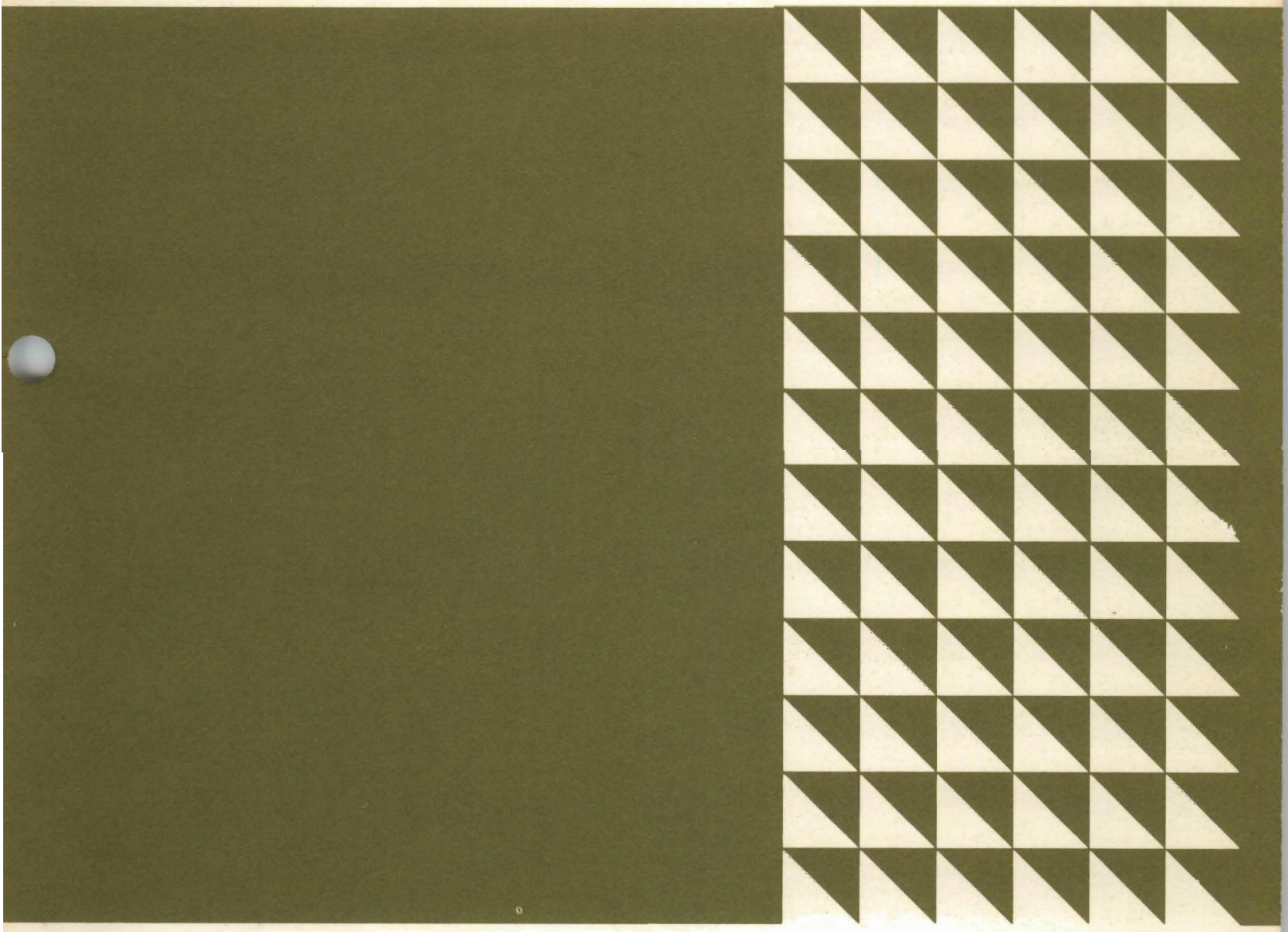
IBM

Introduction to IBM Direct-Access Storage Devices and Organization Methods



Student Text

Introduction to IBM Direct-Access Storage Devices and Organization Methods

Student Text

Preface

This text discusses the physical characteristics and capacities of the following Direct Access Storage Devices available for System/360 Models 25, 30, 40, 50, 65, 67, 75, and 85:

2301 Drum Storage
2302 Disk Storage
2303 Drum Storage
2311 Disk Storage Drive
2314 Direct Access Storage Facility
2321 Data Cell Drive

The file organization methods and access methods provided for these devices by the IBM System/360 Operating System, the IBM System/360 Disk Operating System, and the IBM System/360 Basic Operating System are also discussed. The uses of direct access storage, basic terminology, and the establishment of controls for a direct access system are other topics addressed by this text. Most of the chapters end with student exercises, answers to which may be found at the end of the manual.

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

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

2305 Fixed Head Storage
Available for a System/360 Models 85, 195 or a System/370 Models 145, 155, 165, or 195.

3330 Disk Storage
Available for a System/360 Models 85, 195 or a System/370 Models 135 through 195.

Reference material for these devices is listed in the Bibliography.

Minor Revision (December 1971)
This edition, GC20-1649-5, is a minor revision of the previous edition, GC20-1649-4. The significant changes in this and previous editions are indicated by:

1. A vertical line to the left of the affected text where only part of a page is changed.

2. A bullet (•) to the left of the title of each figure that has been changed.

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.

Contents

Chapter 1: Introduction

This chapter presents System/360 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.

Terminology

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 rather than 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. System/360 Operating System, one of the programming systems available for System/360, 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.

Byte. A byte is the smallest addressable unit of information on the System/360. It is composed of eight data bits and a parity bit. A bit means a binary digit — that is, one that can represent two conditions (0 or 1). A punching position in a card is similar to a bit; it is either punched or not punched.

The bits in a byte can represent several different things, depending on how they are interpreted:

- Zoned decimal format: One byte represents one character. The bit "values" of the eight data bits in a byte are ZZZZ8421. Various combinations of the zone (Z) and numeric bits make it possible to represent 256 different characters. In numeric fields, the sign of the field is represented by the zone bits of the low-order byte. The field +1234 in zoned decimal requires four bytes: | Z1 | Z2 | Z3 | +4 |
- Packed decimal format: This is efficient for decimal numeric information because only four bits are required to represent any decimal digit. In packed decimal format, one byte represents two decimal digits. The bit "values" are 84218421. The four low-order bits of the low-order byte of a packed decimal field always contains the sign of the field. The field +1234 in packed decimal requires three bytes: | 01 | 23 | 4+ |. Numeric information must be in this format for decimal arithmetic operations and for editing operations (suppression of high-order zeros, insertion of commas, decimal points, etc.).
- Binary format: With this format, a byte represents a binary (numbering system with a base of 2) value rather than a decimal (base of 10) value. The only knowledge about the binary numbering system required for this text is that one byte represents a maximum decimal value of $2^8 - 1$ or 255 and two bytes represent a maximum decimal value of $2^{16} - 1$ or 65,535.

Any combination of these formats is permissible either internally in core storage or externally on DASD, although binary fields are usually grouped together in a logical record.

A byte in core storage actually consists of nine bits: the eight data bits (discussed above) and a parity bit that is used to check the validity of data transfer. A byte must always have an odd number of 1-bits. The parity bit is set to 0 or 1 as required to establish odd parity. A parity bit for each byte is not used in DASD recording. Instead, another method of validity checking is used (see "Checking" in Chapter 3).

Uses of Direct Access Storage

Inline 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 "inline", 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 inline 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 inline provides solutions to systems problems for which previous solutions were impractical.

As an example of an inline 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 inline, 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 inline, it should not be inferred that inline processing is a unique requirement of inventory applications, or that the inline 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 information 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 and response. In demand deposit accounting, the question might arise: "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 proce-

dures 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 code, the data would be sorted into 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 main core storage can be reduced because only the optimum number of program steps for processing data need be in core 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 core 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 core 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 core 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 random 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 core storage positions.) 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 primary core storage is exceeded may additional processing time be required for further transfers of subroutines.

Three of the four operating systems available for System/360 require that a DASD be used for program residence. In these cases, IBM programs (such as Assembler or RPG language translators), user programs, and IBM control programs (such as routines to handle interrupts and perform job-to-job transition) must be on DASD.

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 several 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 accumulated 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 continu-

ous 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.

Sorting with a Direct Access Storage System

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.

Chapter 2: System/360 Direct Access Storage Devices

Several DASD's are available for System/360 Models 30, 40, 44, 50, 65, 67, 75, and 85, and, unless noted otherwise under "Physical Description", each one can be used on any of the above models of System/360. The devices by type and number are:

Drive with removable disk packs: 2311 and 2314
Drive with nonremovable disks: 2302
Drum: 2301 and 2303
Data cell drive: 2321

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).

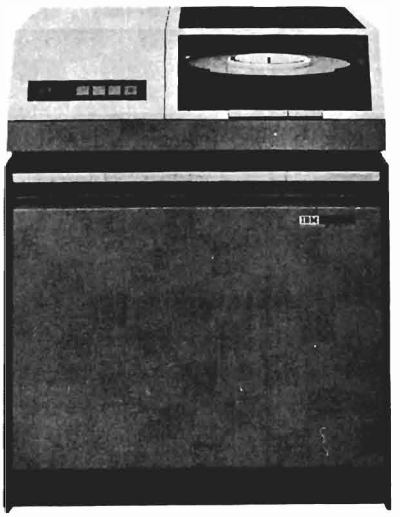


Figure 1. 2311 Disk Storage Drive

Physical Description

2311 Disk Storage Drive (see Figure 1). The 2311 is a drive with removable disk packs. It can be used with System/360 Models 25, 30, 40, 44, 50, 65, 67, 75, and 85. The packs, when removed from the drive, are enclosed in protective covers. Each pack consists of six 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 ten recording surfaces.

2314 Direct Access Storage Facility (see Figure 2). The 2314 consists of five or nine drives, depending upon the model, and a control unit. All five or any eight of the nine drives can be online at a time. The ninth drive is available for backup if one of the other drives requires servicing or maintenance. The 2314 can be used with System/360 Models 30, 40, 50,

Figure 2. 2314 Direct Access Storage Facility

65, 67, 75, and 85. This device uses removable disk packs similar to those on the 2311. The packs are larger, however, each consisting of 11 disks with 20 of the surfaces used for recording.

2302 Disk Storage (see Figure 3). The 2302 is a drive with nonremovable disks. It can be used with System/360 Models 30, 40, 50, 65, or 75. It is available in two models. Model 3 contains one module of disks; Model 4 contains two modules, one mounted above the other. Each module consists of 25 disks, similar to those in a 2311 pack, but 25 inches in diameter. In each module, 46 of the surfaces are used for recording.

2303 Drum Storage. The 2303 is a vertically mounted drum. Data is recorded on its outer surface. It can be used with System/360 Models 40, 50, 65, 67, 75, and 85.

2301 Drum Storage. The 2301 is a drum similar in appearance to the 2303. It can be used with System/360 Models 65, 67, 75, and 85.

2321 Data Cell Drive (see Figure 4). The 2321 is a drive with removable data cells. It can be used with System/360 Models 30, 40, 50, 65, 67, and 75. The cells, when removed from the drive, are enclosed in protective covers. Each cell is divided into 20 subcells (see Figure 5). Each subcell contains ten strips, which are the recording medium of this device. The strips are made of tape with a magnetic oxide coating on one side. Each strip is 2¼ inches wide, 13 inches long, and about three times as thick as magnetic tape. The strips are held in place by the bottom of the cell and by the adjacent strips.

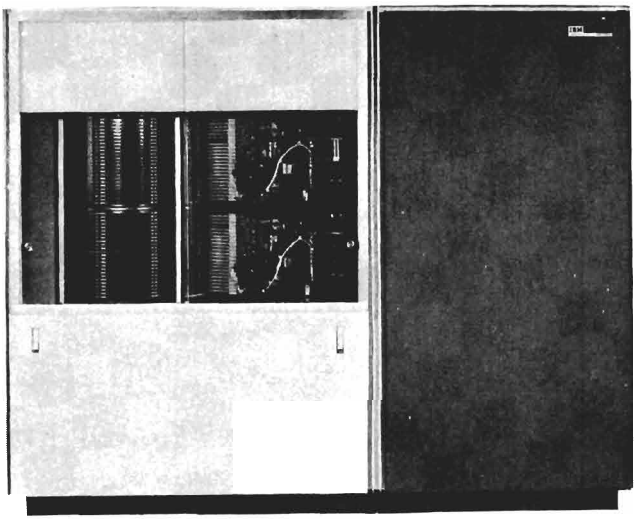


Figure 3. 2302 Disk Storage Drive

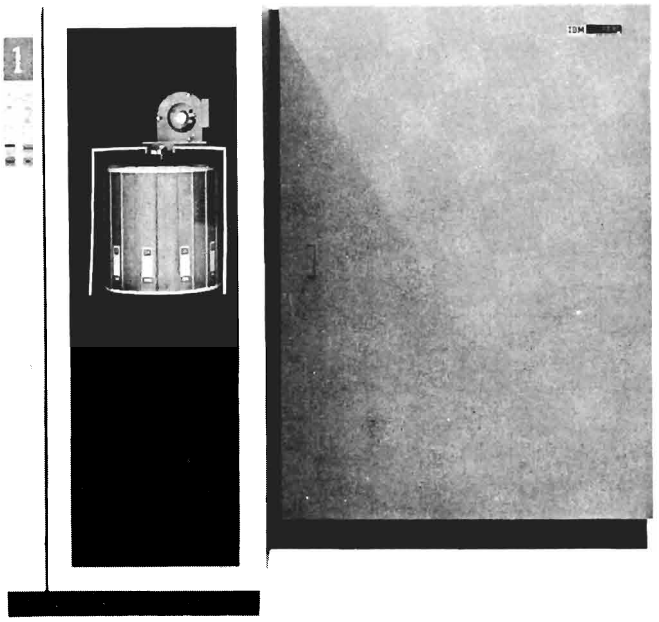


Figure 4. 2321 Data Cell Drive

A complete drive contains 2000 strips or recording surfaces. When the drive is in operation, ten cells must be mounted. Some of these may be ballast cells that do not contain strips; their purpose is to balance the drive properly if the full recording capacity is not required.

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. This definition applies to the 2321 Data Cell, as well as to the other devices, since the strip being processed is wrapped around a drum. The tracks are concentric, not a spiral like a phonograph record.

Except for the 2301 drum, 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 6. 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.

