note that none of the sharing computers is aware of what another is doing with the data on the shared devices. Data integrity is the responsibility of the using program. For this reason, the hardware command Device Reserve may be issued by a program to retain exclusive use of one or more shared devices while a critical update to data is being performed. Device Release is issued to terminate the exclusive reservation.

If one of the shared devices has been reserved for exclusive use, the CPU channel through which the Device Reserve was issued will lock out any *other* channel, on the same or a different CPU, from accessing the device. However, the reserving channel must still vie for use of the programmable switch to gain access to its reserved device.

Reasons for Sharing

There are several reasons an installation would elect to share data between computing systems:

- Scheduling of jobs is simplified and operator intervention minimized. Instead of being moved from one system to another as the volume remains mounted and available to each system able to access the data by means of the two (four) channel switch or string switch.

- Updating of data is minimized. One update to a shared data set is needed, instead of the multiple updates that would be required if each of several computing systems had its own copy of the data set.

- Backup and switchover in the event of hardware failure is facilitated in a multi-computer installation if the needed data is accessible to surviving computers without moving it.

- Direct access storage space may be saved, as one copy of the data is required instead of multiple copies.

Sharing direct access devices does require extra consideration and planning in the areas of operations, programming, performance, and recovery.

Rotational Position Sensing

Rotation position sensing (RPS) is an optional feature. Block multiplexer channel support is a prerequisite for RPS. If present, the RPS feature should be on every drive of the string. It will be supported in DOS/VS after June 1974. RPS is transparent at the programmer level if OS/VS or DOS/VS access methods, e.g. SAM, DAM, ISAM, or VSAM, are used. The RPS feature is optional because the performance advantages cannot be realized in all system configurations and environments. The 3330, 3340 and 2305 direct access storage devices have the RPS feature as a standard part of the device configuration.

Rotational position sensing reduces the time the channel is busy searching for a record. This procedure permits a search command to be initiated just before the desired record is positioned under the read/write heads.

To accomplish this, a "sector" concept is employed. The tracks in each cylinder of a disk storage drive are divided into equally spaced sectors; each record on the track has a sector location as well as a record address. Although the sector location is not physically indicated on the tracks, the sector number is stored at the beginning of all read, write, and search commands. When chained to a read, write, or search CCW, the read sector command provides the sector number required to access the record processed by the previous command. A subsequent set sector command can be used to fetch the sector number from main storage to reposition the track at that record. This type of operation is particularly useful in write verification (Figure 3.14) and sequential disk processing operations.

The sector in which a record is recorded is a function of the length of all records that precede it and its sequential position on the track.

The following example shows some of the advantages of using rotational position sensing to locate and retrieve records.

Without RPS

Channel program 1.

Command	Selector Channel and Storage Control Status
Seek	Available as soon as the storage control accepts the seek address.

Channel program 2.

Command	Selector Channel and Storage Control Status

Search ID Equal	Busy (average 12.5 ms on the 2314).
TIC *-8	
Read Data	Busy.

With RPS

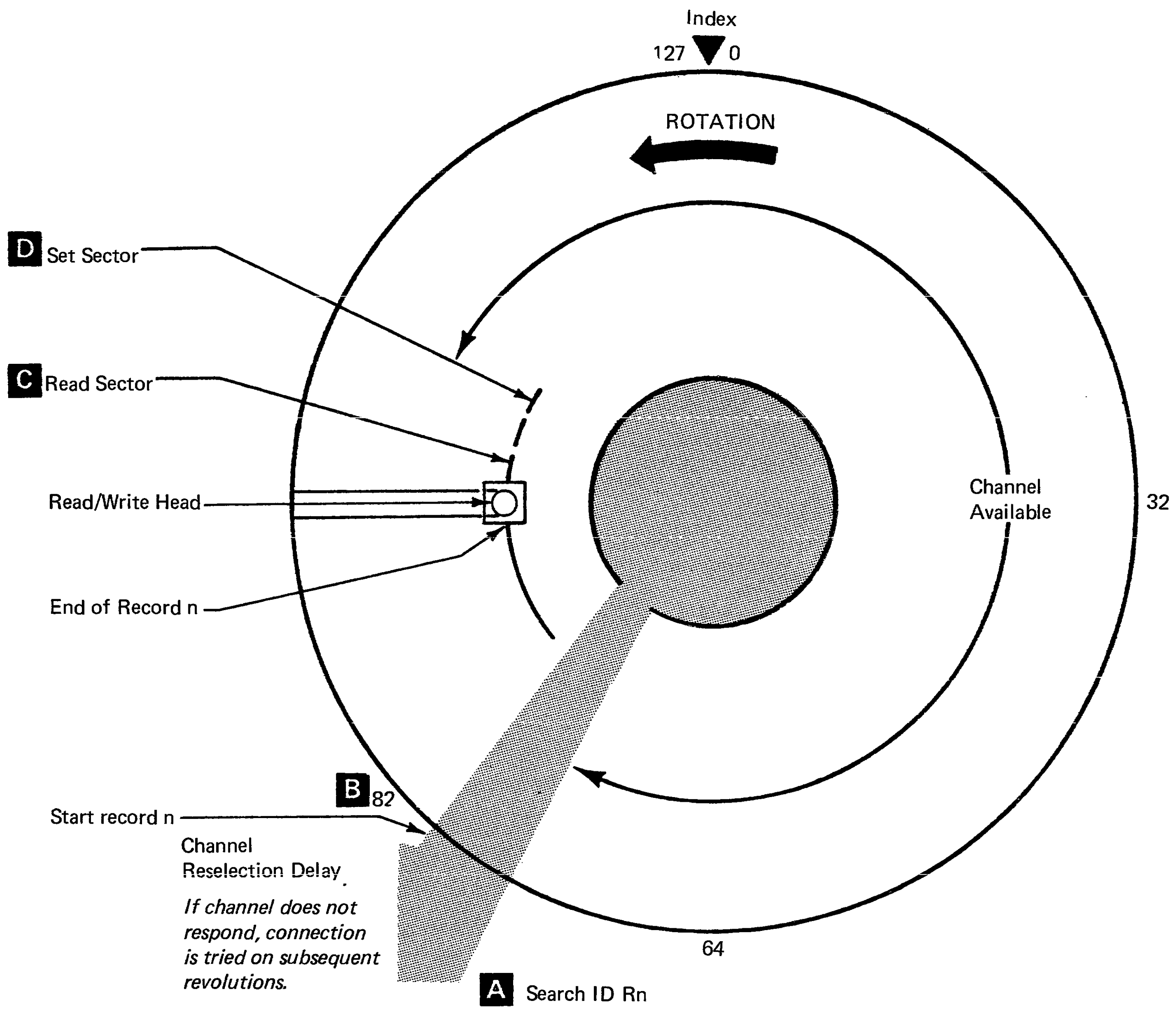
When the sector address is known or can be calculated, the following channel program can be used:

Command	Block Multiplexer Channel and Storage Control Status
Seek	Available during access movement.
Set Sector	Available until sector is located.
Search ID Equal	Busy (average 250 ms on the 3330).
TIC *-8	Normally the first ID read is that of the desired record and the TIC is not executed.
Read Data	Busy.

Note that with RPS only one channel program is required to locate the record and transfer the data. This eliminates a seek I/O interrupt and the I/O processing required to schedule a data transfer channel program.

Also, the channel and disk storage are available during access motion and rotational positioning, allowing seek and set sector operations to be overlapped with other I/O operations on the storage control and channel.

RPS is used in conjunction with block multiplexing. This significantly reduces the time required by the I/O channel and control unit to search for a particular record location on a track (as pointed out in the previous example). RPS permits multiple requests to be active on the channel at the same time, thereby providing increased device utilization. Depending on the number of devices and degree of multiprogramming within the system, the increase device utilization improves the direct-access storage device throughput.



Channel program for write verification of record n.

Seek

A Search ID Rn
TIC *-8

B Write data Rn

C Read Sector (82)

D Set Sector (82)
After channel reselection:
Search ID Rn
TIC *-8
Read data Rn

3330 – 128 Sectors (shown in this Figure)
2305 – Model 1 – 90 Sectors
Model 2 – 180 Sectors

Figure 3.14 Rotational Position Sensing

4

Introduction

Direct Access Storage Devices (DASD) have steadily grown in capacity and speed, but only data with a relatively high value and use can be maintained on DASD permanently, for two reasons. First, when we calculate the cost per megabyte of DASD storage (including the cost of drives, control units and packs), it is substantially higher than the corresponding cost for tape. Second, for many users, the total volume of data collected and maintained cannot fit on even the maximum configuration of DASD. For these reasons, the data processing community has come to recognize the need for an economical mass storage device.

Tape systems also have grown in terms of capacity (recording densities) and in speeds. However, three characteristics of tape devices limit their usefulness. First, data stored on tape is inherently sequential, so that random or direct accessing is impractical. Second, tape volumes are typically not mounted until they are called for, whereas most DASD packs tend to remain more or less permanently mounted. This causes human intervention, which implies both a time delay in mounting the tape and a larger possibility for human error than is typically encountered with DASD. Third, usually only one data set is stored on each reel of tape. An entire, unsharable device is required for each mounted reel of tape, regardless of the activity rate. This can result in low utilization of tape drives and it adds to scheduling complexity.

It would be beneficial, in most installations, to have another alternative for data storage—a mass storage system. Such a system should have:

- Capacity equivalent to a tape library.
- Availability and mounting of volumes under control of the operating system.

- Data organizable in the variety of methods available on DASD.

- Data transfer rates approximately equal to DASD.

- Cost per megabyte of data storage closer to tape costs than to disk costs.

The 3850 Mass Storage System (MSS) is IBM's response to these requirements.

Consider another trend in data processing: with the advent of larger memories, the user can sustain a higher level of multiprogramming than before. That is, more tasks can be handled concurrently.

Each task—whether a batch job step or initiated by a terminal user—requires access to *data sets.* These data sets reside on *volumes,* which must be mounted on I/O *units.* Increasing the number of concurrent tasks increases the requirements for available units, volumes, and data sets. Thus, as the number of concurrent tasks supported by a system increases, the data *requirement* of the system increases.

In the past, as a system's requirement for data has increased, the only solution has been to attach more tape drives and disk drives to the system. However, the additional drives—along with the additional tape reels and disk packs required by these drives—generate operational problems, such as:

- The library storage and retrieval procedures for tape reels and disk packs grow larger and more cumbersome. The time for a volume to be located in the library increases and the time for a volume to cycle from demount to the library and become available for re-use grows significantly longer. (Already, in some installations, this time is as long as 24 hours.)

- Security of volumes in the computer room usually is sacrificed for speed of recycling through the library.

- Operators must mount/demount more volumes on more devices, increasing the time between mount request and mount complete, and increasing the probability of mount errors and subsequent rerun/recovery costs.

- The costs of the I/O devices increase and the requirement for floor space in the library and the computer room grows larger.

Once the additional I/O units are installed in a complex environment to support more concurrent tasks, it is difficult to maintain balanced systems utilization. There will be occasions when even more devices are needed, while at other times not all available units will be required. The MSS addresses these problems–satisfying user requirements for an economical, large capacity storage device with a sophisticated technology that extends the concepts of virtual storage to the I/O components of a computer system.

The Mass Storage System—an Overview

The IBM 3850 Mass Storage System consists of IBM 3330 Disk Storage and one or two IBM 3851 Mass Storage Facilities.

The MSS provides very large data capacity, all under system control. The capacity range of MSS is from 35 billion (35×10^9) bytes to 472 billion (472×10^9) bytes (4,720 3336-1 volumes).

Data under control of the MSS is stored in DASD format images on low cost media (magnetic tape in data cartridges) until it is needed. The MSS transfers data from the data cartridges to DASD (3330 drives), when it has been requested, in a process called *staging*. Once data has been staged, it behaves precisely like any other data resident on a 3330 in terms of organization and accessibility. When the data is no longer needed, and the space it occupies on DASD is required for other data, any "cylinder" containing new or updated data is *destaged* back onto the data cartridge (Figure 4.1).

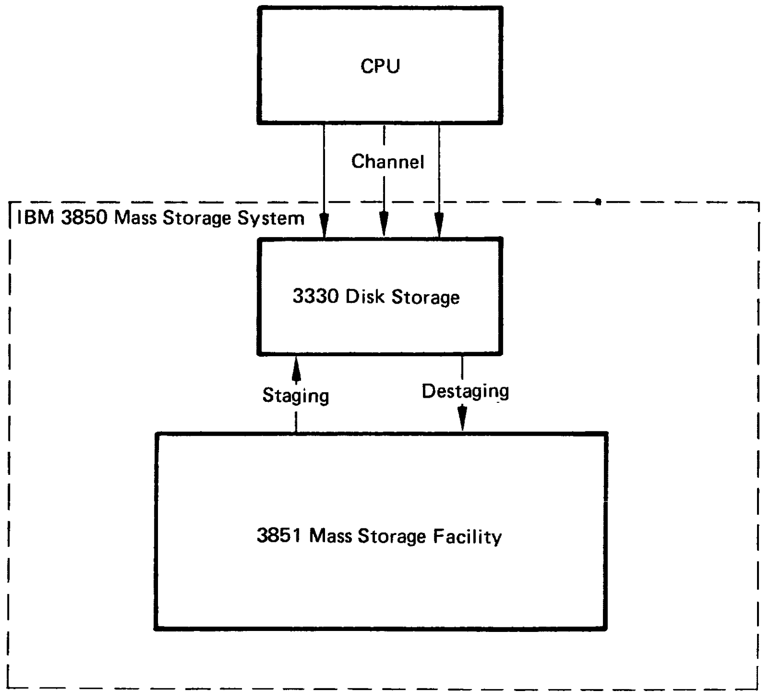
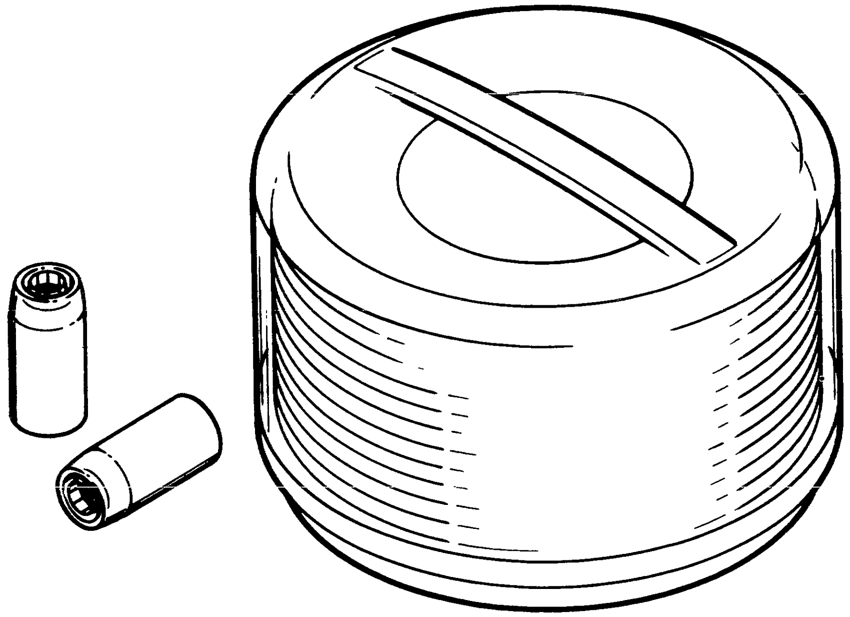


Figure 4.1 Mass Storage System Data Flow

All cartridges under control of the system are physically within the 3851 Mass Storage Facility (MSF) of the MSS. Floor space for storage media is greatly reduced. Many tape data sets may now be stored as DASD data sets, reducing or even eliminating tape drive and tape reel requirements.

Each data cartridge can hold up to 50 megabytes of data, so two cartridges are required to store the data capacity of an entire 3336 Model 1 disk pack (Figure 4.2). Indeed, two cartridges are always initialized together and the pair is referred to as a *mass storage volume* (MSV) and the MSV is managed by the Mass Storage System. 3336 Model 11 disk packs contain the equivalent of two MSVs.



1 Mass Storage Volume=2 Cartridges=1 3336 Model 1 Disk Pack=100 Million Bytes

Figure 4.2 Mass Storage Volume Capacity

Because MSVs when staged appear as normal DASD volumes to OS/VS, the operating system issues mounts for these volumes onto DASD drives. However, only the portions of a volume that are requested will be staged onto DASD. The MSS hardware has been designed to present to OS/VS the image of having more DASD devices available than are physically present (the *virtual device* concept). OS/VS and user programs request data as before, and the hardware translates these requests to obtain data from its actual location on physical DASD or in the Mass Storage Facility.

This eliminates the time to manually pick tape reels from a library and greatly reduces mount times. The cycle time of a volume through a library system is dramatically shortened, because *all* volumes are stored and retrieved under control of the system.

To assist in managing up to 4,720 volumes, IBM provides new operating system functions and utilities. These routines help convert data sets to mass storage volumes, keep track of space availability, group mass storage volumes for simplicity of management, and create backup.

The MSS is a hierarchical storage mechanism that combines many of the advantages of tape and disk systems with potential for operational benefits in a complex environment (Figure 4.3).

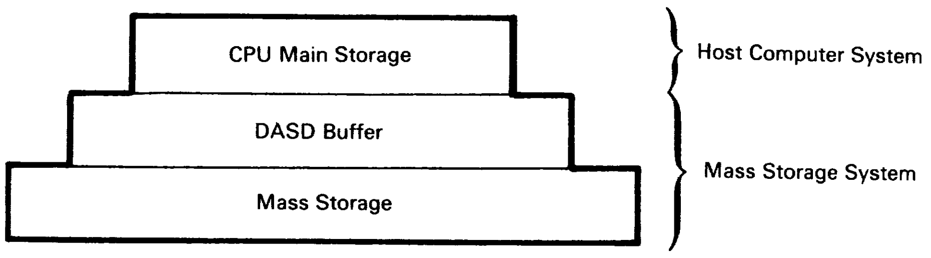


Figure 4.3 Mass Storage System Hierarchical Store

Data is contained in the Mass Storage Facility, the DASD buffer, or main storage as it is required, and migrates up and down in the hierarchy as its use dictates.

The Mass Storage System—Hardware Functional Description

The major components of the Mass Storage System are (Figure 4.4):

- The 3851 Mass Storage Facility (MSF)
- The 3830 Model 3 Storage Control
- The 3333 Disk Storage and Control (Models 1 or 11)
- 3330 Disk Storage Drives (Models 1, 2, or 11).

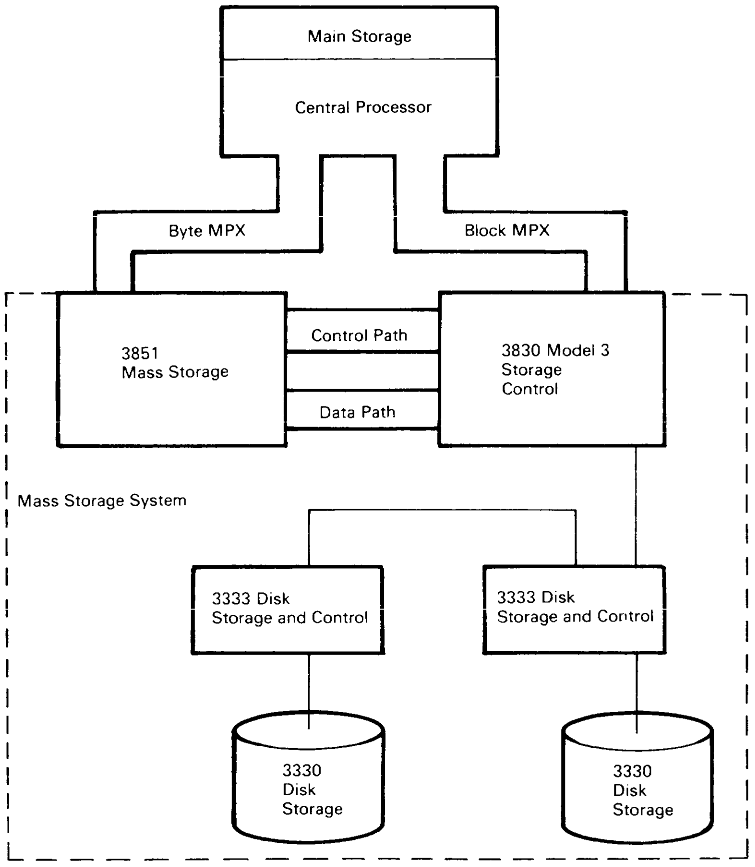


Figure 4.4 Mass Storage System Configuration

Optionally, the System/370 Integrated Storage Control feature on the Models 158 and 168 can be equipped with a Staging Adapter. The Staging Adapter provides identical function to two 3830 Model 3 Storage Controls except for the number of channels that can transmit data to host processor(s). When a Staging Adapter is installed on an Integrated Storage Control, the entire ISC (both sides) is equipped to operate in the MSS. One half of the ISC cannot be so equipped without the other. In this chapter references to the 3830 Model 3 Storage Control also apply to an Integrated Storage Control with the Staging Adapter.

3851 Mass Storage Facility (MSF)

Each 3851 Mass Storage Facility contains up to 236 billion (236 x 10^9) bytes of data and moves data to and from DASD as needed. Two 3851 MSFs can be included within the same 3850 Mass Storage System.

The 3851 Mass Storage Facility includes these major components:

A. Data cartridges to store the data in the storage cells.

B. Data Recording Controls (DRCs) and Data Recording Devices (DRDs)–to transfer data between the MSF and the 3830 Storage Control.

C. Cartridge Access Station–to accept data cartridges into or remove them from the MSF.

D. Accessors to retrieve and deliver data cartridges and accessor controls to control the accessor movements.

E. Mass Storage Control–to provide overall control of MSS functions.

Most of the 3851 Mass Storage Facility components are shown in Figure 4.5. Data is stored on data cartridges that reside in the cartridge storage cells. When data is requested, the accessor moves to the appropriate storage cell, selects the cartridge from the cell, and transports it to a data recording device. The cartridge is then loaded into the DRD, which reads data from the tape and transmits it to a 3830 Storage Control through the data recording control.

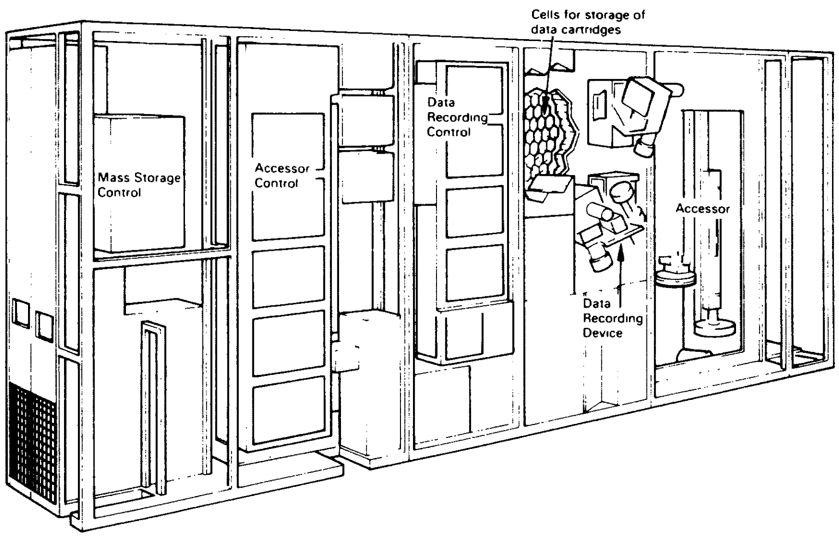
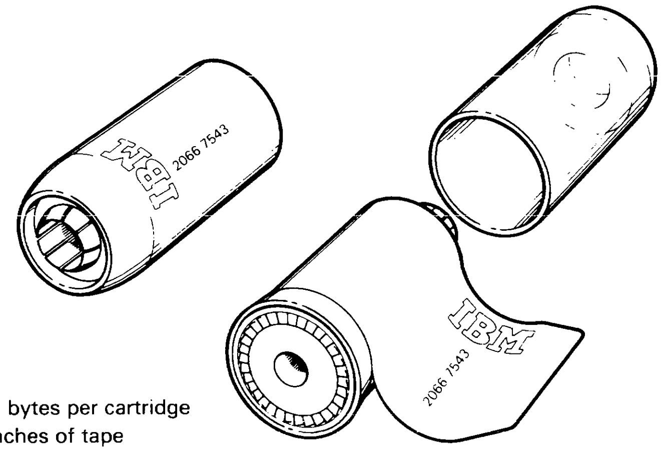


Figure 4.5 3851 Mass Storage Facility Components

When the required data has been completely staged, the data cartridge is returned to its home storage cell. The staging/ destaging operations are supervised and directed by the Mass Storage Control.

Data Cartridge

The 3850 Mass Storage Facility uses data cartridges to store data. Each cartridge can contain up to 50.4 million bytes of data; the cartridge is approximately 2 inches (5.08 cm) in diameter and 4 inches (10.16 cm) long. The cartridge holds a spool of tape 770 inches (17.5 m) long (Figure 4.6).



50.4	million bytes per cartridge
770	total inches of tape
677	inches of usable tape
2	inches diameter
4	inches length

Figure 4.6 *Data Cartridge*

Data is recorded on the magnetic tape in the image of 3330 Disk Storage cylinders. Each cylinder is recorded on a fixed location on the tape, and specific cylinders can be located by unique identifiers along the edge of the tape. Each cartridge is capable of storing 202 cylinders in the 3330 format. Therefore, two cartridges are the equivalent of one 3336 Model 1 Disk Pack.

The *mass storage volume* is the storage reference for data sets within the Mass Storage System, just as tape volume and disk pack volume are the storage references for tape and disk data sets.

Mass storage volumes within the 3851 Mass Storage Facility have the following characteristics:

404 cylinders

19 tracks per cylinder

13,030 bytes per track maximum

100 megabytes

Data Recording Device

The 3851 Mass Storage Facility includes one to four pairs of data recording devices (DRDs) and one data recording control (DRC) for each DRD pair for reading and recording data from and onto cartridges. The DRD also has a high speed search capability that makes storing multiple data sets on one cartridge practical.

Cartridge Access Station

The 3851 Mass Storage Facility has a cartridge access station which allows manual entry and removal of cartridges. There are separate ports for entry and exit of cartridges.

Accessors

Two accessors and associated accessor controls and power supplies are included in every model of the 3851 Mass Storage Facility. The accessors provide for the movement of data cartridges in the Mass Storage Facility; that is, from a storage cell to a data recording device and back.

The Mass Storage Control (MSC)

The Mass Storage Control (MSC) in the 3851 Mass Storage Facility controls all staging and destaging operations. The MSC's functions include:

1. Accepting requests for data from up to four System/370 CPUs, Models 145, 155I, 158, 165II, 168, 158 multiprocessor and 168 multiprocessor.

2. Determining the location of data by referring to an inventory list of cartridges and mass storage volumes.

3. Allocating space in eight-cylinder increments on staging DASD for data to be staged.

4. Instructing the accessor controls to move a cartridge containing the requested data from its storage cell to a data recording device.

5. Initiating the staging of the data from the cartridge in the DRD to the allocated space on disk storage.

6. Performing error recovery procedures, alternate path retry, and device reallocation as needed during the staging/destaging operation.

7. Monitoring the amount of allocable disk storage space. When the amount of allocable space becomes less than a specified threshold (or when the host system so instructs), initiating deallocation and destaging of changed cylinders to create additional allocable space available for other data set requests. The criteria for selection of cylinders to be destaged is based on a least-recently-used (LRU) algorithm.

8. Maintaining usage and error statistics by component and cartridge.

9. Recording the physical configuration of the MSS:

   - Data and control paths

   - Status of components

   - Automatic switching of components

10. Maintaining a record of the location, attributes, and status of all mass storage volumes.

As OS/VS controls the movement of data between 3330 Disk Storage and the CPU, the Mass Storage Control controls the movement of data between Disk Storage and the Mass Storage Facility.

Model and Feature Options of the 3851 Mass Storage Facility

There are eight models of the 3851 Mass Storage Facility; they differ in (1) size and (2) the number of MSCs they have.

The following table shows the differences among models:

	Models A1,B1*	Models A2,B2*	Models A3,B3*	Models A4,B4*
Billion bytes (10^9) of storage capacity	35	102	169	236
Maximum Number of Data Cartridges	706	2,044	3,382	4,720
Mass Storage Volumes	353	1,022	1,691	2,360
Data Recording Controls	1	2	3	4
Data Recording Devices	2	4	6	8
System 370 Channel Interfaces with Two Channel Switch, Additional	4	4	4	4
MSC Ports with MSC Twin Port Feature**	2	2	2	2

* "A" Models have one MSC each; "B" Models have two (one is for backup).

** One MSC Port allows the MSC to attach to eight Mass Storage System elements.

Figure 4-7

An *element* is a 3851 Mass Storage Facility, a 3830 Model 3 Storage Control or an Integrated Storage Control with the Staging Adapter (each ISC counts as two elements). The maximum configuration with one MSF consists of the MSF and seven 3830 Model 3 Storage Controls (eight if the Twin Port feature is installed). The maximum configuration with two MSFs consists of the two MSFs and fourteen 3830 Model 3 Storage Controls. When the MSS configuration includes two MSFs, each staging drive must have a path to each MSF. Paths to an MSF can be provided by attaching the 3830 to each MSF or by attaching two 3830s (one to each MSF) addressing a common set of 3333s via the String Switch. This effectively limits the number of staging drives in a configuration with two MSFs to 128. A minimum MSS configuration consists of an MSF Model A-1, one 3830 Model 3 Storage Control (or an Integrated Storage Control with Staging Adapter), and two 3333 Disk Storage and Controls (must have four disk drives).

Each model can be equipped with two special features:

1. *Two Channel Switch, Additional,* which expands the number of System/370 channel interfaces on the MSC from the standard two to four. With this feature, the MSC can attach to a maximum of four central processors or a maximum of two multiprocessing systems.

