Disk System
Concepts and Programming
Programmer's Guide

Disk System
Concepts and Programming
Programmer's Guide

Preface

Before your System/3 Disk System arrives, much time will be devoted to planning. Some activities involved in this planning are:

1. Selecting and educating personnel.
2. Planning the physical site for the system.
3. Studying and documenting current applications.
4. Designing computer application.
5. Writing and testing programs.
6. Converting data files.
7. Documenting operating procedures.

As a programmer you may be involved in most or all of these activities. This book explains designing and programming computer applications.

This book assumes, then, that you understand the applications in your company and that you have read the *IBM System/3 Disk System Introduction,* Form C21-7510.

From the *Introduction* you should know the functions of five IBM System/3 Disk System devices: Multi-Function Card Unit, Processing Unit, Printer, Disk Storage Drive, and the Printer-Keyboard. You should also know three characteristics of the System/3 disk: cylinders, tracks, and sectors. You must know these functions and characteristics to efficiently design your computer applications.

Chapters 1 and 2 of this manual explain in detail the disk concepts you must know to design applications for System/3. These chapters specifically explain how to:

- Choose the organization of a disk file
- Design disk records, calculate the size of a disk file, and document this information

Chapters 3, 4, and 5 explain how to use disk files with:

- The RPG II language
- Operation Control language
- The Disk Sort program.

None of the topics in this book are completely described. Three manuals are available for further reference.

IBM System/3 Disk System Reference Manual, Form C21-7512

IBM System/3 Disk System Utility and Sort Programs Reference Manual, Form C21-7522

IBM System/3 Disk System RPG II Reference Manual, Form C21-7504

First Edition

Changes are continually made to the specifications herein; any such change will be reported in subsequent revisions.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Programming Publications, Department 425, Rochester, Minnesota 55901.

Contents

This chapter is composed of four parts that describe various characteristics of disk files. The first part explains two ways to organize disk files: sequential and indexed. The second part describes the several ways indexed files can be processed. The third part explains several functions involving disk files:

1. Creating a file
2. Adding records to a file
3. Deleting records from a file
4. Updating records in a file
5. Reorganizing a file

The fourth part presents factors to consider when choosing one of the organization methods.

A thorough knowledge of these topics is needed to understand Chapter 3 which explains how to use RPG II for disk files.

ORGANIZING A DISK FILE

An important part of any data processing job is *file organization.* File organization is the arrangement of records in a file. The purpose of this section is to explain how disk files are organized. Since some disk files are organized like card files which you are familiar with, let's briefly review their organization.

Cards are normally arranged in a file in a particular sequence. For instance, a personnel file might be in alphabetic sequence by employee name, or it might be in numeric sequence by employee number. Data such as employee name and employee number are control fields. Recall that control fields determine the order of the cards in a file.

When card files are processed, cards are read one after another in order. To process only certain cards, all cards must be read. Thus, it takes longer to process a few cards in a large file than to process a file containing only the desired cards. Card sorters and programs, such as the Card Sort/Collate program, can be used to decrease card processing time by selecting only the desired cards and then processing them.

Also, if a card file is needed in one sequence for one job and another sequence for another job, the file must be sorted for each job. For example, salesmen receive a monthly report of their commissions. To produce this report the monthly transaction cards must be sorted in salesman number sequence. Then when the monthly sales report (the amount of sales for each item) is produced, the same transaction cards must be sorted by item number. The same file is used for both reports, but it is arranged in two different orders according to two different control fields: one by salesman number and one by item number.

Sequential Disk Files

A disk file can be organized and processed like a card file. Such a disk file is called a *sequential* file.

When a sequential disk file is processed like a card file, records are processed one after another in the order they occur. An example of a sequential disk file is an employee master file. This file contains information needed for various reports concerning each employee, such as payroll checks. Since checks are usually produced in order by employee number, records are processed in order. The lowest employee number is processed first and so on until the last record, the highest employee number, is processed.

Why, then, should a sequential file be placed on disk instead of cards? The major advantages are storage and speed. A disk can hold the same amount of data as 25,600, 96-column cards. Not only can large amounts of data be stored on a disk, but one disk pack is more convenient to handle than many cards. The disk storage drive also provides faster processing of records than the MFCU.

Sequential disk files, such as the employee master file, are processed *consecutively.* Consecutive processing means records are processed one after another in the order they occur. To process only certain records all records must be processed, or at least read. In this case, as with cards, consecutive processing can be time consuming.

Indexed Disk Files

It would be helpful, then, if disk records were available like books in a library where you go to an index, find the location where the book is stored, go to the right shelf, and get the book you want. No one would read all books in the shelves before reading the desired book. Likewise, it would

be desirable to skip the records not needed in a job and process only the desired ones. Disk files can be organized to overcome the limitation of consecutive processing; they can be *indexed.*

An indexed disk file is organized somewhat like the books in a library; it has two parts, an *index* and the records. The index contains two facts about each record in a file. First the contents of the record's *key field* appear in the index. A key field contains data that uniquely identifies a record. For example, customer number may be the key field for a customer master record. Then, the *disk address* follows the key field. The disk address represents the location on the disk where the record is stored. For each record, then, there is an *entry* in the index that contains

An index contains the same number of entries as there are records in the file. A file with 2000 records has 2000 entries.