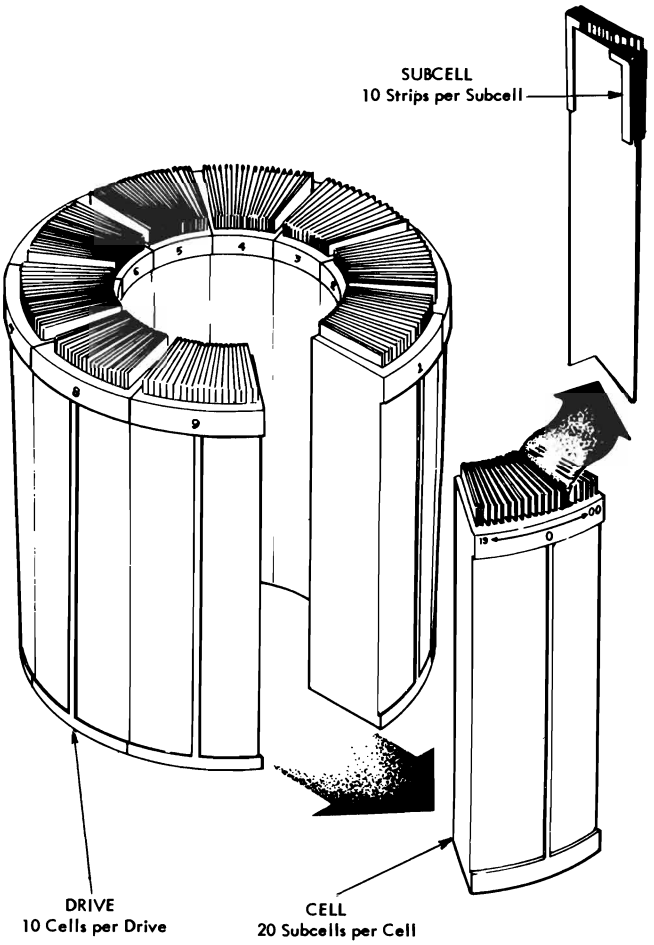
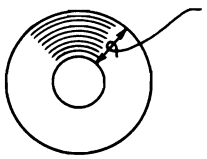


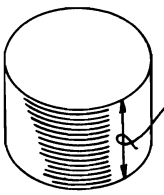
Figure 5. 2321 drive, cell, subcell



2311 Disk Drive: 200 tracks per surface (plus 3 alternates); 3625 bytes per track.

2314 Storage Facility: 200 tracks per surface (plus 3 alternates); 7294 bytes per track.

2302 Disk Drive: 492 tracks per surface (plus 8 alternates); 4984 bytes per track.



2303 Drum: 800 addressable tracks; 4892 bytes per track.

2301 Drum: 200 addressable tracks; 20483 bytes per track.

2321 Data Cell Drive: 100 tracks per strip. 2000 bytes per track. The last four strips of each cell (400 tracks per cell) are reserved as alternates.

Figure 6. 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.

2311 Disk Drive (see Figure 7). 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 203 different positions, thus giving access to all the tracks. Each arm has two read/write heads. There are ten heads in all — one for each recording surface.

2314 Storage Facility. Each drive of the 2314 has a comb-type mechanism like the 2311. Each mechanism has 20 read/write heads — one for each recording surface.

2302 Disk Storage (see Figure 8). Each module of the 2302 has two comb-type mechanisms; one services the inner 250 tracks of each surface, the other services the outer 250 tracks. Each mechanism has 46 read/write heads — one for each recording surface. Model 4, since it consists of two modules, has four access mechanisms. The two (or four) access mechanisms of each 2302 move independently of each other.

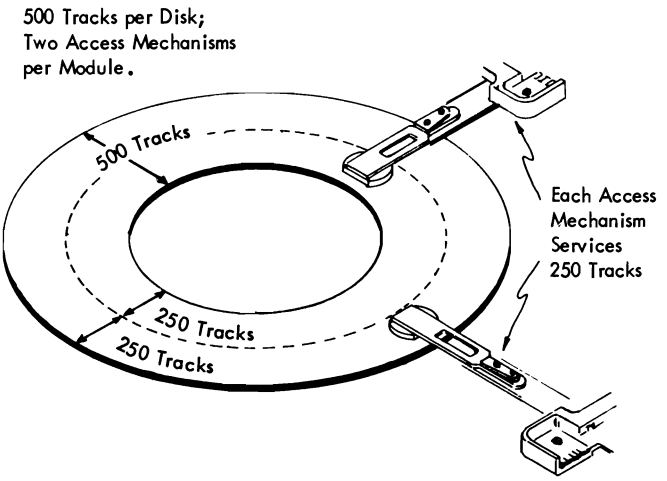


Figure 8. 2302 access mechanism

2303 Drum. The 2303 access mechanism includes a read/write head for each addressable and spare track. The heads are fixed in position on several vertical racks that surround the drum. Data is transferred to or from the drum as the single recording surface rotates past the fixed heads.

2301 Drum. The 2301 access mechanism is as described for the 2303 except that there is a set of four read/write heads for each of the addressable and spare tracks, since data is recorded in parallel groups of four bits.

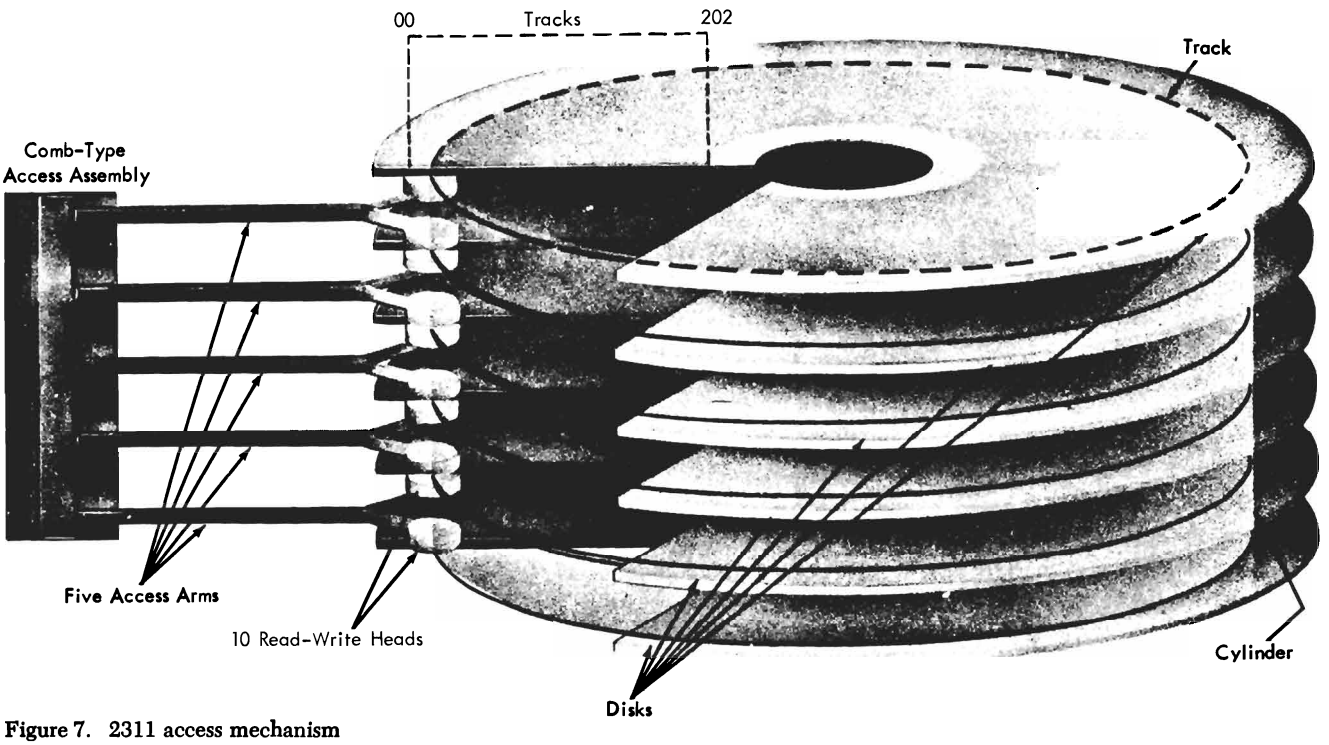


Figure 7. 2311 access mechanism

2321 Data Cell Drive. The array of cells is rotated until the subcell containing the strip to be processed is under a drum that is fixed in position above the array. Each strip has a hole near the top and two index tabs that are in a different position for each of the ten strips in a subcell. The selected strip is isolated as separation fingers push back the index tabs of the adjacent strips (see Figure 9a). As the drum rotates, the pickup head drops down and latches through the hole in the top of the strip (see Figure 9b). As the drum continues to rotate, the strip is withdrawn from the subcell and wrapped around the drum (see Figure 9c and 9d).

As the drum continues rotating, the strip moves past a bar of 20 read/write heads. The heads are positioned at each fifth track of the strip. The bar of heads can move horizontally to five different positions, thus providing access to all 100 tracks of a strip. There is no actual physical contact of head and strip, since there is an air cushion between them.

The strip remains attached to the rotating drum until another strip is selected, a command is given to restore it, or 800 milliseconds (ms) have elapsed since the completion of the last command or series of commands to that 2321. The strip is restored by reversing the direction of rotation of the drum. The strip drops back into its subcell position between the separated adjacent strips.

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. In a System/360 DASD, a large amount of data can be stored in a single cylinder, thus minimizing the movements of the access mechanism. Using the 2311 as an example, physically the pack consists of ten separate horizontal recording surfaces, while from an access point of view it consists of 203 separate vertical cylinders of ten tracks each (see Figure 10).

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.

2311 Disk Drive. 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 ten tracks, which is equal to the number of recording surfaces. A cylinder has a maximum capacity of 36,250 data bytes (3625 bytes per track, 10 tracks per cylinder). A pack has a maximum capacity of 7.25 million data bytes (36,250 bytes per cylinder, 200 cylinders per pack).

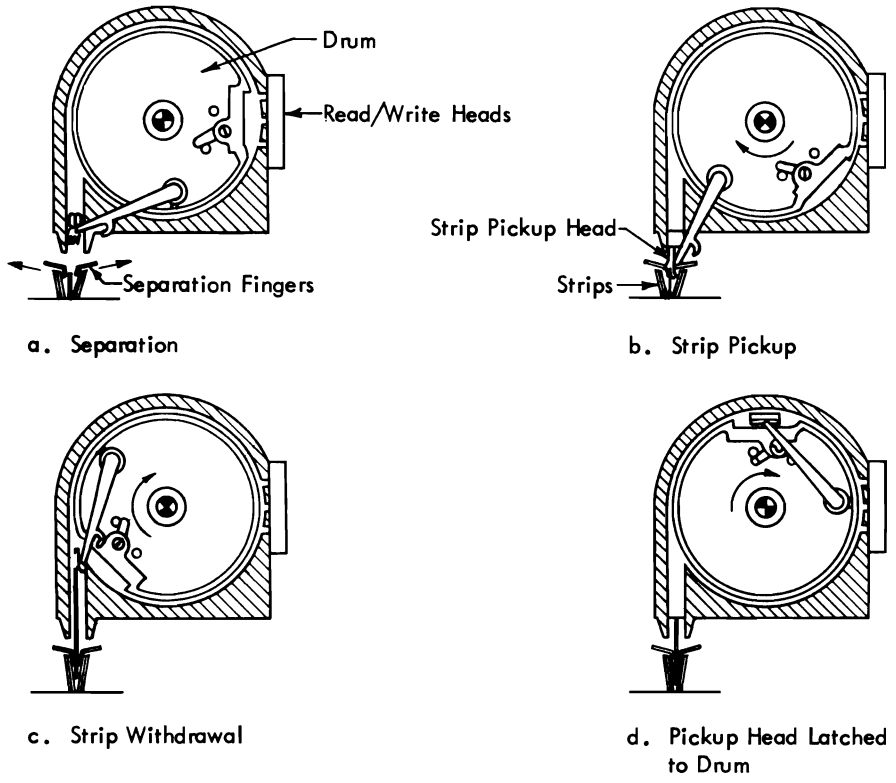


Figure 9. 2321 strip pickup cycle

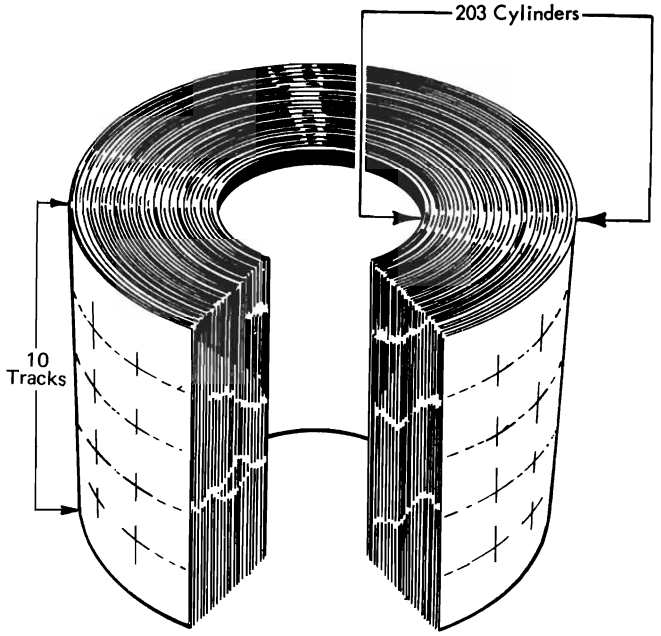


Figure 10. 2311 cylinders

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 233.408 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.

2302 Disk. Each module has 492 cylinders (plus eight alternates), which is equal to the sum of the number of positions to which each access mechanism can move. (Each access mechanism can address 250 cylinders.) Model 4, which consists of two modules, has 984 cylinders (plus 16 alternates). Each cylinder has 46 tracks, which is equal to the number of recording surfaces. A cylinder has a maximum capacity of 229,264 data bytes (4984 bytes per track, 46 tracks per cylinder). A 2302 Model 3 has a maximum capacity of 112.79 million data bytes. A 2302 Model 4 has a maximum capacity of 225.59 million data bytes.

2303 Drum. The cylinder concept does not fully apply to drums. Since the access mechanism is fixed in position, all of the data on the drum is accessible at all times. However, the 800 addressable tracks are divided electronically for addressing purposes into 80 cylinders of 10 tracks each. A cylinder has a maximum capacity of 48,920 data bytes (4892 bytes per track, 10 tracks per cylinder). A 2303 has a maximum capacity of about 3.9 million data bytes. Spare tracks are provided to ensure that each recorded bit can be stored in a magnetically perfect medium. If a defect is encountered on a track, the entire track is disabled by the customer engineer, and one of the spare tracks is substituted. This spare track is given the address of the original disabled track.

2301 Drum. The entire drum is considered as one cylinder consisting of 200 addressable tracks. Each track has a maximum capacity of 20,483 bytes. The 2301 has a maximum capacity of 4.09 million data bytes. Spare tracks are provided to ensure that each recorded bit can be stored in a magnetically perfect medium. If a defect is encountered on a track, the entire track is disabled by the customer engineer, and one of the spare tracks is substituted. This spare track is given the address of the original disabled track.

2321 Data Cell Drive. Each strip contains five cylinders, which is equal to the number of positions to which the bar of read/write heads can move. Since a drive can hold 2000 strips, the entire array contains 10,000 cylinders. Each cylinder has 20 tracks, which is equal to the number of read/write heads. A cylinder has a maximum capacity of 40,000 data bytes (2000 bytes per track, 20 tracks per cylinder). A strip has a maximum capacity of 200,000 data bytes. A complete array has a maximum capacity of 400 million data bytes. (This includes 8 million bytes per array — 400 tracks in each data cell — reserved as alternates.)

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.

- 2311 Disk Drive: As shown in Figure 11, 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 milliseconds (ms); the maximum is 135 ms; the average over the entire pack is 75 ms.
- 2314 Storage Facility: The average access motion time is 20% faster than the 2311. The minimum is 25 ms, maximum 130, and average 60.
- 2302 Disk: The 250 cylinders of each access zone are divided into areas of 60-40-60-40-50 cylinders (see Figure 12). Each area is further divided into groups of 10 cylinders each. Movement within a group takes 50 ms. Movement to a different group or to a different area takes 120 ms or 180 ms, respectively. The minimum is 50 ms; the maximum is 180 ms; the average over an entire access zone is 165 ms.
- 2303 Drum: None, since the access mechanism does not move.
- 2301 Drum: None, since the access mechanism does not move.
- 2321 Data Cell: Access time may include restoring the strip already on the drum and/or picking a new one. As shown in Figure 13, it takes 200 ms to restore a strip, from 75 to 225 ms to rotate the array, and 175 ms to pick a strip. The movement of the bar of read/write heads takes 95 ms; this is overlapped with the restore or pick if either occurs. The minimum access time is 95 ms; the correct strip is already on the drum and only head bar motion is required. The maximum is 600 ms: 200 ms to restore, 225 ms to rotate five cells (the array rotates in both directions, so this is the maximum), and 175 ms to pick. The average for the entire array (assuming that the previous strip has already been restored) is 350 ms: 175 ms to rotate 2½ cells and 175 ms to pick.