2. *MSC Twin Port,* required to increase the allowable number of Storage Controls to 8 in a single MSF configuration and to 14 in a configuration that has two MSFs (an ISC counts as 2 Storage Controls).

Attachment

One or two 3851 Mass Storage Facilities can be attached to IBM System 370 Models 145, 155-II, 158, 165-II and 168, 158 multiprocessor or 168 multiprocessor. If two MSFs are included in an MSS configuration, both must be "A" Models. Both Mass Storage Facilities operate under the control of one Mass Storage Control. The other MSC provides the backup function identical to that in the "B" series. A maximum of two multiprocessor systems may be attached to an MSS.

3830 Model 3 Storage Control

The 3830 Model 3 Storage Control or the Staging Adapter on an Integrated Storage Control Feature performs the following functions:

- Transfers data between disk drives and the CPU, via the 3333.
- Maps virtual device address and logical (cylinder, head and record) address to physical device address and actual (cylinder, head and record) address. For more information, see "Theory of Operations."
- Detects a request for data that is not present on DASD (cylinder fault).
- Requests the MSC to resolve a cylinder fault.
- Transfers data between data recording drives via a data recording control and staging disk drives.

A buffer is provided within the 3830 Model 3 Storage Control to overlap data transfers. Data staged from and destaged to the Mass Storage Facility first goes to the buffer in the 3830 Model 3. While this data is being written from or into the buffer, the 3830 Model 3 can simultaneously transfer data to or from a System/370 CPU.

The 3830 Model 3 has two channel interfaces. One channel interface is used by the Mass Storage Control. The other is used for data transfer to the CPU. Two additional CPU channel interfaces are possible with the addition of the special feature (#8171) Two Channel Switch, Additional. The Integrated Storage Control on the System/370 Models 158 and 168 with the Staging Adapter has two interfaces per path. One interface is used for communication with the Mass Storage Control and the other is for data transfer to the CPU. One additional channel interface is possible per path with the addition of special feature (#7905) Two Channel Switch for ISC.

Up to thirty-two 3330-type drives can be attached to one 3830 Model 3; however, only sixteen 3330 Model 1 or eight 3330 Model 11 drives can be used as staging drives. (The Mass Storage Control considers a 3330 Model 11 Drive as two 3330 Model 1 drives when it is used as a staging drive.)

The 3830 Model 3 Storage Control can attach to a maximum of four data recording controls in a Mass Storage Facility. Each path of an ISC with the Staging Adapter can be similarly attached.

The 3830 Model 3 or ISC with the Staging Adapter has an expanded addressing capability: up to 64 unique addresses per channel interface (maximum of 192 per 3830-3 or 128 per Integrated Storage Control path) are possible. This expanded addressing capability is necessary so that more virtual volumes than physical drives can be simultaneously mounted. (See "Theory of Operations.")

The 3333/3330 units function exactly as before.

The Mass Storage System—Theory of Operations

In order to better understand the operation of the Mass Storage System, keep in mind that the space on all staging drives under control of the MSS is used as buffer storage between the Mass Storage Facility and the CPU. This space is managed by the Mass Storage Control.

Staging Units

Some of the disk drives connected to a 3830-3 can be designated for use as buffers (in which case they are called by any of the names: *staging units, staging devices, staging DASD* or *staging drives*), while others may be designated as standard disk drives (thus, *non-staging units, devices, DASD,* or *drives*).

Although 3336-1s or 3336-11s can be used on a staging drive (thus becoming *staging packs*), virtual volumes (data staged from mass storage volumes) are always in the format of 3336-1 (100 megabyte) packs. Staging packs must be formatted in a special manner:

- The volume serial numbers of all staging packs have the same first four characters (user chosen). The last two characters of the volume serial number must be the MSS identification number.
- The VTOC must be on cylinder zero, track two.
- The VTOC indicates that the entire pack is filled by a single data set (hence, OS/VS cannot allocate space on this pack, which has the effect of reserving the pack for space allocation by the Mass Storage Control).

The format of a staging pack is such that –

- A non-staging pack *cannot* operate on a staging drive without being reinitialized.
- A staging pack *cannot* operate on a non-staging drive without being reinitialized.

Virtual Drive Concept

The implementation of virtual drives is one of the most significant concepts in the MSS. With the MSS creating the image of many 3330 drives, the system is able to satisfy more concurrent requests for devices and volumes and should enable more jobs to run at the same time than with conventional tape/DASD.

Virtual drives are implemented at the 3830-3 level. The standard unit addressing consists of three hexidecimal digits (12 bits). The first four bits designate the channel number, which remains unchanged. However, the *physical* characteristics of the system from the channel level down and assignment of the other eight unit-address bits are:

- A maximum of four 3830-3s can be attached to a single channel, so it takes two bits to select which 3830-3 is being addressed on the channel.
- A maximum of four 3333s can be attached to a particular 3830-3, so it takes two more bits to designate which 3333 is to be selected on the 3830-3.
- A maximum of eight direct access storage drives can be connected to a particular 3333, so it takes 3 bits to specify which drive is being selected.

Thus, to actually designate which drive is being addressed from a channel takes 7 bits. But there are eight bits in the non-channel portion of the device address. Mathematically, this provides a *virtual* addressing capability on each channel interface on a 3830-3 of 64 units–regardless of whether there are only 2 drives or the maximum of 32 attached to the 3830-3.

Tape Recording Organization

Understanding how the MSS stages and destages data requires understanding how data is organized on data cartridges.

The first step to this is an awareness of how tape is threaded onto the data recording devices (DRDs). The tape is wrapped around a read/write mandrel in a helix-like position (Figure 4.8). The read and write heads revolve within the mandrel. If one were to take the tape out of its cartridge and lay it flat, he would see that the resulting recording pattern is a diagonal *stripe* (Figure 4.9).

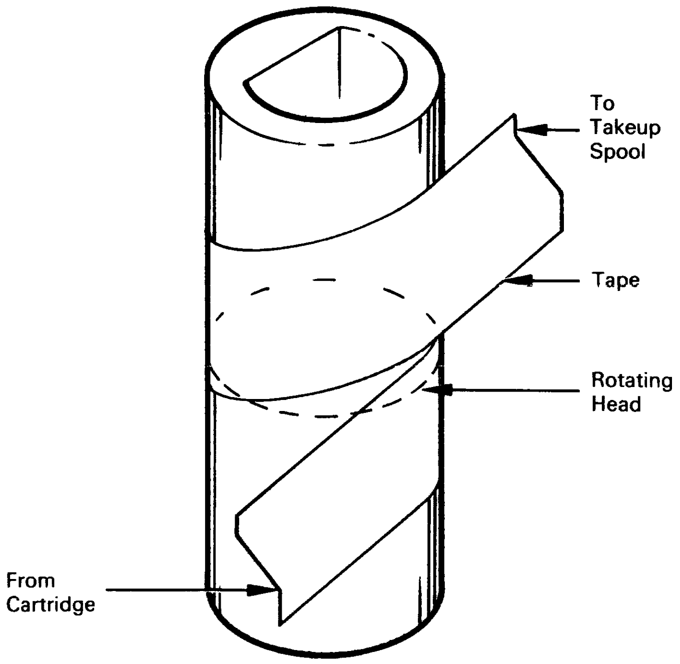


Figure 4.8 Tape Threaded in Recording Position

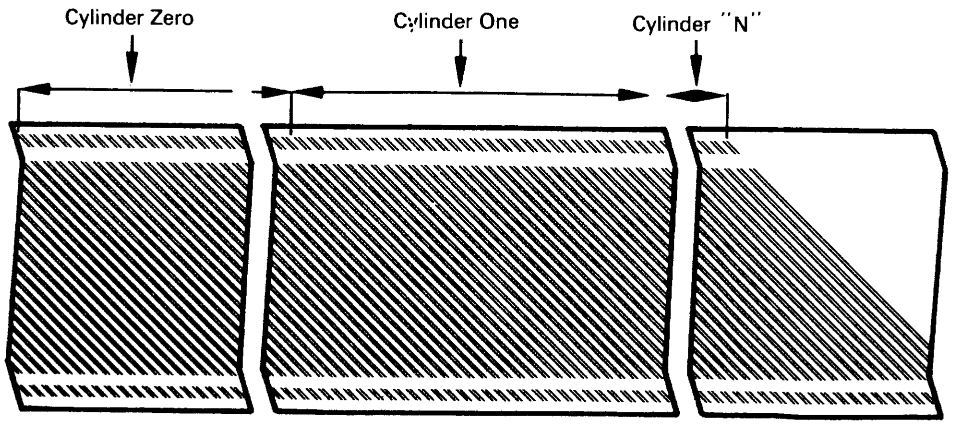


Figure 4.9 Stripe Format

The stripes on the tape are numbered starting with zero. The first ten stripes contain the cartridge label area. There is room for 4096 data bytes per stripe There are 67 stripes allocated per 3336-1 cylinder image. Four of these stripes are alternates and one is a separator. Each stripe carries its own identification or stripe number.

Servo information along the outside edges of the tape assists in positioning the tape properly in the DRD. Included in this information is the stripe number, which assists in the high-speed search for a cylinder on a volume.

Each group of 67 stripes corresponds to a particular cylinder address on a virtual volume enabling a search of stripe addresses to locate a selected cylinder. Since data is always staged and destaged in cylinder increments, data is recorded sequentially from the first track in the cylinder to the first stripe of the 67 stripes assigned to that cylinder. An end of track indication is recorded by the hardware as this condition is encountered. Count, key (if any), and data are recorded, but home address, gaps, and DASD ECC information are not.

Access Method Services Utilities

The use of the Access Method Services utility functions is mandatory to support the MSS. The new functions are implemented via new and modified utility commands specified in utility control statements. These commands are unique to the MSS environment and can be classified as follows:

- Cartridge management commands
- Volume management commands
- Group management commands
- Report generation commands
- MSC control commands
- VSAM data set definition commands

The first four classifications are provided to assist those persons designated as "space managers" to effectively use the data storage capacity of the Mass Storage System.

Manual Cartridge Entry

Entry of data cartridges into the MSF is accomplished by simply placing a cartridge into the cartridge access station. When this is done, the MSF automatically selects the cartridge from the cartridge access station loads the cartridge into a DRD, and reads the cartridge label. If this cartridge is not part of a mass storage volume, it is considered to be a "scratch" cartridge and available for subsequent use as part of a mass storage volume. All *new* data cartridges are considered scratch cartridges. If the cartridge is a scratch, it is then delivered to an empty MSF cell, and an MSC table (Scratch Cartridge List) is updated to reflect the availability and location of the scratch cartridge.

If the cartridge is part of a mass storage volume (not a scratch cartridge), then the cartridge is delivered to an empty MSF cell and an MSC table (Transient Volume List) is updated to reflect the location and identity (mass storage volume serial number and cartridge serial number) of the data cartridge. The MSC will also send an unsolicited message to the MSSC - MASS STORAGE SYSTEM COMMUNICATOR in the primary host CPU. This message contains information that identifies the new cartridge as part of a mass storage volume. The information is then used to update the MSVI - MASS STORAGE VOLUME INVENTORY data set by the MSSC, so that the Inventory data set will reflect the physical presence of the new cartridge now located in the MSF.

Other Utility Functions

Other utility functions include:

- DASD utilities
- Recovery utilities
- Service utilities

The primary DASD utility to support MSS in IEHDASDR. New utility control statements allow the user to specify that the pack being initialized is to be used on a staging drive as a staging pack. This affects the allocation of alternate tracks and other characteristics of the pack.

The major Recovery utilities in the MSS environment are used in the recovery of the MSVI data set. A set of utilities assists the user in the reconstruction of the MSVC data set, using the MSVI Journal data set, if reconstruction is required.

The primary Service utilities for MSS are provided as diagnostic aids to IBM Field Engineers and the MSS Space Manager. An example is the System Data Analyzer, which is described in the "Serviceability" section of this publication.

User Programs

The 3850 MSS was designed for ease of installation in the OS/VS environment. Access to application data sets is via the existing device support for 3330-1, 2, or 11 disk storage devices. Users of tape applications can look at the introduction of MSS as a device conversion to "virtual" 3330s. Tape applications that use device-independent coding techniques can be moved to the MSS with only minor JCL changes. Users of 3330 disk applications can quickly take advantage of MSS by moving data sets to mass storage volumes, and making minor JCL changes.

The primary considerations for executing user programs in the MSS environment relate to JCL, and the access method support needed by the application.

OS/VS Access Method Support of MSS

VSAM

Some options important in an MSS environment have been added to VSAM. Staging options allow the user to choose among three different staging procedures:

1. Staging a data set at OPEN time, which should be the normal way of operation.

2. Staging and binding a data set at OPEN time. A bound data set cannot be destaged by the MSC until the data set is unbound or the volume on which it resides is demounted. Data sets required by high performance applications should be bound.

3. No staging of data at OPEN time. Data is staged one cylinder at a time as required by the host CPU(s). This is called "cylinder fault" mode and could be used for infrequent accesses to very large data sets.

Destaging options permit the user to specify two procedures:

1. Synchronous destaging of a data set at CLOSE time. This option causes the modified cylinders to be completely destaged before control is returned to the user program. If hardware errors are encountered during the destaging process their presence is communicated to OS/VS allowing immediate recovery action.

2. No destaging of a data set at CLOSE time. Modified cylinders are destaged by the least-recently-used (LRU) algorithm of the MSC or when a volume is demounted.

These options are recorded in the VSAM catalog so that they can be used without program or JCL change.

ISAM

No modification has been made to ISAM. This means that there is no support for the staging of ISAM data sets by OPEN. All data is accessed in "cylinder fault" mode. ISAM data sets should be converted to VSAM data sets and processed using the ISAM-VSAM compatibility function of VSAM, which eliminated accessing data in cylinder fault mode.

Other Access Methods

BSAM, QSAM, BDAM, BPAM, EXCP are changed to support staging at OPEN time. But the "BIND" and "CYLINDERFAULT" options are not supported for these non-VSAM data sets. These data sets are always staged at OPEN time.

Figure 4.10 shows the operation of the common access methods

Operation		VSAM	SAM, DAM, PAM, EXCP	ISAM
STAGE	**Stage** Data Set at OPEN Time	Yes (optional)	Yes	-
	Stage and Bind Data Set at OPEN Time	Yes (optional)	-	-
	Cylinder Fault Access to Data Set (No Staging at OPEN)	Yes (optional)	-	Yes
DESTAGE	**Synchronous Destage** of Modified/New Data Cylinders at CLOSE Time	Yes (optional)	-	-
	Asynchronous Destage of modified/new data cylinders based on Least Recently Used (LRU) or volume de-mount	Yes (optional)	Yes	Yes

Figure 4.10 OS/VS Access Method Stage and Destage Options

System Availability and Data Integrity

Computer-based systems have become increasingly important to businesses, governments and other organizations. For this reason, increasing attention has been given to systems availability and data integrity.

System availability is having the system when you want it. Elements of system availability are:

- *Reliability* of the components.
- *Continuity* of system functions.
- *Serviceability* of the components.

Reliability means that the system stays up. When components become unreliable, they may disrupt continuity of system functions. When that happens, the serviceability of the components becomes an important consideration.

Reliability is the measure of the probability—at best an estimate—that the system will do what it is designed to do for a given period of time.

Continuity involves answers to:

- How often will a malfunction occur?
- What effect will malfunctions have on system capability?
- How long will it take to fix?
- How often will preventive maintenance be performed?
- When will engineering improvements be installed?

Serviceability questions are phrased as how easy and fast is a system to fix and maintain and what effect does restoring one element have on another.

Data Integrity means the ability of the system to store, maintain, update, and move data without alteration due to a malfunction. Important data integrity considerations are:

- *Recovery* of data after system failures.
- *Data Security* from unauthorized access, change, or exposure.

Recovery is an automated function in the sense of returning to a normal state after a temporary error in the system. Recovery is also the process of manually restoring a system when it is unable to recover automatically.

Data security refers to protecting data from unauthorized disclosure, modification or destruction by accidental or intentional means.

System Perspective

To look at these aspects of a specific computer-based system, both IBM and the user must have the same understanding and perception of the system. A *system* is not merely an interconnection of computer hardware units, software, and firmware. A system should be perceived—by everyone concerned with it—as a dynamic combination of resources:

- Hardware
- Software
- Firmware
- Data
- Operators
- Application programs
- Normal operating procedures
- Contingency procedures
- Computer room environment
- Management

IBM's 3850 Mass Storage System, properly employed with good systems design, provides an opportunity for improved resource management over conventional tape/DASD in these areas:

1. Tape handling is completely automated. This reduces problems caused by manual handling as dirty leaders, damaged tape, bent reel flanges, etc.

2. Improvements in the recording quality of the tape, reduction in the recording head-to-tape separation, and advances in error correction code provide not only improved data density on the cartridge but more reliable reproduction of the data compared to tape/DASD.

3. Because much more data can be under system(s) control when compared to tape/DASD, normal operating occurrences such as misfiled, mislabeled or lost reels of tape and mismounts are nearly eliminated.

4. Contention for a unique tape data set is also reduced because the data, once staged, can be shared among two, three or four operating systems from DASD, assuming that the data set has been qualified as sharable.

5. Extensive hardware replication and sophisticated, automatic error recovery procedures are integral to the design of the Mass Storage System.

6. The Mass Storage System has been designed so that most maintenance can be accomplished concurrently with productive use of the system.

Disk Pack Swapping

A disk pack can be removed from one staging drive and placed on another of the same model, as is possible with current DASD, to assist in preventive maintenance and help resolve intermittent drive failures. To accomplish this, you must turn off the drive containing the disk pack to be swapped and vary off another staging drive or a designated spare drive on the same 3333 string. The vary off command causes the data on a staging drive to be destaged. Data in use will be restaged on another staging drive. Move the address plug, the pack and VARY ON the new drive.

Serviceability

Serviceability is important because it is a positive influence on availability. The better the design for serviceability of a system, the smaller the impact on availability of failures within that system. A key design objective for the 3850 Mass Storage System has been to maximize concurrent maintenance. In fact, where a Mass Storage System configuration provides replicate components, almost all scheduled and unscheduled maintenance, including problem diagnosis and checkout, can be done while the storage system is in use. This overlap is achieved by varying off only those components required for service. The system continues to operate in degraded performance mode. Another design objective has been to provide better maintenance tools to reduce "long calls". These objectives are served by:

1. Comprehensive Maintenance Library Manuals that are effectively cross referenced and easy for Customer Engineers to use.

2. Improved error checking in the hardware, when compared to previous IBM products, and a more extensive display of sense data.

3. New microdiagnostic programs which are entered in the MSC and 3830 Model 3 by the Customer Engineer from a diskette using a CE console. This is done while the MSS is operating in a normal environment.

4. Improvements to On Line Tests (OLTs) so that the diagnostic intelligence is concentrated in microdiagnostic programs below the host CPU. Some of the functions provided by the improved diagnostic programs are to:

- Isolate different types of failures
- Report cartridge store addresses and conditions of access errors
- Measure accessor motion
- Report addresses and the number of control checks
- Analyze paths
- Test interfaces
- Update microcode

5. A program called System Data Analyzer (SDA) designed to help the Customer Engineer diagnose existing problems more quickly and anticipate failures. Sense data, buffered-usage-log data, and data logged in exceptional circumstances are automatically entered into the SYS1.LOGREC data set during MSS operation. Because (1) this data is voluminous, and (2) the interrelationships of MSS components are complex, the program is valuable in processing logged information.

Data Integrity

The integrity of data stored in the 3850 Mass Storage System is maintained by:

- The extensive use of checking circuitry and diagnostics.
- Parity.
- A new error correction method, Extended Group Coded Recording, for recording data on cartridges.
- DASD error correction coding.
- Multiple recording of cartridge serial numbers.
- Automatic reconstruction of 3830 Model 3 tables.
- Maintaining duplicate Mass Storage Control tables.

Checking and Diagnostics

Data circuits in the data recording control, data recording device, 3830 Storage Controls, Integrated Storage Controls, 3333 Disk Storage and Control, and the 3330 Disk Storage Drive are hardware checked while in use. When they're not in use, a Loop Read to Write diagnostic command checks the data recording control circuits in the Mass Storage Facility. A similar diagnostic command checks data circuits in the DASD subsystem.

Physical travel of each accessor is monitored and the actual physical location address is compared with the address in the command sent to the accessor control that initiated accessor movement. This ensures that the accessor is selecting from the intended location. Each accessor is also equipped with a series of physical interlocks to prevent damage in the event of a failure in the accessor control system.

Extended Group Coded Recording

Data is recorded on cartridges using the Extended Group Coded Recording method. This error correcting code and associated circuitry can correct up to 32 of 208 bytes on-the-fly.

DASD Error Correction Code

As data is transferred from the host channel or from the MSF to disk storage (write operation), the 3830 Model 3 storage control removes the parity bit associated with each byte. It then computes the error correction code bytes, which are written after each recorded area. The correction code bytes, coded to represent the data in the recorded area, are used for both error detection and correction.

As data is transferred from disk storage to the channel or to the MSF (read operation), each area is inspected by the 3830 Model 3 storage control and the error correction code bytes are recalculated for each area. The 3830 correction code corrects single bursts of 11 bits or less.

If a correctable data error is detected in the home address, count, or key areas, the 3830 Model 3 storage control internally executes the error correction function through the use of command retry. If an error in the data area is detected, the host correction function is determined by the system error recovery procedures.

The correction code bytes are removed and proper parity is generated by the 3830 Model 3 storage control before the data is transferred to the channel or to the MSF.

Multiple Recording of Cartridge Labels

When tape is threaded on a data recording device, the cartridge label is read and compared by the Mass Storage Control with the required cartridge label contents. An unequal compare automatically invokes microprogrammed error recovery procedures; an equal compare allows the cartridge to be used. Because of the importance of this label, it is recorded twice in the cartridge label area. The cartridge serial number is also imprinted on the tape so that it can be visually inspected through the transparent cartridge shell.

3830 Model 3 Tables

The 3830 Model 3 contains tables in a control store area which serve the following purposes:

- The Virtual Address table provides information about each virtual address assigned to the 3830-3.
- The Virtual Volume Information table tells about each Virtual Volume mounted such things as whether it is read only, busy, shared and/or reserved.
- The Page Status table allows the conversion of virtual page numbers to real page numbers and virtual unit addresses to logical unit addresses.
- The Logical to Real Conversion table allows the conversion of logical unit addresses to real unit addresses.

Although these tables reside in the 3830-3, they are maintained by the Mass Storage Control. Should a malfunction occur in the 3830-3, the Mass Storage Control refreshes the tables without manual intervention.

Backup

Good systems design requires backup data both for data integrity and to provide a resource for recovery when the MSS is unable to recover automatically or when media damage occurs.

The MSS and host operating systems provide:

- Mass Storage Volume backup
- Mass Storage Volume archiving
- Generation Data groups
- Data set copy

Data Security

Data security is a technical question to be resolved by the use of management-approved and enforced system security procedures, manual procedures, and attention to the physical environment. Data security is an environmental consideration which involves the system functions of:

Identification	Provides the system with a unique, recognizable name for each person, device or system resource attempting to access a system resource.
Authorization	Controls the access of each person, device or system component to a system resource.
Audit	Provides a record of all accesses and attempted accesses to a system resource for review, enforcement and evaluation of approved security measures.
System Integrity	Is the ability of the system to protect itself from unauthorized attempts to compromise or bypass security controls or features.

The IBM 3850 Mass Storage System and OS/VS support these systems functions through password authorization of utility commands, by providing the mass storage volume attributes DASDERASE and READONLY, by providing RPQs designed for physical protection, and by placing more data under system control.

Password Protection

All the utility commands are authorized, because the Mass Storage System Communicator requires an APF Authorization. A user should generally place the utilities in an APF authorized, password-protected library.

The Access Method Services utility commands check data set and VSAM catalog passwords when performing volume operations. For each non-VSAM data set on a volume, the utility commands require the *read-level* password when copying or converting the volume and the *write-level* password when modifying the volume serial number, making the volume inactive, or replacing the volume with a backup copy. For each VSAM data set on a volume, the utility commands require the *master-level* password when copying or converting the volume, making the volume inactive, or replacing the volume with a backup copy. In the above cases, you can specify the master-level password of the VSAM catalog that owns the volume and the utility commands will bypass data set password checking for the VSAM data sets.

Data set passwords are not checked when a copy of a volume is scratched, because the copies cannot be mounted. The SCRATCHV commands, however, can ensure that the volume being scratched is a copy from the volume records maintained in the Mass Storage Volume Inventory.

Conversion

The 3850 Mass Storage System is an extension of the virtual storage concept to data storage. Potential users of the MSS are installing central processors with virtual storage and VS1 or VS2 now. They are also engaged in expanding their application bases.

Conversion of existing applications to the MSS can usually be viewed as an I/O change, thereby conserving resources for continued application expansion. The initial conversion should appear as a tape to DASD conversion, one that is generally familiar.

The design objectives for MSS to ease conversion are:

- Use current production source code without redesign or reprogramming.
- Use current programs without re-compiling or re-linkediting.
- Limit effort to minimal Job Control Language (JCL) changes.

MSS for most users will meet these objectives. This is especially true for users who maintain and develop installation standards.

Compatability Considerations

The compatibility objective is to provide a means by which current applications, using tape or 3330 media, can be stored by the Mass Storage System with minimum effort and expense. From a conversion viewpoint, this has been achieved by making the MSS compatible with the 3330 Disk Storage. With the exception of device-dependent code, the conversion has been reduced to converting a tape data set to a disk data set.

The virtual volume concept is compatible with present 3330 DASD operation. The virtual volume concept is simply an expanded addressing system that permits the host CPU(s) to address and access mass storage volumes as if they were additional 3330 volumes, automatically mounted on demand. The conversion to the Mass Storage System is essentially a conversion to 3330 DASD. In conversion, the user does not have to concern himself with managing the data below the 3330 (data flow between the MSF and the 3330).

For the application programmer, the Mass Storage System is an IBM 3330 with nearly unlimited space. For the systems programmer, the expanded addressing system does not change the parameters he must review; it only increases the total number.

Because the Mass Storage System is compatible with the IBM 3330, the user can convert and test his tape media program/data prior to installation.

VSAM/ISAM Relationships

The access method and catalog recommended for use with Mass Storage System is the Virtual Storage Access Method (VSAM). The current OS/VS access methods also function with the MSS.

Many users will install VSAM and convert at least their Indexed Sequential Access Method (ISAM) data sets. With Access Method Services Utilities, sequential and indexed sequential data sets can be easily converted to the VSAM format. The compatibility routines for existing customer programs, using ISAM, provide access to the VSAM data sets.

ISAM can function with the MSS, but only in a cylinder fault mode. The Mass Storage System will stage a cylinder of data on a cylinder fault. This option can be used for low volume or infrequently accessed ISAM data sets.

VSAM and the Mass Storage System

Virtual Storage Access Method (VSAM) is a DASD access method that operates on relocate versions of System/370 with OS/VS. VSAM supports DASD drives in a device independent manner and therefore the virtual volumes of the Mass Storage System are automatically supported because of their DASD characteristics. Mass Storage System features beyond those of virtual volumes require additional support. These new features include data staging and destaging and volume manipulation.

The VSAM interface for data set open/close/extension is unchanged for Mass Storage System support. The effect of using this interface is unique to the Mass Storage System and is dependent on the VSAM data set options selected through Access Method Services. The following description describes the direct results of using these current interfaces in terms of the Access Method Services external interfaces specified and the Job Control Language parameters selected.

VSAM Catalog and Catalog Management

On OS/VS1 and OS/VS2 systems, any VSAM user catalog can reside on a mass storage volume. The VSAM master catalog cannot be accessed if it resides on a mass storage volume. No change to VSAM catalog management is necessary to support user catalog residence. No check is made during VSAM master catalog or user catalog definition for mass storage volume residence.

Time Sharing Option

Your Time Sharing Option (TSO) programs can operate with the Mass Storage System without any modification of your existing Time Sharing Option data. Foreground jobs will still execute in the same manner as they do in a system that does not contain the Mass Storage System, provided that data sets are on real volumes. However, when archived data sets are used, these data sets can be on mass storage volumes.

Background jobs will be handled just as they are today. When the background job is finished with the data, the data is archived in the Mass Storage System. This means that you must have real drives to be used by Time Sharing Option data and that in the transfer of data to and from the Mass Storage System and Time Sharing Option drives, the CPU channels are used.

How Must the Application Programmer's Job Change

To the application programmer, Mass Storage Volume Control functions mean having to know less about DASD volumes and the Job Control Language than you would have to know in a DASD environment. It means less Job Control Language impact when preparing a job stream. It leaves space management of the volumes, on which space is allocated to the space manager and lets the application program concentrate on the application area.

Where Are the System Programmer Job Changes?

The major change to the system programmer activity is the introduction of two new device types: The Mass Storage Control and the staging drives (3330Vs). Another consideration for the system programmer is the Mass Storage Control tables. Any hardware configuration change, either to the host CPU or CPUs or to the arrangement of the internal I/O units associated with the Mass Storage System, requires an update to the Mass Storage Control's configuration tables.

Introduction to File Organization 5

Records in a file must be logically organized so that they can be retrieved efficiently for processing. This chapter discusses some factors to be considered in selecting a method of organization. It also presents an introduction to the methods of file organization supported by the IBM operating systems for System/360 and System/370.

Data File Characteristics

The inherent characteristics of the file must be considered in selecting an efficient method of organization:

Volatility. This term refers to the addition and deletion of records from a file. A static file is one that has a low percentage of additions and deletions, while a volatile file is one that has a high rate of additions and deletions. No matter how the file is organized, additions and deletions are of significant concern, but they can be handled more efficiently with some methods of organization than with others.