Ordered and Unordered Records

The records in an indexed file or a sequential file can be *ordered* or *unordered.* An ordered file means the records are arranged in order according to some major control field or by frequency of use. An unordered file means the records are not in any particular order.

For example, a wholesale distributor organizes his file of inventory items as an indexed file. The records are loaded on the file by frequency of use (unordered). Thus, the most active items are at the beginning of file. However, the index is created in ascending order by item number. When the file is used to write customer orders, most of the records needed are located in a small area of the file. The total time to process the orders is less than if these records were scattered throughout the *entire* file.

PROCESSING INDEXED FILES

As stated previously, indexed files overcome the limitation of consecutive processing which is often used with a sequential file. Indexed files can be processed several ways because the index provides several ways to find records.

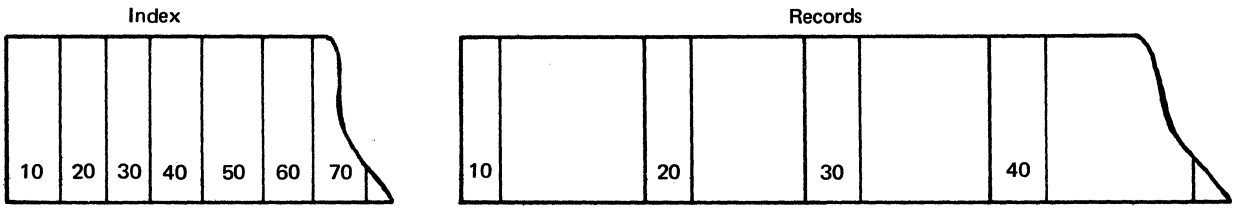
Sequential Processing

When an indexed file is processed sequentially, the keys are processed one after the other in ascending order. If the records are not in order on the disk, they can be processed in order by using the index. There are two ways to sequentially process an indexed file.

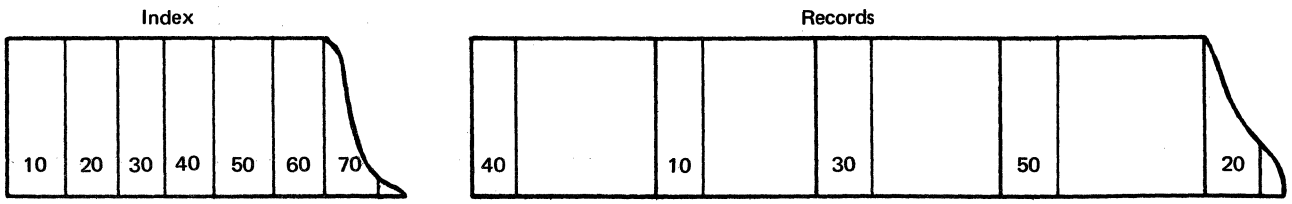
Sequential by Key

One way to sequentially process an indexed file is *sequentially by key.* Sequential by key means processing records in the order of the keys. This method is used to process *all* records in a file regardless of their order. To illustrate this processing method, note the differences and similarities between File A and File B.

FILE A



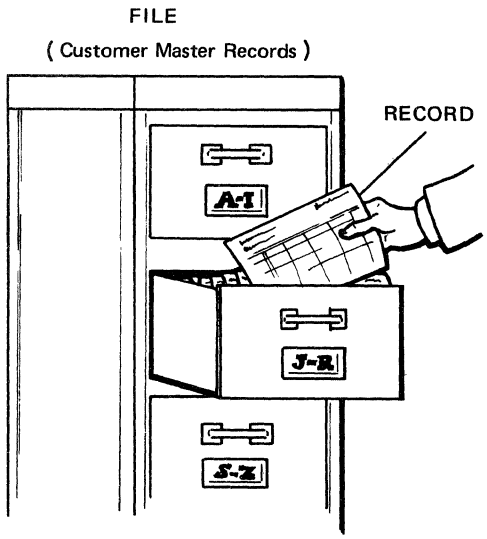
FILE B



Both files contain the same records. Both indexes are in order according to key field. The only difference between the two files is the order of the records. The records in File A are in order according to key field. The records in File B are unordered. *All* records in both files can be processed *in order* by specifying sequential by key. (You specify processing method in your RPG II programs.)

Sequential Within Limits

The second way to sequentially process an indexed file is *sequential within limits.* Sequential within limits is like processing all the records one after another in order, in a filing cabinet drawer that is marked to identify that group of records.



Sequential within limits processing is best suited to jobs where records are processed in groups at specified times. For example, a wholesale company prepares monthly statements of each customer's charges. Statements are written on the tenth, twentieth, and last day of each month. On the tenth, statements are written for all customers whose names are in the first part of the alphabet (A-I). Statements for the other two parts, J-R and S-Z, are written on the 20th and the last day, respectively. This type of statement writing is called *cyclic.*

Sequential within limits can also be used to process several groups of records at one time. Sequential within limits is not illustrated in Chapter 3, *RPG II Disk Programming.* If you must use this capability, refer to *IBM System/3 Disk System RPG II Reference Manual,* Form C21-7504.