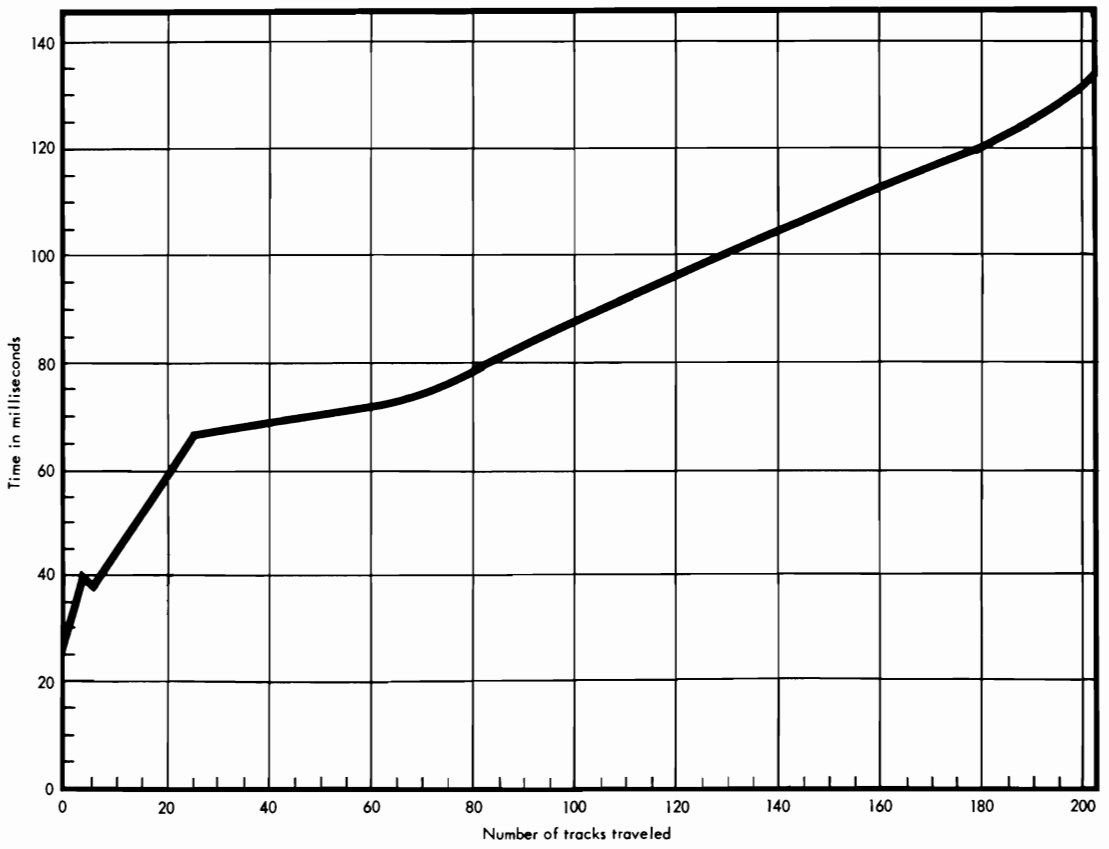


Figure 11. 2311 access time

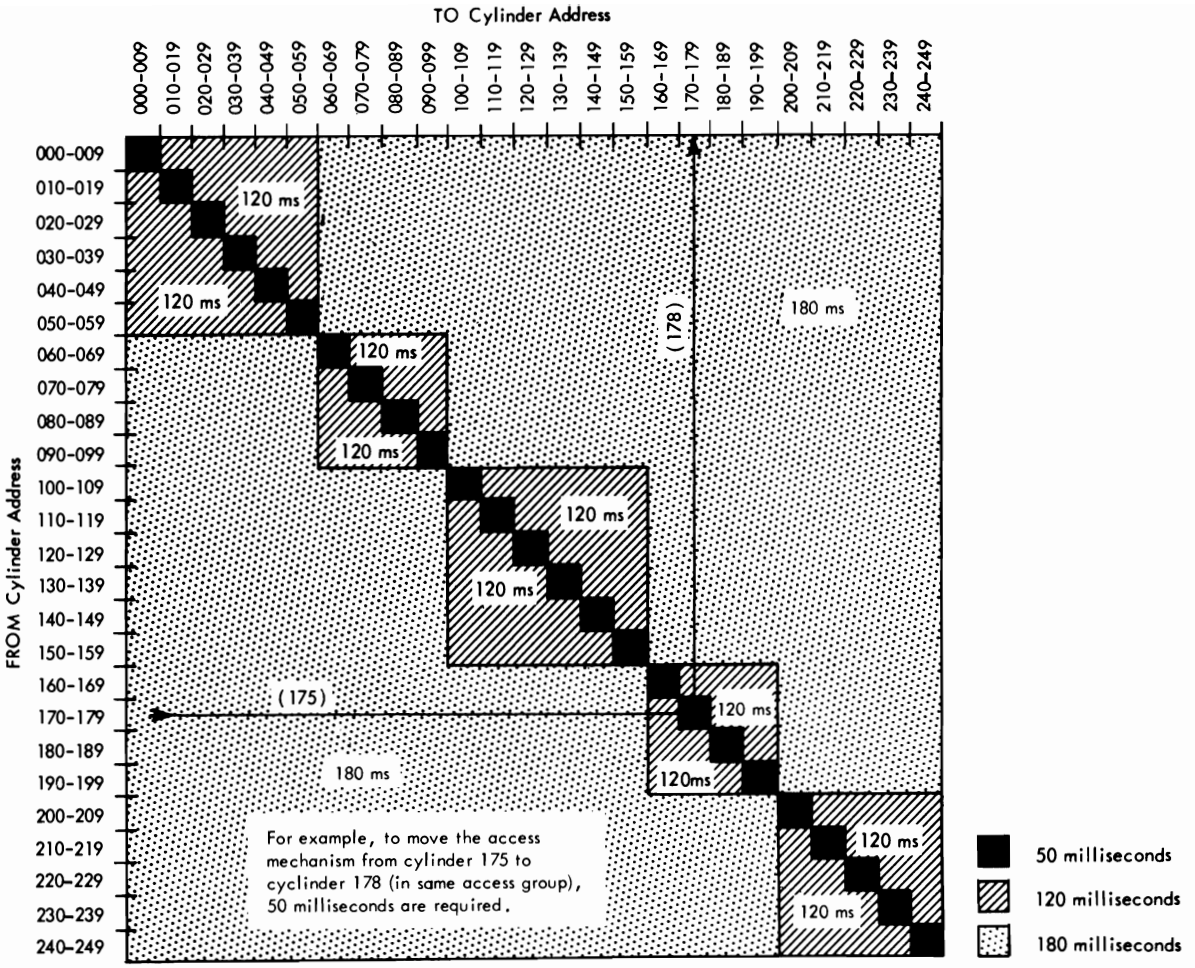


Figure 12. 2302 access time

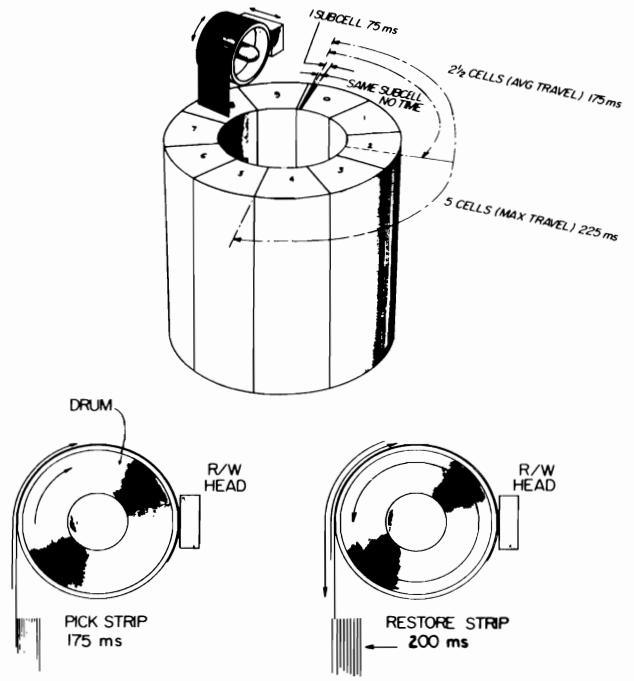


Figure 13. 2321 access time

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*
2311 disk drive	25 ms	12.5 ms
2314 storage facility	25 ms	12.5 ms
2302 disk	34 ms	17 ms
2303 drum	17.5 ms	8.6 ms
2301 drum	17.5 ms	8.6 ms
2321 data cell	50 ms	25 ms

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*
2311 disk drive	156 KB	0.0064103
2314 storage facility	312 KB	0.0032051
2302 disk	156 KB	0.0064103
2303 drum	303.8 KB	0.0032916
2301 drum	1200 KB	0.0008333
2321 data cell	55 KB	0.0181818

*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. In the practice exercises in this text, a simplified approach has been used to reduce the number of computations required to complete the timing exercises. This approach allows a full rotation for each read (or write) to include both rotational delay and data transfer. 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.

Exercises

It is not expected that the details presented in this text be memorized; refer to the text as necessary, therefore, to complete the following exercises. The answers are listed at the end of the text.

1. Complete the table of DASD characeristics below. Do not include any alternate recording areas in the capacities.

Device	Storage Medium	Cylinders	Tracks Per Cylinder	Bytes per			Access Motion (ms)			Rotation (ms) (Full)	Transfer Rate (KB)
				Track	Cylinder	Device (Million)	Min.	Max.	Avg.		
2311											
2314		Pack: Model A1 Total: Model A2 Total:				Pack: Model A1 Total: Model A2 Total:					
2302		Model 3: Model 4:				Model 3: Model 4:					
2303											
2301											
2321		Strip: Array:									

2. What accounts for the fact that the transfer rate of the 2314 is twice that of the 2311?

3. What unique characteristic of the 2301 partially accounts for the extremely high transfer rate?

4. Which of the direct access devices provide offline storage?

5. In designing a file, it has been determined that nine cylinders are required.

   a. If the device is a _____, a _____, or a _____, it will make no difference which nine adjacent cylinders are used, as far as access motion time is concerned.

   b. If the device is a _____, the nine cylinders should be within the same _____ _____; if the device is a _____, the nine cylinders should be within the same _____ in order to minimize access time.

6. With the 2302, state the access motion time between the following cylinders:

   a. 000 to 059 _____
   b. 059 to 060 _____
   c. 050 to 059 _____

7. In designing a file, it has been decided that there will be 10,000 records and that each record will be 200 bytes long. Given the number of records per track stated below, calculate how many tracks and cylinders will be needed on each device.

   a. 2311: 13 records/track, _____ tracks, _____ cylinders
   b. 2314: 23 records/track, _____ tracks, _____ cylinders
   c. 2302: 18 records/track, _____ tracks, _____ cylinders
   d. 2303: 16 records/track, _____ tracks, _____ cylinders
   e. 2301: 61 records/track, _____ tracks
   f. 2321: 7 records/track, _____ tracks, _____ cylinders, _____ strips

8. Approximately what percentage of the total number of cylinders is required for the files in problem 7?

a. 2311:_____ d. 2303:_____
b. 2314-A1: _____ e. 2301:_____
c. 2302-3: _____ f. 2321:_____

9. Assume that the files in problem 7 have been recorded starting at cylinder 0, track 0 (and for the 2321, strip 0 of a subcell). If the entire file is to be processed in physical record sequence, calculate the access motion time required for the job. Do not include the time to access the first record.

a. 2311:_____ms d. 2303:_____ms
b. 2314:_____ms e. 2301:_____ms
c. 2302:_____ms f. 2321:_____ms

10. As shown in problem 9, access motion time is negligible if a file is being processed consecutively. The significant time is rotational delay and data transfer. Allowing a full rotation per record, reading the entire file of 10,000 records would take approximately:

a. 2311:_____min. d. 2303:_____min.
b. 2314:_____min. e. 2301:_____min.
c. 2302:_____min. f. 2321:_____min.

11. Given the following layout of a record, indicate whether you would select packed decimal or zoned decimal format for each of the fields, and the number of bytes each will require.

	Characters	Format	Bytes
a. Employee number	6		
b. Name	18		
c. Number of dependents	2		
d. Social Security number	9		
e. Salary	6		
f. YTD gross earnings	7		
g. YTD withholding tax	6		
h. YTD FICA	5		
Total	59		

Chapter 3: DASD Control Units

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.

There are actually three control units for the different devices. As already mentioned, the 2314 storage facility has a self-contained control unit. The control unit for the 2301 drum is the 2820. It can control up to four 2301s.

The control unit for the other four devices is the 2841. The general rule for the 2841 is that it can control up to eight access mechanisms of varying types in any combination. Exceptions to that rule exist for the attachment of 2303s, and of both 2303s and 2311s to the same 2841.

A maximum of two 2303s may be installed on any 2841. Since both the 2303 and the 2311 use the 2841 as a source of power, the number of 2311s that can be attached to a 2841 in combination with 2303s is restricted. If one 2303 is attached, a maximum of three 2311s may also be attached. If two 2303s are attached, no 2311s may be attached. The rules for attaching both 2311s and 2303s to a single 2841 do not restrict, however, the ability of the 2841 to handle eight access mechanisms. Thus, if one 2303 and three 2311s were attached (a total of four access mechanisms), four more access mechanisms that are not 2311s or 2303s could also be attached – for instance, 2321s. If two 2303s were attached (a total of two access mechanisms), devices totaling six additional access mechanisms could also be attached.

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 (where it is parallel-by-byte) and the devices (where, except in the case of the 2301, it is serial-by-bit).

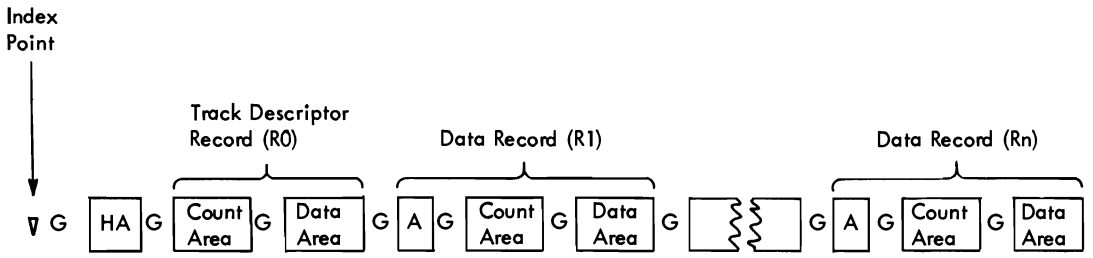
Checking

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.