Activity. The percentage of activity is one of the factors to be considered. If a low percentage of the records are to be processed on a run, the file should probably be organized in such a way that any record can be quickly located without having to look at all the records in the file.

The distribution of the activity is also a consideration. With some methods of organization, some records can be located more quickly than others. The records processed most frequently should certainly be the ones that can be located most quickly.

The amount of activity also makes a difference. An active file (that is, one which is frequently referred to) must be organized very carefully, since the time involved in locating records may amount to an appreciable period of time. At the other extreme, an inactive file may be referred to so infrequently that the time required to locate records is immaterial.

Size. A file so large that it cannot all be online (available to the system) at one time must be organized and processed in certain ways. A file may be so small that the method of organization makes little difference, since the time required to process it is very short no matter how it is organized.

The growth potential of the file is also a consideration. Usually, files are planned on the basis of their anticipated growth over a period of time. Initial planning must also consider how growth that exceeds this size will eventually be handled.

Processing Characteristics

The distinction between the organization of a master file and the order of the input detail records processed against that file is important. In *sequential processing*, the input transactions are grouped together, sorted into the same sequence as the master file, and the resulting batch is then processed against the master file. When tape and cards are used to store the master files, sequential processing is the most efficient means of processing. Direct access storage devices are also very efficient sequential processors, especially when the percentage of activity against the master file is high.

Non-sequential processing is the processing of detail transactions against a master file in whatever order they occur. With direct access devices, non-sequential processing can be very efficient, since a file can be organized in such a way that any record can be quickly located.

It is possible, on a run, to process the input transactions against more than one file. This saves setup and sorting time. It may also minimize control problems, since the transactions are handled less frequently.

It is feasible to handle unscheduled transactions. This is particularly significant in a teleprocessing system or in a system where there are many inquiries about the data in the files.

It is not necessary to wait until a batch of transactions has been accumulated to make processing worthwhile. The transactions can be processed inline – that is, as soon as they are available. If it is not necessary to do inline processing of all transactions, most of them can be batched for scheduled runs, and only high-priority or exceptional transactions processed inline – that is, as soon as they enter the system.

The use of a DASD to store a master file makes it possible to choose the processing method to suit the application. Thus some applications can be processed sequentially, while those in which the time required to sort or the delay associated with batching is undesirable can be processed non-sequentially. Real savings in overall job time can only be made by combining runs in which each input affects several master files; the details can be processed sequentially against a primary file and non-sequentially against the secondary files, all in a single run. This is the basis of inline processing.

Methods of Organization

Five methods of organization for direct access devices are supported by IBM programming systems. They are described briefly in this section.

Sequential Organization. In a sequential file, records are organized solely on the basis of their successive physical locations in the file. The records are generally, but not necessarily, in sequence according to their keys (control numbers) as well as in physical sequence. The records are usually read or updated in the same order in which they appear. For example, the hundredth record is usually read only after the first 99 have been read.

Individual records cannot be located quickly. Records usually cannot be deleted or added unless the entire file is rewritten. This organization is generally used when most records are processed each time the file is used.

Partitioned Organization. A partitioned file is one that is divided into several members. Each member has a unique name. Members may be called by name for processing. Members may be added or deleted as required. The records within the members are organized sequentially and are retrieved or stored successively according to physical sequence.

Partitioned organization is used mainly for the storage of sequential data, such as programs, subroutines, and tables. For example, a library of subroutines may be a partitioned file whose members are the subroutines.

Indexed Sequential Organization. An indexed sequential file is similar to a sequential file in that rapid sequential processing is possible. Indexed sequential organization, however, by reference to indexes associated with the file, makes it also possible to quickly locate individual records for non-sequential processing. Moreover, a separate area of the file is set aside for additions; this obviates a rewrite of the entire file, a process that would usually be necessary when adding records to a sequential file. Although the added records are not physically in key sequence, the indexes are referred to in order to retrieve the added records in key sequence, thus making rapid sequential processing possible.

In this method of organization, the programming system has control over the location of the individual records. The user, therefore, need do very little I/O programming; the programming system does almost all of it, since the characteristics of the file are known.

Direct Organization. A file organized in a direct manner is characterized by some predictable relationship between the key of a record and the address of that record on a DASD. This relationship is established by the user. This organization method is generally used for files whose characterisitcs do not permit the use of sequential or indexed sequential organizations, or for files where the time required to locate individual records must be kept to an absolute minimum.

This method has considerable flexibility. The accompanying disadvantage is that although the programming system provides the routines to read or write a file of this type, the user is largely responsible for the logic and programming required to locate records, since he establishes the relationship between the key of the record and its address on the DASD.

Virtual Storage Access Method (VSAM) Data Organization. The data organization for VSAM differs from the preceding organizations so as to establish a data organization that will be, from a user's point of view, device independent. The data organization should be suitable for all kinds of accessing (keyed, addressed, direct and sequential) and should be extendable to anticipated requirements. Once a user adopts the VSAM organization, his data will be portable from system to system. This will facilitate migration from smaller systems to larger systems.

Data records of fixed or variable length are stored in the same format in both key-sequenced and entry-sequenced data sets. The records of a key-sequenced data set are in collating sequence, defined by a key field in the records; the records of an entry-sequenced data set are in the same sequence as the order in which they are entered in the data set. An index is used to physically locate and sequentially order the records of a key sequenced data set.

IBM Operating Systems

Operating systems are part of the programming systems support supplied by IBM. The operating systems include access methods which schedule and control the transfer of data between real storage and I/O devices.

Operating systems that support direct access devices are discussed in this text. They differ from one another in the operating system functions provided and in the machine configuration supported.

Since this text must deal with the operating systems in rather general terms, refer to the texts cited in the Bibliography for specific information on a particular operating system.

Sequential, indexed sequential, and direct methods or organization are supported by all the operating systems. OS and OS/VS also supports partitioned organization. The operating systems allow users to concentrate their programming efforts on processing the records read and written by the access method routines. The responsibility of the assembler language programmer in the area of input/output is essentially to describe the files to be processed and then issue instructions to cause records to be transferred to real storage, and instructions to cause records to be transferred to I/O devices.

The access methods are divided into two catagories: queued access methods and basic access methods.

The queued access methods are used in situations where the sequence in which records are to be processed is known to the system and the programmer wishes the operating system to perform anticipatory buffering and scheduling of I/O operations using the buffers (I/O areas) requested by the user. (More than one I/O area and/or a work area can be specified for a file.) As soon as a channel and device are free, the system can read the next record(s) into the buffers or write the preceding record(s) from the buffers at the same time that the current record is being processed. Therefore, more than one record is normally in real storage at the same time, so that processing and I/O operations can be overlapped. A queued access method, if the records are blocked, performs automatic blocking and deblocking and makes the next logical record available to the user when he issues the next input statement. Queued access methods are provided for sequential organization and indexed sequential organization.

The basic access methods are used when the operating system cannot predict the sequence in which records are to be processed or when the programmer does not want some or all of the automatic functions that are performed by the queued access method. Since the system does not provide anticipatory buffering and scheduling, these can be performed through user programming. Basic access methods read and write physical, not logical, records. Thus, blocking and deblocking of records is (in most basic access methods) the user's responsibility.

As previously implied, access methods are identified primarily by the file organization to which they apply. For instance, we speak of a basic access method for direct organization. Although an access method is identified with a particular organization, there are times when an access method identified with one organization can be used to deal with a file usually thought of as organized in a different manner. Thus, a file that is considered to be a directly organized file is formatted and must be created with the basic access method for sequential organization. It is processed non-sequentially with the basic access method for direct organization.

Virtual Storage Access Method (VSAM) is designed to meet most of the common data organization needs of both batch and inquiry processing. Batch processing requires the efficiency of a sequential organization; inquiry processing requires efficient direct access for random requests. The two types of processing are intermixed in the processing of a common data base.

Both of VSAM's two organizations permit both direct and sequential access. The key-sequenced organization provides quick sequential retrieval in collating sequence; the entry-sequenced organization is suitable for quick record entry and for sequential processing where sequence is not important. A key-sequenced data set can be processed by key as well as by record address; it provides a convenient method of identifying data records. The records of an entry-sequenced data set are identified only by their addresses within the data set.

The two data organizations and VSAM's range of access options permit the user to select the combination that best suits his application.

6 Sequential Organization

In a sequential file, records are written one after the other – track by track, cylinder by cylinder – at successively higher addresses. The records are usually in key sequence.

Description of Records

Records may be fixed- or variable-length, blocked or unblocked, or undefined.

The records may be formatted with or without keys. If the file is always processed sequentially, as is normally the case with this method of organization, there is no point in formatting with keys. If for some reason there is an appreciable amount of non-sequential processing, records should be formatted with keys so that they can be located more quickly.

DASD Storage Requirements

The amount of DASD storage required is simply enough to hold all the records in the file. The area should be large enough for the maximum number of records, although it is permissible to have the file extend over several noncontiguous areas.

Timing

Sequential Processing

The time required is one seek per cylinder and one read per record (or block of records). Remember that in this text we are using a simplified timing approach of allotting a full rotation for each read (or write) to include both rotational delay and data transfer.

Non-Sequential

Non-sequential processing of a sequential file is, at best, very inefficient. If it is done infrequently, the time required to locate the records may not matter. There are several ways to program non-sequential processing, with significant differences in the time required. The slowest way is to read the records sequentially until the desired one is located. On the average, half of the file would have to be read. A sequential search takes less time if the records are formatted with keys. The search is done on Search Key High or Equal at the speed of one revolution per track. When the search condition is satisfied, the corresponding record is read.

Another way of processing a sequential file in a non-sequential fashion is first to perform a binary search of the file in order to determine in which small section of the file the desired record is located. Then only that small section need be searched in full. A binary search of an eight-cylinder file formatted with keys is illustrated in Figure 6.1. The last record in cylinder 4 is read and compared with the search argument. Then the last record in either cylinder 2 or 6 is read and a comparison performed again. Then, depending on the result of that comparison, the last record in either cylinder 1, 3, 5, or 7 is read and compared against the search argument. This last comparison indicates in which one of the eight cylinders the desired record is to be found. That cylinder can then be searched in full.

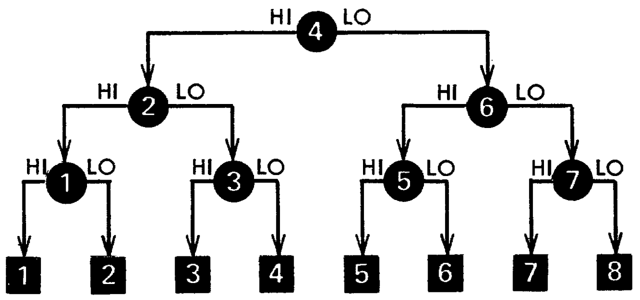


Figure 6.1 *A binary search of an eight-cylinder file*

File Maintenance

Additions and deletions require a complete rewrite of a sequential file. This is desirable from a timing standpoint only if additions and deletions can be combined with another job that also requires reading and updating all the records.

Uses for Sequential Organization

Sequential organization is used on direct access storage devices primarily for tables and intermediate storage rather than for master files. Its use is recommended for master files if they have a high percentage of activity and if virtually all processing is sequential.

Operating System Functions

Queued Access Method

The queued access method is used for creating a sequential file and for reading or updating all of the records in physical sequence. The operating system takes care of any required blocking or deblocking of records. It provides anticipatory buffering, overlap of input/output with processing and error checking.

Basic Access Method

The basic access method does not provide anticipatory buffering and blocking/deblocking routines. The basic access method can be used to read or write records formatted with keys (OS, OS/VS only)or without keys (OS,OS/VS and DOS). It can be used, to a limited extent, to store and retrieve records non-sequentially. Note that the DOS basic access method for sequentially organized files does not permit the processing of files formatted with keys. A basic access method for directly organized files may be used in DOS to create and process (sequentially or non-sequentially) such files. A corresponding access method exists in OS that can be used to process sequential files formatted with keys non-sequentially.

User Options

The operating system performs a Write Verify after write operations if the user so requests. OS only supports the Record Overflow feature.

7 Partitioned Organization

A partitioned data set consists of several sequential units or members. The data set also includes a directory containing the name and beginning address of each member. This method of organization is supported only by OS.

Description of Records

The records in the members may be fixed-or variable-length, blocked, unblocked, or undefined, and may be formatted with or without keys. The records in all the members must have identical formats. Members are stored one after another in the order in which they are written.

The directory contains one record for each existing or projected member of the data set. The directory records are grouped into 256-byte blocks, each containing as many records as will fit into the block. The directory records, which are in alphabetic sequence by member name, vary from 12 to 74 bytes in length, depending on how much user data is included in addition to the member's name and starting address. Each directory block has an eight-byte Key Area containing the name of the last member in the block.

DASD Storage Requirements

Enough DASD storage is required to hold the sequentially organized members and the directory. As new members are added, OS allocates additional area if the original area is full. If the directory is full, however, no new members can be added until the file is reorganized. A deleted directory entry can be reused. Deleted member Data Areas cannot be reused.

Operating System Functions

The basic access method is always used for partitioned organization. The members are created or processed through use of the basic access method for sequentially organized files after the name has been entered into the directory or the starting address has been determined.

8 Indexed Sequential Organization

An indexed sequential file is a sequential file with indexes that permit rapid access to individual records as well as rapid sequential processing. An indexed sequential file has three distinct areas: a prime area, indexes, and an overflow area. Each area is described in detail below.

Prime Area

The prime area is the area in which records are written when the file is first created or subsequently reorganized. Additions to the file may also be written in the prime area. The prime area may span multiple volumes and (in OS and OS/VS) consist of several noncontiguous areas. The records in the prime area are in key sequence.

Prime records must be formatted with keys. They may be blocked or unblocked. If blocked, each logical record contains its key and the key area contains the key of the highest record in the block.

Indexes

There are two or more indexes of different levels. They are created and written by the operating system when the file is created or reorganized.

Track Index (See Figure 8.1)

This is the lowest level of index and is always present. Its entries point to data records. There is one track index for each cylinder in the prime area. It is always written on the first track (s) of the cylinder that it indexes.

Each track index may contain a special first record called a "Cylinder Overflow Control Record" (see "Overflow Area"). The rest of each track index consists of alternating normal and overflow entries. There is a pair of entries for each prime data track in the cylinder. The normal entry contains the home address of the prime track and the key of the highest record on the track. The key of the overflow entry is originally the same as the normal entry. The data area contains 255 to indicate "end of chain." It is changed when records are added to the file (see "Additions Procedure").

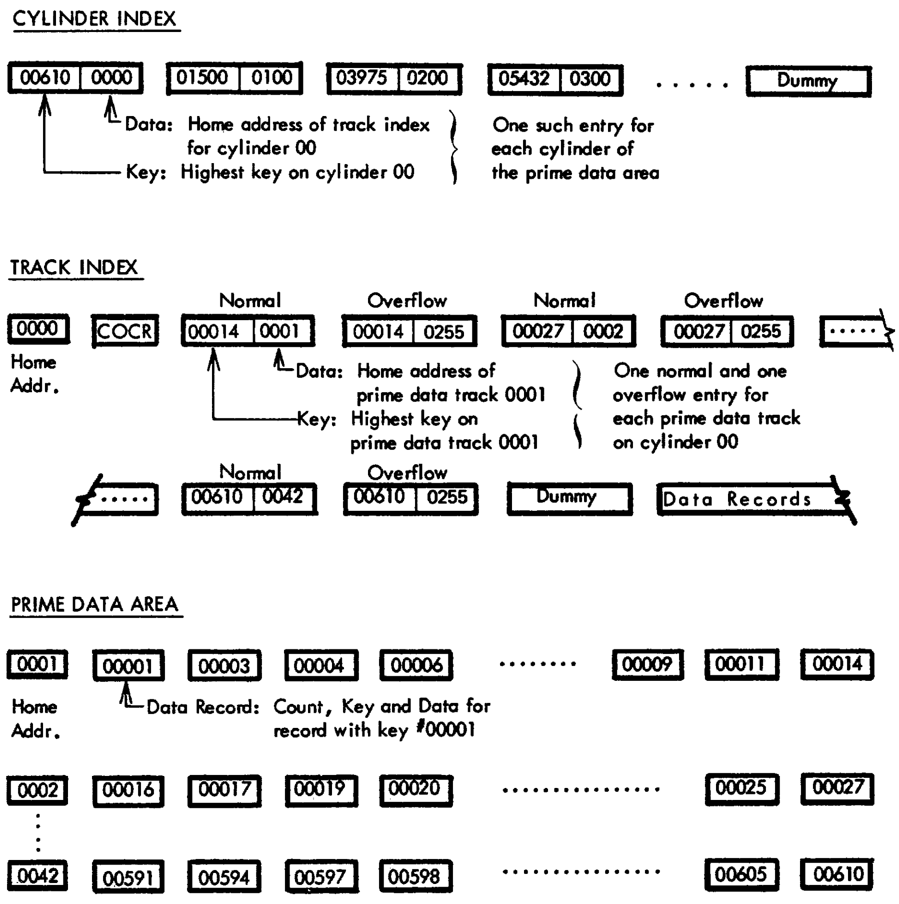


Figure 8.1 *An indexed sequential file with no additions*

The last entry of each track index is a dummy entry indicating the end of the index. The rest of the index track contains prime records if there is room for them. In this case, the first pair of entries in the index refers to this track.

Each index entry (normal,overflow, or dummy) has the same format. It is an unblocked, fixed-length record consisting of a Count Area, a Key Area and a Data Area. The length of the Key Area is as specified by the user. It contains the key of the data record to which the entry points, except for the dummy entry whose key is all 1-bits (highest in collating sequence). The Data Area is always ten bytes long. It contains the full address of the track or record to which the index points and other information such as the level of index and type of entry. The Data Area of the dummy entry is null (all 0-bits). For simplicity, in Figure 8.1 only the cylinder and head portion of the address in the Data Areas is shown.

Cylinder Index (See Figure 7.1)

This is a higher level of index and is always present. Its entries point to track indexes. There is one cylinder index for the file. It may be on a different type of DASD than the rest of the file. In OS and OS/VS it may be placed in an independent index area, an independent overflow area, or in the prime area.

The cylinder index consists of one entry for each cylinder in the prime area, followed by a dummy entry. The entries are formatted in the same fashion as the track index entries. The Key Area contains the key of the highest record in the cylinder to which the entry points. The Data Area contains the Home Address of the track index for that cylinder.

If the prime area is not filled when the file is created, the last cylinder index entries are inactive. These inactive entries have all 1-bits in the Key Area and a null Data Area, just like the dummy entry. The track indexes for prime cylinders that do not yet contain data records also have inactive entries. The inactive entries provide for adding higher records to the end of the file or for expanding the file when it is reorganized.

Master Index

This is the highest level of index and is optional. It is used when the cylinder index is so long that a search through it is too time-consuming. It is suggested that a master index be requested when the cylinder index occupies more than four tracks.

A master index of one level consists of one entry for each track of the cylinder index and is formatted in the same way as the cylinder index. The Data Area of each entry contains the Home Address of the track of the cylinder index to which the entry points. The Key Area contains the highest key in the cylinders indexed by that track of the cylinder index.

OS and OS/VS permits three levels of master indexes and allows them to be written in an independent index area, an independent overflow area, or in the prime area. Each bears the same relationship to the next lower one as the lowest one bears to the cylinder index. That is, if the user specifies that he wants a master index if the cylinder index exceeds four tracks, there will be a second master index if the first one exceeds four tracks and a third master index if the second one exceeds four tracks.

Overflow Area

There are two types of overflow areas: a cylinder overflow area and an independent overflow area. Either one or both may be specified for an indexed sequential file. Records are written in the overflow area(s) as additions are made to the file.

Cylinder Overflow Area (See Figure 8.2)

A certain number of whole tracks, as specified by the user, are reserved in each cylinder for overflows from the prime tracks in that cylinder. When a cylinder overflow area is specified, record 0 (the track descriptor record) of each track index is used as a Cylinder Overflow Control Record (COCR, Figure 8.2). The Operating Systems use the COCR to keep track of the address of the last overflow record in the cylinder and the number of bytes left in the cylinder overflow area. OS and OS/VS also uses this record for additional information needed when the file has variable-length records.

An advantage of having a cylinder overflow area is that additional seeks are not required to locate overflow records. A disadvantage is that there will be unused space if additions are unevenly distributed throughout the file.

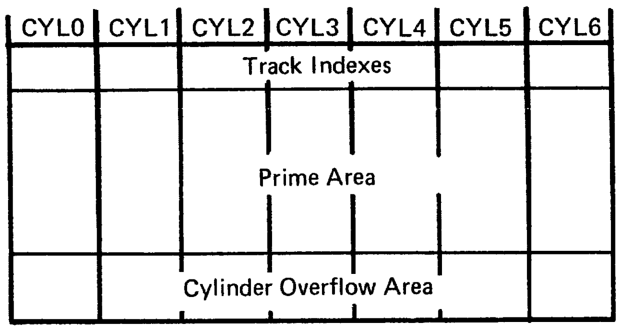


Figure 8.2 Cylinder overflow area

Independent Overflow Area (See Figure 8.3)

Overflows from anywhere in the prime area are placed in a certain number of cylinders reserved solely for overflows. The size and unit location of the independent overflow area are as specified by the user. The area must, however, be on the same type of DASD as the prime area.

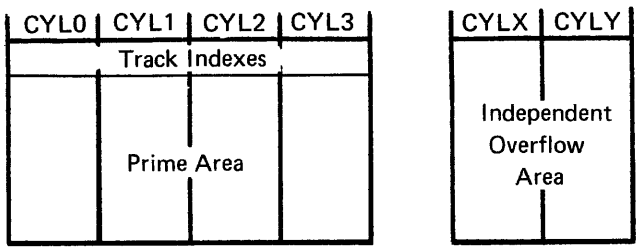


Figure 8.3 Independent overflow area

An advantage of having an independent overflow area is that less space need be reserved for overflows. A disadvantage is that accessing overflow records takes additional seeks.

A suggested approach is to have cylinder overflow areas large enough to contain the average number of overflows caused by additions and an independent overflow area to be used as the cylinder overflow area are filled.

Overflow Records

Overflow records must be unblocked. They must be formatted with keys. They may be fixed-length or, in OS and OS/VS, variable-length. If prime records are blocked, the key of an overflow record is contained in both the Key Area and the Data Area so that all logical records have the same format.

The first field in the Data Area of an overflow record is a link field. It is used to chain together in key sequence the records that have overflowed from a prime track. The link field is ten bytes long and contains the same type of information as the Data Area of index entries. If an overflow record is not the last link in a chain, its link field so indicates and contains the address of the next overflow record in the chain. If an overflow record is the last link in a chain, its link field so indicates and points back to the track index.

The fact that an overflow record has a link field while a prime record does not is of significance to the user only in that the link field requires space on the DASD and in core storage. The operating system presents logical records to the user in such a way that he is not aware of the difference in formats.

Additions Procedure

As records are added to the file, they are no longer physically in key sequence. They are still logically in key sequence, however, through use of the track indexes and link fields. Three different situations may occur when a record is added to the file, and following is a discussion of each situation.

First Addition to a Prime Track (See Figure 8.4)

The new record (key 00010) is written in its proper sequential location on the prime track. The rest of the prime records are moved up one location. The bumped record (00014) is written in the first available location in the overflow area. The record is placed in the cylinder overflow area for that cylinder if it exists and if there is space in it; otherwise, it is placed in the independent overflow area. The Key Area of the normal index entry is changed, since record 00011 is now the highest record on the track. The Data Area of the overflow index entry is changed; it now contains the address of the overflow record. The first addition to a track is always handled in this way. Any record that is higher than the original highest record on the preceding track but lower than the original highest record on this track is written on this track. Record 00015, for example, would be written as the first record on track 0002, and record 00027 would be bumped into the overflow area. Note that no change to higher-level indexes is required. Record 00611 would be written as the first record in the second cylinder. Record 00610 is still and will remain the highest record in the first cylinder.

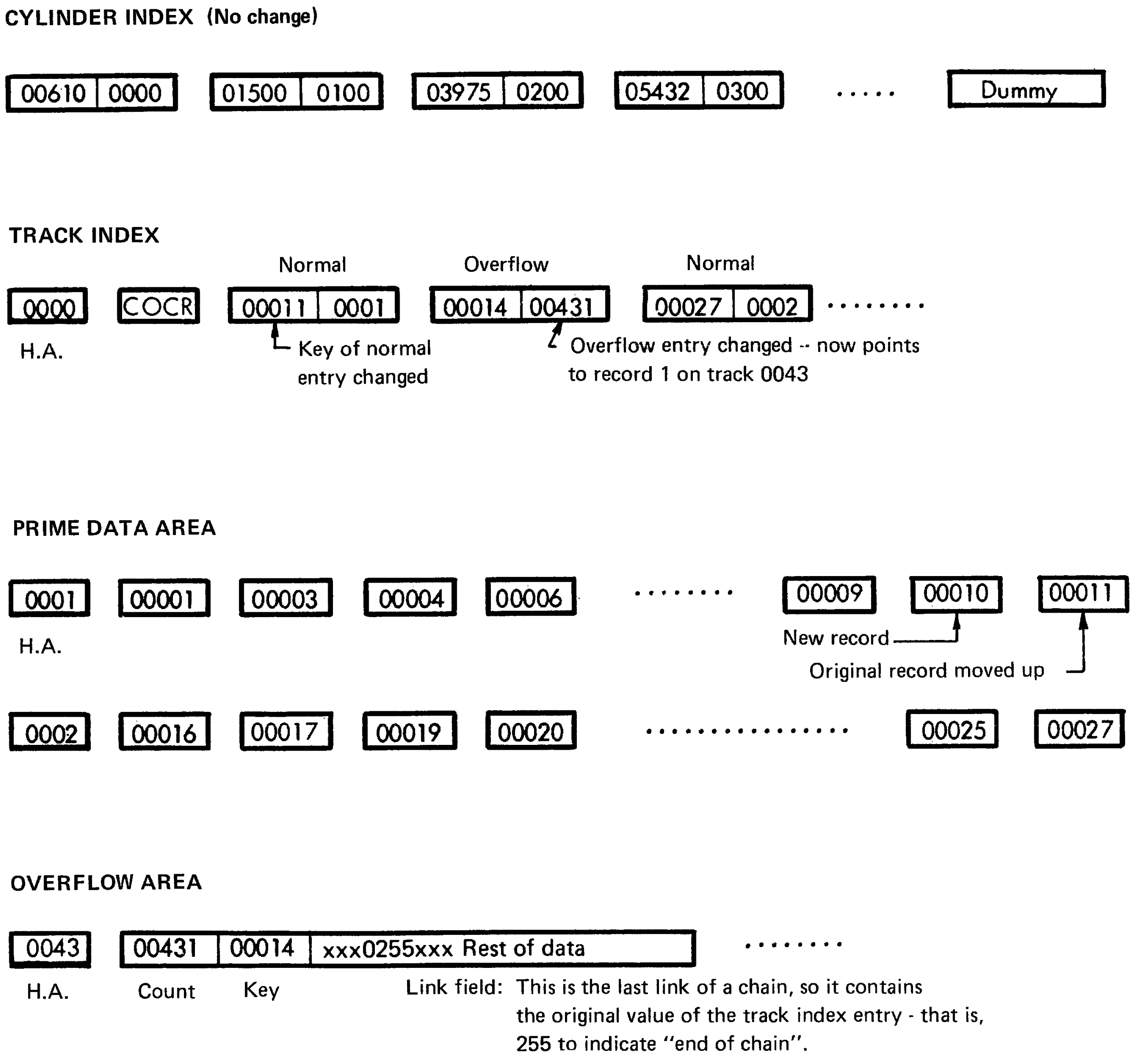


Figure 8.4 *An indexed sequential file after the first addition to a prime track*

Subsequent Additions to a Track (See Figure 8.5)

Subsequent additions are written either on the prime track where they belong or as part of the overflow chain from that track. If the addition belongs between the last prime record on a track and a previous overflow from that track (as is the case with record 00013), it is written in the first available location in the overflow area, with its link field containing the address of the next record in the chain. The link field of a previous overflow may need to be

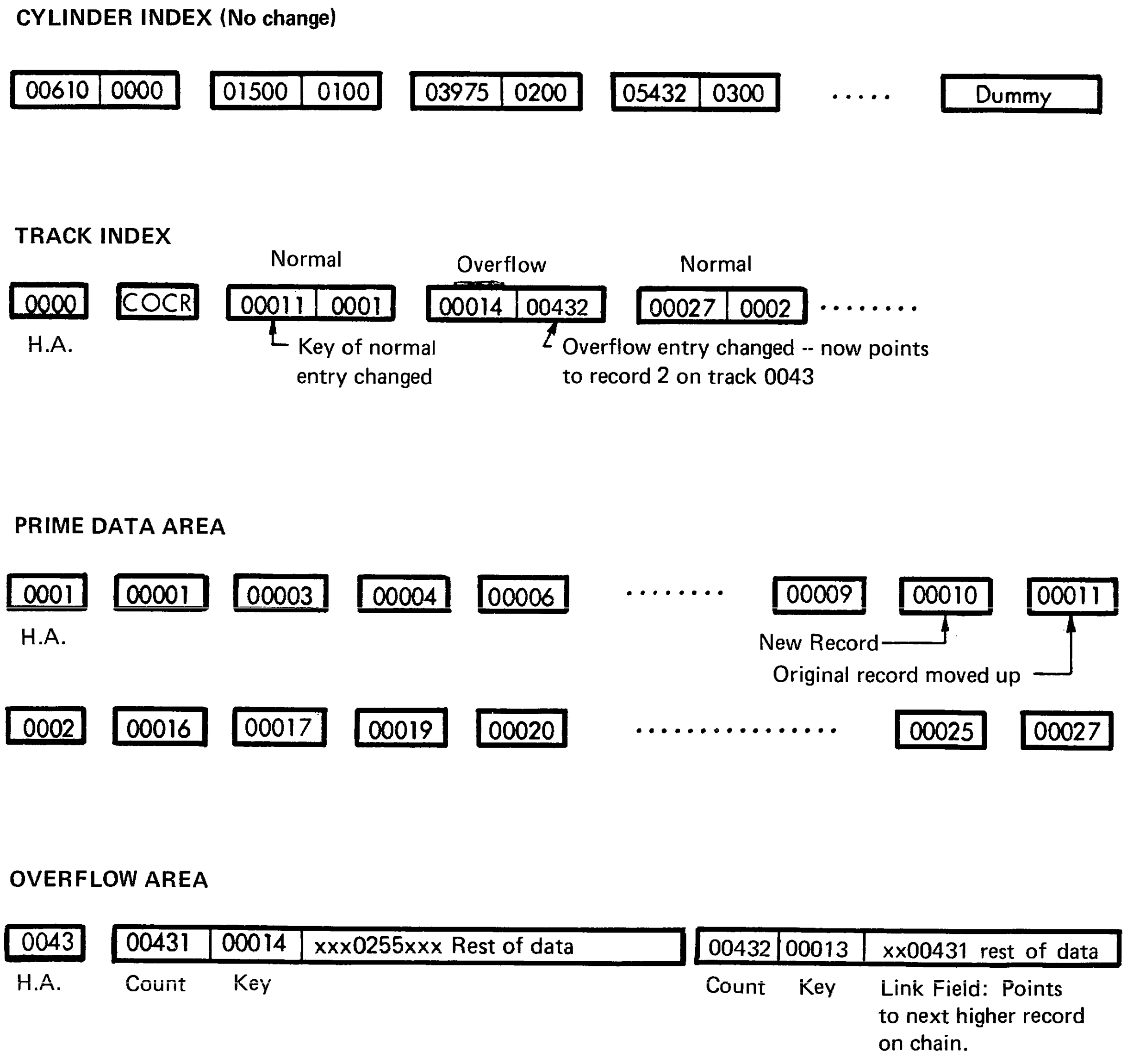


Figure 8.5 *An indexed sequential file after subsequent additions to a track*

changed; it is not necessary in this example. Because the Data Area of the overflow index entry always refers to the address of the lowest key in a chain, it is changed if necessary (as in this example).