Random Processing

Indexed files can also be processed randomly. This type of processing also uses the index and is called *random by key.* This method permits processing of one particular record without regard to its relation to other records. You must specify the key of the record (through RPG II). The key is then found in the index; the disk address (adjacent to key) is used to locate the record; and the record is transferred to storage for processing. For example, records in a customer master file are to be updated to reflect current information. The transaction cards are not in order. The disk record is found by matching the customer number in the card with the key (customer number) in the index. The disk address, adjacent to this key, is then used to find the record.

Often an indexed file is used in several different jobs each of which requires a different processing method. For example, during statement writing, a customer file may be processed sequentially within limits to allow cyclic statement writing. During a billing job, the same file may be processed randomly by key to allow updating master records with unsorted transaction cards. Then during an aged trial balance job (each customer's outstanding balance is printed), the file may be processed sequentially by key.

REVIEW – FILE ORGANIZATION AND PROCESSING

So far you have studied two ways to organize a disk file: sequential and indexed. You also learned that a sequential file can be processed consecutively, one record after another in order, and that an indexed file can be processed sequentially and randomly. Sequential processing can be by key or within limits. Random processing is by key.

CREATING AND MAINTAINING DISK FILES

Before a disk file can be processed, it must be written on a disk. The first time a file is written on the disk, the process is called *loading* or *creating.* A card file can be used to create a disk file. You record in cards the data you want on the disk. Then you write an RPG II program that transfers the data from the cards to the disk. Chapter 3 explains how to write such an RPG II program.

Once a file is created, *file maintenance* is often necessary. File maintenance means performing those functions that keep a file current for daily processing needs. Some file maintenance functions common to both sequential and indexed files are: *adding, deleting,* and *updating* records. Adding means putting a record in a file after the file is created. Deleting means identifying a record so it won't be processed with the other records. Updating means adding or changing some data in a record.

Creating a Sequential File

A sequentially organized disk file is similar to a sequentially organized card file. The records are either ordered, in ascending or descending order by some control field, or unordered. However, regardless of order when a sequential file is created, records are placed consecutively, one after another on the disk. Both tracks in one cylinder are filled, then both tracks in the next cylinder, until the whole file is placed on the disk. Figure 1 shows this process.

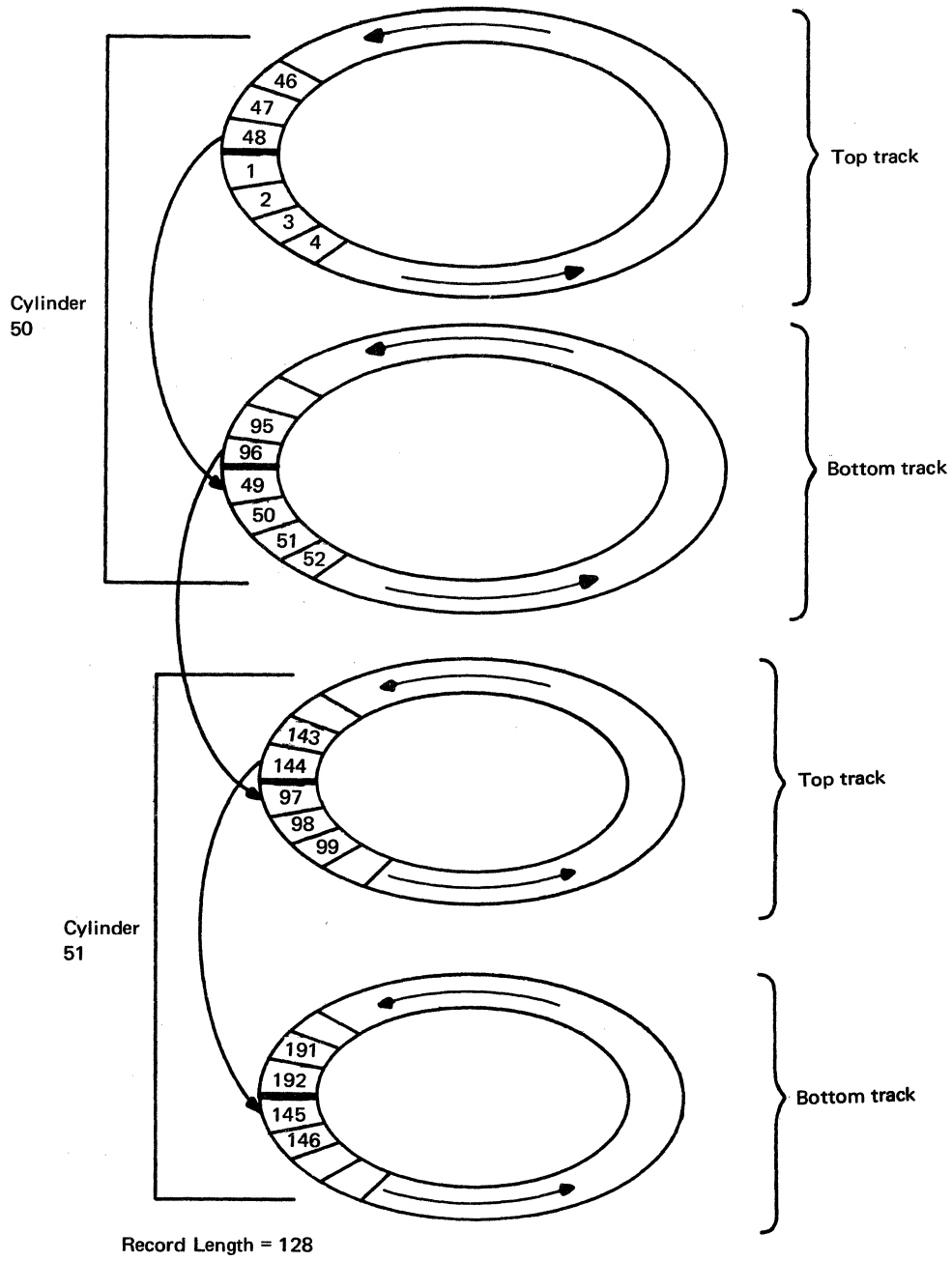
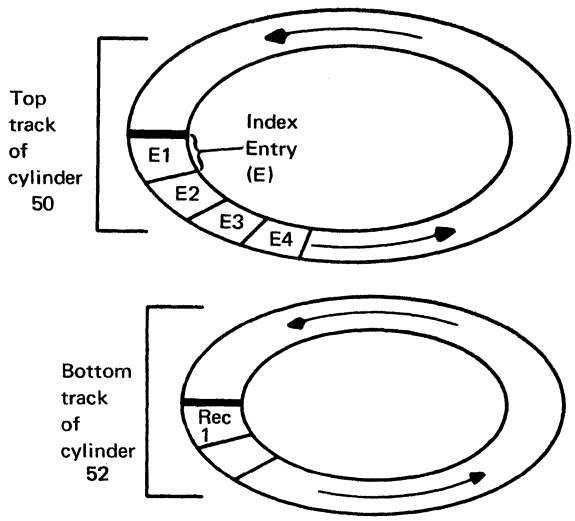