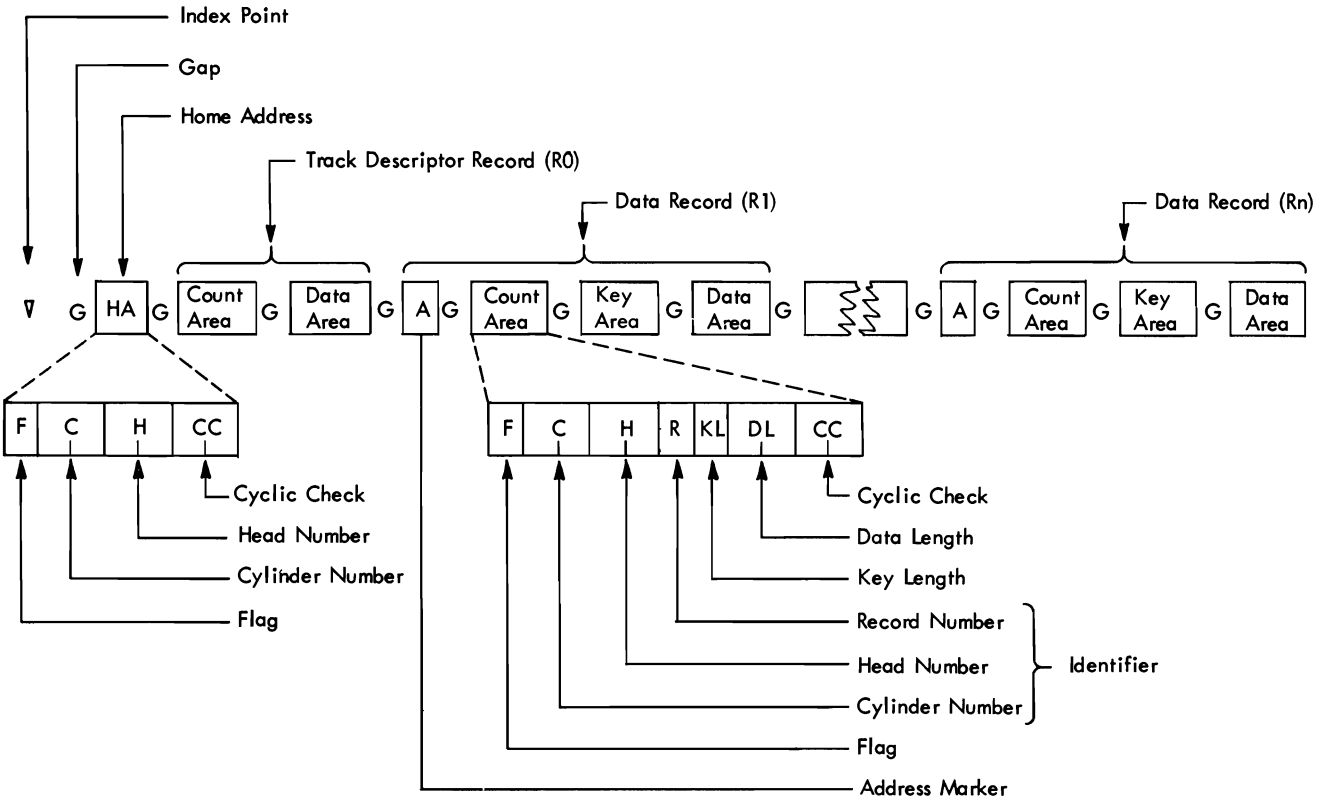
There are two advantages to this method of checking. It detects more errors than can be checked with a parity check. It also saves storage space on the devices; checking requires 16 bits per data area rather than one bit per byte.

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 areas) as well as data information (see Figure 14).



A. Count-data format



B. Count-key-data format

Figure 14. 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 14B, 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 14. 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 14, 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 14A), 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 14B) 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.

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 15 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 15. Cross-referencing between original track and alternate track via R0

Record Formats

When using IBM programming systems, logical records may be in one of five formats, as shown in Figure 16. 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.

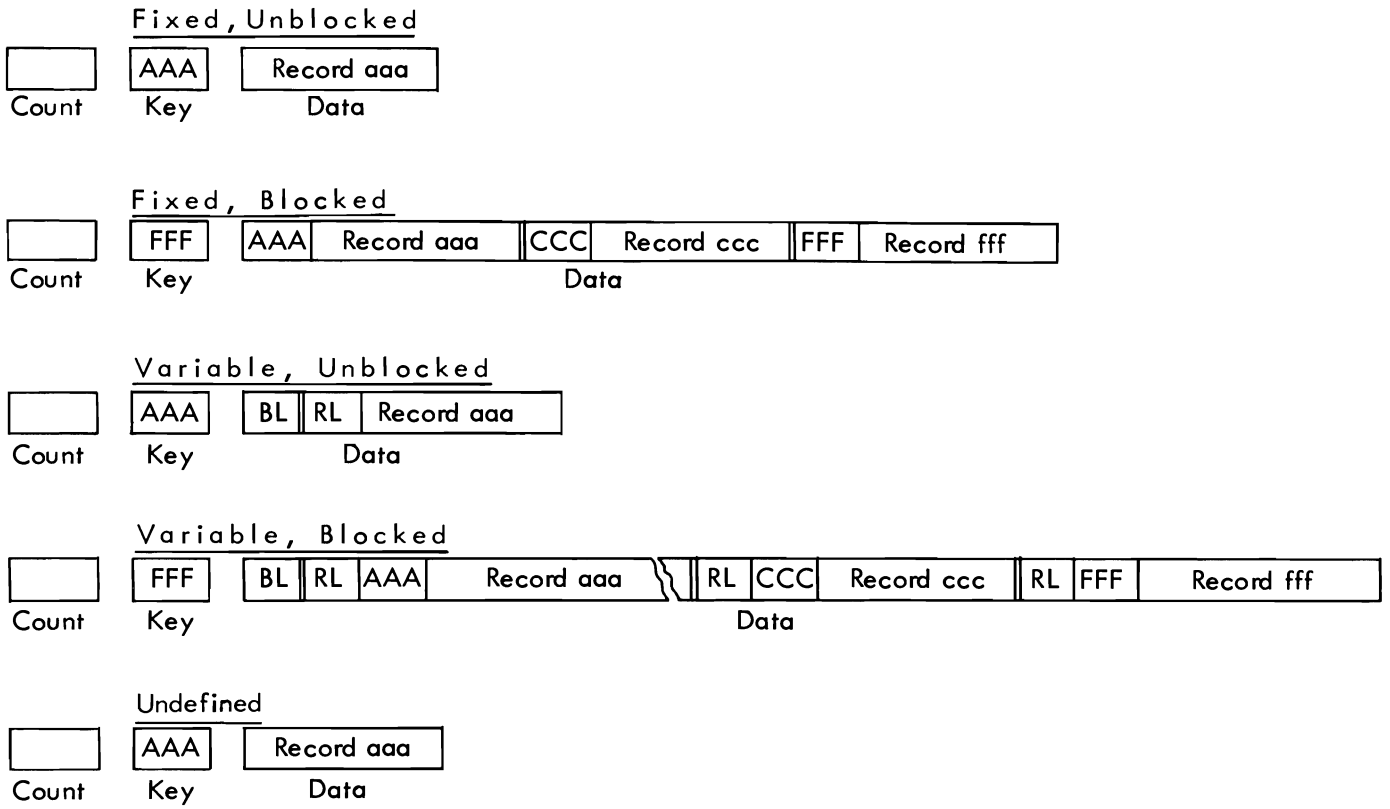


Figure 16. Record formats

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) is the first four bytes of the block; it indicates the number of bytes in the block including itself. RL (record length) is the first four bytes of the logical record; it 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 below, 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 17 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 2311, records are unblocked and formatted without keys, and data length is 120 bytes. There will be 19 records per track.
- Device is the 2321, records are unblocked and formatted with keys, data length is 100 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 — in this example, 108. There will be 9 records per track.
- Device is the 2302, records are blocked and formatted without keys, blocking factor is 3, and logical record length is 200 bytes. The data area will be 600 bytes, so there will be 7 blocks of 3 records each or 21 logical records per track.

In some cases, the table in Figure 17 cannot be used and the number of records per track for a given record design must be calculated using the formulas shown in Figure 18. The formulas are different for each device because the gap lengths required by each device are different. The formulas in Figure 18 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					
2311	2314	2302	2303	2301	2321		2311	2314	2302	2303	2301	2321
3625	7294	4984	4892	20483	2000	1	3605	7249	4964	4854	20430	1984
1740	3521	2403	2392	10175	935	2	1720	3476	2383	2354	10122	920
1131	2298	1570	1558	6739	592	3	1111	2254	1550	1520	6686	576
830	1693	1158	1142	5021	422	4	811	1649	1139	1104	4968	406
651	1332	912	892	3990	320	5	632	1288	893	854	3937	305
532	1092	749	725	3303	253	6	512	1049	730	687	3250	238
447	921	634	606	2812	205	7	428	878	614	568	2759	190
384	793	546	517	2444	169	8	364	750	527	479	2391	154
334	694	479	447	2157	142	9	315	650	460	409	2104	126
295	615	425	392	1928	119	10	275	571	406	354	1875	103
263	550	381	346	1741	101	11	244	506	362	308	1688	85
236	496	344	308	1585	86	12	217	452	325	270	1532	70
213	450	313	276	1452	73	13	194	407	294	238	1399	58
193	411	286	249	1339	62	14	174	368	267	211	1286	47
177	377	264	225	1241	53	15	158	333	245	187	1188	38
162	347	244	204	1155	44	16	143	304	224	166	1102	29
149	321	225	186	1079	37	17	130	277	206	148	1026	21
138	298	209	169	1012	30	18	119	254	190	131	959	15
127	276	196	155	952	24	19	108	233	176	117	899	9
118	258	183	142	897	20	20	99	215	163	104	844	
109	241	171	130	848	15	21	90	198	152	92	795	
102	226	161	119	804	10	22	82	183	142	81	751	
95	211	151	109	763	6	23	76	168	132	71	710	
88	199	143	100	726		24	69	156	123	62	673	
82	187	135	92	691		25	63	144	116	54	638	
77	176	127	84	659		26	58	133	108	46	606	
72	166	121	77	630		27	53	123	102	39	577	
67	157	114	70	603		28	48	114	95	32	550	
63	148	108	64	577		29	44	105	89	26	524	
59	139	102	58	554		30	40	96	83	20	501	

• Figure 17. Track capacity table

Device	Track Capacity (in bytes)	Bytes Required by Physical Data Records			
		Data Records (except for Last)		Last Data Record	
		without key	with key	without key	with key
2311	3625	$61 + \frac{537\ (DL)}{512}$	$81 + \frac{537\ (KL + DL)}{512}$	DL	$20 + KL + DL$
2314	7294	$101 + \frac{534\ (DL)}{512}$	$146 + \frac{534\ (KL + DL)}{512}$	DL	$45 + KL + DL$
2302	4984	$61 + \frac{537\ (DL)}{512}$	$81 + \frac{537\ (KL + DL)}{512}$	DL	$20 + KL + DL$
2303	4892	$108 + DL$	$146 + KL + DL$	DL	$38 + KL + DL$
2301	20483	$133 + DL$	$186 + KL + DL$	DL	$53 + KL + DL$
2321	2000	$84 + \frac{537\ (DL)}{512}$	$100 + \frac{537\ (KL + DL)}{512}$	DL	$16 + KL + DL$

• Figure 18. Track capacity formulas

• The *track capacity* figure is the number of bytes left for data records after subtracting 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 2311, for instance, requires 61 bytes for this information. This formula sometimes includes a fixed factor to account for the allowable deviation in the position of the record. The 2311 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 2311 is 20.

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

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

The formulas in Figure 18 are used rather than the table in Figure 17 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 2311, how many records can be placed on each track? The solution is as follows:

Bytes for each data record except the last =

$$81 + \frac{537\ (6 + 50)}{512} = 81 + 58 = 139$$

Bytes for the last data record $= 20 + 6 + 50 = 76$

$$\text{Records per track} = 1 + \frac{3625-76}{139} = 1 + 25 = 26$$

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 below.

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 is ended when the end of the cylinder is reached. In case of drum devices, 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 core storage.
- Search Identifier High or Equal
- Search Key Equal. This causes a search of the Key Areas. This and the other search key com-

mands 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 is 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 core storage and checked. The Cyclic Check bytes of the area are not transferred to core 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.

Verification of transfer from the control unit to the device is performed by following a write command with a read command coded in such a way that no data is transferred to core storage. Data is transferred from the device to the control unit, however, and the Cyclic Check is performed. This operation, called

Write Verify, is performed by the IBM programming system after every 2321 write operation and may be optionally specified by the user for write operations on the other DASDs. If Write Verify is performed, one extra revolution is required for each write command. Write Verify should be specified for permanent files. For those cases where the 2321 is used without IBM programming support, it is recommended that the programmer always provide for a Write Verify operation.

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.
- 2820 – Controls the 2301 drum.
- 2841 – controls the other four devices (2311 disk drive, 2302 disk, 2303 drum and 2321 data cell).

Additional Storage

This feature is available only for the 2841. It permits the attachment of eight more 2302 access mechanisms in addition to the eight access mechanisms that the standard 2841 can control.

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 2302s, 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.

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 and the 2820 and available as a special feature on the 2841.

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.

Two Channel Switch

The Two Channel Switch feature enables the control unit to be shared by two channels and also allows individual devices (access mechanisms) to be reserved for the exclusive use of either of the two channels. The two channels may be attached to the same CPU or different CPU's. Channel switching and device reservation is under program control. The Two Channel Switch feature is limited to eight access mechanisms.

When installed on the 2841, 2820, or the 2314 facility without a 2844, the control unit can control input/output operations for its devices via either of the two channels. Assume that a 2841 with a Two Channel Switch feature is attached to selector channels 1 and 2. If selector channel 1 is busy with the operations of another control unit, the 2841 can perform operations through selector channel 2 if it is not busy.

When a 2844 Auxiliary Control Unit is installed on a 2314 facility, 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.

When a control unit is attached to two channels that are attached to two different CPU's, it permits the two CPU's to share the same direct access devices.

Exercises

12. It is desired to attach five 2311's and *one* of the following to one 2841. Place a check in the correct column for each case.

	IMPOSSIBLE	OK	REQUIRES ADDITIONAL STORAGE FEATURE
a. Three 2321's			
b. Two 2303's			
c. One 2302-3			
d. Two 2302-4's			

13a. The validity of data being transferred between core storage and the control unit is checked by means of the _____ _____ associated with each _____.

b. The validity of data being transferred between the control unit and a direct access device is checked when data is read, by means of the _____ _____ associated with each _____.

14. A record will always have a _____ Area and a _____ Area. It may or may not have a _____ Area.

15. All of the following information is recorded on a formatted track. Indicate in which area(s) each one is located.

a. One-byte record number: _____
b. Four-byte cylinder/head number: _____
c. Two-byte cyclic check: _____
d. Flag indicating track condition: _____
e. Key that identifies a record: _____
f. Length of key area: _____
g. Length of data area: _____

16. How many data records can be recorded per track for each of the following?

a. 2321, unblocked records, data length of 200 bytes, formatted with a key that is 8 bytes: _____

b. 2302, unblocked records, data length of 250 bytes, formatted without keys: _____

c. 2311, unblocked records, data length of 150 bytes, formatted with a key that is 10 bytes: _____

17. Given logical records 100 bytes long, to be formatted without keys on a 2311, how many logical records per track are there for each of the following?

a. Unblocked: _____ c. Blocked 5: _____
b. Blocked 2: _____ d. Blocked 8: _____

18. Given logical records 120 bytes long, to be formatted without keys on a 2302, what blocking factor will allow the most logical records per track? (The block is not to exceed 1000 bytes. _____

19. It is desired to write one 200-byte record followed by as many 75-byte records as will fit on a 2311 track. All records are to be unblocked and formatted without keys. How many of the 75-byte records can be written on a track? _____

Chapter 4: Introduction to File Organization

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 the System/360.

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.