If the addition belongs on a prime track (as would be the case with record 00005), it is written in its proper sequential location on the prime track. The bumped record (00011) is written in the first available location in the overflow area. The Key Area of the normal index entry is changed (to 00010). The link field of a previous overflow and the Data Area of the overflow index entry are changed if necessary.

Note the logical similarity between the normal and overflow index entries. The normal entry indicates that a sequence of records starts at the beginning of track 0001, the last record having a key of 00011. The overflow entry indicates that a sequence of records (chained together by the link fields), starts with the second record on track 0043, the last record having a key of 00014.

Although the cylinder overflow area may eventually contain overflows from all prime tracks in the cylinder, and the independent overflow area may eventually contain overflows from anywhere in the file, each prime track has its own chain.

Addition of High Keys

A record with a key higher than the current highest key in the file is placed on the last prime track containing data records if that track is not full. If that track is full, the record is placed in the overflow area. The sequence link for these records is chained to the last prime track containing data records. The Key Area of higher level indexes is changed to reflect the addition.

Variable-Length Records

One approach to variable-length records is to use trailer records. A trailer record is an extension of a master record. It is separate from the master and written as required. Using an open-item accounts receivable file as an example, the master records contain information common to all accounts, and the number of invoices sufficient for most of the accounts, while the trailer record contains more invoices. A master may have as many trailer records associated with it as are required.

The trailer records may be written immediately after the associated master record. Since duplicate keys are not allowed, it is necessary to add a digit or character to the true key. Thus 123A would be the master record for account number 123; 123B would be the first trailer, 123C the second trailer, and so forth.

The trailer records may be written as a separate file. This approach would be advantageous if many jobs referenced only the master records. Reference between a master record and its trailer record can be effected by having a link field in each record. The master record would contain the address of the first trailer record, the first trailer record would contain the address of the second, and so forth. The trailer file would probably be written and processed using the basic access method for directly organized files. The logic of handling the trailer records as a separate file is more complex and requires more programming by the user than the first approach described above.

Operating System Functions

Queued Access Method

The queued access method for indexed sequential files is used when reading or updating the records in key sequence. The entire file may be processed, or processing may begin at a specified key or record number. The operating system takes care of all searching of the indexes and link fields and any required deblocking and presents the next sequential logical record to the user. OS and OS/VS provides anticipatory buffering and overlap of input/output with processing.

Basic Access Method

The basic access method for indexed sequential files is used when adding records to the file. The operating system writes the new record, rewrites existing records as required, rewrites index entries and link fields as required, and takes care of blocking if required.

This access method is also used when reading or updating records directly. The user supplies the key of the desired record. The operating system takes care of all searching of the indexes and link fields, along with any required deblocking, and either presents the specified logical record to the user or indicates that it could not be found.

9 Direct Organization

This chapter discusses some commonly used methods of direct organization, as well as the access methods provided for files so organized. The user is not restricted to the methods of organization discussed here; they are presented as suggestions only.

General Characteristics

With direct organization, there is a definite relationship between the key of a record and its address. This relationship permits rapid access to any record if the file is carefully organized. The records will probably be distributed nonsequentially throughout the file. If so, processing the records in key sequence requires a preliminary sort or the use of a finder file.

Addressing

With direct organization, the user generally develops a record address that ranges from zero to some maximum. Track addresses on most DASD's, however, are noncontiguous. For example, the address of the last track on the first cylinder of a 2314 is 0019, while the first track on the next cylinder is 0100. Furthermore, the file may start at other than the first track of a device and it may occupy several nonadjacent areas.

Most operating systems allow the user to refer to a record in several ways:

(1) Relative Track Address – here the user presents to the system a 3 byte binary number in the form TTR where:

TT	is the position of the track relative to the first track on which the data set resides. The first track of the data set always has a relative position of 0.
R	is the number of the block relative to the first block of data on the track specified TT. The first block of data on a track has a relative value of 1.

(2) Relative Track and Key – here the user specifies a 3 byte binary number which the system converts to a relative track address. The track is then searched for the record which has the key specified by the user.

(3) Relative Block Address – here the user presents the system with a 3 byte binary number that indicates the position of a block in relation to the first block of a data set. The first block of a data set always has a relative block address of 0.

(4) Actual Address – here the user presents a pattern of characters that, without further modification, identifies a unique direct storage location. The format is an 8 byte address (MBBCCHHR).

Directly Addressed File

With direct addressing, every possible key in the file converts to a unique address. This makes it possible to locate any record in the file with one seek and one read.

Using the Key as the Address

In order to be able to use the key of a record directly as its address, the records must be fixed-length and the keys must be numeric. One computation is required. Divide the key by the number of records per track; the quotient equals the relative track address, and the remainder plus one (record 0 is used as a capacity record) equals the record number.

This method of direct addressing not only allows minimum disk time when processing directly, but is also ideal for sequential processing since the records are written in key sequence. A possible disadvantage is that there may be a large amount of unused direct access storage. A location must be reserved for every key in the file's range even though many of them are not used.

Using a Cross-Reference List

With this method, each record in the file is assigned an address and a cross-reference list of keys and assigned addresses is maintained. The list may be a printed one. Some clerical and keypunch time is required for each transaction, since the address must be looked up and included in the input to the job. Controls must be tight, since the list, as well as the file, must be kept up to date. The list may itself be a file recorded on a DASD. Although any record can be located directly when its address is known, time is required to look up the address in the list. Indexed sequential organization is a variation of this method.

Indirectly Addressed File

Indirect addressing is generally used when the range of keys for a file includes such a high percentage of unused ones that direct addressing is not feasible. For example, employee numbers range from 0001 to 9999 but only 3000 of the possible 9999 numbers are assigned. Indirect addressing is also used for nonnumeric keys.

With indirect addressing, the range of keys for a file is compressed to the smaller desired range of addresses by some sort of address conversion. Address conversion techniques inevitably cause "synonyms" – two or more records whose keys convert to the same address. Two objectives must be considered in selecting a technique: (1) every possible key in the file must convert to an address in the allotted range, and (2) the addresses should be distributed evenly across the range so that there are few synonyms.

A record that is written at the address to which its key converts is called a "prime record." Any other records whose keys convert to this address are "synonyms." What to do about synonym records is discussed later in this chapter, but the point to be made now is that synonyms should be kept to a minimum because of the additional time required to locate these records.

A way to minimize synonyms is to allot more space for the file than is actually required to hold all the records. The term "packing factor" means the percentage of allotted locations that are actually used. For an indirectly addressed file, an initial packing factor of 80-85% is suggested. For example, a 10,000-record file packed 83% would be allotted space for 12,000 records.

Address Conversion

There are many address conversion techniques. Selecting a good one for a particular file may require some trial and error. A suggested goal is no more than 20% synonyms. If converting to track address, count only the synonyms in excess of the number of records per track.

Division/Remainder Method

It is suggested that this technique be tried first, because it is a simple one that often gives good results. The key is divided by a prime number (a number evenly divisible only by itself and by one) that is close to the number of addresses allotted to the file. The remainder is used as the address.

Example 1: Load 8000 200-byte records on a 2314, converting to track address.

a. With 80% packing, 10,000 locations are required.

b. Can load 20 records per track, so 500 tracks are required.

c. A prime number close to 500 is 499.

d. Divide the key by 499.

e. The remainder (000 to 498) equals the relative track address.

Example 2: Same as above, but converting to record address.

a. A prime number close to 10,000 is 9973.

b. Divide the key by 9973.

c. Divide the remainder by the number of records per track (20).

d. The quotient equals the relative track address; the remainder plus one equals the record number.

This method can also be used with nonnumeric keys. Using binary arithmetic will probably give better results than using decimal arithmetic, since the uniqueness of the letters and special characters in the key is retained.

The division/remainder method automatically achieves the first objective mentioned earlier – that is, to have all keys convert to addresses within the allotted range. Whether it achieves the second objective for a particular file – that is, to have few synonyms – can be determined only by trying it.

Digit Analysis

Since the primary objective of an address conversion technique is to develop addresses spread evenly across a range, it may be possible to make use of any existing evenness in the distribution of the keys.

Figure 9.1 shows the output of a digit analysis program that counted the number of times each digit appeared in each position of the keys of a particular file.

If allotting 20,000 locations for the 16,045 records, the keys must convert to addresses that range from 00000 to 19999. Since positions 7, 8, 9 and 10 of the key are evenly distributed, they may be used as the four low-order digits of the address. Of the four positions chosen as the four low-order digits of the address, use one position as the basis of forming the high-order digit of the address; if that position is odd, use 1 as the high-order digit of the address; if even, use zero.

If converting to track address, divide the record address developed above by the number of tracks required for the 20,000 records. The remainder equals the relative track address.

TOTAL NUMBER OF RECORDS 16,045

DIGIT	KEY POSITION										
	1	2	3	4	5	6	7	8	9	10	11
0	16045			1852	5168	1807	1738	1574	1597	1579	87
1			4408	3147	5638	2120	1748	1652	1651	1599	235
2		2198	3792	1174	4958	1745	1743	1587	1569	1604	334
3		576	2231	2724	281	1684	1610	1620	1576	1603	9371
4		1195	2459	1194		1378	1617	1647	1652	1619	3164
5		12076	3155	1267		1647	1688	1580	1605	1645	1939
6				1243		1560	1606	1538	1611	1625	565
7				1228		1329	1450	1560	1598	1557	253
8				1227		1415	1411	1630	1618	1622	76
9				989		1360	1434	1657	1568	1592	21

Figure 9.1 *Digit analysis table*

Folding

The key is split into two or more parts, which are added together. The sum, or part of it, is used as a relative address.

Examples of folding a key of 7 4 6 2 9 8:

	746 +	298 =	1044	(split in half)
74 +	62 +	98 =	234	(split in thirds)
	769 +	428 =	1197	(alternate digits)

Radix Transformation

The key is transformed to a different radix or base. Excess digits are discarded, leaving an address of the required length.

Example of converting a key of 4 2 3 5 6 to radix 11 to produce a four-digit address:

$(4 \times 11^4) + (2 \times 11^3) + (3 \times 11^2) + (5 \times 11^1) + (6 \times 11^0) =$
$58564 + 2662 + 363 + 55 + 6 = 61650$
Use 1650 as the relative address.

Evaluation of Results

The selected address conversion technique should be applied to the entire file of keys and carefully evaluated before deciding to use it. When evaluating a conversion technique, it is not sufficient to calculate the percentage of synonyms. The expected average number of reads (revolutions) per record should also be developed. For example, if ten keys convert to addresses 1, 2, 3, 4, 5, 6, 7, 8, 1, 2, then 20% are synonyms. Assuming that one read (revolution) is required for each of the first eight, and two reads for each of the last two, the average number of reads per record is 1.2. If the keys convert to addresses 1, 2, 3, 4, 5, 6, 7, 8, 1, 1, however, 20% are synonyms, but if one read is required for each of the first eight, two for the ninth, and three for the tenth, the average number of reads per record is 1.3.

The evaluation, then, should be based on the average number of reads (revolutions) per record. An average of 1.2 is considered to be good. The final question may be: "Does this rather rapid access time justify the additional preinstallation planning and programming required for a directly organized file?" Another question may be whether it justifies the effort involved in the development of a new conversion technique and the subsequent reprogramming that might be necessary if the directly organized file were later relocated on another type of DASD. A directly organized file is relatively device-dependent, since it implies a specific relationship between the key of a record and its address on a DASD.

Description of a Directly Organized File

With direct organization, records may be fixed-length, variable-length, or undefined.

Records may be formatted with or without keys. If the file is indirectly addressed and conversion is to track address, the records should be formatted with keys for efficiency. If not, each record on the track must be read to determine whether it is the desired one.

The records may be blocked or unblocked. If they are blocked, the user is responsible for all blocking and deblocking, because in the access method for directly organized files the operating system handles physical records rather than logical records. If the file is indirectly addressed, the records are probably unblocked. The problems that may occur with blocked records are discussed later in this chapter.

With most directly organized files, R0 of each track is used as a capacity record. It contains the address of the last record written on the track and is used by the operating system to determine whether a new record will fit on the track. The capacity records are updated by the operating system as records are added to the file. They do not account for deletions. Once a track is full, it remains full as far as the operating system is concerned (until the file is reorganized), even though the user deletes records.

An indirectly addressed file generally consists of just one logical area, which may actually be several nonadjacent physical areas. The location of synonym records is up to the user, but they are generally put in unused locations in the main (and only) area. Synonym records can be put in a separate area if the user desires. The disadvantage of doing this is that each synonym record will require an additional seek. If there is just one area, and if a good conversion technique has been selected and the file is not packed too tight, synonym records are likely to be in the same cylinder as the prime record, thus eliminating the need for an additional seek.

File Creation and Maintenance

With indirect addressing, the logic of creating and maintaining the file depends mainly on the synonym records. The area in which to place the synonym records has already been discussed. Now the problem of how to locate them quickly must be considered.

Approaches to the handling of synonym records are discussed in the following sections: chaining and progressive overflow, two approaches used by the DOS programmers, and the extended search option of the BSAM and BDAM access method creation and retrieval routines for OS/VS. They are discussed in order to point out how the maintenance of a file depends on the way in which it was created and the interaction between operating system functions and the user's programming. They are presented as suggestions only; other more complex and possibly more efficient approaches are possible. All three approaches assume that:

- The records are unblocked.
- The records are formatted with keys.
- Address Conversion is to track address.
- Synonym records are placed in unused locations.

Chaining Method

One record on each track is used, and maintained by the DOS programmer, as a chaining record to provide a link between the prime track and the track on which the synonym record was placed. Synonym records are written on the next higher available track.

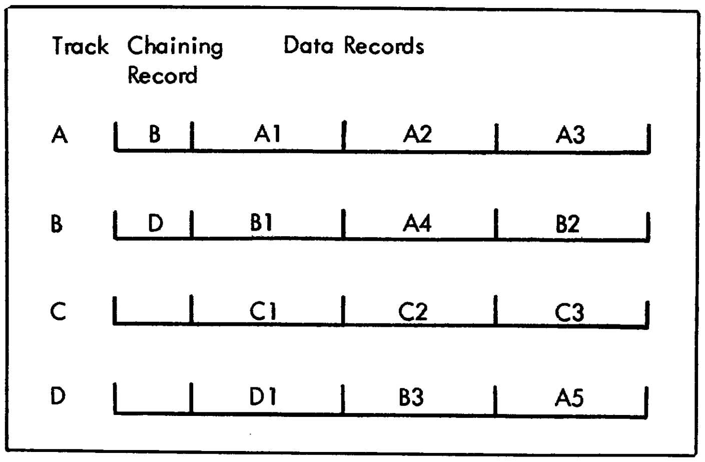


Figure 9.2 Chaining

Figure 9.2 shows a chained file. The sequence in which the records were loaded was A1, B1, A2, D1, C1, A3, C2, A4(synonym), B2, C3, B3 (synonym), A5 (synonym). The following questions and answers explain how records in a chained file are located:

Q. If looking for a "A" record, where does the search begin?

A. On track A. Searches always begin with the prime track.

Q. If an "A" record is not found on track A, what is the next track searched?

A. Track B. Searches always continue at the track specified in the chaining record.

Q. If the "A" record is not found on track B, what is the next track searched?

A. Track D.

Q. If a "C" record is not found on track C, what is the next track searched?

A. None. The blank chaining record shows that there are no more "C" records.

Creation of the File

The way in which the records are loaded may have a significant effect on the average number of reads that it will take to locate them.

The file may be completely loaded in one pass. The results of this one-pass load and the number of reads required to subsequently locate each record are shown in Figure 9.3 This example and those following show one record per track for simplicity. The logical results will be the same with multiple records per track. With the one-pass load, the record is written on its prime track, if there is room. If the track is full, the record is written on the next available track, and the chaining record of the home track is updated. Assume that prime records require one read, first synonym requires two reads (prime track and placement track), second overflows require three reads (home track, first placement track, second placement track), etc. Notice that record C should have been a prime record, but a synonym from track 1 took its place first.

Key	Home Track	Where Loaded	Chaining Address	Number of Reads
A	1	1	2	1
B	1	2	3	2
C	2	3	4	2
D	7	7	9	1
E	5	5	-	1
F	6	6	-	1
G	8	8	-	1
H	7	9	10	2
I	2	4	-	3
J	7	10	-	3

Average reads per record = 1.7

Figure 9.3 One-pass load

The file may be loaded in two passes. The results of this are shown in Figure 9.4 On the first pass, only prime records are loaded. On the second pass, the synonym records are loaded and the chaining records updated. Because all prime records are written on their prime track, less chaining is required, and the average number of reads per record has decreased.

Key	Home Track	Where Loaded		Chaining Address	Number of Reads
		Pass 1	Pass 2		
A	1	1		3	1
B	1	-	3	-	2
C	2	2		4	1
D	7	7		9	1
E	5	5		-	1
F	6	6		-	1
G	8	8		-	1
H	7	-	9	10	2
I	2	-	4	-	2
J	7	-	10	-	3

Average reads per record = 1.5

Figure 9.4 Two-pass load

Additions to the File

The logic of making additions to a chained file is a combination of pass 1 and pass 2 of the load routines. The same problem that was illustrated with a one-pass load will eventually occur: there should be room for the new record on its prime track, but it is already filled with synonyms from other tracks. There is no really effective, simple solution to this problem. Placing the new record where it belongs involves a dump and reload of all affected records, which can be very complicated and time- consuming. For example, try to add record D2 to the sample file shown in Figure 9.2 (assume that this is only part of the file and that a location is available somewhere). The complexity is due to the fact that when converting to track address, a track may contain snyonyms from more than one prime track. A suggested solution is to ignore the problem for the time being and write the record on the next higher track on which there is available space. The situation will be corrected when the file is reorganized.

Deletions from the File

Records to be deleted may be tagged in some way and omitted when the file is reorganized. If the operating system is responsible for finding locations for new records, there is no point in literally deleting records since the capacity record is not updated to reflect this.

Reorganization of the File

Particularly with a volatile file, a change in the distribution of the keys may adversely affect the results of the converting technique and the speed with which the file can be referenced. Directly organized files may therefore require frequent reorganizations. The operating system maintains no statistics as it does with indexed sequential organization. Therefore the user should, at least periodically, calculate the average number of reads per record to ensure that the existing organization continues to provide the desired degree of efficiency. Reorganization will be needed less frequently if the user develops more complicated addition and deletion routines than those that have been discussed.

As with indexed sequential, there are two ways to handle reorganization. The file can be written elsewhere and then, on a separate run, re-created in the original area, or it can be reorganized directly into a different area of direct access storage.

Progressive Overflow Method

As with chaining, synonym records are written on the next higher available track. The difference is that there is no chain from the prime track to the next available track. The links in the chain are simply consecutive tracks.

Creation of the File

With progressive overflow, a one-pass load produces the same results as a two-pass load. Some of the records may be written in different locations, but the average number of reads per record is the same. Figure 9.5 shows the results of a one-pass load of the same file used to illustrate the loading of a chained file. Note that the average number of reads (revolutions) per record is higher than those shown in Figure 9.3 and 9.4 because all tracks between the prime track and the one where a synonym record is located must be searched. Note that a search without a read takes place for all tracks except the one on which the desired record is located.

Key	Home Track	Where Loaded	Number of Reads
A	1	1	1
B	1	2	2
C	2	3	2
D	7	7	1
E	5	5	1
F	6	6	1
G	8	8	1
H	7	9	3
I	2	4	3
J	7	10	4

Average reads per record = 1.9

Figure 9.5 *Progressive overflow*

Additions to the File

The logic is the same as the load routine.

Deletions from the File

The comments made for the chaining method apply here also.

Reorganization of the File

The comments made for the chaining method apply here also.

Progressive Overflow Compared to Chaining

The chaining method of handling synonyms requires somewhat more complicated load and addition programs but in some circumstances it may result in a shorter search for snyonym records. If a fairly low packing factor is used, however, synonyms will usually be located on the track that follows the prime track, and progressive overflow with a track-by-track search will result in timing equivalent to that provided by the chaining method described. Progressive overflow with extended search provides the fastest timing. Only when the packing factor approaches 100% will the time required for progressive overflow increase significantly.

Extended Search

To utilize the extended search option, the data set must be formatted with:

- Unblocked records.
- Records formatted with fixed length keys.
- Unused record spaces in the data set contain standard dummy records. That is records having the first byte of the key

set to a hex value of 'FF' and the first byte of the data portion containing the block reference count.

- Conversion of keys can be to relative track or relative block.
- Synonym records will be placed in the next higher available space. That is a space containing a dummy record.

You may request that the system begin its search with a specified starting location and continue for a certain number of records or tracks. This option can be used to request a search for unused space in which a record can be added.

To use the extended search option, you must indicate to the access method the number of track (including the starting track) or number of records (including the starting record) that are to be searched. If you indicate a number of records, the system may actually examine more than this number. In searching a track, the system searches the whole track (starting with the first record); it therefore may examine records that preceed the starting record or follow the ending record.

If you specify a number equal to or greater than the number of tracks allocated to the data set or the number of records within the data set, the entire data set is searched in an attempt to satisfy your request.

Additions to the File

Because the entire file must be formatted with standard dummy records (records having the first byte of the key set to a value of hex 'FF') in unused record spaces, additions are in reality updates to dummy records (replacement of the dummy record with the actual record and its key). Since dummy records contain a value of hex 'FF' in the first byte of the key, you cannot use keys with a hex 'FF' value in the first byte and utilize the extended search option.

Deletion to the File

The comments made for the chaining method apply here also.

Reorganization of the File

The comments made for the chaining method apply here also.

Activity Loading

With an indirectly addressed file, the sequence in which the records are loaded may have a significant effect on the time to locate records, regardless of how overflows are handled. The average number of reads per record depends on the frequency with which each record is processed as well as on the number of reads required to locate it. Figure 9.6 shows what a drastic difference the method of loading makes when 20% of the records (I and J) account for 80% of the activity. The example uses progressive overflow.

If uneven distribution of activity is a characteristic of the file, the most active records should be loaded first, so that they have the greatest probability of being home records. If activity statistics are not available before installation of the system, they can be accumulated once the system is installed. Activity statistics should continue to be accumulated, since the distribution of activity may change seasonally or over another time period. The file can then be sorted into the current activity sequence as part of each reorganization.

Loading in key sequence:

Key	Home Track	Where Loaded	Number of Reads	Frequency of Reference	Reads Times Frequency
A	1	1	1	2.5%	.025
B	1	2	2	2.5%	.050
C	2	3	2	2.5%	.050
D	7	7	1	2.5%	.025
E	5	5	1	2.5%	.025
F	6	6	1	2.5%	.025
G	8	8	1	2.5%	.025
H	7	9	3	2.5%	.075
I	2	4	3	40%	1.200
J	7	10	4	40%	1.600

Average reads per record = 3.1

Loading in activity sequence:

Key	Home Track	Where Loaded	Number of Reads	Frequency of Reference	Reads Times Frequency
I	2	2	1	40%	.400
J	7	7	1	40%	.400
A	1	1	1	2.5%	.025
B	1	3	3	2.5%	.075
C	2	4	3	2.5%	.075
D	7	8	2	2.5%	.050
E	5	5	1	2.5%	.025
F	6	6	1	2.5%	.025
G	8	9	2	2.5%	.050
H	7	10	4	2.5%	.100

Average reads per record = 1.225

Figure 9.6 Effect of loading sequence on timing

Blocked Records

Although blocking records is advantageous as far as direct access storage utilization is concerned, it may have an adverse effect on timing when direct organization is used. Moreover, as already noted, the user is responsible for all blocking and deblocking routines.

Directly Addressed File

Blocking presents no problems if direct addressing is used. It simply requires a different computation of the address of a record:

1. Divide the key by the number of logical records per track. The quotient equals the relative track address.

2. Divide the remainder from step 1 by the number of records per block. The quotient plus one equals the identifier (record number of the block). The remainder equals the position of the logical record within the block which can be used in the blocking and deblocking routines.

A point to remember is that when adding a record to the file, an entire block must be written.

Indirectly Addressed File

The problem here is that there is no logical key to a block of indirectly addressed records. Therefore, the points already discussed will have to be modified as follows.

Records are formatted without keys. Address conversion is to record address (actually block address) as shown in the second example under "Division/Remainder Method" in this chapter. The prime number to be used is one close to the number of blocks allotted for the file. In step c divide by the number of blocks per track.

When loading the file or making additions to it, the user has to read and search an entire block. If it is full, the next sequential block (progressive overflow) or the next block in the chain (chaining) is read and the search continued until a location is found. Note that if the chaining method is used, the linkage is between blocks, not between tracks.

If the primary reason for using direct organization is to minimize the time required to locate records, the effect of blocking on timing should be carefully evaluated.

Operating System Functions

For direct organization only a basic access method is provided. The operating system does not provide automatic buffering and overlap of input/output with scheduling. Macros are provided, however, so that the user can program these functions if he knows in advance which record he will want next. This access method is used for writing new records and for reading and updating existing records as already discussed.

Introduction to Virtual Storage Access Method (VSAM) 10

What Are the Requirements for an Access Method?

In data processing today, it is common for a computer installation to do a number of different types of processing. An installation must provide for one combination or another of data-base processing, online processing, batch processing, inquiry and transaction processing, communications, and multiple CPUs under the control of different operating systems. This variety requires an access method that provides:

- High performance of retrieval and storage – independent of previous insertions of records into data sets and uninterrupted by requirements to reorganize data sets or copy them for backup.

- Applicability to different types of processing that require different kinds of access and different levels of performance (such as online and batch processing).

- Simplicity of use by means of a common set of instructions for different types of access, simplified JCL (job control language), and optimization of the use of space in auxiliary storage.

- Protection of data: security against unauthorized access and integrity through prevention of intentional or accidental loss of data.

- Recovery of data: the ability to recover catalogs and data sets in the event of failure or damage.

- Central control over the creation, access, and deletion of data sets and over the management of space by keeping data-set and storage information in one place and making it independent of JCL and processing programs.

- Ability to move data from one operating system to another in a format that is common to both systems.

- Independence from type of storage device: freedom from physical record size, control information, and record deblocking.

- Ease of conversion of data and programs from other access methods to the new access method.

What is VSAM

VSAM is a set of programs (an access method) for use with OS/VS1 and OS/VS. VSAM is used with direct-access storage devices to provide fast storage and retrieval. Figure 10.1 shows how VSAM relates a processing program and stored data.

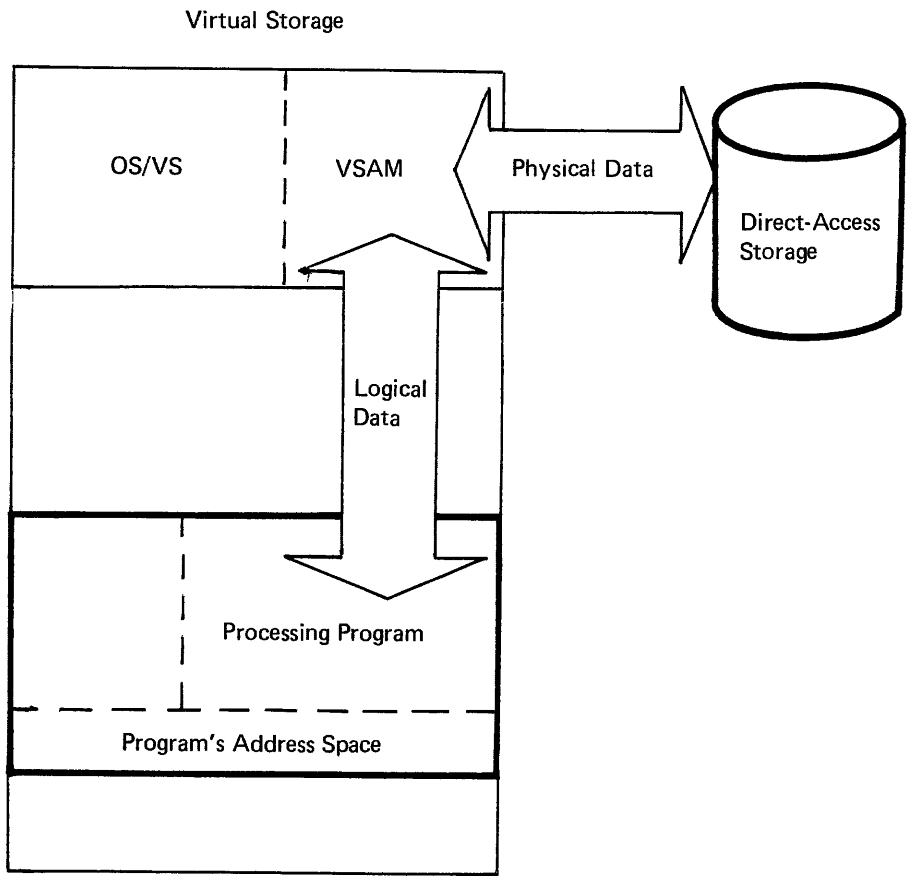


Figure 10.1 *VSAM's relative position. VSAM relays data between the processing program and direct-access storage.*

How Does VSAM Meet These Requirements?