Figure 1. Writing Records on a Disk

The numbers on the tracks in the figure correspond to the number and position of each record. Record 1 is written at the beginning of the top track of cylinder 50. Records 2 through 48 are written, in order, on the top track of cylinder 50. Then, records 49 through 96 are written, in order, on the bottom track of cylinder 50. When cylinder 50 is filled, cylinder 51 and subsequent cylinders are filled in the same manner.

In this example, each record is 128 positions long. Recall that a sector is 256 positions long, and that a track contains 24 sectors. So, in this example, two records are written in each sector; 48 records are written on each track; and 96 records are written on a cylinder.

The 97th record is written at the beginning of the top track of cylinder 51. The next 95 records are written on cylinder 51. Ninety six more records are written on cylinder 52, cylinder 53, and so on until all records are written on the disk.



The index for a certain file takes five tracks. So index entries are written on cylinders 50 and 51 and the top track of cylinder 52.

Records are written beginning in the first sector of the bottom track of cylinder 52. Both the index area and the record area must start at the beginning of a track.

Creating an Indexed File

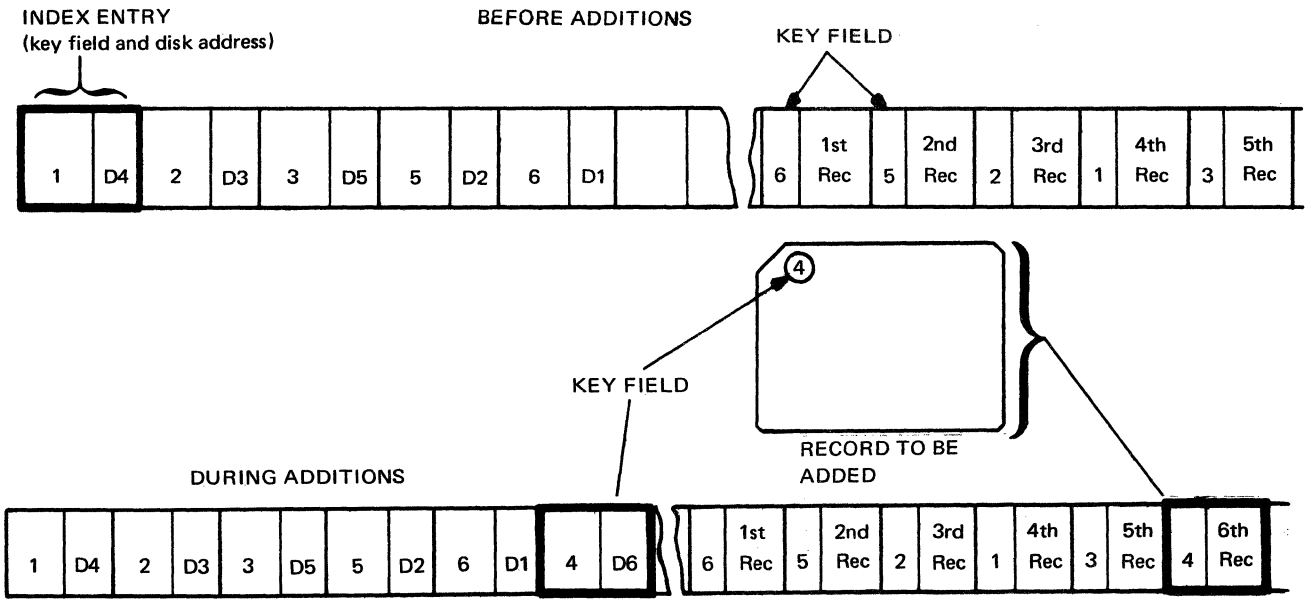
When an indexed file is created, the index is created as the records are written on the disk. The index area precedes the area where records are placed. In an indexed file, the records are either ordered or unordered, but the index contains key fields which are *always* ordered.