Random processing is the processing of detail transactions against a master file in whatever order they occur. With direct access devices, random 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 un-

desirable can be processed randomly. 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 randomly against the secondary files, all in a single run. This is the basis of inline processing.

Methods of Organization

Four methods of organization for direct access devices are supported by IBM programming systems. They are described briefly in this section and discussed fully in the following chapters.

Sequential Organization. In a sequential file (the IBM System/360 Operating System term is "data set" instead of "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 random 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 (Random) Organization. A file organized in a direct (random) 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 characteristics 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 betwen the key of the record and its address on the DASD.

IBM Operating Systems

Operating systems are part of the programming systems support supplied by IBM. The operating systems include IOCS (input/output control system) routines to schedule and control the transfer of data between core storage and I/O devices.

Three System/360 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. The three systems are:

- IBM System/360 Operating System, which requires a CPU with a minimum of 64K (K = 1024) positions of core storage.
- IBM System/360 Disk Operating System, which requires a CPU with a minimum of 16K positions of core storage.
- IBM System/360 Basic Operating System, which requires a CPU with a minimum of 8K positions of core storage.

In the remainder of this chapter, and in the following chapters, which discuss the methods of organization in detail, the term "operating system" applies to all three unless otherwise noted. Where there are differences, "OS" means the Operating System, "DOS" means the Disk Operating System, and "BOS" means the Basic Operating System.

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

One difference between the operating systems is in the direct access devices that are supported. OS sup-

ports all of them; DOS supports the 2321 Data Cell Drive, the 2311 Disk Storage Drive, and the 2314 Direct Access Storage Facility; BOS supports the 2311 only.

Sequential, indexed sequential, and direct methods of organization are supported by all the operating systems. OS also supports partitioned organization. The operating systems allow users to concentrate their programming efforts on processing the records read and written by the IOCS 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 GET or READ instructions to cause records to be transferred to core storage, and PUT or WRITE instructions to cause records to be transferred to I/O devices.

These instructions issued by the user are called "macro instructions". For one macro instruction, a number of machine instructions and commands are assembled. These in turn call into use certain IOCS routines. The I/O macro instructions and associated IOCS routines are referred to as "access methods" and are divided into two categories: queued access methods and basic access methods.

The queued access methods provide GET and PUT macro instructions. These 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 core 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 GET. Queued access methods are provided for sequential organization and indexed sequential organization by BOS, DOS, and OS.

The basic access methods provide READ and WRITE macro instructions. These 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. The system does not provide anticipatory buffering and scheduling. Macros are provided, however, to help the user program these functions. Moreover, READ and WRITE macros read and write physical, not logical, records. Thus, blocking and deblocking of records is (in most basic access methods) the user's responsibility. Basic access methods are provided in OS for all types of organization. BOS and DOS provide a basic access method for direct and indexed sequential organizations. DOS also provides a basic access method for sequential organization.

The BOS, DOS, and OS programmer specify the I/O macros in different ways. This text, however, does not discuss the way in which these macros are written, but only the various methods of file organization supported by OS, BOS, and DOS and the capabilities of the access methods.

In BOS and DOS terminology, the various access methods are referred to collectively as "logical IOCS". In most situations, the programmer will find it advantageous to use the access methods (logical IOCS), since their use greatly reduces programming effort. Occasionally, however, it may be necessary for him to control the I/O devices directly. He can do this by using the EXCP macro to call upon a lower level of IOCS called "IOS" in OS and "physical IOCS" in BOS and DOS. In the following chapters on file organization, any restrictions noted apply only when the access methods (logical IOCS) are being used.

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, in OS, 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 at random with the basic access method for direct organization.

The facilities for creating and processing files on DASD are discussed in the next four chapters. Utility programs are also provided by each of the operating systems. In addition, a number of stand-alone 2311 utility programs for operations outside the control of an operating system are provided in the category of Basic Programming Support. This category also includes Initialize Disk (2311) and Initialize Data Cell (2321) programs.

Please note that the DASD timings in the practice exercises for Chapters 5, 7, and 8 are approximate times. In addition, these times are not estimates of total job time, since the practice exercises do not consider factors beyond the scope of this text, such as problem program processing time, access method processing time, and control program time.

Chapter 5: 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 random 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.

Random Processing

Random 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 random 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 random 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 19. 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.

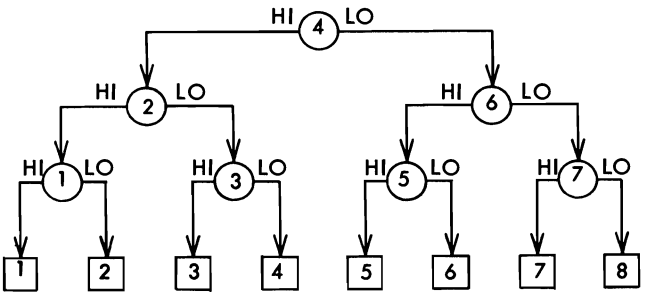


Figure 19. A binary search of an eight-cylinder file

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 for sequentially organized files is available only with OS and DOS. Anticipatory buffering and blocking/deblocking routines are not provided. The basic access method can be used to read or write records formatted with keys (OS only) or without keys (OS and DOS). It can be used, to a limited extent, to store and retrieve records at random. 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 at random) such files. A corresponding access method exists in OS that can be used to process at random sequential files formatted with keys.

User Options

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

Exercises

The following questions apply to a sequential file on a 2311 Disk Storage Drive. There are 10,000 logical records, each 160 bytes long. The records are to be formatted without keys.

20. If the block is kept under 1000 bytes, what is the optimum blocking factor for maximum storage utilization?

21. If the optimum blocking factor is used, how many cylinders are required for the file?

22. What is the disk time required to read the entire file sequentially? Use one revolution per read.

23. What is the disk time required to read 5000 of the records in random sequence?

Chapter 6: Partitioned Organization

A partioned 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 areas 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. Three special macro instructions are provided. FIND causes the starting address of a specified member to be returned to the user so that the member can be processed. STOW causes the name of a member to be entered into, changed, replaced, or deleted from the directory. BLDL causes a list to be built in core storage. This list contains the starting address and user information from the directory for each specified member. The FIND macro can then be directed to an entry in the list rather than to the directory in order to minimize access time.

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. See "Operating System Functions", in Chapter 5.

Chapter 7: 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 only) consist of several noncontiguous areas. The records in the prime area are in key sequence.

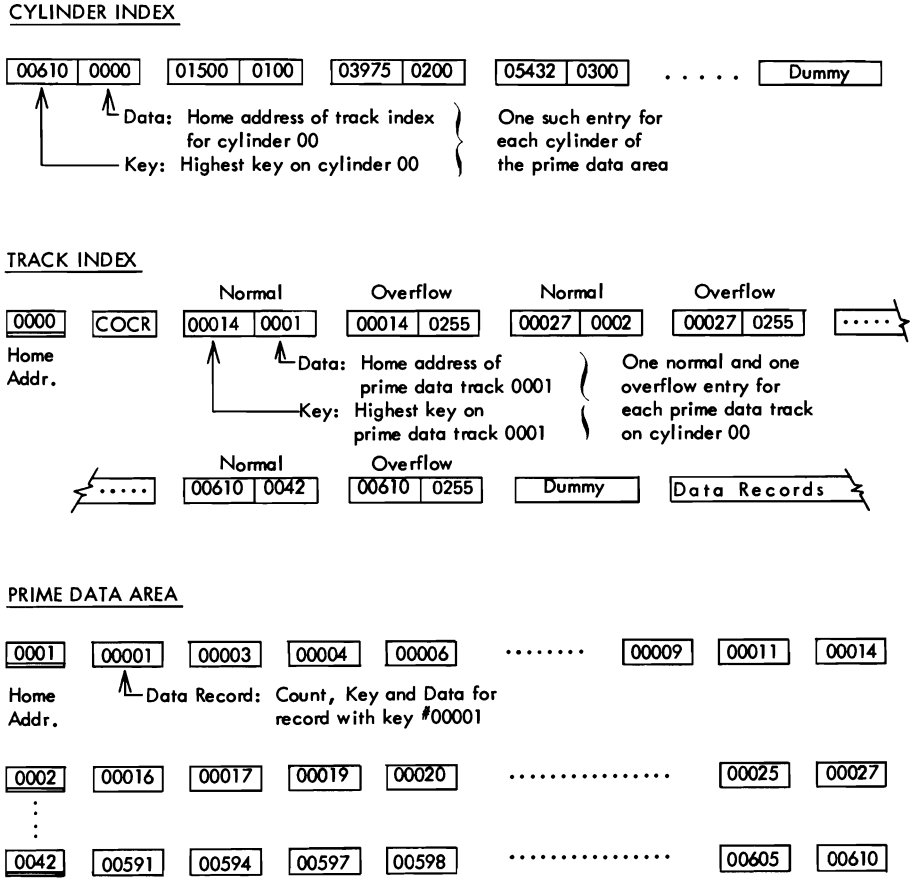
Prime records must be formatted with keys. They may be blocked or unblocked. If blocked, each logical record contains its key and the key area contains the key of the highest record in the block.

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 20)

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.



• Figure 20. An indexed sequential file with no additions

Each track index may contain a special first record called a "Cylinder Overflow Control Record" (see below, "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 below, "Additions Procedure").

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 20 only the cylinder and head portion of the address in the Data Areas is shown.

Cylinder Index (See Figure 20)

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 BOS and DOS it is written wherever the user specifies. In OS 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.

BOS and DOS permit one level of master index for the file and require that it be written immediately before the cylinder index. 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 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 21)

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 20). BOS, DOS, and OS 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 also uses this record for additional information needed when the file has variable-length records. OS uses two bytes of the COCR for this purpose. These two bytes are blank in BOS and DOS.

An advantage of having a cylinder overflow area is that additional seeks are not required to locate over-

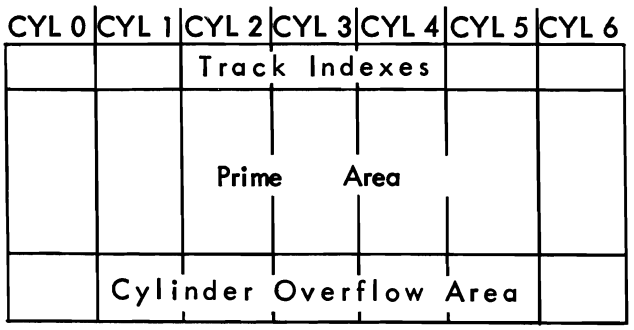


Figure 21. Cylinder overflow area

flow records. A disadvantage is that there will be unused space if additions are unevenly distributed throughout the file.

Independent Overflow Area (See Figure 22)

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.

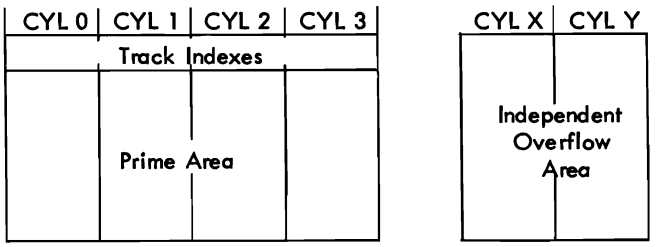


Figure 22. Independent overflow area

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 areas are filled.

Overflow Records

Overflow records must be unblocked. They must be formatted with keys. They may be fixed-length or, in OS only, 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.

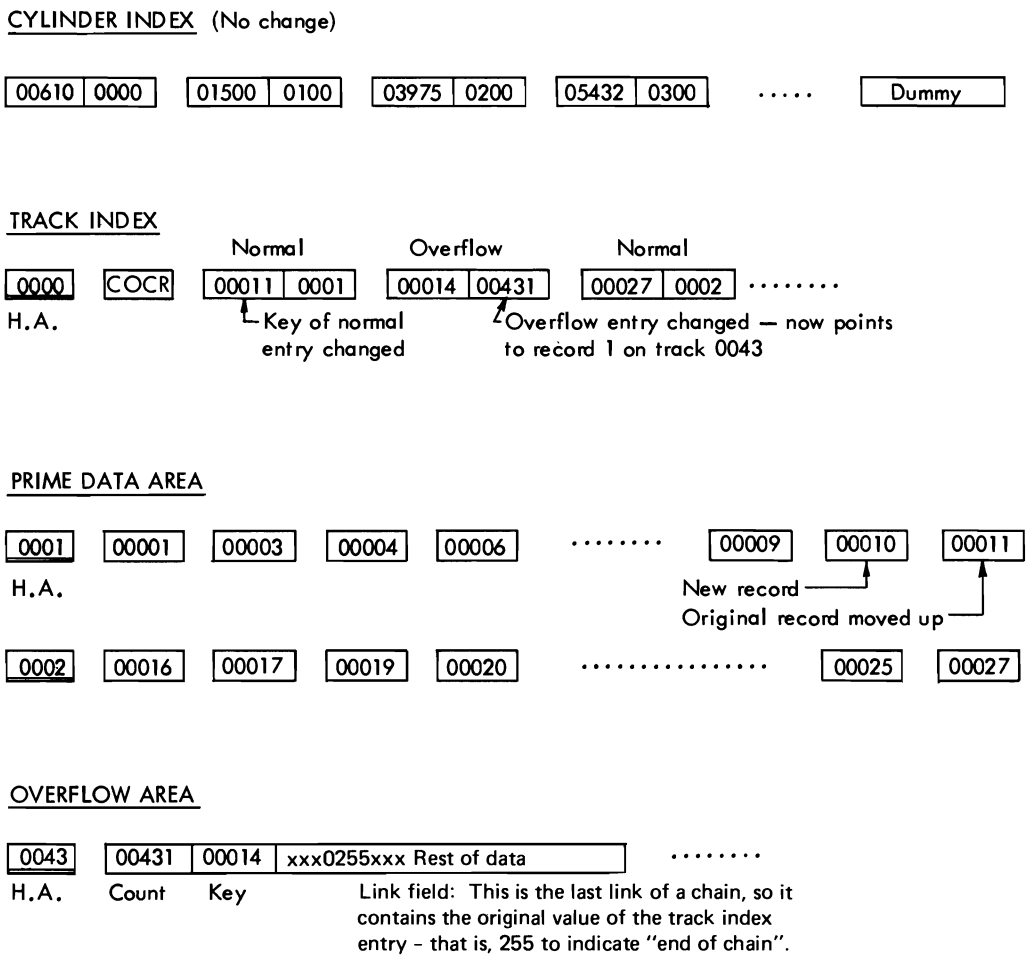
The first field in the Data Area of an overflow record is a link field. It is used to chain together in key sequence the records that have overflowed from a prime track. The link field is ten bytes long and contains the same type of information as the Data Area of index entries. If an overflow record 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. Each is discussed below.