VSAM provides an approach to meeting these requirements through:

- A format for storing data independently of type of storage device.
- Routines for sequential or direct access and for access by key, by relative address, or by relative record number.
- Options for optimizing performance.
- A comprehensive catalog for defining data sets and auxiliary-storage space.
- A multifunction service program (Access Method Services) for setting up catalog records and maintaining data sets.

High Performance

VSAM's high performance is due to an efficiently organized index, performance options for reducing disk-arm movement and rotational delay, and distributed free space for fast insertion of records and minimal reorganization. The size of the index is reduced by compressing keys to eliminate redundant information. The type of index used for a data set is also used for VSAM catalogs.

VSAM's method of inserting records into a data set provides access whose speed following a large number of insertions is equivalent to the speed of access without previous insertions. Free space is used for efficient automatic reorganization of data sets: inserted records are stored and addressed in the same way as original records, and space given up by deletions is reclaimed as free space within the control interval.

Applicability to Different Types of Processing

VSAM is designed to meet most of the common data-organization needs of both batch and online processing. Batch processing requires the efficiency of sequential and indexed data; online processing requires efficient direct access for random requests. VSAM permits both direct and sequential access. Access can be by key, by relative address, or by relative record number. Different types of processing can be intermixed in the processing of a common data base. You can select the type of access or the combination of types that best suits your application.

TSO (Time Sharing Option), a subsystem of OS/VS2, can execute Access Method Services commands as TSO commands, dynamically allocate a VSAM data set, and execute a program that uses VSAM macros to process the data set.

VSAM also provides options and macros for sharing a pool of I/O-related control blocks, channel programs, and buffers among several VSAM data sets open at the same time.

Simplicity of Use

There is a common way of requesting the different types of access (sequential or direct, by key, by relative address, or relative record number), so that the same instructions are learned and used for achieving different results.

All VSAM data sets are cataloged, so JCL is simplified. Minimal JCL parameters are required for describing data sets.

VSAM uses default values to establish the size of control intervals and control areas in which data is stored and to manage virtual storage space for I/O (input/output) buffers. Programmers can think in terms of the application, not in terms of the internal workings of VSAM.

Individual data records are passed to a processing program, application data alone is processed by the program. Application programmers do not need to know the format of control blocks. They need not be concerned either with storage devices and device addresses or with different formats for fixed-length or variable-length records.

Protection of Data

VSAM protects data by means of its design and its integrity and security options. *Integrity* means the safety of data from inadvertent destruction or alteration; *security* means the protection of data from unauthorized use or purposeful destruction or alteration. VSAM writes records in a way that does not expose data to loss, even if an I/O error occurs. You can specify optional passwords for levels of protection (read-only, update, control interval, and full access) and include your own routine to check a requester's authority to gain access to data. You can select options for formatting data sets before data is stored in them and for verifying write operations for data integrity. VSAM also provides various levels of exclusive control for data to be shared between subtasks, regions, and operating systems.

Recovery of Data

VSAM catalogs that are defined with the optional recovery attribute allow data to be recovered. Recovery is based on information recorded on the volumes controlled by the catalog as well as in the catalog itself.

Central Control

The VSAM catalog brings together extensive information about data sets and storage space. Access Method Services controls the definition and deletion of data sets and the alteration of information about them in the catalog. Its use is authorized by passwords assigned to the data sets or to the catalog itself. Consequently, the management of your inventory of data sets is centralized and made independent of the use of JCL or the actions of processing programs. Space for data sets can be allocated or deallocated without mounting volumes, because the information describing the contents of VSAM spaces on those volumes is contained in the catalog. You can assign a data set to volumes by ranges of keys that are controlled by the catalog.

Portability of Data Between Systems

VSAM's technique for storing records uses a format that is common to OS/VS and DOS/VS (disk operating system/virtual storage). Communication with VSAM is very similar for both operating systems, except for JCL. Access Method Services includes functions that facilitate moving data sets and volumes from one operating system to another.

Device Independence

VSAM is independent of particular types of storage devices, because it addresses a record in a data set without respect to the physical attributes of auxiliary storage, but with respect to the displacement of the record from the beginning of the data set. The unit in which data is transmitted between virtual and auxiliary storage does not depend on the size of the physical records in which data is stored physically on a volume.

Ease of Conversion

VSAM provides for easy conversion of indexed sequential data sets to VSAM format and the continued use of your existing ISAM (indexed sequential access method) programs to process converted data sets and new VSAM data sets. Access Method Services converts a sequential or an indexed sequential data set to VSAM format. To process the converted data set with the ISAM program, a set of interface routines within VSAM interpret each ISAM request and issue the appropriate VSAM request.

In VS2 systems, the OS catalog has been replaced by a VS2 master catalog. Access Method Services is used to convert entries in an OS catalog in entries in an existing VS2 master catalog or a VSAM user catalog.

What Machines Can VSAM Be Used With?

You can use VSAM on IBM System/370 CPU Models 135 (OS/VS1 only), 145, 155 (Model 2), 158, 165 (Model 2), and 168. Each of these CPUs must have the dynamic address translator that is required by OS/VS1 and OS/VS2 and either the advanced control program support feature or the conditional swapping feature. VSAM is designed to take full advantage of the benefits of virtual storage.

The IBM direct-access storage devices that you can use are the IBM 2305 (Models 1 and 2) Fixed Head Storage, the 2314 Direct Access Storage Facility, the 2319 Disk Storage, the 3330 Disk Storage, the 3330 (Model 11) Disk Storage, the 3340 Disk Storage, and the 3850 Mass Storage System.

Getting to Know What VSAM Is and Does

What Are VSAM's Three Types of Data Sets?

VSAM has key-sequenced, entry-sequenced, and relative record data sets. The primary difference among the three is the order in which data records are loaded into them.

Records are loaded into a *key-sequenced data set* in key sequence: that is, in the order defined by the collating sequence of the contents of the key field in each of the records. Each record has a unique value in the key field, such as employee number or invoice number. VSAM uses an index and optional free space to insert a new record into the data set in key sequence.

Records are loaded into an *entry-sequenced data set* without respect to the contents of the records. Their sequence is determined by the order in which they are physically arranged in the data set: their entry sequence. New records are stored at the end of the data set.

Records are loaded into a *relative record data set* in relative record number sequence. The data set is a string of fixed-length slots, each of which is identified by a relative record number. When a record is inserted, you can assign the relative record number or allow VSAM to assign the record the next available number in sequence. No index is used.

When you create a data set, you define it, together with its index, if any, in a *cluster*. A cluster may be a key-sequenced data set, which consists of a data component and an index component, or an entry-sequenced or relative record data set, which consists of only a data component.

VSAM stores the records of each type of data set in the same way in a fixed-length area of auxiliary storage called a *control interval*. We can better discuss the three types of data sets if we first look at the control interval in perspective with the other logical divisions of a data set and see how and why VSAM uses it for storing records.

The Use of Control Intervals

A control interval is a continuous area of auxiliary storage that VSAM uses for storing data records and control information describing them. It is the unit of information that VSAM transfers between virtual and auxiliary storage. Its size may vary from one data set to another, but for a given data set the size of each control interval in it is fixed, either by VSAM or by you, within limits acceptable to VSAM. VSAM chooses the size based on the type of direct-access storage device used to store the data set, the size of your data records, and the smallest amount of virtual-storage space your processing program will provide for VSAM's I/O buffers.

A control interval is independent of particular types of storage devices. For instance, a control interval that fits on a track of one type of device might span several tracks if the data set were moved to another type of device, as Figure 10.2 illustrates.

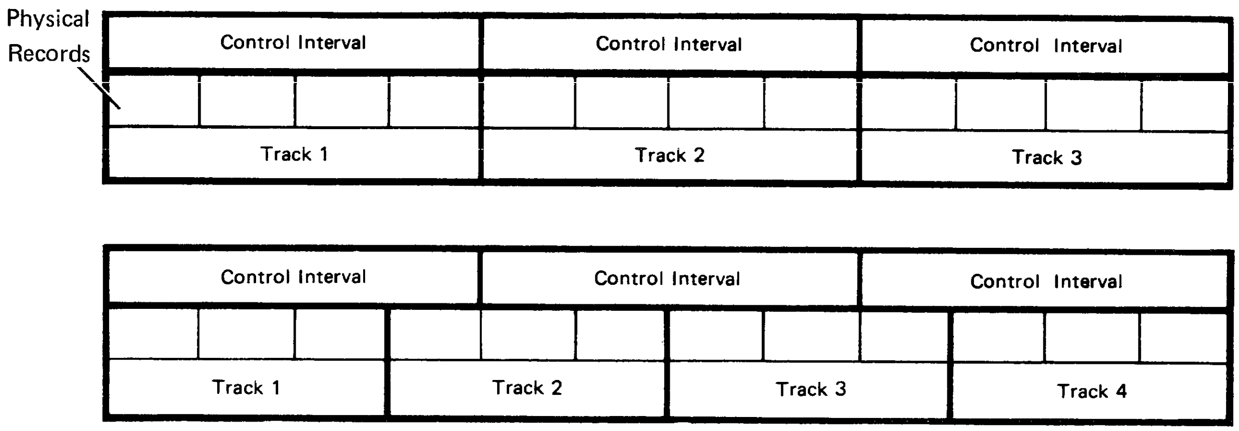


Figure 10.2 Control Intervals Are Independent of Physical Record Size

The Control Interval in Perspective

How does a data set relate to the physical attributes of auxiliary storage? And how does a control interval relate to a data set?

A volume can contain areas for VSAM's use and areas for the use of other access methods of the operating system. A storage area defined in the volume table of contents for VSAM's exclusive use is called a *data space*. It can be extended beyond its original size to include up to 16 areas (*extents*) that need not be adjacent to one another on the volume.

A data set is stored in a data space or data spaces on one or more volumes on direct-access devices of the same type. When you define a data set, you can allocate enough space to have some left at the end of the data set for additions. Otherwise, when additional space is needed, VSAM automatically extends the data set by the amount of space indicated in the definition of the data set in the catalog. It can be extended beyond its original size to include up to 123 extents, or to a maximum size of 2^{23} (approximately 4,290,000,000) bytes. Figure 10.3 illustrates the relationships among volumes, data spaces, and data sets. The figure shows portions of data sets A and C stored in different data spaces on different volumes.

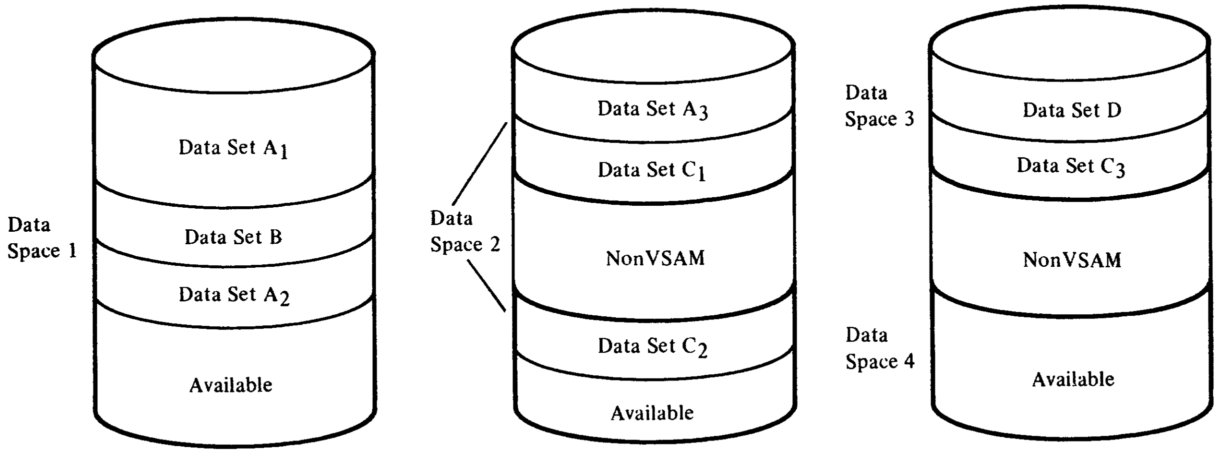


Figure 10.3 Relationship Among Storage Volumes, Data Spaces, and Data Sets

A data set is made up of control intervals. A group of control intervals makes up a *control area*. It is the unit of a data set that VSAM preformats for data integrity as records are added to the data set. In a key-sequenced data set, control areas are also used for distributing free space throughout the data set as a percent of control intervals per control area and for placing portions of the index adjacent to the data set.

VSAM fixes the number of control intervals for each control area in the data set. For a key-sequenced data set, the size of a control area is determined on the basis of the space-allocation request, user-specified or default index and data control-interval size, and available buffer space.

The Method of Storing a Record in a Control Interval

The records of a key-sequenced or entry-sequenced data set may be either fixed or variable in length; the records of a relative record data set are always fixed in length. VSAM treats them all the same. It puts control information at the end of a control interval to describe the data records stored in the control interval: the combination of a data record and its control information, though they are not physically adjacent, is called a *stored record*. When adjacent records are the same length, they share control information. Figure 10.4 shows how data records and control information are stored in a control interval. The data records are stored at the beginning of a control interval, and control information at the end.

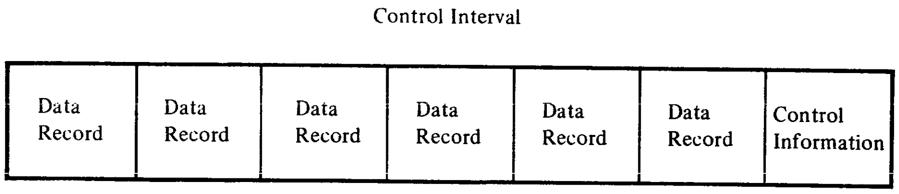


Figure 10.4 Placement of Data Records and Control Information in a Control Interval

When you define a data set, you can specify enough buffer space for the control intervals in the data set to be large enough for your largest stored record. The maximum control interval size is 32,768 bytes.

Key-sequenced and entry-sequenced data set records whose lengths exceed control interval size may cross, or span, one or more control interval boundaries. Such records are called *spanned records.* A spanned record always begins on a control interval boundary and fills one or more control intervals within a single control area. As shown in Figure 10.5 the control interval that contains the last segment of a spanned record can contain unused space. This free space can be used only to extend the spanned record; it cannot contain all or part of any other record. You must specify your intent to use spanned records when you define the data set.

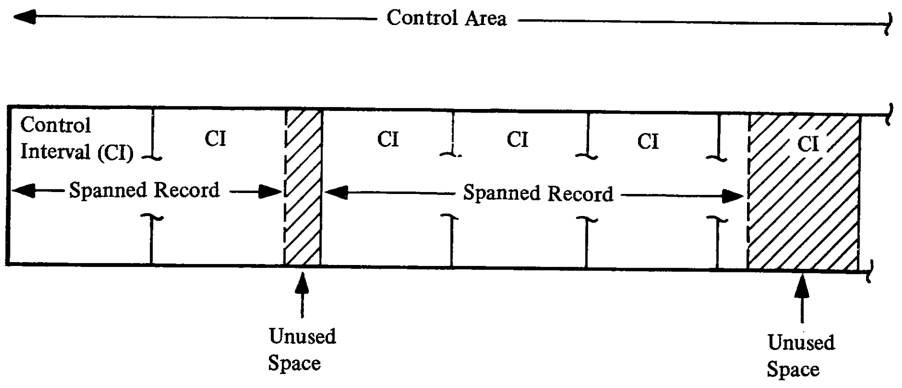


Figure 10.5 Control Intervals That Contain Spanned Records

A data record is addressed not by its location in terms of the physical attributes of the storage device (such as the number of tracks per cylinder), but by its displacement, in bytes, from the beginning of the data set, called its *RBA (relative byte address).* The RBA does not depend on how many extents belong to the data set or on whether they are in different data spaces or on different volumes. For relative byte addressing, VSAM considers the control intervals in the data set to be contiguous, as though the data set were stored in virtual storage beginning at address 0.

Key-Sequenced Data Sets

A key-sequenced data set is always defined with an index that relates key values to the relative locations of the data records in a data set. (This index is the *prime index*, in contrast to *alternate indexes*, which are discussed later. The prime index and distributed free space used to insert a new record in key sequence are discussed in the paragraphs that follow.

An index relates key values to the relative locations of the data records. A key in the index is taken from a record's key field, whose size and position are the same for every record in the data set, and whose value cannot be altered. VSAM uses an index to locate a record for retrieval and to locate the collating position for insertion.

An index has one or more levels, each of which is a set of records that contains entries giving the location of the records in the next lower level. The index records in the lowest level are collectively called the *sequence set*; they give the location of control intervals containing the data records. The records in all the higher levels are collectively called the *index set*; they give the location of index records. The highest level always has only a single record. The index of a data set with few enough control intervals for a single sequence-set record has only one level: the sequence set itself.

Figure 10.6 illustrates the levels of a prime index and shows the relationship between a sequence-set index record and a control area. The figure shows that the highest-level index record (A) controls the entire next level (records B through Z); each sequence-set index record controls a control area.

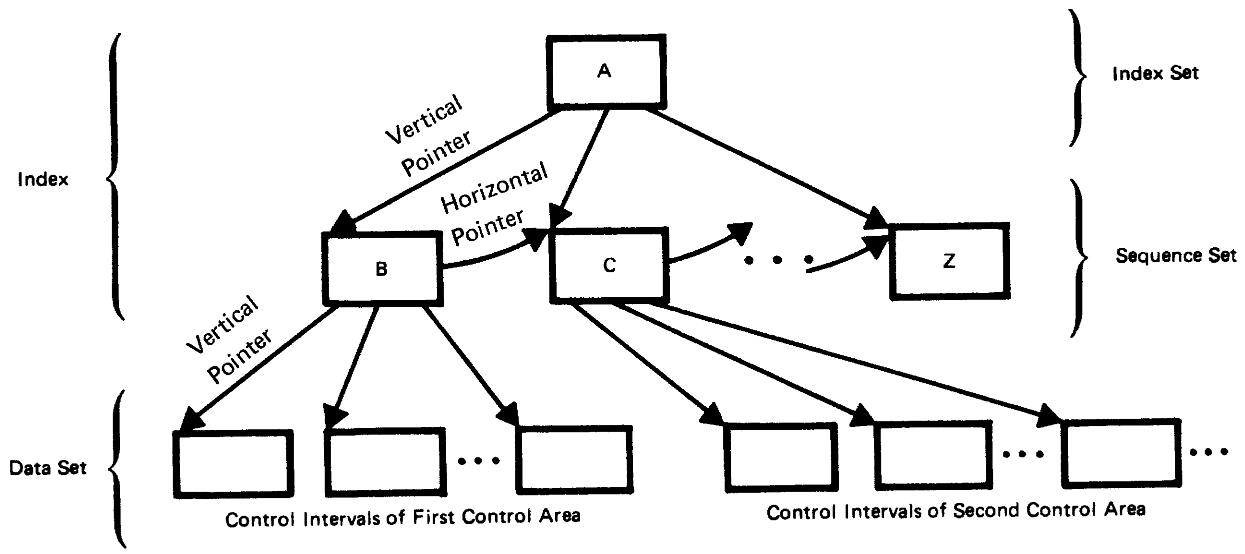


Figure 10.6 Relationship Among the Levels of a Prime Index and a Data Set

An entry in an index-set record consists of the highest key that an index record in the next lower level contains, paired with a pointer to the beginning of that index record. An entry in a sequence-set record consists of the highest key in a control interval of the data component, paired with a pointer to the beginning of that control interval. Not all data records have sequence-set entries, for there is only one entry for each control interval in the data set.

For direct access by key, VSAM follows *vertical pointers* from the highest level down to the sequence set to find a vertical pointer to data; for sequential access by key, VSAM refers only to the sequence set. It uses a *horizontal pointer* in a sequence-set record to get from that sequence-set record to the one containing the next key in collating sequence so it can find a vertical pointer to data. Figure 10.6 shows both vertical pointers and horizontal pointers.

VSAM increases the number of entries that an index record of a given size can hold by a method of *key compression*: it eliminates from the front and the back of a key those characters that aren't necessary to distinguish it from the adjacent keys. Compression helps achieve a physically smaller index by reducing the size of keys in index entries. Key compression, in an index of a particular physical size, allows you to gain access to many more records than you could otherwise. Key compression is entirely transparent to the user.

The number of control intervals in a control area equals the number of entries in a sequence-set index record. This equality has important uses in:

- Placing the sequence-set index record adjacent to the control area on a single cylinder.

- Distributing free space throughout a data set as a percent of free control intervals in each control area.

When you define a key-sequenced data set, you can specify that free space is to be distributed throughout it in two ways: by leaving some space at the end of all the used control intervals and by leaving some control intervals completely empty. The amount of free space in a used control interval and the number of free control intervals in a control area are independent of each other. Figure 10.7 shows how free space might be set aside in each control area of a data set. The sequence-set record for a control area contains an entry for each free control interval, as well as for each of those that contain data.

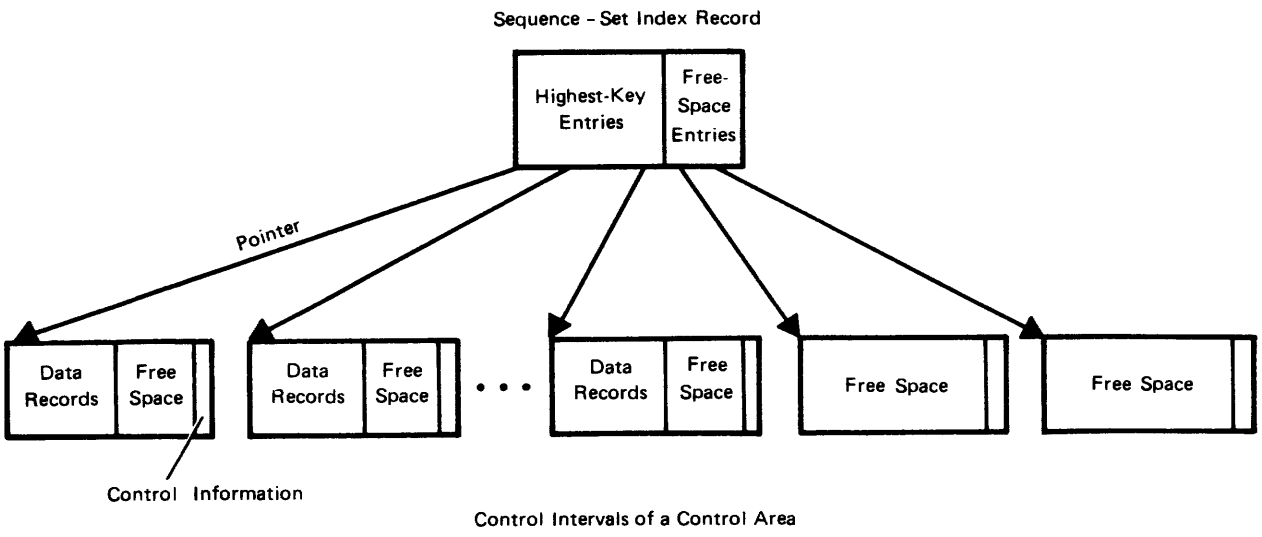


Figure 10.7 Distribution of Free Space in a Key-Sequenced Data Set

Besides the space that you distribute when you create a key-sequenced data set, space that becomes available within a control interval when a record is shortened or deleted from the data set is automatically reclaimed by VSAM and can be used when a record is lengthened in place or directly inserted into the control interval.

Reclaiming space and using distributed free space may cause RBAs of some records to change. As Figure 10.7 illustrates, free space within a used control interval is between the data in the front and the control information in the back. If a record is deleted or shortened, any succeeding records in the control interval are moved to the left and their RBAs are changed so that the space vacated can be combined with the free space already in the control interval.

Conversely, an insertion or a lengthening causes any succeeding records in the control interval to be moved to the right into free space and their RBAs to be changed.

The discussion this far has assumed that there is enough free space in the control interval for a new or lengthened record. The following paragraphs describe what VSAM does when there is not enough free space in the control interval to contain the record. For simplicity, only insertion is referred to explicitly.

If the record to be inserted will not fit in the control interval, there is a *control interval split*: VSAM moves stored records in the control interval to an empty control interval in the same control area, and inserts the new record in its proper key sequence. The number of records moved depends on the position of insertion of the new record and on the type of insertion.

For sequential insertion, records are inserted in the control interval leaving any specified free space; when the next record to be inserted will not fit in the control interval, the records with keys greater than the key of the record to be inserted are moved to a new control interval. The new record is inserted in the old control interval. If there are no records in the control interval with a key greater than the new record, the new record is placed in a new control interval.

For direct insertion, approximately half of the records in a control interval are moved when a control-interval split is required.

Figure 10.8 illustrates a control-interval split and shows the resulting free space available in the two affected control intervals. Some of the records in the control interval that is too full for insertion are moved to a free control interval, and the new record is inserted into the control interval according to key sequence. Because the number of records in the first control interval is reduced, subsequent insertions revert to the simpler case, instead of becoming more complex.

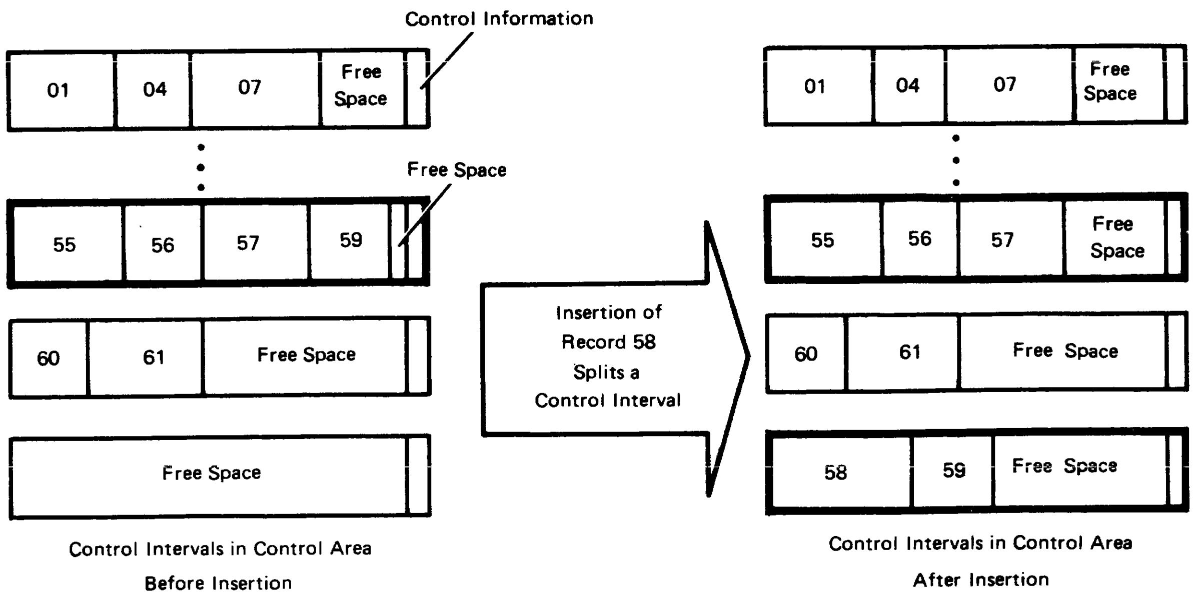

Control Information
01
04
07
Free Space
Free Space
55
56
57
59
60
61
Free Space
Free Space
Control Intervals in Control Area Before Insertion
Insertion of Record 58 Splits a Control Interval
01
04
07
Free Space
55
56
57
Free Space
60
61
Free Space
58
59
Free Space
Control Intervals in Control Area After Insertion

Figure 10.8 Splitting a Control Interval for Record Insertion

If the control intervals involved in a split are not adjacent, the physical sequence of data records is no longer the same as their key sequence. In Figure 10.8, the entry sequence of the records in the last three control intervals on the right is: 55, 56, 57, 60, 61, 58, 59. But the sequence-set index record reflects the key sequence, so that, for keyed sequential requests, the data records are retrieved in the order: 55, 56, 57, 58, 59, 60, 61.

Should there not be a free control interval in the control area, an insertion requiring a free control interval causes a *control area split*: VSAM establishes a new control area, either by using space already allocated or by extending the data set, if the initially allocated space is full and you provided for extensions when you defined the data set. VSAM moves the contents of approximately half of the control intervals in the full control area to free control intervals in the new control area and inserts the new record into one of the two control areas, as its key dictates. Since about half of the control intervals of each of these control areas are now free, subsequent insertions won't require control-area splitting. Splitting should be an infrequent occurrence for data sets with sufficient distributed free space; splitting a control area does make it possible, however, to insert records into a key-sequenced data set without previously distributed free space.