Adding Records to a Sequential File

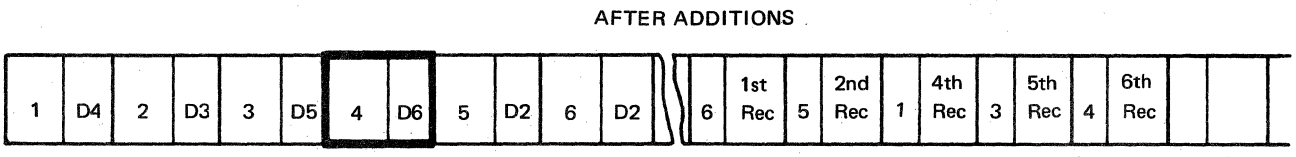
When records are added to a sequential file, they are written at the end of the records already in that file. Records are so added whether the records already present are ordered or unordered.

Adding Records to an Indexed File

When a record is added to an indexed file it is also written at the end of the records already in that file. The index entry (key field and disk address) for that record is written at the end of the current entries in the index area.



After all records are added, the index is automatically sorted so the keys of the new records are in ascending sequence.



Deleting Records

When a record becomes inactive, you will no longer want to process it with the other records. A record is not physically removed from a file during regular processing; thus, it is necessary to identify or tag the record so it can be by-passed. One way of tagging such a record is to put a code in a particular location in the record. This code is referred to as a *delete code.* When the file is processed, your program must check for the delete code; if the code is present, the program bypasses that record. The delete procedure is the same for both sequential and indexed files.

Updating Records

The update function can also be the same for both indexed and sequential files. The record to be updated is read into storage. The data to be changed is changed in storage, and then the record is written on the disk in its original location.

Reorganizing a File

Another file maintenance function performed on indexed files is *reorganization.* Reorganizing means re-creating an indexed file. Reorganization of an indexed file may be necessary for two reasons: to increase processing efficiency and to free disk space.

For example, an indexed file was created with the records in ascending key field order. Since that time several records were added to the file. Recall that these records are added at the end of the file, but the keys are in order in the index. When this file is processed sequentially by key, the disk access arm must move back and forth between the sequenced records (those originally created) and the added records. Often processing time for a particular job increases in this situation. During reorganization the added records can be placed in sequence in the re-created file.

Another reason to reorganize a file is that often the space reserved for a file becomes filled as additions are made. (Planning file space is discussed in Chapter 2). Reorganizing is a means of freeing space since inactive records, those with a delete code, can be physically removed from a file during reorganization.

Reorganization, then, is a means of physically placing added records in sequence with the records originally created. It is also a means of physically removing records that are tagged for deletion. The IBM Disk Copy/Dump program can be used to reorganize an indexed file. The *IBM System/3 Disk System Reference Manual*, Form C21-7512, explains how to use the program to reorganize indexed files.

REVIEW – FILE MAINTENANCE

This section explained file creation and maintenance. The file maintenance functions explained were: adding records, deleting records, updating records, and reorganizing a file. Creating and reorganizing a file are usually performed independently of other processing. Adding, deleting, and updating can occur separately, but most usually occur during processing of a specific job. Examples explaining how these functions are coded in RPG II are contained in Chapter 3.

CHOOSING A FILE ORGANIZATION METHOD

The first two sections described several characteristics of the System/3 Disk System and explained the flexibility they provide to perform a variety of jobs. Due to this flexibility and variety, it is important for you to analyze each of your jobs and choose the file organization method that gives you the best possible performance.

In many cases, the type or organization for a file is immediately evident. Some applications, however, may require additional study because of their complexity, because a file is used in several jobs, or because unusual processing is required. Studying existing applications is an important aspect of planning for a data processing system. Decisions in this area must be made *before* programming begins, since efficiency of your data processing installation may be affected. This section described factors to consider when making these decisions.

There are no absolute rules for choosing a file organization method. However, several characteristics of the file to consider are:

1. Use of the file
2. Volatility of the file
3. Activity of the file
4. Size of the file

Each of these four characteristics is discussed next. Following these discussions are two examples. One example explains why a sequential file is chosen; the other describes why an indexed file is chosen.

How is the File Used?

The answer to this question takes priority over all other considerations. To answer this question you must ask several other questions.

First, *is the file a master file?* Recall that a master file is fairly permanent, is generally used in several jobs, and is often used with several other files. For example, a customer file contains a record for each customer. Each record may contain such data as customer name and address, shipping information, credit status, accounts receivable, and sales information. Although certain data in a record such as accounts receivable, may change, the record remains in the file as long as the customer does business with the company. (These changes are made with a transaction file.) Since this file contains so much information about each customer, it may be used in several jobs to produce various reports. Likewise, the file may be used with several other files, master or transaction.

Recall that a transaction file contains records of a less permanent nature than a master file, and transaction files contain data that is used to update a master file.