First Addition to a Prime Track (See Figure 23)
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 23. An indexed sequential file after the first addition to a prime track

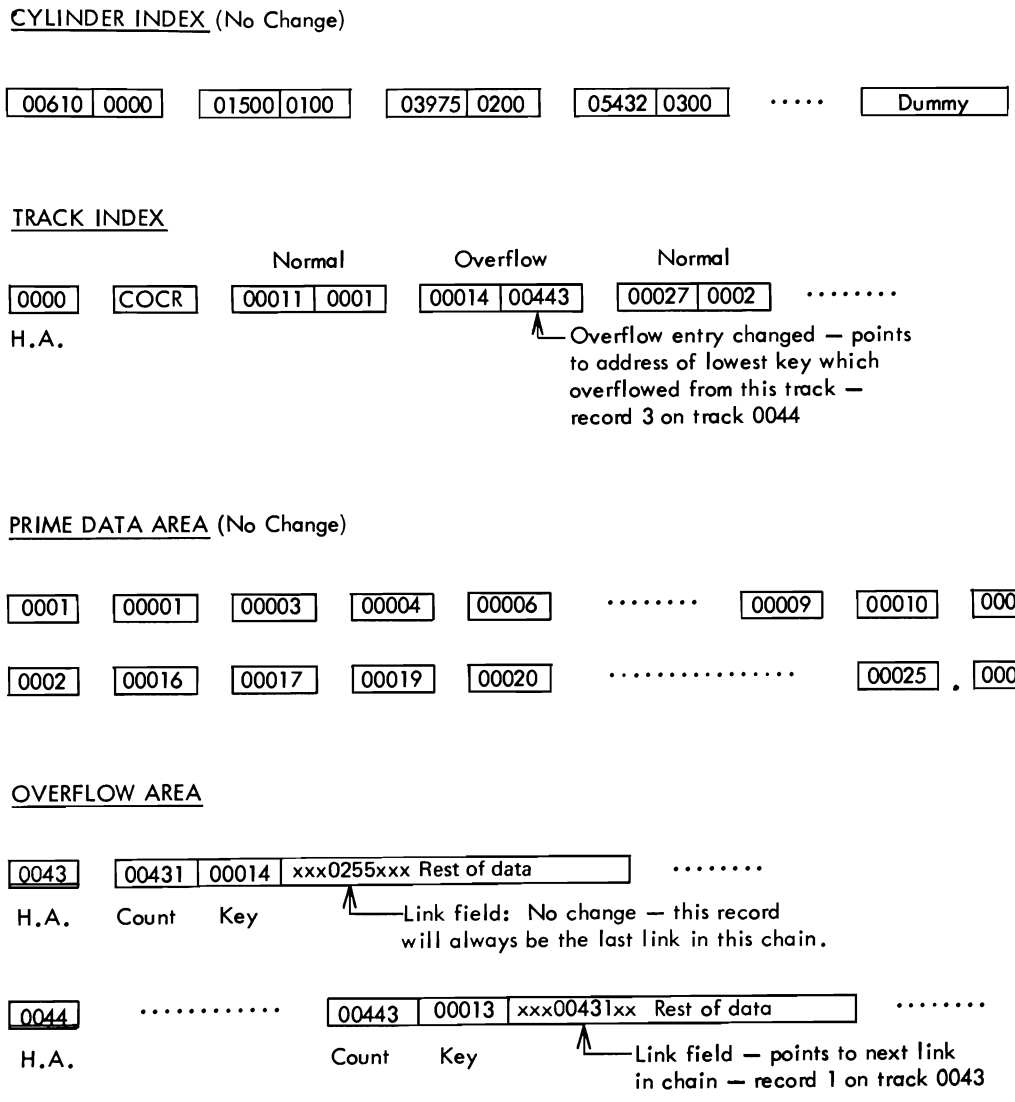
Subsequent Additions to a Track (See Figure 24)

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 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 third record on track 0044, 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.



• Figure 24. An indexed sequential file after subsequent additions to a track

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.

If a number of "high-key" records must be added to the file, it is possible in BOS and DOS to extend the prime area and load the new records there.

Timing

None of the following detail is programmed by the user. The steps shown indicate the general logic of the IOCS routines generated by the operating system from the user's GET macro.

Sequential Processing

The logical steps required to retrieve all of the records in key sequence are as follows (assuming that the prime area consists of one contiguous area):

1. Initialize.
   a. Position access mechanism at the track index of the first cylinder of the file.
   b. Search the track index and read into core two entries from the track index (the current overflow entry and the next normal entry).
2. Read and present to the user each record on the specified prime track. If the end-of-file record is read, go to the end-of-file routine.
3. If the current overflow entry has been changed, read and present to the user each record in the overflow chain and then go to step 5.
4. If the current overflow entry has not been changed, go to step 5.
5. If the next prime entry is a dummy (that is, if all the records in this cylinder have been read), seek the next cylinder and go to step 7.
6. If the next prime entry is not a dummy, go to step 7.
7. Search the track index and read into core the next pair of entries (the overflow entry of the next prime track to be processed and the normal entry for the prime track following that).
8. Go to step 2.

Note that reference to the cylinder index (and master index) is necessary only for the initial positioning at the beginning of the file and that reference to the track index is necessary only once for each prime track.

The basic disk time to process the file sequentially is:

1. One seek for each prime cylinder.
2. One search and read (and write, or write and Write Verify, if updating) for each data record (or block of records) in the file.
3. One search and read for each pair of entries in the track indexes.

The search for and reading of a data record should be estimated at one rotation (following our procedure of allowing one rotation to include both rotational delay and data transfer time). Since the index entries are such short records, the search for and reading of a pair of track index entries should be rated at half a rotation instead of a full rotation. The seek for each prime cylinder could be rated at minimum access time rather than average access time, since the file is located on one contiguous area. Additional time is required if the file has an independent overflow area.

In some cases it may be desirable to process only a section of an indexed sequential file. This can easily be done. Assume that an indexed sequential payroll file is in sequence by man number within department. It might be necessary, for instance, to process the portion of the file that begins with the record identified as man number 20, department 05. A SETL macro is used to specify the key of the first record to be processed (in this case 0520). The programming to process the first record and the ones following it would be the same as for normal sequential processing, except that the user's program must, if it is not necessary to process through the end of the file, recognize when the last desired record has been processed and issue an ESETL macro at that point to terminate the sequential accessing of the file.

In OS and DOS only, the SETL macro can specify only the prefix of a key and thus ask that the processing of a file start with the first record containing that prefix in its key. Assuming the payroll file mentioned above, the SETL macro could specify 0500, causing the processing to start with the first record of department 05. This prefix capability of the SETL macro frees the user from the necessity of knowing the entire key of the first record of a certain class to be processed. Again, the ESETL macro can be used to terminate the sequential accessing of the file.

Random Processing

The logical steps required to retrieve specified records in random sequence are:

1. Read transaction.
2. Search the cylinder index for Key High or Equal.
3. When search is satisfied, read corresponding Data Area.

4. Seek to the referenced cylinder.

5. Search the track index for that cylinder for Key High or Equal.

6. When search is satisfied, read corresponding Data Area.

7. If it is a normal entry:
   a. Search the referenced prime track for Key Equal.
   b. Read and present to the user the corresponding Data Area. If the prime records are blocked the operating system searches the block in core and presents the specified logical record to the user.

8. If it is an overflow entry:
   a. Search the referenced overflow track for Identifier (record address) Equal.
   b. Read corresponding Key and Data Areas.
   c. If it is the specified record, present to the user.
   d. If it is not, repeat steps a and b, using the address from the link field until the specified record is found.
   e. If the end of the chain is reached, go to record-not-found routine.

Thus the disk time required to locate a record is two seeks (to the cylinder index and to the prime cylinder) and three reads (of the cylinder index, track index, and data record). As in the discussion of sequential processing of an indexed sequential file, each read of a data record should be rated at one full rotation, and each search and read of an index track at half a rotation.

Additional disk time is required if the file has a master index or if there is an independent overflow area. Additional time is also required if the specified record is other than the first overflow record in an overflow chain. Notice in Figure 24 that record 00013 is referenced directly by the overflow index entry.

The disk time can be minimized by placing the cylinder index on a different access module from the rest of the file. Then the cylinder index access mechanism does not have to move after it is first positioned. (This approach, however, is not always helpful in an OS or DOS multiprogramming environment, since a file processed by another task might be located on the same module as the cylinder index and thus require the movement of the access mechanism.)

OS, however, offers the user the option of searching the file's highest-level index in core. The entire index is read once and held in core for the duration of the job. Thus the minimum disk time to locate a record can be reduced to one seek and two reads (of the track index and the data record). DOS offers the user the option of holding all or part of the cylinder index in core. If only a portion of the cylinder index is kept in core, a new portion of the cylinder index will be read into core when the key of the record to be processed is not within the range of keys in the "in core" cylinder index.

File Maintenance

The user writes his own load and maintenance programs using the macros provided by the operating system. Additions are handled by the operating system as already discussed. The user may choose to write a separate additions program or to include the additions procedure as part of another job.

The file must be reorganized periodically for three reasons: (1) the overflow area will eventually be filled, (2) additions increase the time required to locate records at random, and (3) the prime area may contain too many deleted records. The frequency of reorganization depends on the volatility of the file and on the user's timing and direct access storage requirements. There are two ways to handle reorganizations: (1) The file can be written sequentially into another area of direct access storage or some other storage medium and then re-created in the original area. (2) It can be reorganized in one pass into some other area of direct access storage, in which case none of the area occupied by the original file can be used by the reorganized file.

The operating system maintains statistics that are pertinent to reorganization. These statistics are written on the direct access device and may be read by the user at any time. The statistics maintained are the number of cylinder overflow areas that are full, the number of unused tracks in the independent overflow area, the number of references to non-first overflow records, and (in OS only) the number of records marked for deletion.

BOS and DOS do not handle deletions in any way. The usual approach is for the user to tag deleted records in some way and then omit them when the file is reorganized.

OS offers a delete option to the user. A record to be deleted is tagged by writing all 1-bits in the first byte of the logical record. If a tagged record is bumped off the prime track by a subsequent addition, it is not rewritten in the overflow area. When the file is reorganized, any tagged records remaining in the prime area can be omitted from the reorganized file by the user. When the file is processed sequentially, records tagged for deletion are not retrieved for processing. When the file is processed in random sequence, tagged records are retrieved like any other record and thus should be

checked for the deletion code by the user's program.

Variable-Length Records

As already noted, BOS, DOS, and OS do not permit variable-length records.

One approach to variable-length records that does not require programming system support of 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 provides anticipatory buffering and overlap of input/output with processing; with BOS and DOS, this can be programmed to a limited extent by the user.

This access method is also used with OS for creating the file — both for the initial loading and for reorganization. With BOS and DOS, the basic access method is used. The operating system formats all the tracks in the specified areas, writes the data records, and creates and writes all the index entries.

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 at random. 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.

User Options

Following is a list of the available options already discussed.

All levels of operating system:
- Prime records blocked or unblocked.
- Master index if cylinder index exceeds specified length.
- Cylinder or independent overflow area, or both.
- Write Verify.
- Reorganization statistics.

OS only:
- Three levels of master index.
- Highest level index held and searched in core.
- Delete option.

DOS only:
- All or part of the cylinder index can be held and searched in core.

Exercises

24. With OS, when processing an indexed sequential file at random, it takes a minimum of _____ seek(s) and _____ rotation(s) to locate each record. Name the four situations that will require additional time.

The following questions all apply to an indexed sequential file with the following characteristics:

Device is the 2311 Disk Storage Drive.
Logical records are 160 bytes long, including a 7-byte key.
Blocking factor is 5.
File is designed for 10,000 prime records.
One track per cylinder is reserved for overflow.
The area assigned to the file is one contiguous area.
The operating system used is OS.

25. How many logical records will fit on each prime track?

26. How many logical records will fit on each overflow track?

27. Assuming that each track index will require less than a full track, this leaves the rest of that track for prime records. In addition, how many full tracks per cylinder are left for prime records?

28. How many entries will each track index have in addition to the COCR?

29. Using the track capacity formulas given in Chapter 3, calculate the total number of bytes required for the index entries for each track index.

30. How many prime records (logical records) will fit on the rest of each track index track?

31. Except for higher-level indexes and the cylinder overflow area, how many cylinders will the file require?

32. How many entries will the cylinder index have?

33. How many tracks will the cylinder index require?

34. If the cylinder index were to be held in core during a job, how many bytes would be required for it?

35. Should a master index be specified for this file?

36. What is the disk time required to read the entire file sequentially? Assume that the overflow areas are half full.

37. What is the disk time required to read 5000 of the logical records in random sequence? Assume that the cylinder index is serviced by the same access mechanism as the rest of the file and is not to be held in core.

38. What is the disk time required to read 5000 of the logical records in random sequence if the cylinder index is serviced by a different access mechanism from the rest of the file?

39. What is the disk time required to read 5000 of the logical records in random sequence if the cylinder index is held and searched in core?

Chapter 8: Direct (Random) Organization

This chapter discusses some commonly used methods of direct (random) 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 2302 is 0045, 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.

OS and DOS allow the user to refer to a relative track address. If N tracks are allotted to a file, the user refers to relative track 0 through N-1. IOCS converts this to the corresponding absolute track address. With BOS, the user programs the steps to convert the relative track address to an absolute track address of the format shown in Figure 25. Each byte in the address is a binary number. With all levels of the operating system, if the user wants to refer to a particular record, he must supply either its key or its identifier (cylinder number, head number, and record number) as well as the track reference.

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.

Byte	2311		2321	
0	0-244	volume	0-244	volume
1	0		0	
2	0		0-9	cell
3	0		0-19	subcell
4	0-199	cylinder	0-9	strip
5	0		0-4	cylinder
6	0-9	head	0-19	head

● Figure 25. Absolute track address

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 at random, 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 computation. This technique is called "randomizing". It inevitably causes "synonyms" — two or more records whose keys randomize to the same address. Two objectives must be considered in selecting a randomizing technique: (1) every possible key in the file must randomize 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 where it "belongs" (at the address to which its key randomizes) is called a "home record". Any other records whose keys randomize to this address are "overflow records". What to do about overflow 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 overflow 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.

A way to minimize overflows is to randomize to track address rather than to record address. If randomizing to record address, every synonym causes an overflow. As shown in Figure 26, 30% of the records are synonyms and 30% are overflows. If randomizing to track address, there are many synonyms, but no overflow until a track is full. As shown in Figure 26, 70% of the records are synonyms (the two A3 records, B2, B4, two C2 records, and C3), but there are no overflows. If randomizing is to record number, the commands to locate a record are Seek, Search Identifier Equal, Read Data. If randomizing is to track address, the commands are Seek, Search Key Equal, Read Data. Both sets of commands take the same amount of time.

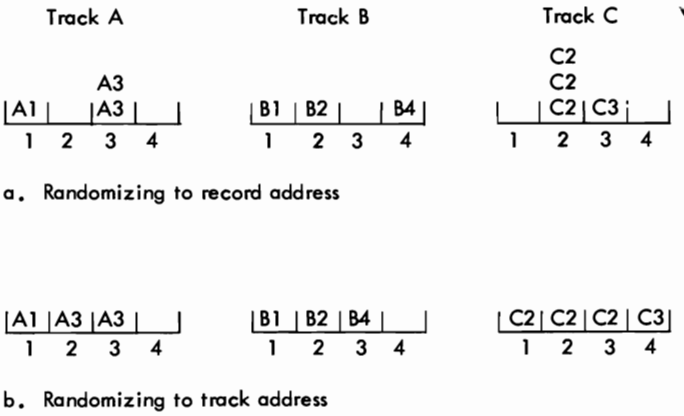


Figure 26. Synonyms and overflows

Randomizing Techniques

There are many randomizing 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 randomizing 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 2311, randomizing to track address.

a. With 80% packing, 10,000 locations are required.
b. Can load 13 records per track, so 770 tracks are required.
c. A prime number close to 770 is 769.
d. Divide the key by 769.
e. The remainder (000 to 768) equals the relative track address.

Example 2: Same as above, but randomizing 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 (13).

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 a randomizing 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 27 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 randomize 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 randomizing 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.

Automatic Programmed Address Conversion

This 1401 program (1401 01.4.034) can be used to develop a randomizing technique. The program first performs a digit analysis of the keys in a file. It then develops a table of constants based on the counts in the digit analysis. The table can be written on a direct access device and used for randomizing by looking up the constant for each digit in a key, adding them together, and multiplying the result by a constant which is the number of locations allotted to the file.