Key-sequenced data sets are appropriate for most applications. You can use the full range of VSAM's processing options to gain access to your data by a key field rather than some location-dependent manner. A simplified approach to planning is to assume that you will store your records in key-sequenced data sets and handle as exceptions those applications that are more suited to entry-sequenced or relative record data sets.

Entry-Sequenced Data Sets

No prime index is associated with an entry-sequenced data set. When a record is loaded or subsequently added, VSAM indicates its RBA to you. You must keep track of the RBAs of the records yourself to gain access to them by direct processing. One way to keep track is to build your own index, or cross reference table.

Sequential access with an entry-sequenced data set is similar to that of QSAM (queued sequential access method).

You can use direct access with an entry-sequenced data set in a way similar to BDAM (basic direct access method) by preformatting the data set with records of your choice (filled with blanks, for instance) and providing a routine that randomly associates an RBA with the key field of a record in the data set and thus distributes records throughout the data set. To store a record initially, you convert its key field to an RBA, retrieve the preformatted record at that RBA, and store the new record back at that RBA. The routine must have a procedure for determining an alternate RBA when two or more keys are converted to the same RBA. To retrieve a record subsequently, you convert its key field to its RBA and determine the alternate RBA, if one is required.

An entry-sequenced data set is appropriate for applications that require no particular ordering of data by the contents of a record. Thus, it is well-suited for a log or a journal in which the order corresponds to a sequence of events.

Relative Record Data Sets

A relative record data set has no index. It has a string of fixed-length slots, each of which has a relative record number from 1 to n, the maximum number of records that can be stored in the data set. Each record occupies a slot and is stored and retrieved by the relative record number of the slot. Relative record number 9 in Figure 10.9, for example, occupies the ninth slot; whether slots 1 through 8 are filled makes no difference.

Records in a relative record data set are grouped together in control intervals, just as they are in a key-sequenced or an entry-sequenced data set. Each control interval contains the same number of slots, the size of which is the record length you specified when you defined the data set. The number of slots in a control interval is determined by the control interval size and the record length.

Relative record data sets can be processed by key or by control interval. With keyed access, a relative record number is treated like a key. You can update records in place, delete records, and insert new records into empty slots. You can use a relative record data set in much the same you would use a BDAM (basic direct access method) data set in which the data records are not ordered by their contents or their entry sequence.

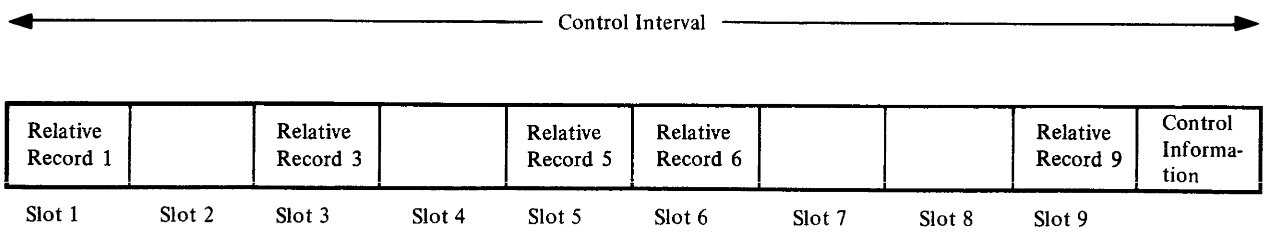


Figure 10.9 The First Control Interval Within a Relative Record Data Set

Figure 10.10 shows the comparison between a *Key-Sequenced* (KSDS), and *Entry-Sequenced* (ESDS), and a *Relative Record Data Set* (RRDS).

FIGURE 10.10 CONTRASTS THE THREE TYPES BY LISTING THE ATTRIBUTES OF EACH.

Key-Sequenced Data Set	Entry-Sequenced Data Set	Relative Record Data Set
Records are in collating sequence by key field.	Records are in the order in which they are entered.	Records are in relative record number order.
Access is by key through an index or by RBA.	Access is by RBA.	Access is by relative record number, which is treated like a key.
May have one or more alternate indexes.	May have one or more alternate indexes.	May not have alternate indexes.
A record's RBA can change.	A record's RBA cannot change.	A record's relative record number cannot change.
Distributed free space is used for inserting records and changing their length in place.	Space at the end of the data set is used for adding records.	Empty slots in the data set are used for adding records.
Space given up by a deleted or shortened record is automatically reclaimed within a control interval.	A record cannot be deleted, but you can reuse its space for a record of the same length.	Space given up by a deleted record can be reused.
Can have spanned records.	Can have spanned records.	Cannot have spanned records.
Can be reused as a work file unless it has an alternate index.	Can be reused as a work file unless it has an alternate index.	Can be reused as a work file.

Figure 10.10 Comparison of Key-Sequenced, Entry-Sequenced, and Relative Record Data Sets

Alternate Indexes Used with Key-Sequenced and Entry-Sequenced Data Sets

An alternate index provides a unique way to gain access to a related base data set, so that you need not keep multiple copies of the same information organized in different ways for different applications. For example, a payroll data set indexed by employee number can also be indexed by other fields such as employee name or department number.

This section describes the components of an alternate index and explains how they are related to the base data set. It defines new terms associated with alternate indexes, describes alternate-index records, keys, and pointers and describes how alternate indexes are maintained.

Base Clusters and Alternate-Index Clusters

In terms of access, an alternate index performs the same function as the prime index of a key-sequenced data set. The data set over which the alternate index is built is the *base cluster.* It can be a key-sequenced or an entry-sequenced data set, but not a relative record or a reusable data set.

In structure, the alternate index is similar to a cluster. It consists of an index component and a data component. The index component is identical in structure, format, and function to the prime index of a key-sequenced cluster. Likewise, the format of the alternate-index data component is identical to the format of the data portion of a key-sequenced data set. Therefore, each entry in the sequence set of an alternate-index index component points to a control interval in the alternate-index data component.

When building an alternate index, you can use as the *alternate key* any field in the base data set's records having a fixed length and a fixed position within each record. The alternate key must be in the first segment of a spanned record. For each alternate key, the data component of the alternate key itself, followed by a pointer that is the prime key or RBA of the base data record that contains the alternate key. If more than one base data record contains the alternate key then the alternate index record contains a pointer to each base data record. These duplicate, or *nonunique* keys are discussed in the section "Alternate Keys" in this chapter.

Alternate-Index Paths

A *path* logically relates a base cluster and each of its alternate indexes. It provides a way to gain access to the base data through a specific alternate index. You define a path through Access Method Services. You must name it and you can give it a password, if you choose. The path name subsequently refers to the base cluster/alternate-index pair. This means that when you refer to a path, both the base cluster and the alternate index are affected. Figure 10.11 shows how two paths can relate two alternate indexes to a single base cluster.

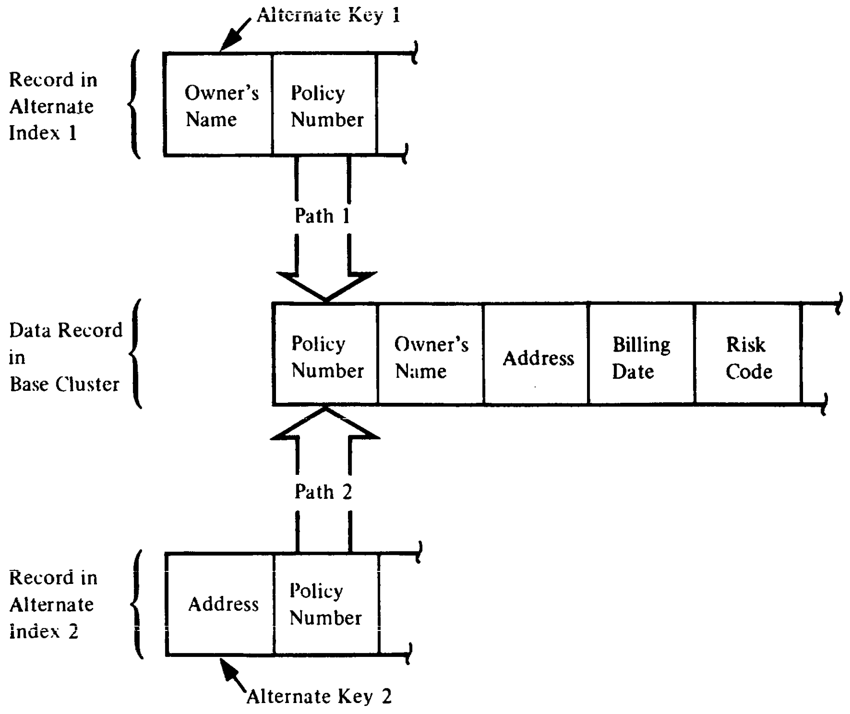


Figure 10.11 Two Alternate Indexes Over a Single Key-Sequenced Data Set

Alternate-Index Records

Each record in the data component of an alternate-index is variable-length and contains system header information, the alternate key, and at least one pointer to a base data record.

System Header Information

System header information is fixed length and indicates:

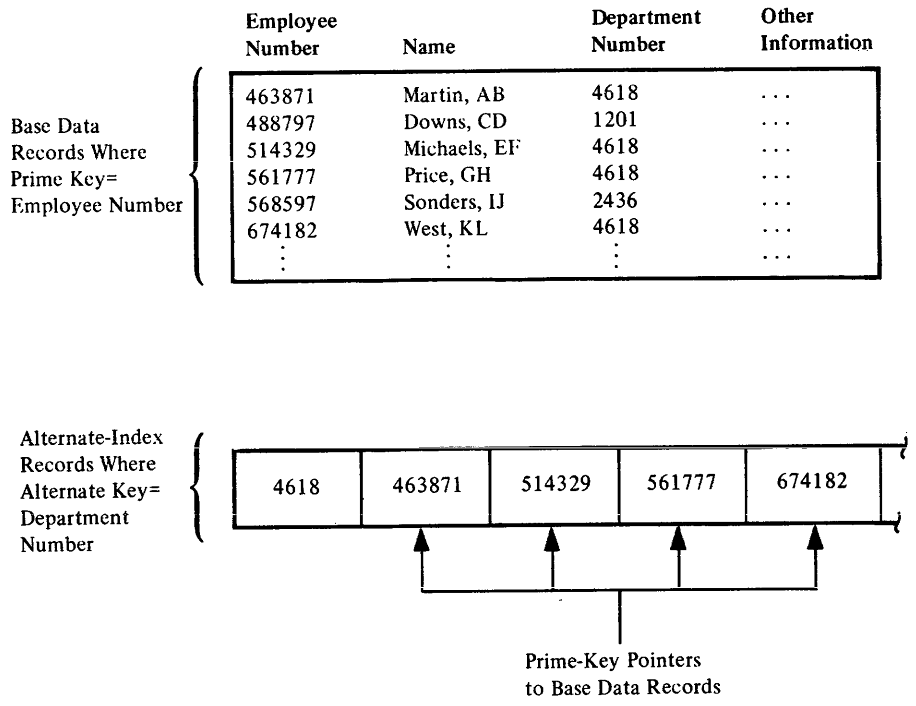
- Whether the alternate index record contains (1) prime keys or RBA pointers and (2) unique or nonunique keys.
- The length of each pointer.
- The length of the alternate key.
- The number of pointers.

Alternate Keys

Unless the base data records span control intervals, any field in the base data records that has a fixed length and a fixed position within the record can be an alternate key. The alternate key must be in the first control interval of a spanned record. When an alternate index is created, the alternate keys are extracted from the base data records and ordered in collating sequence. If you build several alternate indexes over a base cluster, the alternate key fields of the different alternate indexes may overlap each other in the base data records. They can also overlap the prime key.

Keys in the index component of an alternate index or of a key-sequenced base cluster are compressed. Keys in the data component of an alternate index are not compressed. That is, the entire key is represented in the alternate-index data record.

An alternate key may refer to more than one record in the base cluster. For example, if an alternate index is established by department number over a payroll data set organized by employee number, there will be several employees with the same department number, as shown in Figure 10.12. In other words, there will be several prime-key pointers (employee numbers) in the alternate-index record: one for each occurrence of the alternate key (department number) in the base data set. When multiple pointers are associated with a given alternate key value, the alternate key is said to be *nonunique*; if only one pointer is associated with the alternate key, it is *unique*.



Employee Number	Name	Department Number	Other Information
463871	Martin, AB	4618	. . .
488797	Downs, CD	1201	. . .
514329	Michaels, EF	4618	. . .
561777	Price, GH	4618	. . .
568597	Sonders, IJ	2436	. . .
674182	West, KL	4618	. . .
⋮	⋮	⋮	. . .

Base Data Records Where Prime Key= Employee Number
Alternate-Index Records Where Alternate Key= Department Number
4618
463871
514329
561777
674182
Prime-Key Pointers to Base Data Records

Figure 10.12 Nonunique Alternate Keys

Alternate-Index Pointers

An alternate index uses prime keys if the base cluster is a key--sequenced data set and RBAs if the base cluster is an entry-sequenced data set.

For a nonunique key, like department number in Figure 10.12, multiple pointers are associated with it. The pointers are ordered by their arrival times. That is, if a base data record is updated with a key change (for example, an employee number in Figure 10.12 is changed), or if a new record is inserted with the same alternate key value (department number in Figure 10.12), the new prime-key pointer is added to the end of the alternate-index record. In the case of a key change, the old pointer is deleted.

A prime-key pointer has the same length as the prime key field of the base data record it points to. The maximum number of pointers that can be associated with a given alternate key is 32,767, provided the maximum possible record length for spanned records is not exceeded.

Alternate-Index Maintenance

VSAM assumes alternate indexes are synchronized with the base cluster at all times and makes no synchronization checks during open processing; therefore, all structural changes made to a base cluster must be reflected in its alternate index or indexes. This maintenance is called *index upgrade*. You can maintain your own alternate indexes or you can have VSAM maintain them. When the alternate index is defined with the UPGRADE attribute, VSAM updates the alternate index immediately when there is a change to the associated base data cluster. VSAM opens all the UPGRADE alternate indexes for a base cluster whenever the base cluster is opened for output (but not control interval processing).

All the alternate indexes of a given base cluster that have the UPGRADE attribute belong to the *upgrade set*. The upgrade set is updated whenever a base data record is inserted, erased, or updated. The upgrading is part of a request and VSAM completes it before returning control to your program. If the upgrade fails because of a logical error, VSAM attempts to nullify any modifications made to the base data or to other alternate indexes, and the request that caused the upgrade is rejected.

If you specify NOUPGRADE when the alternate index is defined, you must provide a way to reflect insertions, deletions, and changes made to the base cluster in the associated alternate index.

When a path is opened for update, JCL allocates the base cluster and all the alternate indexes in the upgrade set. If allocating the alternate indexes is unnecessary, you can specify NOUPDATE and cause JCL to allocate only the base cluster. VSAM, in that case, does no automatic upgrading.

A VSAM Catalog's Use in Data and Space Management

A master catalog is required with VSAM, and any number of user catalogs are optional. Almost everything that is true of the master catalog is true of user catalogs, but user catalogs have special uses and there are significant differences between the VS1 and VS2 catalogs that we will discuss after we consider the general functions of a VSAM catalog.

VSAM catalogs are a central information point for all VSAM data sets and the direct-access storage volumes containing them. The information describing a volume and the data sets on it is extensive enough to enable VSAM to allocate and deallocate data sets on the volumes without the volumes being mounted on a device of the system. The catalogs also provide VSAM with information needed to authorize access to data sets, compile usage statistics on them, and relate RBAs to physical locations. Defining a VSAM data set automatically builds the appropriate catalog entry containing all the necessary information.

All VSAM data sets on a volume must be cataloged in the same VSAM catalog, and that catalog must be the one that owns the volume. This may be either the master catalog or a user catalog. A VSAM data set has an entry in only one catalog.

Information Contained in the Records of a Catalog

Besides data set records, a VSAM catalog has records describing direct-access volumes in terms of the allocation of data spaces and the location of available space. VSAM can allocate and deallocate space on cataloged volumes that are not mounted. However, when allocating space to a data set, if there is not sufficient space available in the data space or data spaces on a volume, you must use the Access Method Services DEFINE space command to get the additional space the data set needs.

Information in a Data Set Record

Data set records provide the information required to make the connection between a data record's RBA and its physical location in terms of a storage volume's physical attributes. Besides the type of storage device and a list of volume serial numbers, a VSAM catalog keeps other data set information, including:

- A pointer to the location of each extent of the data set.
- Statistics on the results of operations performed on the data set and its records, such as the number of insertions and deletions and the amount of free space remaining.
- Attributes of the data set determined when it was defined, such as control-interval size, physical record size, number of control intervals in a control area, and, for a key-sequenced data set, location of the key field.
- Password protection information.
- An indication of the connection between: the index and the data components of a key-sequenced data set; the index and data components of an alternate-index cluster; the alternate index and the base cluster of a path; and an alternate-index upgrade set and its base cluster.
- Information used to determine whether a key-sequenced data or index component has been processed without the other.
- Information about the volume(s) on which the data set is stored.

Information in a Volume Record

Volume information in a VSAM catalog provides the information required to keep track of data spaces and free storage areas. A VSAM catalog contains this sort of volume information:

- The volume serial number and device characteristics.
- The location of data spaces on a volume.
- The location and size of free areas available for allocation to data sets.

From this information, you can derive:

- The count of data spaces and data sets on a volume.
- The location of data sets within data spaces on a volume.
- An indication of the data spaces associated with a data set.

The Special Uses of User Catalogs

User catalogs can improve VSAM reliability and facilitate volume portability.

Improving Reliability

User catalogs are useful for improving reliability. By putting the catalog information of some of your data sets and storage volumes into user catalogs, you decentralize control, allow for the partitioning of applications, and at the same time achieve increased reliability.

Moving Volumes from One Operating System to Another

Because all VSAM data sets must be cataloged, moving a volume from one operating system to another requires that catalog information describing the volume and the data sets on it be moved along with the volume.

If you want to be able to move a volume or volumes from one OS/VS system to another, or from an OS/VS system to a DOS/VS system, define a user catalog on one of the volumes and define the volumes and the VSAM data sets on them in the user catalog. You can then transport the volumes by demounting them and removing them from the first system, taking them to the second system, and remounting them. You use Access Method Services to disconnect the user catalog from the master catalog of the first system and to connect a pointer to it in the master catalog of the second system. Any number of user catalogs can be used in this way.

You can also move individual data sets from one system to another by using Access Method Services, but the use of user catalogs for single volume portability is the most convenient way to achieve data set portability.

How the VS1 Master Catalog Differs from the VS2 Master Catalog

In OS/VS1, the system contains a VSAM master catalog as well as a system catalog. In OS/VS2, the system catalog is a VSAM catalog that also serves as the VSAM master catalog. The differences are important, particularly in the areas of naming conventions and search strategies.

The VS1 VSAM Master Catalog

The VS1 system catalog points to the VSAM master catalog, which can contain catalog entries for VSAM and non VSAM data sets (except for those belonging to generation data groups) and pointer entries for any number of optional user catalogs. Figure 10.13 illustrates how data sets can be cataloged among the system catalog, the master catalog, and user catalogs.

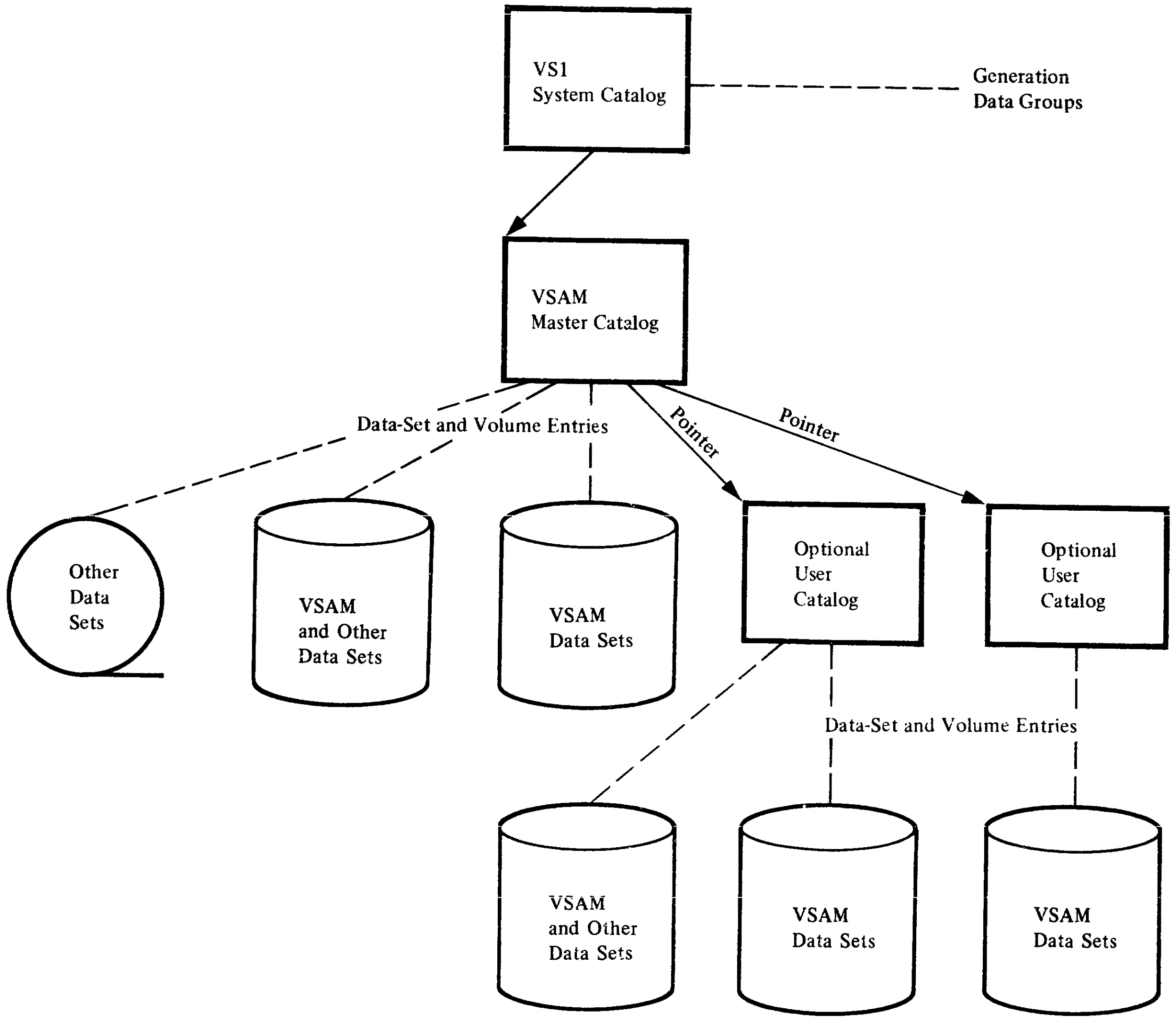


Figure 10.13 Cataloging VSAM and NonVSAM Data Sets in a VSAM Catalog

VSAM catalogs are searched before the system catalog, for VSAM data sets and data sets of other access methods. When you execute a program to process a data set, the order in which the catalogs are searched is:

1. Any user catalog or catalogs specified for the job step.
2. Any user catalog or catalogs specified for the job when none is specified for the job step.
3. The master catalog.
4. The system catalog.

Use caution in naming your data sets. Because the VSAM catalog is always searched first, it is possible to lose access to a data set cataloged in the system catalog if it has the same name as a data set in the VSAM catalog.

The VS2 Master Catalog

In VS2, the system catalog is the VSAM master catalog. It can handle both OS and VSAM data sets. As in OS, you can gain access to only one catalog per system at system initialization, and that catalog, called the master catalog, must contain entries for all the system data sets. Figure 10.14 compares the OS system catalog and the VS2 master catalog.

OS System Catalog	**VS2 Master Catalog**
Contains only OS data set entries.	Contains entries for OS data sets, VSAM data sets, VSAM user catalogs, and OS CVOLS.
One catalog during initialization.	One catalog during initialization.
Must be on IPL volume.	Need not be on IPL volume.
System controls the search strategy.	You control the search strategy.
All catalogs named SYSCTLG.	All catalogs can be named by the user.

Figure 10.14 Comparison of the OS System Catalog and the VS2 Master Catalog

The master catalog is established at system generation time, and without it, you can't define user catalogs, data spaces, or data sets. The volume on which the master catalog is defined must be permanently mounted.

The master catalog can contain pointers to OS catalogs (CVOLs), VSAM and other data sets, optional VSAM user catalogs, and generation data groups in non VSAM data sets. Figure 10.15 illustrates how data sets and catalogs might be arranged within a basic VS2 catalog structure.

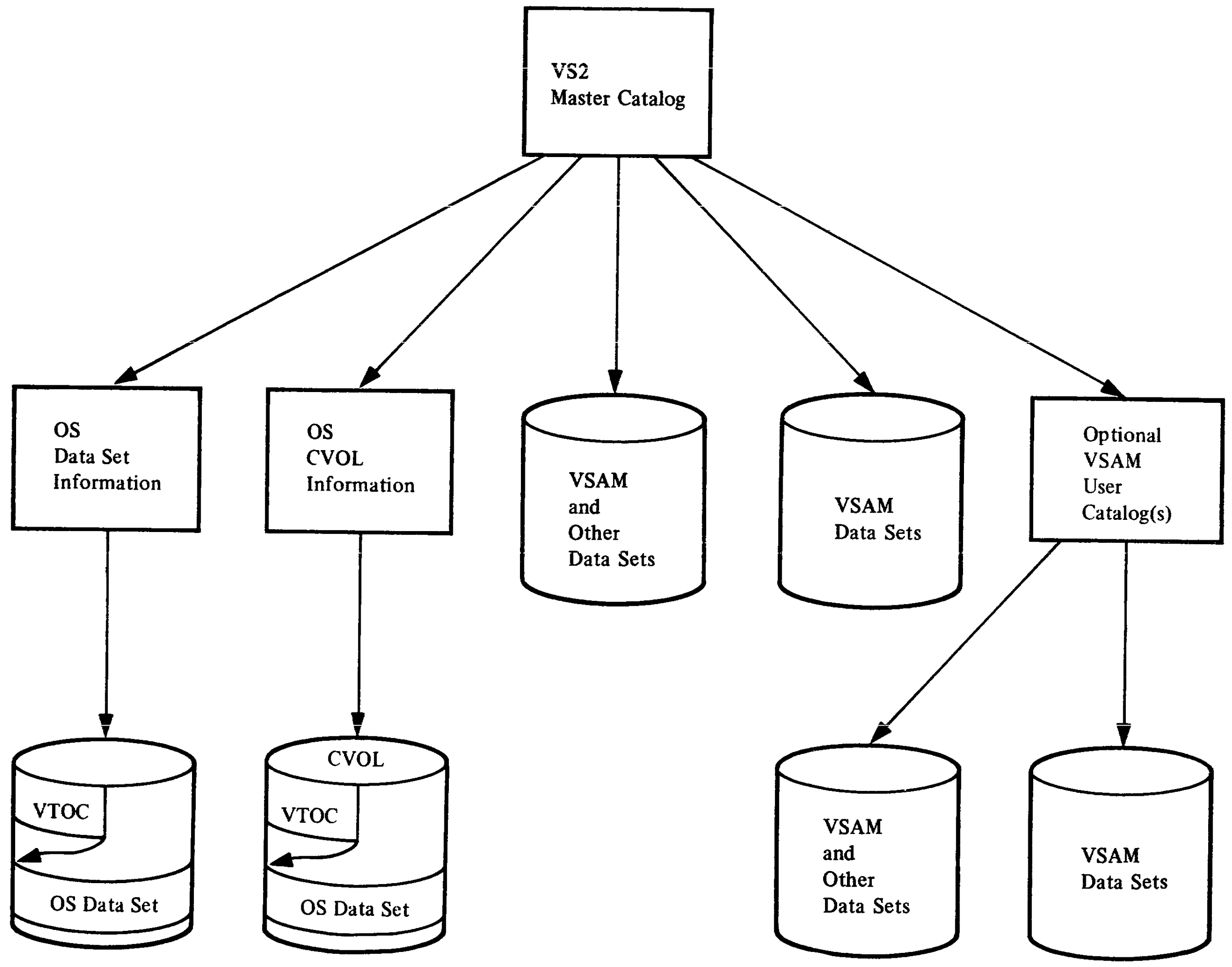


Figure 10.15 Cataloging VSAM and NonVSAM Data Sets in the VS2 Master Catalog

In VS2, alias names can be assigned to a non VSAM data set entry, a catalog connector entry (CVOL), and a user catalog. Such an entry contains a pointer to the beginning of a chain of alias entries. Each alias entry contains three pointers: one to the non VSAM or CVOL entry, one to the next alias entry, and one to the previous alias entry.

When you execute a program to process a data set, the order in which the catalogs are searched is:

1. Any user catalog or catalogs specified for the job step.

2. Any user catalog or catalogs specified for the job when none is specified for the job step.

3. The master catalog, unless the data set is qualified (contains periods) and the qualifier (the characters up to the first period) is the name or the alias name of a catalog. In that case, that catalog is searched rather than the master catalog.

Because of this order of search, a qualified data set name and an unqualified data set name cannot exist in the same catalog, if the unqualified name is the same as the first qualifier of the qualified data set name. For example, the master catalog could not contain the data set 'ABC.123' and an alias 'ABC' for a CVOL or user catalog.

Utility Functions Carried Out By Access Method Services

Access Method Services is a multifunction service program that you use to define a VSAM data set and load records into it, build an alternate index, convert OS catalogs to VSAM master or user catalogs, convert a sequential or an indexed sequential data set to the VSAM format, list VSAM catalog records or records of a data set, copy a data set for reorganization, create a backup version of a data set, recover from certain types of damage to a data set, and make a data set portable from one operating system to another.