Thus, when choosing a file organization method for a master file, the major question to ask is: *What are the processing requirements of the file*? To answer this question, you must study the applications in which the file is used and ask:

- Is the file used with other files or in several jobs?
    1. If so, what is the organization of the other files?
    2. If used with transaction files, are the transaction records ordered or unordered?
- Must the file be sorted for any jobs?
- Must the file provide for inquiry?

Using a Master File with Several Files or in Several Jobs

If a master file is used with several files (a transaction file, another master file, or both), the master file can be either sequential or indexed. The determining factors are the processing requirements of the various runs that will be using the file and the organization of the other files.

If transaction files are ordered (sorted in the same sequence as the master file), then the master file may be either sequential or indexed. However, to process unordered transaction files against a master file, the master file must be indexed and processed randomly by key.

If the master file is used in several jobs, and records must be processed both in order and randomly, then indexed is the better organization. Records can be processed in order (sequential by key) and randomly (random by key).

Sorting a Master File

If the master file must be sorted for some jobs, a sequential file is easier to sort than an indexed file. The Disk Sort program cannot produce a sorted indexed file. That is, an indexed file can be sorted; but the sorted file will be a sequential file. Instead of keeping the sorted file as the master file, the original file must be kept. Chapter 5 explains the Disk Sort program in more detail.

Inquiring Against a Master File

Most businesses need to get information from a file on an *inquiry* basis. An inquiry is a request for information from some type of storage.

Some jobs that emphasize the importance of immediate inquiry and response are:

Demand Deposit Accounting	*What is the balance of account number 133420?*
Inventory Control	*How many of part number 55632 are on order?*
Manufacturing	*How many sub-assemblies of part number 16414 are on hand?*

Payroll	*What are the year-to-date earnings for employee number 13862?*

The System/3 Disk System provides for inquiry. The ability to obtain the desired information and the speed with which it can be obtained depend upon the organization of the file.

Where inquiry is required, a critical question in choosing the best file organization method is: *How fast must the inquiry be answered?* The less critical the response time, the greater the choice of organization and processing methods.

To decide how fast the inquiry must be, ask a series of questions. *Can the answer to the inquiry wait until the next updating of the specific master file?* If it can, then these inquiries can be treated as additional transaction records and so processed. File organization, in this case, could be either sequential or indexed depending on other processing needs. If the inquiry cannot wait, additional questions must be asked.

Can the answer wait until the end of the present computer run? If so, the disk pack containing the specific master file is mounted at the completion of the current job; the inquiry program is loaded; and the file is processed to produce the required answers. Obviously, response time varies considerably depending on (1) the job that is in progress when the inquiry arrives and (2) the organization of the file inquired against.

An indexed file processed randomly by key will usually provide the best response time. (If the desired record were the first record in a sequential file, the response is also fast, but this type of inquiry is rare.)

Review—File Use

You must consider several factors to answer the question: How is the file used? Specific answers are impossible without knowing the specific file and the applications that use it. However, if you consider the processing requirements of the file, you have some guidelines for arriving at specific answers.

How Volatile is the File?

The number and frequency of records added to or deleted from a file affect the type of file organization chosen. *Volatility* refers to number of additions and deletions. High volatility means many records are added and deleted; low volatility means few records are added or deleted.

Recall that for both sequential and indexed files records are added at the end of the current records. If a file is sequential and the control fields of the added records are higher than the last record on the file, additions cause no problem. However, if they are not higher, and processing of the file depends on the records being processed in control field order, additions do cause a problem. In this situation, records added at the end of the file would be out of sequence.

To avoid this problem, the disk file can be re-created when such additions are made. To re-create a sequential file you write an RPG II program. Job 4 in Chapter 3 illustrates this process.

However, if additions are made to an indexed file, there is no need to rewrite the file. Records are also added at the end of the file, but the keys are in ascending order in the index. Thus, if the records must be procedded in order, they can be processed sequentially by key. Thus, one of the advantages of an indexed file is that additions and deletions can be handled without rewriting the file.

However, as the number of additions increases, the efficiency of sequentially processing an indexed file decreases. Sequentially processing the added records by key requires more time than processing the records from continuous positions on the disk. This increase occurs because additional access arm movement is required to read records at the end of the file. The arm must move back and forth between the index and the records. Even if the original records are in sequence, the added records are not. The arm must make one additional move for each added record that is processed.

Thus, for a highly volatile file where records must be processed in order, a sequential file with consecutive processing is best. However, if a highly volatile file does not require processing records in order, the file can be indexed and processed randomly by key.

If a highly volatile file requires both sequential and random processing, an indexed file is best. In this case, to overcome the decrease in time due to arm movement, the file can be reorganized.

Recall that reorganization means building a new indexed file from the old one by physically merging added records in key order and excluding all records tagged for deletion. Processing with the reorganized file is highly efficient. As additions and deletions occur, this efficiency gradually diminishes, until you reach a point where reorganization again becomes advisable.

Review – Volatility

The critical questions concerning volatility are:

1. What are the processing requirements for the file?

2. Is volatility high or low?

Possible answers are:

1. If all records must be processed in order and volatility is high, a sequential file is suggested.

2. If only certain records are processed and volatility is high, an indexed file (random by key) with planned reorganization is suggested. If this file is used in other jobs where records are needed in order, it can be processed sequentially by key.