The program also evaluates the results of the randomizing and prints the average number of seeks per

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 27. Digit analysis table

record and a summary of synonyms which shows the number of addresses with one record assigned, the number of addresses with two records assigned, and so on up to twelve.

Either the digit analysis phase or the evaluation phase can be used alone as an aid in developing some other randomizing technique.

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 randomizing technique should be applied to the entire file of keys and carefully evaluated before deciding to use it. When evaluating a randomizing 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 randomize 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 randomize 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 randomizing 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 (OS only), or undefined. Since BOS and DOS permit undefined records, they, too, can handle variable-length records.

Records may be formatted with or without keys. If the file is indirectly addressed and randomization 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 (assembling a block of logical records before using the WRITE macro) and deblocking (searching the block read into core by the READ macro for the desired logical record), 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 (which are originally written for a file by a utility program in BOS and DOS, and a user-written program in OS) 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 overflow records is up to the user, but they are generally put in unused locations in the main (and only) area. Overflow records can be put in a separate area if the user desires. The disadvantage of doing this is that each overflow record will require an additional seek. If there is just one area, and if a good randomizing technique has been selected and the file is not packed too tight, overflow records are likely to be in the same cylinder as the home record, thus eliminating the need for an additional seek.

File Creation, Maintenance, and Processing

As already noted, the user has complete freedom in deciding where records are to be located in a directly organized file. The logic and programming are his responsibility.

When creating or making additions to the file, the user may specify the location for a record by supplying the track address and identifier, or he may supply just a track address and let the operating system find a location for the record. If there is room on the specified track, the operating system writes the record (and updates the capacity record for files constructed with capacity records). If the specified track is full, OS continues searching on successively higher tracks until a location is found. This search continues for as many tracks as the user has specified, to a maximum of the entire file. If a maximum search is specified and the end of the file is reached, the search resumes at the beginning and continues until a location is found or until the original track is reached. With BOS or DOS, if the specified track is full, the user must supply another track address.

When reading or updating the file, the user must supply a track address and either the identifier or the key of the desired record. When an identifier is supplied, the operating system reads or writes back that specific record. When a key is supplied, the operating system searches for that key and, upon finding it, reads or writes back the corresponding Data Area. If the key is not found, the operating system so indicates to the user. A search by key may be a restricted or an extended one. On a restricted search, only the track specified by the user is searched. On an extended search, OS continues searching on successively higher tracks for as many tracks as the user has specified; BOS and DOS continue searching to the end of the cylinder.

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

Two approaches to the handling of overflow records are discussed in the following sections: chaining and progressive overflow. They are discussed in order to point out how the maintenance and processing 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. Both approaches assume that:

- The records are unblocked.
- The records are formatted with keys.
- Randomization is to track address.
- Overflow records are placed in unused locations in the main (and only) area.

Chaining Method

One record on each track is used as a chaining record to provide a link between the home track and an overflow track. Overflow records are written on the next higher available track.

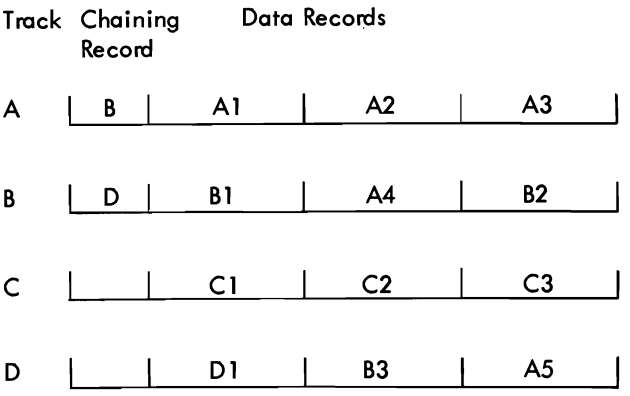


Figure 28. Chaining

Figure 28 shows a chained file. The sequence in which the records were loaded was A1, B1, A2, D1, C1, A3, C2, A4 (overflow), B2, C3, B3 (overflow), A5 (overflow). The following questions and answers explain how records in a chained file are located:

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

A. On track A. Searches always begin with the home 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 29. 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 home 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 home records require one read, first overflows require two reads (home track and overflow track), second overflows require three reads (home track, first overflow track, second overflow track), etc. Notice that record C should have been a home record, but an overflow from track 1 took its place first.

The file may be loaded in two passes. The results of this are shown in Figure 30. On the first pass, only home records are loaded. On the second pass, the overflow records are loaded and the chaining records updated. Because all home records are written on their home track, less chaining is required, and the average number of reads per record has decreased. The general logic of a two-pass load is shown in Figure 31.

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 29. One-pass load

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 30. Two-pass load

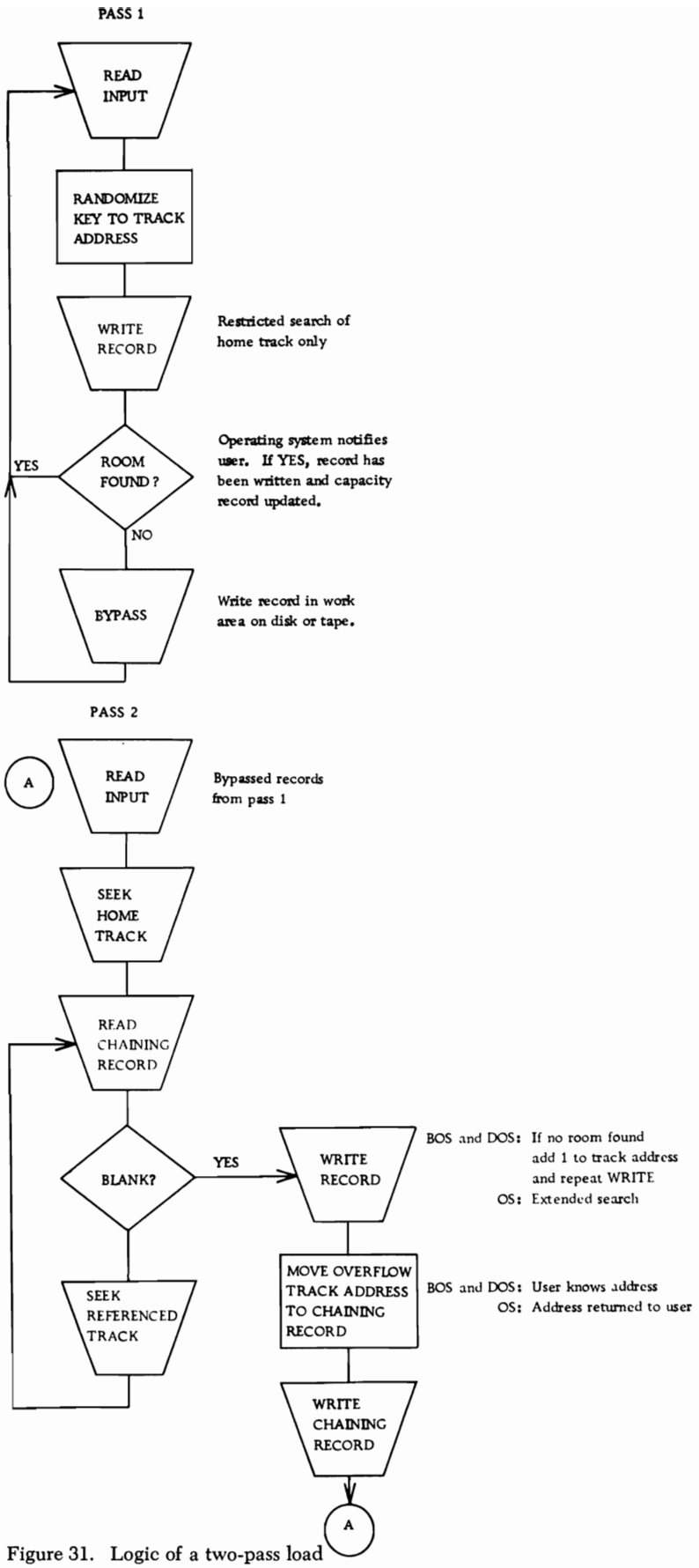


Figure 31. Logic of a two-pass load

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 home track, but it is already filled with overflows 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 28 (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 randomizing to track address, a track may contain overflows from more than one home 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 randomizing 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.

Processing the File

The general logic of processing a chained file is shown in Figure 32. Note that locating an overflow record actually requires two additional reads (revolutions) for each link in the chain: (1) a read of the chaining record, and (2) a search of the overflow track followed by a read if the key is found. When evaluating a randomizing technique for a file with chaining, three reads should be allowed for the first overflow, five for the second, and so forth when computing the average reads per record.

The same logic can be used when the file is to be processed sequentially. The input to this job is a finder

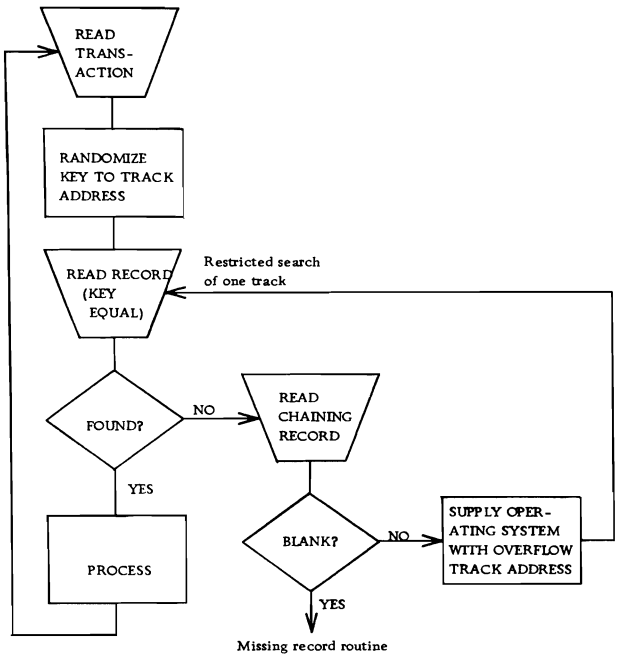


Figure 32. Processing a chained file

file on any storage medium of all the keys in sequence. An alternate approach is to sort the file into key sequence and then process it using the sequential access method.

Progressive Overflow Method

As with chaining, overflow records are written on the next higher available track. The difference is that there is no chain from the home track to the overflow 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 33 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 Figures 29 and 30 because all tracks between the home track and the one where an overflow 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. The general logic of a one-pass load is shown in Figure 34.

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 33. Progressive overflow

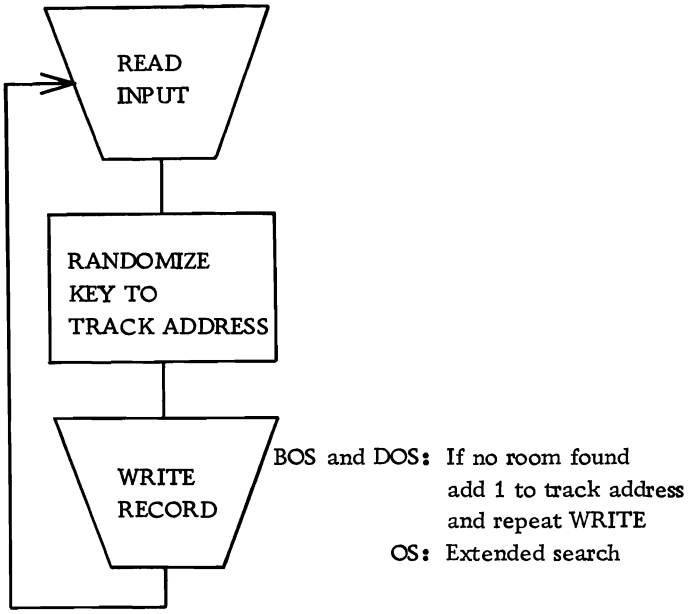


Figure 34. Logic of a one-pass load for a progressive overflow file

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.

Processing the File

When processing the file, either of two approaches is possible: an extended search (Figure 35) or a track-by-track search (Figure 36). A track-by-track search takes longer than an extended search to locate overflow records, since the capacity record is read if the desired record is not found on a track. A track-by-track search, however, signals a missing record condition earlier (when the system encounters the first track that is not full) than an extended search.

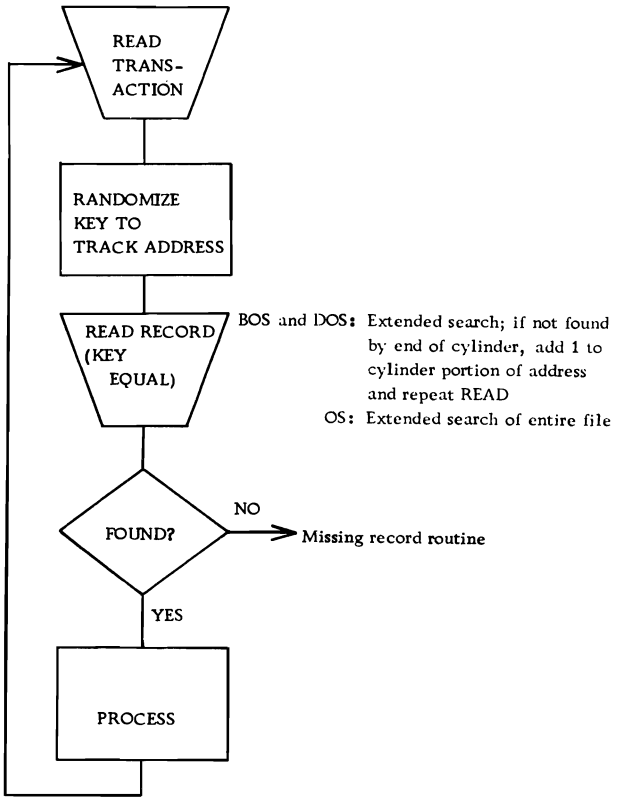


Figure 35. Processing a progressive overflow file with extended search

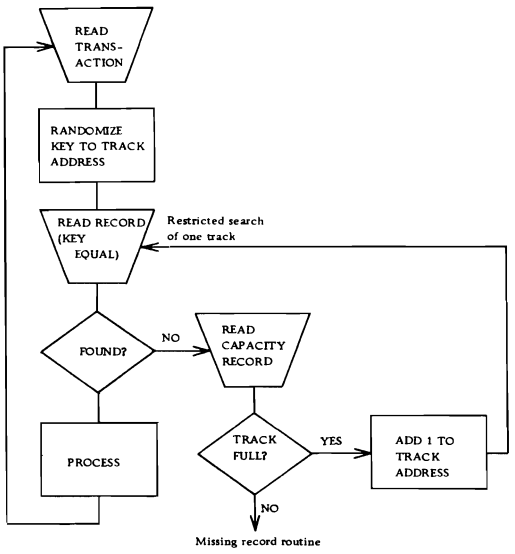


Figure 36. Processing a progressive overflow file with a track-by-track search

Progressive Overflow Compared to Chaining

The chaining method of handling overflows requires somewhat more complicated load and addition programs but in some circumstances it may result in a shorter search for overflow records. If a fairly low packing factor is used, however, progressive overflows will usually be located on the track that follows the home 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.

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 37 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

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 37. Effect of loading sequence on timing

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.

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 read and write file commands transfer entire Data Areas. Therefore, the load and addition programs would have to be updating operations — that is, reading and then writing back. For example, when adding a record to the file, an entire block must be written. If the user simply set up an output area in core storage, moved the new record to its proper location, and issued the WRITE macro, any other records already in the block would be destroyed. Instead, the existing block must be read, the new record moved to its proper location, and the block rewritten.