You tell Access Method Services what to do by giving a command and descriptive parameters through an input job stream or by calling it in a processing program and passing it a command statement. In OS/VS2 you can also execute Access Method Services from a TSO (Time Sharing Option) terminal, either by executing a program that calls it, by executing it directly and giving commands and parameters through an input data set, which can come from a TSO terminal, or by entering one of the TSO commands that is identical to Access Method Services commands.

A set of conditional statements (IF, ELSE, DO, END, SET) allows you to alter the sequence of execution of a series of commands by testing or resetting codes that Access Method Services sets to indicate the completion status of each command.

There are commands and groups of commands in Access Method Services for:

- Defining and deleting data sets and listing catalog records.
- Building alternate indexes.
- Copying and listing data sets.
- Moving catalogs and data sets from one operating system to another.
- Aiding in recovery from damage to data.
- Converting OS catalogs to VSAM catalogs.
- Listing tape volumes that were mounted at the time of a check point.
- Controlling command execution by testing or setting condition codes.
- Establishing diagnostic-aids and printed-output options.

Figure 10.16 shows a list of tasks that Access Method Services commands can be used to perform. The left-hand column shows tasks that you might want to perform. The middle column more specifically defines the tasks. The right-hand column shows the commands that can be used to perform each task.

Operation		Command
Attach	a user catalog to the master catalog	DEFINE, IMPORT
Catalog	a VSAM data set	DEFINE
	a nonVSAM data set	DEFINE
Change	a data set's description in the catalog	ALTER
	the device type of the volume on which the catalog resides	REPRO
Connect	a user catalog to a master catalog	IMPORT
	an OS/VS CVOL catalog to a master catalog	DEFINE

Figure 10.16 (Part 1 of 3). Tasks and Commands

Operation		Command
Convert	a data set to VSAM format	REPRO
	a VSAM data set to sequential format	REPRO
	OS/VS catalog entries to VSAM Catalog entries	CNVTCAT
Copy	a catalog	REPRO
	a data set	REPRO
Create	a backup copy of a data set	REPRO, EXPORT
	a catalog	DEFINE
	an alias	DEFINE
	a VSAM data set	DEFINE
	a generation data group	DEFINE
	a page space	DEFINE
Define	a catalog	DEFINE
	a VSAM data set	DEFINE
	a data space	DEFINE
	a generation data group	DEFINE
	a nonVSAM data set	DEFINE
	a page space	DEFINE
Delete	a catalog	DELETE
	an alias	DELETE
	a VSAM data set	DELETE
	a data space	DELETE
	a generation data group	DELETE
	a nonVSAM data set	DELETE
	a page space	DELETE
Disconnect	a user catalog	EXPORT
Enter	a data set in the catalog	DEFINE
List	a password	LISTCAT
	a data set	PRINT
	contents of the catalog	LISTCAT
	tapes mounted at a checkpoint	CHKLIST

Figure 10.16 (Part 2 of 3). Tasks and Commands

Operation		Command
Load	records into a data set	REPRO
Modify	a data set's description in the catalog	ALTER
Move	a catalog to another system	EXPORT, IMPORT
	a VSAM data set to another system	EXPORT, IMPORT
	a nonVSAM data set to another system	REPRO
Password Protect	give or change VSAM data set or catalog	DEFINE
	add a password to an existing VSAM data set or catalog	ALTER
	delete a password	ALTER
	list passwords	LISTCAT
	replace a password	ALTER
Print	a data set	PRINT
Recover	from possible loss of data	VERIFY
Release	a user catalog from the master catalog	EXPORT
Rename	a data set	ALTER
Uncatalog	a data set	DELETE
Unload	a data set	REPRO
Verify	end-of-file	VERIFY

Figure 10.16 (Part 3 of 3). Tasks and Commands

Moving Data Sets from One Operating System to Another

The EXPORT and IMPORT commands allow you to transport individual data sets between OS/VS systems or between OS/VS and DOS/VS systems. Figure 10.17 compares volume and data-set portability. Data portability is achieved by moving volumes or by moving individual data sets.

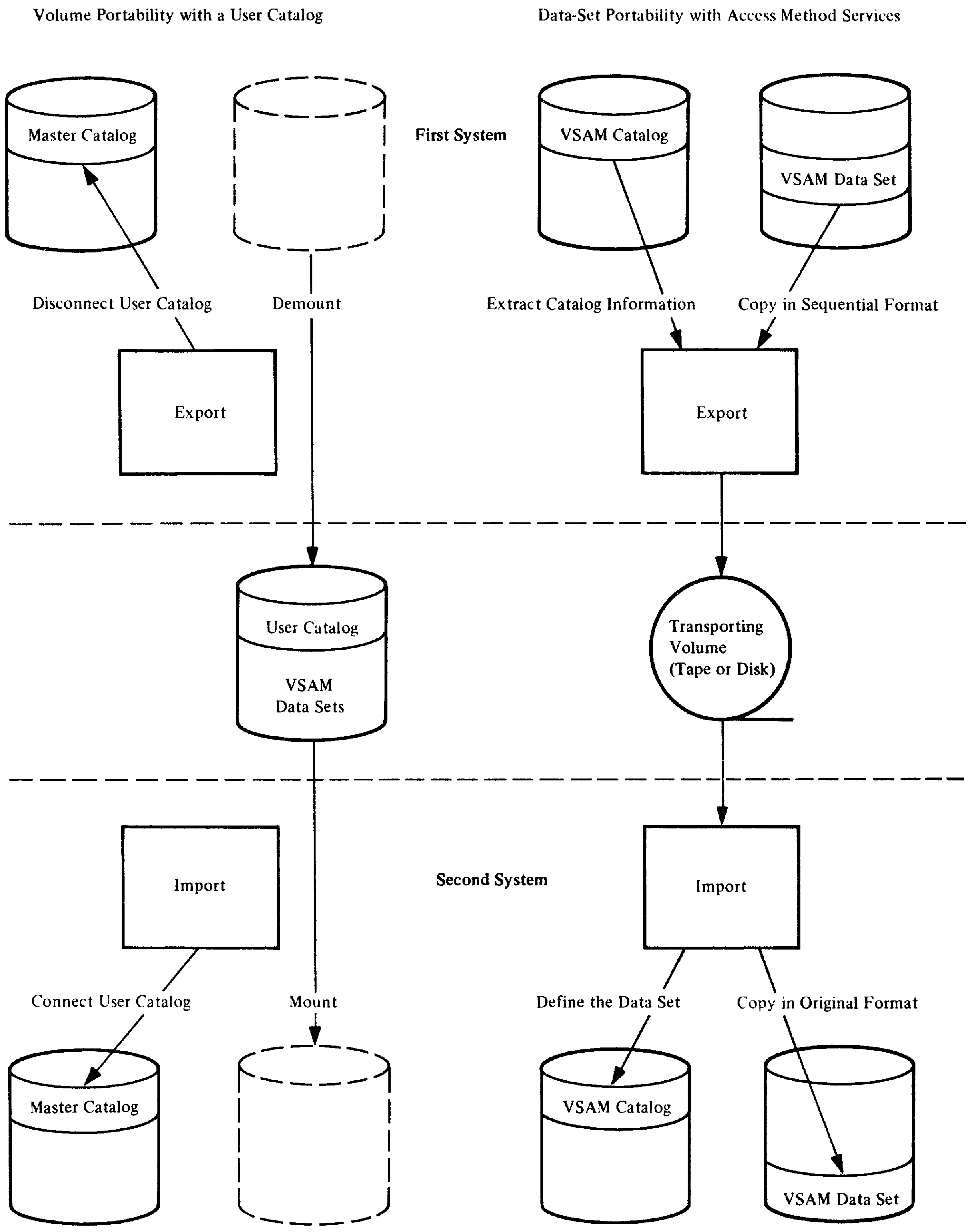


Figure 10.17 Comparison of Volume Portability and Data-Set Portability

EXPORT: Extracting Catalog Information and Making a Data Set Portable

The EXPORT command instructs Access Method Services to copy an entry-sequenced data set, a relative record data set, or a key-sequenced data set and its index (other than a VSAM catalog) in the format of a sequential data set onto a storage volume to be transported to another operating system. The transporting volume may be magnetic tape or disk. Access Method Services also extracts information from the catalog entry that defines the object to be transported and copies it onto the transporting volume. The information is used to define the object automatically in a VSAM catalog in the other operating system.

EXPORT can be used to make a backup copy of a data set. Should that data set become inaccessible, you can use the IMPORT command to introduce the exported copy back into the system.

Exportation is either permanent or temporary. In permanent exportation, Access Method Services deletes the catalog record and frees the storage space; in temporary exportation of an object, both the sending and the receiving operating systems have a copy of it, and you may specify that one or both of the copies are not to be modified. A copy so protected can only be read. You may free the copy for full access with the ALTER command.

You use EXPORT to disconnect a user catalog from a master catalog when you are moving the user catalog to another system. The user catalog is not copied, but remains on its original volume in its original form.

Paths are not exportable by themselves but are included in exports of alternate indexes or clusters; alternate indexes are exported as key-sequenced data sets. To permanently export a cluster and the alternate indexes associated with it, first export the alternate indexes and then the cluster.

IMPORT: Loading a Portable Data Set and Its Catalog Information

The IMPORT command instructs Access Method Services to define the entry-sequenced data set, the relative record data set, or the key-sequenced data set and its index on the transporting volume in the catalog that you specify, using the catalog information extracted in exportation. The object itself is stored in its VSAM format in a data space that is defined in the specified catalog.

You use IMPORT to define a pointer to a user catalog in the master catalog. The user catalog is not copied, but remains on its original volume in its original form.

You can use the EXPORT and IMPORT commands to prepare a backup version of an entry-sequenced data set and its catalog record, a key-sequenced data set, its index, and their catalog records, or a relative record data set and to load the backup copy if it is needed. When you import a backup copy, the catalog record is regenerated.

To import a cluster and the alternate indexes associated with it, first import the cluster and then the alternate indexes. IMPORT will automatically reestablish all the paths that existed when the data sets were exported.

Use the IMPORT command to introduce back into the system the backup data sets produced by the EXPORT command.

When exporting data sets from one device type and importing them to another device type, you can delete and redefine your data set with space parameters that are appropriate to the new device. The new data set must be empty. Also, it must be the same type of data set, and if indexed, it must have the same key length and position as the old data set. IMPORT uses this empty data set rather than defining a new one based on exported catalog information.

Mass Storage System (MSS)

The 3850 Mass Storage System can be used with OS/VS to store a massive amount of data online to the operating system. It is described in the *Introduction to the IBM 3850 Mass Storage System (MSS)* and the *OS/VS Mass Storage System (MSS) Planning Guide*.

When you have the Mass Storage System, you can define VSAM data spaces, user catalogs, and data sets and non VSAM data sets on mass storage volumes. The master catalog and VS2 page spaces cannot be stored on mass storage volumes.

A VSAM catalog may have defined in it both objects stored on direct-access storage volumes and objects stored on mass storage volumes. In particular, the data component of a key-sequenced cluster may be stored on a mass storage volume and the index component on a direct-access storage volume, or *vice versa*. However, if the sequence set is imbedded in the data, both components must be stored on a mass storage volume or on a direct-access storage volume. For example, both components of a user catalog must be stored on a mass storage volume or on a direct-access storage volume.

Space for an object larger than one cylinder that is stored on a mass storage volume should be allocated in cylinders to optimize data transferral between mass storage and direct-access storage.

Access Method Services for the Mass Storage System provides a set of commands for the management of mass storage volumes.

Access Method Services for managing VSAM catalogs provides parameters for options of the Mass Storage System.

A user catalog that is stored on a mass storage volume is always staged and bound when it is opened – that is, it is retained in direct-access storage until it is closed. Not binding a user catalog might degrade performance.

How Can the Time Sharing Options (TSO) Be Used with VSAM?

TSO is a subsystem of OS/VS2 that provides conversational time sharing from remote terminals. You can use TSO with VSAM and Access Method Services to:

- Execute Access Method Services commands directly as TSO commands.
- Execute a program to process a VSAM data set.
- Execute a program to call Access Method Services.
- Dynamically allocate a VSAM data set and execute a program that uses VSAM macros to process the data set.
- Allocate a VSAM data set by way of a LOGON procedure and execute a program that uses either VSAM or ISAM macros to process the data set.

VSAM data sets must be cataloged in the master catalog or in a user catalog. The master catalog is allocated when the system is initialized; you can allocate and gain access to a user catalog by making it the STEPCAT of a LOGON procedure or by using the naming conventions.

For details about writing and executing programs and allocating data sets with TSO, see *OS/VS2 TSO Terminal User's Guide (GC28-0645)* and *OS/VS2 TSO Command Language Reference (GC28-0646).*

How Can System Management Facilities (SMF) Be Used with VSAM?

SMF is an optional program of OS/VS that provides the means for gathering and recording information that can be used to evaluate system usage. VSAM supplies volume and data-set information to SMF.

For further details about the facilities of SMF and how to use it, see *OS/VS System Management Facilities (SMF)*.

How Can Existing Programs That Use ISAM Be Used with VSAM?

This section is intended for users of ISAM who are converting to VSAM. VSAM's ISAM interface minimizes your conversion costs and scheduling problems by permitting programs coded to use ISAM to process VSAM data sets. To use the interface, you must convert indexed sequential data sets to VSAM data sets (for which you can use Access Method Services), convert ISAM JCL to VSAM JCL, and ensure that your existing ISAM programs meet the restrictions for using the interface.

Comparison of VSAM and ISAM

In most cases, you can get better performance with VSAM while achieving essentially the same results that you can achieve with ISAM; you can also achieve results that you can't achieve with ISAM. The use of your existing ISAM processing programs to process key-sequenced data sets depends upon the extent to which VSAM and ISAM are similar in what they do, as well as upon the limitations of the ISAM interface itself. This subsection describes the similarities and differences between VSAM and ISAM in the areas that you are familiar with from using ISAM and indicates the functions of VSAM that have no counterpart in ISAM.

Comparison of VSAM and ISAM in Common Areas

A number of things that ISAM does are done differently or not at all by VSAM, even though the same practical results are achieved. The areas in which VSAM and ISAM differ are:

- Index structure
- Relation of index to data
- Deleting records
- Defining and loading a data set

These differences are described in the paragraphs that follow.

Index structure. Both a VSAM key-sequenced data set and an indexed sequential data set have an index that consists of levels, with a higher level controlling a lower level. In ISAM, either all or none of the index records of a higher level are kept in virtual storage. VSAM keeps individual index records in virtual storage, the number depending on the amount of buffer space you provide. It optimizes the use of the space by keeping those records it judges to be most useful at a particular time.

11 System Design Consideration

This chapter discusses some factors to be considered when designing a system using direct access storage devices.

Controls are established and used to ensure accuracy throughout data processing operations.

The controls established for direct access storage operations are basically the same as for any system; the difference lies in the manner in which they are applied. With direct processing, new data is entered to update an old master record; the old record is destroyed or erased when the new record replaces it in the file. This updating process may occur once or several thousand different times a day. Because the master record is continually updated, it is more difficult to establish the status of a record at a given time in the past, select the transactions that affected it (which are in random sequence), provide the correct output (which may have related transactions), and maintain control so that all records are still in balance and can be checked. For these reasons, the controls that will keep any type of error from going through the system will certainly increase productive time.

Data Validation at Initial Input

The largest single checking problem exists in the validation of input data at the time it initially enters the system. At this time, the data is on cards, or in the form of card images on magnetic tape, or on paper tape. The entire record should be checked, and any record which cannot be processed by all subsequent programs should be rejected.

Programmed validation checks fall into four catagories: character checking, field checking, batch or level checking, and control field checking. Since all this validation represents an extensive amount of programming, it probably will be desirable to have a separate program for input editing. In such a case, the input data is not actually processed to update the file records until editing is completed.

Several techniques are discussed in this section.

Character Checking

The checking of each character is usually done by examining the characters as a group or field.

Test for Blanks. An indication must be made as to which fields must be blank. If the field requires blanks, a constant of the proper number of blanks is compared against the field, and a test made for an equal condition. An unequal comparison indicates an error condition.

There is a case where, even though certain positions do not in themselves need to be checked for blanks, it may be necessary to perform the check to show up a keypunch error in an adjacent position. For instance, if column 25 is not used, but column 26 can be a blank or contain a 1, then a 1 in column 25 would indicate a keypunch error and, therefore, column 25 should be checked for a blank.

In some cases, because of a keypunch procedure, a field can contain either blanks or zeros. In this case a test for blanks or zeros is made. Usually it is desirable to replace blanks with zeros. When possible, fields should be punched with zeros rather than left blank.

Test for Sign. This type of check is made to ensure that the proper algebraic sign is present for the type of transaction involved.

Test for Numeric. A numeric field is tested to ensure against having interspersed blanks and/or extraneous zone bits. Blanks are replaced by zeros. If the numeric field may not contain zone bits, zones are stripped from the field by the appropriate instructions.

Zone bits over characters that are supposed to be strictly numeric generally indicate that the numeric portion is also a probable error. For instance, if zone bits that are the equivalent of an 11 or X punch are present over a digit 1, it cannot be assumed that 1 is the correct numeric digit, since both a J and a 4 are on the same key of the card punch. Therefore, the intended digit may very well be a 4 and not a 1. If this is the case, the incorrect numeric digit might be caught on the hash or control total check.

Test for Alphabetic. Normally, it is not serious if alphabetic information is omitted, since the phrase "No Description" can be inserted in the record and a message put out to correct the record later. If however, this information is vital to the application, such as the name on a payroll check, an error should be signaled.

Field Checking

These checks are concerned with the contents of fields within records.

Sequence Check. A sequence check is performed if incoming data records must be sequenced for further processing. If applicable, this type of check can be expanded to include a check on multiple records making up one transaction. For example, if three records are necessary to complete a transaction, the program should check to determine whether they are all there, in order. Further discussion of this check is included under "Completeness Check". A check for duplicate records may be inlcuded if it is necessary.

Reasonableness Check. A reasonableness check is a programmed judgment on data to determine whether it is normal. An example is scanning sales for unusual quantities or amounts such as a sale of 50 mink coats, or a $1000 charge from a cosmetics department. Another possible check would be testing a discount percentage to see that it does not exceed 15%. Then again, the check may be more complex and require first that an extrapolation of previous data be made, and then that a test be made to ensure that the new data does not vary by more than a given percentage from the computed expectation.

These examples are obvious, but in practice it may be difficult to determine correct limits on reasonableness; the best solution is to experiment. A constant can be set up for each limit; then as experience is gained or as the situation changes, the appropriate constant can be changed to reflect the new test.

Sometimes data will be entered which is known to be exceptional. In order to process this type of data, the program must include provisions for omitting certain tests or negating their results.

Consistency Check. A check for consistency means that two or more pieces of data are considered in relation to each other. For example, the classification and credit rating of a customer may indicate that he is eligible for discounts on merchandise up to a certain percentage, that his total order may not exceed a specified dollar value, and that he must pay for merchandise on a COD basis. An order from this customer must be checked against these three requirements to ensure that it is consistent with specified credit terms.

Range Check. A range check is usually applied to a code in order to verify that it falls within a given set of characters or numbers.

Special care must be taken if alphabetic, signed and unsigned numeric, and special characters fall within the standard collating sequence of this range. In this case the collating sequence of all possible good and error combinations must be considered. Tables can be used effectively in many range checks.

Limit Check. A limit check places either upper or lower quantitative limits on a field. For example, net pay on a payroll check may be limited to $250; or a total order to be delivered must amount to a minimum of $10 to avoid a delivery charge.

Limits may also be set according to a percentage of a previously used figure. For instance, in updating a master product file on prices, a check can be made that the new price is 10% plus or minus the old price.

Checking That a Code Exists. It is often necessary to verify that a code is valid for a program and does exist. Tables are used for this purpose. The size of the table depends upon the number of valid codes against which a check is made. Various programming techniques are used to search the table for the code and thus determine its existence or nonexistence.

It is possible that a code shown to be nonexistent is a new addition to the valid list, and one that will be included in the figure. When tables are originally set up, therefore, some memory space should be reserved for expansion.

Completeness Check. A completeness check verifies that no fields are missing and that no part of the record has been skipped in sequence. In discussion of checks thus far, a one-card record has been assumed. Since each field was checked, a completeness check was implied. The new consideration here is for multiplecard records that constitute a single transaction.

If all cards in the record are present and in sequence, the program continues making the remaining checks. If an error in number is found in the group of cards making up the transaction, the entire group is rejected.

The group sequence check depends upon how many of the sequenced records appear in memory at one time. If one record at a time comes in, and there is an out-of-sequence condition, the entire batch is rejected. However, if several cards, say three or four, are in memory at the same time, and they are out of sequence within the group, this condition can be program-corrected by selecting the coded records in sequence.

Date Check. A date check on incoming records is done primarily to ensure that the record date is acceptable.

Date is carried on records in various formats. The usual ones are two digits for month, day and year, as in 12 31 66, or a three-character representation of month, as in OCT 12 66. A one-position code for month can be used, such as 1-9 for January to September, and 0, –, + for October, November and December. Day can be compressed from the two digits required for 01-31 to one position by using alphabetic characters A-Z plus 0-4. Year can be carried as either one or two positions – that is, 66, 67, or 6, 7.

Another more concise method of carrying date is to number the working days. This number can start with the first working day the system is operative and continue indefinitely, or it can restart each year, in which case it would contain a digit designating year.

The checks made on date verify that month falls between 01 and 12, day between 01 and 31, and year according to actual year.

In addition, limits are checked for dates in the future or in the past. In order to do this, a decision is made as to how far in the future a record may be dated, or how late the record may be on entering the system. An arbitrary length of time may be used, such as five days, or six months, in either direction. If these limits are exceeded, a message is put out to signal an investigation.

Records with old dates can be reentries to the program and should be distinguished from records that might be rejected as too late.

Self-Checking Number. A self-checking number is one that has a precalculated digit appended to the basic number for the purpose of catching keypunch or transmission errors. Any size number can be checked. For instance, a five-digit code with the self-checking digit would be carried as a six-position code. Normally, the self-checking digit is used with identification codes, such as part number, customer number, or employee number.

There are two techniques for calculating a self-checking digit: the modulus 10 and modulus 11 methods. In both methods the digit is originated by a special device on the IBM 24 Card Punch, 26 Printing Card Punch, or 29 Card Punch, or by an initial calculation operation.

- Modulus 10 Method

  The modulus 10 method, which is completely described in *Self-Checking Number Feature* (G24-1057), is as follows:

  1. The units position and every alternate position of the basic code number are multiplied by 2.

  2. The digits in the product and the digits in the basic code number not multiplied by 2 are crossfooted.

  3. The crossfooted total is subtracted from the next-higher number ending in zero.

  4. The difference is the check digit.

Example:

Basic code number:	6 1 2 4 8
Units and every alternate position of basic code number:	6 2 8
Multiply by 2:	x2
Product:	1 2 5 6
Digits not multiplied by 2:	1 4
Cross-add:	1+ 2+ 1+ 5+ 4+ 6=19
Next-higher number ending in zero:	20
Subtract crossfooted total:	−19
Check digit:	1
Self-checking number:	6 1 2 4 8 1

Other examples:

Basic code number	Self-checking number
45626	456269
30759	307595
73074	730747

- Modulus 11 Method

  The modulus 11 method, which is covered in *Self-Checking Number Feature, Modulus 11, and Its Associated Self-Checking-Number Generator, Modulus 11* (G24-1022), is as follows: Each digit position of any basic number is assigned a "weight" (checking factor). These factors are 2, 3, 4, 5, 6, 7, 2, 3, 4, 5, . . . ,starting with the units position of the number and progressing toward the high-order digit. Any size field may be converted into a self-checking number.

1. Write the number, as illustrated below, leaving space between the digits.

2. Below each digit, starting at the right and working left, place the corresponding checking ("weighting") factor.

3. Multiple each digit by its checking factor and add the products.

4. Since this is a modulus 11 system, divide the sum of the products by 11, and subtract the remainder from 11.

5. The result is the check digit.

```
Example:
Basic number:  9  4  3  4  5  7  8  4  2
From right to
left, the check-
ing factors:   4  3  2  7  6  5  4  3  2
              ---------------------------
Multiply and
add the
products:     36+12+ 6+28+30+35+32+12+ 4 =195
Divide total by 11:  195 ÷ 11 = 17, remainder 8
Subtract:    11-8 = 3 (the check digit)
Self-checking number: 9434578423
```

The self-checking digit is used to verify the correctness of a code by recalculating the check digit and comparing the result with the digit in the record. An equal condition signals that the code is correct.

This type of check catches about 97% of transposition and substitution errors, which are the most common type of keypunch and clerical errors.

The fact that the check digit is the same on recalculation does not mean that the code does in fact exist as a valid code, but only that the combination of digits in the code field is correct. For instance, it is possible for an employee number to check out correctly, but for that employee to be no longer on the payroll.

Borderline Tests. There will be cases when the data in a record just passes the acceptance test – that is, when a particular field is borderline but does not invalidate the record. If, however, several fields in the record are borderline cases, their cumulative effect may cause the record to be unacceptable. This situation should be considered and such records put out for investigation.

Methods for Processing Records Containing Field Errors. In some types of applications it is possible to process records even though they contain erroneous data. Some techniques for dealing with such conditions are:

- *Use of Approximations.* It is often possible, when data is either omitted, unavailable or unreasonable, to use an approximate figure and process the record. A common example of this technique is the use of a minimum charge on utility bills. If a meter cannot be read for a certain billing date, either a standard minimum billing figure is used, or a figure is computed on the basis of average past usage. The record can then be completely processed.

  In some circumstances a special listing must be kept of records that use approximations, and the necessary follow-up must be maintained to replace the approximations with actual figures when they become available. In other situations no special record is necessary since the condition will be self-correcting. Such is the case for utility minimum charges.

  Another use of approximations occurs when certain information is not presently available and a dummy number is used to process the record. For instance, an order received from a new customer who has not yet been assigned a customer number could be processed by using a constant customer number and by putting out a message for follow-up. In cases such as this, a fixed constant is used which is recognizable as an unreal code, quantity, or amount.

- *Invalidating Part of a Record.* In order to continue processing automatically under all conditions, the technique of invalidating or disabling part of a record may be employed. This means that a significant code is inserted in the record to prevent processing of a portion of the input data. Follow-up would, of course, be necessary.

  Another use of this technique is in the updating of a master file. It is possible to include new data in the master which is not valid until a certain date. Before the conversion date it is coded as invalid, and at the proper date it is made available to the program.

- *Unscrambling.* Programs to unscramble data are used in many instances. Unscrambling means rearranging the data by character or digit. This technique can be used in relation to a multicharacter code or an entire record including the quantitative data.

  The unscrambling technique is generally used on data that has originated from paper tape or some other data transmission medium. In the case of paper tape, it is possible that an operator may have put the tape on backwards when converting to magnetic tape, so that all records are reversed.

  The procedure for unscrambling is to read the record in the usual manner and check it. If it is in error, the fields are then checked backwards – that is, from right to left; if still in error, the record is offset one position to the left and checked; if still in error, it is offset one position to the right and checked; and so on. This type of check has innumerable combinations that can be tried. The most successful rearrangements result from experiment.

Batch or Level Checking

A batch or level is a subgroup of a logical file of information. Input data is batched for the purpsoe of balancing small groups of data to control totals. If an error is discovered, the erroneous batch can be rejected without the loss of the entire run. Also, the error can be located more quickly and easily in a small section of a file.

A batch may be made up of groups of records having a common identity, such as department or branch, or it may be made up of a specified number of records, say 500.

The type of batch is generally based on the manner in which the data arrives at the data processing department. if it arrives by department, or location, these would seem to be logical groups despite volume. If, however, data is to be batched in size groupings, the only consideration is convenience in error trackdown. The smaller the batch, the easier it is to find the errors, but since more totals are required, additional clerical and machine time is necessary.

Each batch of input data includes as a first or last record a batch control card, which is created either in the originating department or by a control group within the data processing department. The batch control record contains batch number, date, originating source, record count, hash totals of identifying information, and control totals of quantities and amounts.

As the batch is processed through the edit program, totals of the detail records are accumulated for both accepted and rejected records. If all control totals balance, the batch is accepted; if any do not balance, it is rejected. Complete lists of rejected batches are maintained for follow-up purposes.

The detail records may or may not contain all the information in the batch control card. If the information is present in the detail record, it is checked; otherwise, only record count and control totals checks are made.

If the batch balances, but certain records in it are rejected on other tests, such as reasonableness, the batch may be (1) rejected until the error record is corrected, or (2) reentered with new batch control totals from which the error records have been deleted.

Batch Number Check. A check is made that the batch number in the control card matches the batch number in all the detail records. If any record in the group does not contain the same batch number, it is investigated. If the batch is rejected for this reason, and if the totals for the batch balance, the error is probably a keypunch error in batch number and can be easily corrected.

Batch Record Count. A count is made of all detail records in each batch. This count must balance to the record count in the batch card. An out-of-balance condition indicates missing, additional, or duplicate records that must be checked.

A simultaneous error in record count and in batch number would indicate that an additional record has been picked up in the batch and is probably a record that is missing from another batch.

Batch Control Totals. All quantitative fields in the detail records are accumulated and checked against the batch totals. Any error causes the batch to be rejected. This is the classic check that has always been made on data as it is processed through any data processing system. Except for compensating errors, a balance here is proof that the batch is complete and correct on quantity and amount fields.

Batch Hash Totals. A hash total is an accumulation of digits generally taken from an identification or control field. This type of total is taken solely for checking purposes, since the actual total has no quantitative significance.

Hash totals enable the user to positively identify an added or missing record. For instance, if a record in the amount of $25 were missing from one batch and appeared in another batch where there was also one for $25, it would be difficult to determine on the basis of the amount field which of the two was out of place. However, if the control fields were different, the out-of-place record could be easily identified.