3. If volatility is low, an indexed file is better. There is no need to reorganize this file as often as a highly volatile indexed file. To process all records, the file can be processed sequentially by key. Also, an indexed file allows particular records to be processed (sequential within limits or random by key).

As you can see several factors influence the answers to the two questions. It is impossible to provide specific answers. Only with a well-defined file and well-defined applications can specific answers be given.

How Active is the File?

The next important consideration, after volatility, is the activity of the file. *Activity* refers to the number of records in a file for which there are transactions. Activity is usually expressed as a percentage: for example, 10 percent of the records in a file. Disregarding other processing needs, the higher the activity the better the file is suited to sequential organization. (A high percentage indicates high activity.)

For example, activity on 100 cards out of 1000 in an inventory file means that, at any one time, there are transactions to be posted to 10 percent of the records in the file. As activity increases, consecutive processing becomes more efficient. This implies a sequential file with consecutive processing or an indexed file processed sequentially by key. An indexed file may be necessary for a high activity file, if other processing needs so justify. Low activity may justify an indexed file processed randomly by key.

For an high activity file, you should consider *batch processing.* This means the application does not require transaction records to be processed the moment they occur; some time lag is all right. Transactions can be accumulated, or batched, and processed at certain times. The time lag may be hours, weeks, or even months, depending on the application.

Generally, you may assume that low activity justifies an indexed file processed randomly by key, and high activity justifies a sequential file processed consecutively.

What is the Size of the File?

You must consider on-line capability. Recall that on-line means data is available on the disk for processing. The size of a file is affected by two important file-organization characteristics:

1. A sequential disk file can be written on any number of disks, which can then be mounted and processed consecutively.

2. An indexed file must be entirely on-line.

The fact that indexed files must be entirely on-line imposes physical restrictions on maximum file size. If you must periodically reorganize an indexed file, you must have enough on-line disk space to contain the old file and the new file. If a disk file exceeds the size limitations imposed by indexed organization, you can organize it as a sequential file. However, if other factors require the file to be indexed, you can punch the new file on cards and then create it on disk from the cards, instead of using the Disk Copy/Dump program to reorganize.

Sorting a File

If a file will be sorted by the System/3 Disk Sort program, the size of the file also affects the choice of a file organization method.

The System/3 Disk Sort program uses disk *work areas.* A work area is space on the disk that the program uses to arrange records in the specified order. The size of these work areas must be considered when planning files that need sorting.

When a sequential disk file is sorted, the maximum size of the input file is a little less than half the total on-line disk storage capacity. On a 1-spindle system, the input file can be overlaid by the output file. In this case, the input file is not preserved. On a 2-spindle system, half the total on-line capacity is one pack. The pack that contains the input file can be removed before the sort program starts writing the output file. Another pack can be mounted, and in this manner the input file can be preserved.

If the size of the file to be sorted appears too large for a particular system, another way to sort the file is available.

Instead of sorting the records, an *ADDROUT* sort may be performed. An ADDROUT sort produces a file of relative record numbers. The relative record number can be used by an RPG II program to specify the location of a record in the disk file. The record numbers for a file are sorted into the sequence specified by the control fields. These numbers are written on the disk. They can be used as input to an RPG II program that processes the records in the desired sequence.

The ADDROUT sort offers three advantages:

1. The original file is preserved.
2. The work and output areas must only be large enough to provide space for the record numbers, not for the records.
3. Random processing of a sequential file is possible through RPG II.

REVIEW – CHOOSING A FILE ORGANIZATION METHOD

To choose a file organization method, many factors must be considered. Four major questions must be asked.

1. How is the file used?
2. How active is the file?
3. How volatile is the file?
4. What is the size of the file?

Each of these questions pose additional questions. The four major questions and subsequent questions are not independent. Many times all four factors are important, and often they are equally important. The choice is yours. Only you know the requirements of your installation.

EXAMPLES

The following two examples show how the factors just discussed apply to specific file characteristics and which file organization method is best under these assumed conditions.

Example 1

File: Customer Master File

Characteristics:

1. Applications that use the file:

   Billing—prepared as items are shipped

   Statement Writing—prepared monthly in customer number order

   Aged Trial Balance Report—prepared monthly in customer number order

2. Volatility—30 new records a week; 25 records tagged as deleted.
3. Activity—20 percent highly active records; 10 percent inactive records; 70 percent slightly active records
4. File Size—2000 records at creation time

Since orders occur randomly, the billing job requires the ability to process the file randomly by key. Statements and

the Aged Trial Balance Report are printed in customer number order, so sequential processing is required. Since an indexed file can be processed both sequentially by key and randomly by key, this file lends itself to the index file organization method.

Since activity is low, only 20 percent of the records are processed during billing, an indexed file is better than a sequential file. Recall that records in a sequential file must be processed in order one after the other.

Since 30 new customers are added each week, at the end of three months the file will contain about 2400 records.

30 X 4 = 120 records a month

120 X 3 = 360 records in three months

2000 records at beginning
360 added records in three months

2360 records at end of three months

At the end of the fourth month, 2480 records would be in the file (2360 + 120 = 2480). If at creation time, file size is defined as 2400 records, the fourth month poses a problem: what to do with the 80 extra records? Since 300 inactive records can be deleted during reorganization, the volatility of this file requires quarterly reorganization.