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. Randomization 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 3, divide by the number of blocks per track.

When loading the file or making additions to it, the operating system cannot find a location for a record because it has no way of knowing whether the specified block is full. The user has to read the home block and search it in core to see whether it is full. If it is not, the new record is moved in and the block rewritten. 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.

When processing the file, the user would have to read block after block and search them in core until the desired record was located.

If enough core is available so that the entire track can be handled as one block, the timing will be about the same as for an unblocked file. If there is more than one block per track, it will take longer to locate records — for two reasons: (1) since randomizing is to a smaller "bucket", there will probably be a higher percentage of overflows, and (2) the search itself will take longer because approximately one revolution per block, rather than one revolution per track, is required.

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. In OS, some directly organized files can be created with the basic access method for sequentially organized files.

User Options

Following is a list of the available options that have already been discussed.

All levels of operating system:
- Records fixed or undefined.
- Records formatted with or without keys.
- Restricted or extended search when processing.
- Write Verify.

OS only:
- Records fixed, variable, or undefined.
- Restricted or extended search when writing new records.
- Record Overflow feature, for fixed-length records only.

Exercises

The following questions apply to a file with the following characteristics:

Device is the 2311 Disk Storage Drive.
Logical records are 160 bytes long, including a 7-byte key.
The records are unblocked and formatted with keys.
File is designed for 10,000 records.
The packing factor is 85%.
The records are indirectly addressed.
Progressive overflow with extended search will be used.

40. How many cylinders will be required for the file?

41. What is the disk time required to read the entire file in key sequence? Assume that a finder file is used and that it takes an average of one seek and 1.2 reads to locate each record.

42. What is the disk time required to read 5000 of the records in random sequence?

Chapter 9: System Design Considerations

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 random 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 categories: 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 included 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 future. 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 multiple-card 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. Multiply 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 checking factors:	4	3	2	7	6	5	4	3	2

Mutiply 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 in-

validate 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 purpose 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, calculated 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 campaign.

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 situation 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 system checks to avoid duplication. As 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 2302 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 discussion 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 38. 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, 1966

Account Number	Name	Last Reference		Amount Paid	Balance Due
		Date	Run		
123476	S. J. Wilson	06-25	JRNL	250.00	182.94

Figure 38. 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.

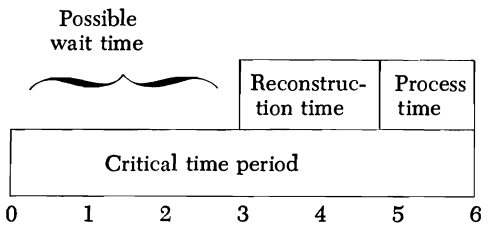
Bypass Procedures

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 considerably 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 designed 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 checkpoint was taken. When this is completed, normal restart procedures can be accomplished and reprocessing begun.

Bibliography

IBM System/360 Component Descriptions — 2841, 2302, 2311, 2321, 2303 (A26-5988) — includes a detailed description of the I/O instructions and file commands available for the System/360 DASD.

IBM System/360 Component Description-2314 Direct Access Storage Facility (A26-3599) — describes the 2314 and presents its functional and operating characteristics.

IBM 2301 Drum Storage Reference Card (X20-1717)

IBM 2302 Disk Storage Reference Card (X20-1706)

IBM 2303 Drum Storage Reference Card (X20-1718)

IBM 2311 Disk Storage Drive Reference Card (X20-1705)

IBM 2314 Direct Access Storage Facility Reference Card (X20-1710)

IBM 2321 Data Cell Drive Reference Card (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.

IBM System/360 Operating System: Supervisor and Data Management Services (C28-6646) — includes a discussion of the file organization and access methods supported by OS.

IBM System/360 Operating System: Supervisor and Data Management Macro Instructions (C28-6647) — describes in detail the OS macro instructions.

IBM System/360 Basic Operating System: Programmer's Guide (C24-3372) — includes a detailed description of the file organization and processing methods supported by BOS.

IBM System/360 Basic Operating System: Assembler with Input/Output Macros (C24-3361) — includes a detailed description of the BOS macro instructions.

IBM System/360 Disk Operating System Data Management Concepts (C24-3427) — includes an introduction to the file organization methods and access methods supported by DOS.

IBM System/360 Disk Operating System Supervisor and Input/Output Macros (C24-5037) — describes the DOS macro instructions.

Management Control of Electronic Data Processing (F20-0006) — discusses methods for controlling data processing procedures.

System/360 Component Description — 2835 Storage Control and 2305 Fixed Head Storage Module (GA26-1589)

Component Summary — 3830 Storage Control — 3330 Disk Storage (GA26-1592)

IBM System/370 System Summary (GA22-7001)

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)

Chapters 1 and 2

Device	Storage Medium	Cylinders	Tracks per Cylinder	Bytes per			Access Motion (MS)			Rotation (ms) (Full)	Transfer Rate (KB)
				Track	Cylinder	Device (Million)	Min.	Max.	Avg.		
2311	Disk	200	10	3625	36,250	7.25	25	135	75	25	156
2314	Disk	Pack: 200 Model A1 Total: 1600 Model A2 Total: 1000	20	7294	145,880	Pack: 29.17 Model A1 Total: 233.408 Model A2 Total: 145.880	25	130	60	25	312
2302	Disk	Model 3: 492 Model 4: 984	46	4984	229,264	Model 3: 112.79 Model 4: 225.59	50	180	165	34	156
2303	Drum	80	10	4892	48,920	3.9	0	0	0	17.5	303.8
2301	Drum	1	200	20,483	4.09 (Million)	4.09	0	0	0	17.5	1200
2321	Strip of Tape	Strip: 5 Array: 10,000	20	2000	40,000	400	95	600	350*	50	55

*Assuming that the previously addressed strip has already been restored. If this assumption cannot be made, average access time is 550 ms.

2. Although the rotation speed is the same, a 2314 track contains approximately twice as many bytes as a 2311 track.

3. With the 2301, data is transferred four bits in parallel rather than one bit at a time, as with the other devices.

4. 2311, 2314, 2321

5. a. 2311, 2314, 2303
   b. 2302, access group
      2321, subcell

6. a. 120 ms (same area, different group)
   b. 180 ms (different area)
   c. 50 ms (same group)

7. a. 10,000/13 = 770 tracks/10 = 77 cylinders
   b. 10,000/23 = 435 tracks/20 = 22 cylinders
   c. 10,000/18 = 556 tracks/46 = 13 cylinders
   d. 10,000/16 = 625 tracks/10 = 63 cylinders
   e. 10,000/61 = 164 tracks
   f. 10,000/7 = 1429 tracks/20 = 72 cylinders/5 = 15 strips (two cylinders will be used on the 15th strip)

8. a. 77 out of 200 cylinders = 39%
   b. 22 out of 1600 cylinders = 1.4% of the total facility
   c. 13 out of 492 cylinders = 2.6%
   d. 63 out of 80 cylinders = 79%
   e. 164 out of 200 tracks = 82%
   f. 72 out of 10,000 cylinders = .7%

9. The only access motion time is from cylinder to cylinder.
   a. 76 moves @ 25 ms = 1900 ms
   b. 21 moves @ 25 ms = 525 ms
   c. 1 move @ 120 ms + 11 moves @ 50 ms = 670 ms
   d. none
   e. none
   f. The 15 strips occupy two subcells, so there is one access to the second subcell at 450 ms (200 + 75 + 175). This leaves 13 strips to be accessed from the same subcell @ 375 ms each (200 + 175) for a total of 4875 ms. The bar of heads moves four times for each of the first 14 strips and once for the last strip for a total of 57 times @ 95 ms per move for a total of 5415 ms.
      450 + 4,875 + 5,415 = 10,740 ms
10. a. (10,000 x 25)/60,000 = 4.2 min
    b. (10,000 x 25)/60,000 = 4.2 min
    c. (10,000 x 34)/60,000 = 5.7 min
    d. (10,000 x 17.5)/60,000 = 2.9 min
    e. (10,000 x 17.5)/60,000 = 2.9 min
    f. (10,000 x 50)/60,000 = 8.4 min
11. All except the name should be recorded in packed decimal format to conserve storage and to provide the proper format for arithmetic and editing operations.

	Characters	Format	Bytes
a. Employee number	6	Packed	4
b. Name	18	Zoned	18
c. Number of dependents	2	Packed	2
d. Social Security number	9	Packed	5
e. Salary	6	Packed	4
f. YTD gross earnings	7	Packed	4
g. YTD withholding tax	6	Packed	4
h. YTD FICA	5	Packed	3
Total	59		44

Chapter 3

12. a. OK – total of eight access mechanisms.
    b. Impossible. No 2311s may be attached with two 2303s.
    c. OK – total of seven access mechanisms.
    d. Requires additional storage feature – total of 13 access mechanisms.
13. a. parity bit, byte
    b. Cyclic Check, area
14. Count Area and Data Area, Key Area
15. a. each Count Area
    b. Home Address, each Count Area
    c. Home Address, each Count Area, each Key Area, each Data Area
    d. Home Address, each Count Area
    e. Key Area and/or Data Area
    f. each Count Area
    g. each Count Area
16. a. 6
    b. 15
    c. 14 (KL + DL = 160)
17. a. 22
    b. 26 (13 blocks of 2 logical records each)
    c. 30 (6 blocks of 5 logical records each)
    d. 32 (4 blocks of 8 logical records each)
18. A blocking factor of 6 will allow 36 logical records per track.
19. 24 of the 75-byte records can be written on a track.
    The 200 character record requires 270 bytes.
    $$61 + \frac{537\ (200)}{512} = 270$$
    The last 75-byte record requires 75 bytes.
    Each of the other 75-byte records requires 139 bytes:
    $$61 + \frac{537\ (75)}{512} = 139$$
    $$\text{Number of records} = 1 + \frac{3625 - 270 - 75}{139} = 24$$

Chapter 5

20. A blocking factor of either 4 or 5 will allow for 20 logical records per track.
    A blocking factor of 5 provides a timing advantage for sequential processing, since fewer file operations will be required. Five, then, is the optimum blocking factor.
21. 10000/20 = 500 tracks/10 = 50 cylinders
22. .9 minutes
    Seek 50 times @ 25 ms = 1250
    Read 2000 blocks @ 25 ms = 50000
    51250 ms/60000 = .9 min.
23. 35.6 hours
    On the average, half the file must be read to locate each record.
    51250/2 = 25625 ms/record
    x5000
    128125 seconds/5000 records
    128125/60 = 2135 min/60 = 35.6 hours

Chapter 7

24. Minimum is 1 seek and 1½ rotations.
Additional time required if:
File has a master index.
Highest-level index is not searched in core.
Specified record is a non-first overflow.
Specified record is in the independent overflow area.

25. 20 logical records per prime track – 4 blocks, each consisting of 5 logical records.
Length of each block is 5 x 160 + 7 for key = 807.

26. 13 logical records per overflow track.
Overflow records are always unblocked. Length of each record is:
160 + 10 (link field) + 7 (Key Length) = 177

27. 8 full tracks for prime records.

28. 19 entries per track index – 2 for the prime records on the index track, 2 for each of the 8 prime tracks, plus the dummy entry.

29. 1862 bytes
Index entries have KL = 7 and DL = 10.

$$\text{Index entry} = 81 + \frac{537\ (17)}{512} = 98$$

19 x 98 = 1862

30. 10 prime records (two blocks) on each index track. 3625 (track capacity) – 1862 (used for index) = 1763 bytes.
Two blocks would require 1754 bytes: 827 for the last one and 927 for the other one.

31. 59 cylinders*
(20 x 8) + 10 = 170 prime records per cylinder
10,000/170 = 59 cylinders

*Note that the same file required 50 cylinders with sequential organization (question 21).

32. 60 entries in the cylinder index.
One for each prime cylinder plus the dummy entry.

33. Cylinder index requires two tracks (actually about 1 2/3).

$$\text{Index entry} = 81 + \frac{537\ (17)}{512} = 98$$

$$\text{Last entry} = 20 + 7 + 10 = 37$$

$$\text{Entries per track} = 1 + \frac{3625 - 37}{98} = 1 + 36 = 37$$

34. 1020 bytes required in core for cylinder index.
Cyclic Checks, gaps, and Address Markers are never read into core; the Count Areas are not required for the logic of searching the index.
Therefore the number of bytes required is enough to hold the Key Area and Data Area of each entry.
60 (7 + 10) = 1020.

35. No, a master index should not be specified.
It would take just as long to search the master index and then one track of the cylinder index as it does to search the two-track cylinder index.

36. 1.15 minutes*

Read – blocks of prime records:	2000
overflow records:	400 (approx.)
	2400 @ 25 = 60000 ms
Search for and read each pair of track index entries (10 x 59)	590 @ 12.5 = 7375 ms
Seek each prime cylinder**	59 @ 25 = 1475 ms
	68850 ms

68850/60000 = 1.15 minutes

*The same file required .9 minutes if sequentially organized (question 22).
**Minimum access time was used, since the file is located on adjacent cylinders.

37. 16.6 minutes

Seek – cylinder index:	75 ms
prime cylinder:	75 ms
Read – per pair of track index entries:	12.5 ms
pair of cylinder index entries:	12.5 ms
data record:	25 ms
	200 ms

200 ms x 5000/60000 = 16.6 minutes
Additional considerations:
1. Non-first overflows would take additional reads.
2. Since the cylinder index occupies two tracks, a a more accurate average would be one rotation rather than half a rotation per read of the cylinder index. This would add about one minute.
3. Since the file occupies about 61 cylinders, an average seek time of about 65 milliseconds is more realistic (see Figure 11). This would reduce the time by 1.7 minutes.

38. 10.4 minutes

Seek – prime cylinder:	75 ms
Read – pair of track index entries:	12.5 ms
pair of cylinder index entries:	12.5 ms
data record:	25 ms
	125 ms

125 ms x 5000/60000 = 10.4 minutes

39. 9.38 minutes

Seek – prime cylinder:	75 ms
Read – each pair of track index records:	12.5 ms
data record:	25 ms
	112.5 ms

112.5 ms x 5000/60000 = 9.38 minutes

Chapter 8

40. 84 cylinders*

    With 85% packing, approximately 11765 locations are required. Can write 14 data records per track, so 11765/14 = 841 tracks/10 = 85 cylinders.

*Note that the same file required 50 cylinders with sequential organization (question 21) and 59 cylinders with indexed sequential (question 31).

41. 17.5 minutes*

    Seek once
    @ 75 ms = 75
    Read 1.2 times
    @ 25 ms = 30
    105 ms x 10000/60000 = 17.5 min

*Note that the same job required .9 minutes with sequential organization (question 22) and 1.15 minutes with indexed sequential (question 36).

42. 8.75 minutes*

    105 ms x 5000/60000 = 8.75 min

*Note that the same job required from 9.38 to 16.6 minutes of disk time with indexed sequential (questions 37, 38, and 39). The difference in total job times between direct and indexed sequential may not be of great importance. If the time required to access and process one record is the most important consideration, the difference in timing may be critical, particularly in a teleprocessing system. Consider the disk times required to access one record for the two different organization methods:

Indexed sequential:	112.5, 125, or 200 ms
Direct:	105 ms

Note, however, that timing may not be the only consideration. The preinstallation planning and programming effort required, the frequency with which reorganization is required, and device independence are some other factors that will influence the choice of an organization method.

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)

GC20-1649-5

Direct Access Storage Devices and Organization Methods Printed in U. S. A. GC20-1649-5

Direct Access Storage Devices and Organization Methods Printed in U. S. A. GC20-1649-5