Control Field Checking

The control field check is not normally made during the edit program. Rather, it is included in the first processing run against the master file. At that time, each detail record is compared with the master on the appropriate control field. Nonmatches must be investigated further – either in the program or manually. The program can, for example, interrogate a code to determine whether the nonmatch is a new product that has not yet been added to the master file. If the nonmatch cannot be resolved by the program, it is put out as an error for follow-up.

Sometimes the job is such that the check must be made during the edit run. If this is the case, it can be done in several ways. A short master record containing only the code numbers can be used for comparison. Or, if the number of codes is small enough, a table can be created in memory and a table lookup done on the code.

Once an error has been found during the edit program, its cause must be determined and the error corrected. The usual procedure for correction is to route the listing of error records and related messages to someone who investigates each record and makes the proper correction. If the errors have resulted from a new application just put on the computer, or if the data has originated at a remote location, the process of tracking down the error is more involved. With a new application, it may be necessary for several experienced people to review the error records.

After the cause of each error has been found and the correction made, the record is reentered into the edit program. The rules for reentries may be different from those for original data. For example, reentered records may be 10 to 30 days late, whereas current records may be a maximum of 5 days late.

In handling errors:

1. Overall control of good data plus error data must be maintained.

2. Reconstruction of the error record from the source data must be possible.

3. The rules on resubmission of corrected records must be clearly defined.

4. Overall controls must be reestablished after correction runs.

Often, input data is edited at more frequent intervals than it is processed. For instance, in a weekly processing run, the input data might be edited daily, while in a daily run of, say, invoices, the input order data might be edited in several batches throughout the day. Thus peak loads on corrections are avoided.

Systems or Internal Controls

An external control on all records is established as close to the originating source as possible. This means that as soon as the data is keypunched or received over transmission media, control totals are established which balance back to accompanying group totals.

A record is also kept as to exactly what data has been received. This may be a manually filled-in form referencing the source department and the number of records, or it may be as elaborate as a machine listing of all incoming records. The point is that the records must be controlled from the moment they come in the door of the data processing department until processing is completed.

The internal controls to be discussed here are directly related to the external controls and must tie back to them.

Systems or internal controls include the checks incorporated into a programmed system, exclusive of the validation checks on input data, for controlling the number of records being processed and the correctness of the machine calculations. Even though input data is acceptable on range and limit checks, calcualted results using these factors may be outside an accepted limit and should also be checked. For instance, factors A and B may satisfy the validation tests made, but A times B, or A divided by B, may be out of range.

Control Totals. Control totals can be taken on amount fields, or quantity fields of like sizes, such as units, dozens, or cases. These totals are added algebraically.

Batch control totals on input data have already been discussed. In addition, overall control totals are used which include totals by various groupings, such as department, branch, or total file. These totals generally are of interest in themselves, since they represent specific control groups.

A balance on all control totals can usually be interpreted as proof that a file is complete and has been processed correctly.

A programming consideration worth noting in regard to control totals concerns the memory space reserved for these totals. It is wise to reserve enough memory positions to accommodate totals for two to four times the normal volume of records going through each program. This is because two days' work may be put through the machine at one time, or volume may suddenly spurt as a result of a current advertising compaign.

Hash Totals. Hash totals have also been mentioned under batch hash totals. A hash total is the sum of the digits of an identifying field. On some machines, hash totals may be taken of the numeric part of alphabetic fields. A hash total is unlike a control total in that the sign is ignored and carries are dropped.

Quantity totals may also be hash totals if all quantity sizes are added together – for instance, units, dozens, and packages.

Hash totals are used for checking purposes only, and are of no interest in themselves.

Crossfooting Checks. Crossfooting, in the checking sense, means cross-adding or subtracting two or more fields and zero-balancing the result against the original result. This is an effective control when total debits, total credits, and a balance-forward amount are maintained in each account; total debits and total credits can be crossfooted to prove that the difference equals the balance forward.

For discussion purposes, assume an accounts receivable application. In posting to accounts in disk storage, the stored program must select for each transaction the proper account record, read it into a working storage area, update it there, and, if posting is correct, write it back in the same disk storage location. In the final phase of posting, the old account record is replaced by the updated one.

The accuracy of posting should be proved between the last two steps; this is the last point at which the old account record is still available. For proof, total debits and total credits are crossfooted and the net result compared with the new balance-forward amount; they should be equal. If they are not, the last step is skipped and the updated record is not returned to disk storage until the error is corrected.

Crossfoot checking can also be used on a recalculate basis by reversing the additions and subtractions. For example, the original calculation would be:

$$+A+B+C+D+E = F$$

and the recalculate:

$$-A-B-C-D-E+F = 0$$

Balancing Partially Processed Data Files. When random transactions or batches are processed against records in disk storage, only the active records are consulted. Since the inactive records are not read, the balancing procedure must depend upon the assumption that they are correct. This assumption is proved by trial-balancing all accounts on some cyclic basis that is frequent enough to enable corrective action.

The remaining control problem rests upon assurance that the active records are processed correctly and that a record which is in error can be detected within the system.

The means for detecting errors with this technique is provided by establishing balance fields in addition to detailed item fields. For accounts receivable records, a total-amount-due field is established which is the crossfoot total of the gross amounts of the individual (invoice) items.

All processing of those records includes crossfooting the record before and after processing to ensure that the record was and remains in a balanced condition. A total of all balances of the affected records "before" is reconciled with the changes and the total of the balances "after". When this is done, the total of the changes may be posted to the total control records, which will then reflect the correct total of all record balances. An example is shown below.

Accounts before processing:

	Item	Item	Balance
Account A	50.00	00.00	50.00
Account D	75.00	75.00	150.00
Total old balance of all accounts			10,000.00

Two cash receipts to be processed:
Transaction A for 40.00
Transaction B for 75.00

Accounts after processing:

	Item	Item	Balance
Account A	10.00	00.00	10.00
Account D	00.00	75.00	75.00
Total balance of affected accounts "before"			200.00
Total transactions			115.00
Total balance of affected accounts "after"			85.00

Since 200.00-115.00=85.00, the procedure checks, and the new control balance of all accounts is reduced from $10,000.00 to $9,885.00.

Such a balancing procedure is no different from that used in manual bookkeeping systems where the total main file is split into daily cycles and a total control covers all cycles.

If subledger controls are used for controlling smaller groups of records, they should be reconciled to the grand total before and after processing runs or at periodic intervals during processing. Provision must be made for restoring changed subledger totals to the last previous reconciled figures, but otherwise changes are made as posting is accomplished. The general philosophy is that if the changes balance in detail, they may be used in the total subledger. If the subledger totals balance similarly, the change may be posted to the grand total.

If they do not balance, the detail records are trial-balanced to the subledger and the subledger to the grand total.

It is noted that if an account which is inactive is out of balance, it will go undetected. However, the procedure outlined guarantees that the last time it was legitimately processed, the record was correct, and that the next time it is processed or trial-balanced, the error will be detected.

Multiplication Checking. Multiplication checking can be done in a variety of ways, depending upon the format of the record.

One of the simplest methods of multiplication verification is to reextend with the multiplier and multiplicand reversed, and zero-balance the products. Another method is to obtain one of the factors from a different source, such as a table lookup based upon an identification code, and zero-balance the recalculation with the original product. Still another method is to total the quantities to be multiplied by the same multiplicand and then do one multiplication per multiplicand instead of several. The product would then be zero-balanced with the total of the individual products.

If the machine time required for multiplication checking is excessive, a check on every hundredth or five-hundredth record may be considered. This check, however, will catch only a consistent machine failure.

Rounding Considerations. Error conditions can be incorrectly signaled as a result of attempting to balance the multiplication of a total against the sum of its parts which have been individually extended and half-adjusted.

In order to avoid error signals on such conditions, it is possible to use a group half-adjustment in the individual extensions. This method requires that an artificial five be introduced only once per group calculation (vs. each calculation) and that the adjustment position be cumulative until the end of the group. The following example illustrates this case:

	Time (Hours)	Rate	Individually Adjusted	Actual Calculation	Accumulated Decimals	Decimal Accumulated Adjustment
half adj.					0¦5	
	2.5	1.25	3.13	3.125	1¦0	3.13
	2.5	1.25	3.13	3.125	0¦5	3.12
	2.5	1.25	3.13	3.125	1¦0	3.13
	0.5	1.25	.63	0.625	¦5	0.62
Total	8.0		10.02	10.000	¦5	10.00
Daily	8.0	1.25	10.00	10.000		

In this example, the individually adjusted extensions of hours times rate add up to .02 more than the group total extension. It can be readily seen that this type of discrepancy could grow substantially if perpetuated through an entire program.

Another method of dealing with the rounding situations is to use a limit on the amount of tolerable error and consider the amount as correct if under the limit. If this method is used, it is preferable that the limit be tested on as small a group of calculations as possible, since it is very difficult to determine whether an error of a fairly large amount is due to thousands of rounding errors or is in fact one large error.

Division Checking. Division is usually checked by multiplication. This is done by multiplying the quotient by the divisor, adding the remainder, and zero-balancing the result against the original dividend. For example, if the original calculation is:

$$A \div B = Q + R$$

the verification is:

$$(Q \times B) + R - A = 0$$

The remainder situation may be handled by the use of formulas that test for successive plus or minus conditions. Examples of such formulas are available in the 602 and 604 reference manuals.

Another possibility for division checking is a multiplication of the dividend by the reciprocal of the divisor and a comparison of this result with the original quotient.

Negative Amount Considerations. Control totals have been defined as being algebraic additions. This recognizes the fact the credit items occur and also that reversing entries are possible for every plus entry.

Because of these negative entries, it is possible to develop totals that bear a strange relationship to each other. For example, consider the case of two sales transactions, one of which paid a commission to a salesman while the other, a credit item, did not:

Net Sales	Commission
+100.00	+6.00
–500.00	
–400.00	+6.00

If these two transactions were the only two processed for this salesman on this day, it would appear that a commission was paid for credit business. Also, if a reasonableness check were applied to the totals, for instance to determine that the commission percentage ranged from 4% to 10%, an error condition would be signaled.

Unexpected results like the above do occur when negative numbers are being processed. Consideration should be given to such possibilities, and procedures should be developed to handle them properly.

Processing Nonstandard Input and Output. Processing programs that are run after the edit program do not include editing as such. However, they do incorporate a similar principle, in that they must provide a programming path for nonstandard conditions. For instance, a program may be set up to expect three types of input per transaction. If one type is missing, it may be desirable to have the program skip that transaction, continue processing other transactions, and send out a message about the transaction and the missing data.

The point is that programs should be written to continue to run under as many conditions as possible. Error messages would, of course, be put out on every error or nonstandard operation. In programming, one should never decide that a condition will not occur. Experience shows that if it can happen, it will happen.

Record Coding. File data destruction, when it does occur, is often the result of programming error. Some of the causes have been (1) attempting to run a program before thorough testing, (2) entering incorrect beginning or ending addresses for sequential file changes, and (3) blanking records. To avoid having such incidents occur unnecessarily, a code can be placed in each data record and matched against a constant associated with the proper program. This establishes the fact that the program has the right to work with the given record. Although not foolproof, it will prevent a large percentage of accidental program errors. It requires few instructions and little storage space.

Messages. Messages are usually associated with error conditions, but they are also used with control totals. The principal rule in regard to messages is that they should be clear, complete, and concise.

An error message should identify the error record, specify what is wrong with it, and use as few memory positions as possible. For example, a message such as:

INVOICE 12345 AMOUNT OVER LIMIT

is not sufficient to describe the actual case. A better message would be:

INVOICE 12345 PROD 6789 AMT OVER $500.

QUANTITY 25 PRICE $100.00 AMT $2500.00

This message enables the control clerk to determine that if a quantity of 25 is reasonable, the error condition is in the price. In this example, it is probable that the price is incorrect in the master record and should be $10 rather than $100.

Message standards can be set up that will aid in proper format and content.

If a sufficient amount of memory is not available for the necessary error messages, a coding system can be used. The original program detecting the error would then put out an error code and the identifying information. When the error message tape is printed, the codes can be translated into English and the identifying information inserted into the message format.

Undetectable Errors. In input data, errors can occur which defy detection. They result from human mistakes and can be in detail transactions, or, worse yet, in data used to update a master file.

An example of an undetectable error in a detail transaction is the case where a customer phones an order for twelve pieces of an item and the order clerk writes down 11. The quantity is punched as 11, and since 11 is as valid to the program as 12, it is processed as 11. Not until the customer receives only 11 pieces is the error found.

While it should be realized that human errors undetected by the program can occur, this should in no way detract from the use of a comprehensive set of checks. The vast majority of error conditions are detectable and can be discovered by a complete checking operation.

Output Controls

Controls on output are more difficult to establish because the resulting output may not coincide with input. For example, if an input control total were taken on quantity, it might not balance with the invoice because of back orders or items deleted from inventory. Had the total been on a part number or hash total, a substitute item would have caused an out-of-balance condition. One solution to a problem such as this is to obtain totals of the quantities that were invoiced or back-ordered, as well as sales that were lost or adjusted. A tally of these totals should balance with the input control total.

Since the volumes processed by a system are normally so great that the taking of external totals on the output documents themselves would be too unwieldy and costly, a more practical approach would be to control by batch. The number of documents processed would be totaled and compared with input controls to prove the inclusion of all. As an output control, forms can be prenumbered so that the total number of documents invoiced (output) could be balanced.

For example:

Input documents		2362
Invoices	2200	
Lost sales (incomplete orders)	31	
Back orders (complete orders)	131	
		−2362
		0000

There are other output controls, such as systematic manual checks, statistical sampling, physical inventories, and analysis of reports. The last of these may well fit into the category of systematic checks if they are reports that are created weekly or monthly.

Many means can be devised for output controls, but the degree of control should depend to some extent on the type and number of input as well as process controls and the nature of the job. Payroll checks, for instance, should have the ultimate in controls, whereas an invoice for chain stores may have few output controls from the data processing department.

Intermediate controls are generally included in output controls because they are the result of a given run. In addition, however, they may be carried forward to another system or run before they have any meaning for balancing purposes. This injection of the time element requires other considerations. The time difference in taking the control totals may vary from a few minutes to days. The specific situation must govern this; however, two guides are:

1. Keep the time period between control totals *as reasonably* short as possible.
2. Provide for convenient systems and physical handling of the controls.

Built-in Checks. Advantage should be taken of all automatic and built-in checks, all transfers to output devices are parity-checked and the devices themselves have automatic checks. For instance, printers have setup checks, and tapes have dual-gap heads to ensure accurate recording. Direct access files have a very positive checking method in their Cyclic Check.

Accounting controls that are too tight can hamper processing; inadequate controls can make the processed data worthless. Controls should therefore be used wisely. Only those that satisfy a need should be included, and they should be simple and easy to maintain.

Program Testing

Programs must be thoroughly checked out before they are put into production. An attempt should be made to anticipate and allow for all possible errors, exceptions and unusual combinations of circumstances. Routines or separate programs must be written for correcting errors when they do occur. For example, updating files while printing a report with the wrong form inserted can be disastrous if there is no program written to print the report without updating. Up-to-date copies of all programs should be maintained, and any changes authorized and fully documented.

Direct Access Label Checking

The operating systems require that each direct access volume (a disk pack, data cell, drum, or part of a 2301 served by one access mechanism) must have one 80-position standard volume label. The operating system will check that the volume serial number in the volume label (each volume is assigned a unique volume serial number) matches the volume serial number stated by the user at job initiation time, thus ensuring that the correct volume has been mounted. The volume label also contains the address of the area on the volume that contains the standard file labels of the files that reside on the volume. This area is called the volume table of contents (VTOC). The standard file label or set of standard file labels for a file identifies that file, gives its location or locations on the volume, and contains information to prevent premature destruction of the file. The number and format of the labels required for any one file depend on the file organization structure and the number of separate areas (extents) used by the file. The operating system writes file labels for new files. It also checks the file labels for existing files to ensure that the correct file is online and that a new file being created will not destroy an unexpired file.

The Audit Trail

The audit trail must provide the detailed business information for the period of time that will satisfy legal, accounting, and practical requirements. It must also provide a method of extracting the information that is most economically consistent with the requirements.

In some computer runs, there are no audit trails; such is the case with engineering problems having variables that are entered for trial fits. There will also be runs where added procedures are unnecessary as well as uneconomical because the amount of source data is small and readily available for checking and rerun purposes. Most commercial applications, however, require audit trails – for several reasons:

1. The audit trail is the means for checking any discrepancies that occur.

2. Business has legal requirements to provide this information.

3. The audit trail is necessary for the accountant to perform a valid audit.

4. It is a means of updating master records in a file reconstruction procedure.

Before establishing an audit trail, the length of time that the detail documents are to be retained must be determined. This will be based upon:

1. Legal requirements.

2. The auditor's needs for annual or semiannual audits.

3. The operational requirements of the business.

4. The operational requirements of the data processing department.

The degree of detail required for any one of these may vary over a long period of time, and the source document, depending on the length of time it is required, may remain intact or be microfilmed for condensed storage. Because of storage expense, cost of tapes, maintenance, etc., management should try to condense or summarize the necessary data as much as possible.

There are various ways to establish a good audit trail for data processing systems having direct access storage. In the disucssion that follows, the availability of tape is a basic assumption. It does not preclude the use of cards or other files to accomplish the same results.

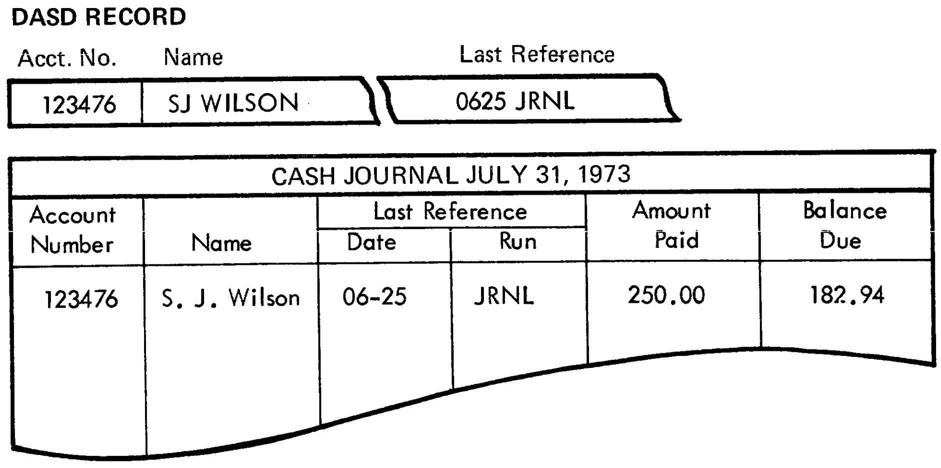
The one basic method of creating an audit trail is through a file dump. By reading the file and writing it on tape, a correct master file is always available as of a given point in time. To make it current, all transactions since the dump must be passed against it for updating. The dump to tape may be the entire file, only the portions used, or only the groups of records affected by a day's runs. In many cases, the speed of files and tapes makes it feasible to perform a file dump on a daily basis.

Where many master records are involved and the transaction volume is low, another method should be investigated. This approach requires a complete file dump less frequently. For example, if an inventory record is used today, a tag is placed in the record and its address is written out on tape. Each successive item going to that same record will find the tag present and *not* write the address. At the end of the day the address tape is used to read the corresponding file records. Each is dated and written on tape. When needed, these tape records are sorted and merged; the merged records, along with those on the master tape file, are read into the processor, where the record with the latest date is used in reconstructing the disk file. This approach has the advantage of taking less daily time than a full dump, and requires further processing only when it is necessary to reconstruct the file. File reconstruction will take longer when it occurs. The user should be cautioned against letting too much time elapse between complete file dumps; sorting and merging them can become quite time-consuming.

A way to create an audit trail when processing at random is by having a program "sign" each record that it updates. An example of this is shown in Figure 11.1 Each record contains a field for the date and source of the last update. As the field is changed, the previous reference can be printed. In the example shown, the reference field will be updated to 0731 CASH. If every update does not result in printed output, an additional field can be included which contains the number of times the record has been updated since the last printout. This information can be useful in tracing errors or unusual conditions.

In most applications there are transactions that require special handling and therefore cannot be processed with the others. A record of these must be kept to avoid creating gaps in control and audit procedures. Processing can be monitored by the stored program and these transactions handled as exceptions. The system can be programmed to notify the operator of them and expedite their handling. Such transactions are held in a pending file and accounted for until completed. Thus they are readily available when an out-of-balance condition occurs or when information about them is needed.

Additional discussion of controls as they are related to data processing systems is found in *Management Control of Electronic Data Processing* (F20-0006).



DASD RECORD

Acct. No.	Name	Last Reference
123476	SJ WILSON	0625 JRNL

CASH JOURNAL JULY 31, 1973

Account Number	Name	Last Reference		Amount Paid	Balance Due
		Date	Run		
123476	S. J. Wilson	06-25	JRNL	250.00	182.94

Figure 11.1 Audit Trail

Reconstruction Procedures

It is necessary to fully plan the type of action to be taken under all conditions that might arise which would prevent normal execution of data processing procedures. Each type of unit making up the system should be considered as nonoperational, and an alternate plan should be devised for each specific unit (as well as combinations of units) in order to continue processing in some manner. These plans should be devised and adhered to in all cases.

The need for reconstruction arises when information in the file is destroyed. Reconstruction methods used will vary depending on job priority, time considerations, processing time necessary to provide reconstruction data, etc.

The first requirement for a file reconstruction procedure is that the data in the file be dumped periodically. The dump can be made either to cards or tape (the latter is the basis for this discussion). The time required for the dump and the frequency with which it is done will vary. In cases where reports are prepared periodically, the file dump can probably be obtained as a by-product.

The feasibility of a daily file dump should be investigated as a starting point. With a daily dump, file reconstruction is greatly simplified in that the file as of yesterday can be loaded into the direct access storage device and today's transactions reprocessed. This approach can, of course, be used even though the file is not dumped every day. The deciding factor is whether another method might cost less or perhaps be more timely.

As the number of direct access storage modules increases, a daily dump of all modules will probably become less desirable unless an auxiliary processor is available.

If it is not feasible to reprocess all transactions that occur in the interval between file dumps, the approach outlined under "The Audit Trail" might be applicable. With it, as each record is updated in the file, the updated record was written on tape. When it becomes necessary to reconstruct a file, the latest status of each active record affected can be selected from this tape and merged with the previous dump tape to provide a current file status as of the last processing cycle. Current date should be included in the record to facilitate selection of the most current record. An advantage of this over-reprocessing is that program changes will have no effect, whereas they could cause different action to be taken if reprocessing were attempted.

It should be recognized that program storage is considered in the same manner as data storage. Each time a program change is made, it must be reflected on a tape record or some other medium to ensure retrieval capability, should reconstruction become necessary.

The method used for reconstruction should be well planned, well documented, thoroughly checked out, and then followed when reconstruction is necessary.

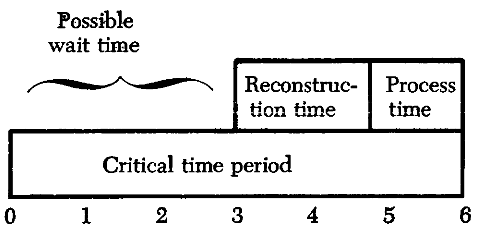
Bypass Procedures

In the event of machine nonavailability during critical time periods, a method of alternate processing must be designed to allow the major portion or most critical portion of the job to continue.

If the computer produces output that governs the immediate action of another part of the operation (stockpicking, for example), the decision may be made to institute the prescribed bypass procedure immediately upon encountering an unusual condition in order to keep this function operational. On the other hand, applications that have a critical period once a month might be able to wait a considerable longer period of time before a bypass operation is begun.

Probably one of the most difficult decisions to make in a multifile application is when to go into a bypass mode of operation. One way to determine the amount of time that can elapse is shown below:

Critical time period:		6 hrs.
Reconstruction time:	2 hrs.	
Process time:	1 hr.	
		–3 hrs.
Total wait time possible:		3 hrs.



The critical time period consists of that time which can elapse without disrupting another operation.

The possibility that each unit in the machine configuration, as well as combinations of units, may be unavailable must be considered in order to establish adequate bypass procedures. One DASD may contain an index to the files on other DASD's, making normal processing impossible when it is inoperative. In such a case, partial processing may be accomplished by dumping the contents of another direct access storage unit and loading the index in its place.

Another approach that might be considered is to have duplicate critical content files, or to dump only these files daily to minimize reconstruction time.

Every application should be desinged to maintain at least a partial processing capability as long as possible before initiating a bypass operation.

Some applications demand assurance that downtime be virtually impossible, and go so far as to require duplexed processors.

For applications requiring fast response, one approach to maintaining operational status when the system is down is to utilize the previous dump tapes and daily action tapes to create a printout which could then be used in a manual operation. Such is the case when there are priority transactions that must be handled immediately. The file contents are printed by using the last dump tape, merging in the action tapes and providing a printout which can then be utilized by clerks to process the priority transactions manually. The results of the clerical processing are fed back to the computer when operational status is restored and post-posting updates the file to reflect the manual action taken.

If once-a-day processing is adequate, the tapes created above could be used to perform tape processing of the application. In considering a tape bypass operation, tape unit availability must be ensured.

In cases where an alternate processor is available, its use for bypass should be considered.

The major factor for satisfactory bypass operation is to have a definite procedure.

Restart Procedures

The theory behind restart is that if for any reason operation is interrupted, there is a time advantage in being able to resume processing without having to start at the beginning of the run.

To accomplish this, a system of checkpoints must be developed whereby the contents of memory are dumped at specified intervals. In many cases a checkpoint occurs at the end of an input or output tape.

The checkpoint routine will dump not only the contents of memory, but also the contents of accumulators and registers, indicators, and input and output records in process.

When a restart is initiated in a tape system, the tapes are repositioned, the contents of memory, registers, etc., are reestablished, and processing then continues.

When direct access media are employed, a new consideration arises, since each updated record has destroyed the prior status of the records. In order to restart, the file must be reestablished as of the last checkpoint. This can be done by dumping an image of the direct access record on tape as soon as it has been read. When a restart is initiated, these records can be used to rewrite the file and establish the status that existed when the corresponding check-point was taken. When this is completed, normal restart procedures can be accomplished and reprocessing begun.

Bibliography

IBM 3830 Storage Control – 3330 Disk Storage Reference Manual (GA26-1592)

IBM 2820 Storage Control – 2301 Drum Storage Component Description (GA22-6895)

IBM 2841 Storage Control – 2311 Disk Storage Drive – 2321 Data Cell Drive – 2303 Drum Storage Component Description (GA26-5988)

IBM 2835 Storage Control – 2305 Fixed Head Storage Facility Component Summary (GA26-1589)

IBM 2314 Direct Access Storage Facility – 2844 Auxiliary Storage Control (GA26-3599)

IBM 2319 Disk Storage Component Description (GA26-1606) and is also described in (GA26-3599)

Introduction to IBM 3850 Mass Storage System (GA32-0028)

OS/VS Mass Storage System Planning Guide (GC35-0011)

REFERENCE CARDS

IBM 2301 Drum Storage	(X20-1717)
IBM 2303 Drum Storage	(X20-1718)
IBM 2311 Disk Storage	(X20-1705)
IBM 2314 Disk Storage	(X20-1710)
IBM 2321 Data Cell Drive	(X20-1704)

These reference cards contain the capacity formulas, a table of bytes per record depending on records per track (down to a data length of five bytes), and a table of transmission time depending on record length.

Additional information can be found in the following references:

System/360 Principles of Operation (GA22-6821)
System/370 Principles of Operation (GA22-7000)
System/370 System Summary (GA22-7001)
Data Management Services Guide (GC26-3746)
Data Management Macro Instructions (GC26-3794)
Supervisor Services and Macro Instructions (GC28-6646)

Direct Access Device Space Management (GY28-6607)
Index Sequential Access Methods Program Logic Manual (GY28-6618)
Sequential Access Method Program Logic (GY28-6604)
Basic Direct Access Method Program Logic (GY28-6617)
A Guide to the IBM System/370 Model 145 (GC20-1734)
A Guide to the IBM System/370 Model 155 (GC20-1729)
A Guide to the IBM System/370 Model 165 (GC20-1730)
DOS Supervisor and I/O Macros (GC24-5037)
VSAM Programmer's Guide (GC26-3838)
VSAM Access Method Services (GC26-3843)
OS/VS Data Management Services (GC26-3786)
OS/VS Data Management Macro Instructions (GC26-3793)
OS/VS Supervisor Services and Macro Instructions (GC27-6979)
OSVS DADSM Logic (SY26-3787)
OS/VS ISAM Logic (SY26-3786)
VS SAM Logic (SY26-3788)
VS BDAM Logic (SY26-3789)

READER'S COMMENT FORM

Introduction to IBM Direct-Access Storage Devices and Organization Methods — GC20-1649-7

Please comment on the usefulness and readability of this publication; suggest additions and deletions, and list specific errors and omissions (give page numbers). All comments and suggestions become the property of IBM.

COMMENTS

Reply Necessary

Yes ☐

No ☐

Name ____________________

Job Title ____________________

Address ____________________

____________________ Zip ____________

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

FOLD ON TWO LINES, STAPLE AND MAIL

YOUR COMMENTS PLEASE. . . . . .

Your comments on the other side of this form will help us improve future editions of this publication. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material.

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

FOLD FOLD

FIRST CLASS
PERMIT NO. 40
ARMONK, NEW YORK

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY:

IBM Education Center, Building 005
Department 78L, Publications Services
South Road
Poughkeepsie, New York 12602

FOLD FOLD

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)

GC20-1649-9

Due to an error this number was printed as -7 it should be -9

Copyright Page should read Major Revision (Dec. 1975) Replaces GC20-1649-8

Introduction to IBM Direct-Access Storage Devices and Organization Methods Printed in U.S.A. GC20-1649-7