File size for 2400 records is approximately 56 tracks; so size is not a problem if the file is an indexed file and it is reorganized every three months. (Chapter 2 explains the calculations needed to determine exact file size.)

Recommended File Organization: Indexed; quarterly reorganization.

Example 2

File: Employee Payroll (Hourly and Salary)

Characteristics:

1. Applications that use the file:

   Payroll Checks--printed in ascending order by employee number

   Payroll Register--printed in ascending order by employee number

2. Volatility--five new employee records a month

3. Activity--98 percent active records; 2 percent inactive records

4. File Size--240 records

Payroll checks and the Payroll Register are printed in ascending order. Employee numbers are assigned consecutively; as a new employee joins the company, he is assigned the next available employee number. Thus all additions are placed at the end of the file, but they are in order.

Since the reports are printed in ascending order, and additions occur in order, the file can be processed in order. High activity and low volatility suggest a sequential file.

Recommended File Organization: Sequential with consecutive processing.

After deciding which file organization method to use, you should design the record and determine file size and location. This chapter explains how to design records and determine file size and location.

DESIGNING A RECORD

The applications that use a certain file determine what data is needed in a record. You should study these applications and then decide the *layout* of the record. Layout means the arrangement of fields in a record. When you design a record, you must consider processing requirements of the record and then determine field length, location, and name.

To illustrate these design considerations, a name and address file is used in this chapter. Each record in the file contains the following data:

Field	*Size (number of positions)*
Customer Number	6
Name	20
Street Address	20
City and State	20
Record code	2
Delete code	1
(Other fields)	47 (total)
	116 TOTAL

Determining Field Size

Field size depends on the nature of the data in the field. First, the length of the data may vary. In the example, name is 20 positions. The length of each customer's name varies, but 20 positions should be sufficient for all names. Secondly, all data in a field may be the same length. For example, customer number is six positions, and all six positions are used in each record.

There are no firm rules for determining field size. The major problem involves fields with variable length data. For example, if name is planned as 15 positions, and a new customer has 19 characters in his name, a problem arises when adding his record to the file. To avoid this problem, try to estimate the largest length of the data that will be contained in a field. Use this length to determine field size.

Providing for a Delete Code

Recall that records are not automatically deleted. You must place a delete code on a record with an RPG II program. Then when the file is processed, you must check for this code with an RPG II program. In the example, if a customer becomes inactive, we don't want to process his record. Thus, a 1-position field is included to provide for a delete code.

Providing Extra Space

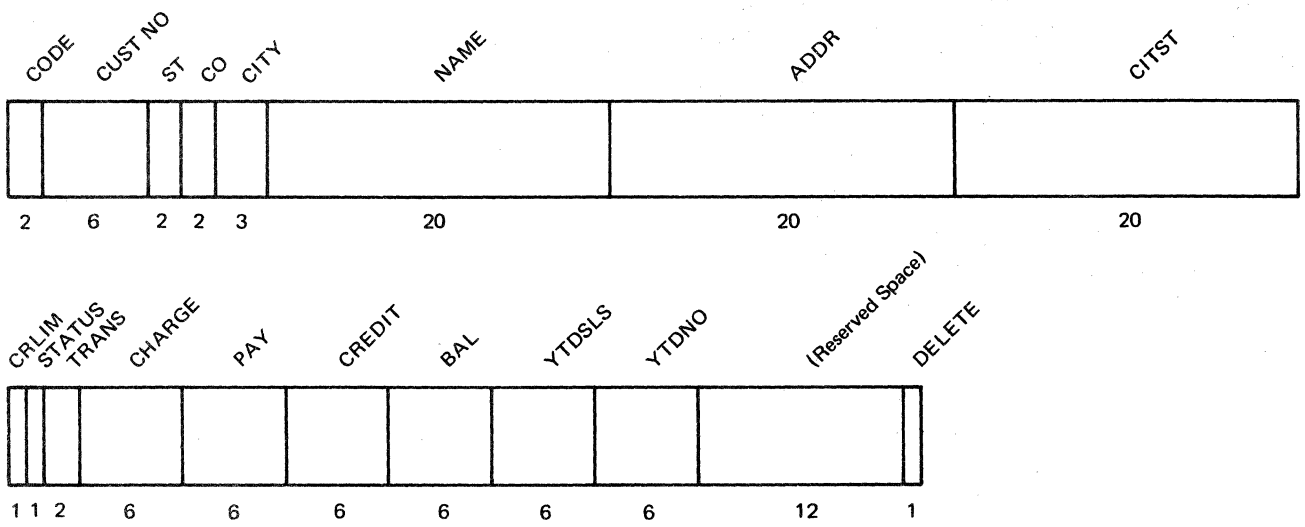
At this stage in planning, it is often wise to allow for data to be added to a record. For example, suppose this name and address file were created with the fields described, and at a later time each customer's zip code is needed. If all positions in the record are used, there is no place to add zip code. Since record length is not yet established, we can allow for such additions to this record. Although it is often difficult at the planning stage to imagine what data might be added, it is wise to reserve extra space.

Naming Fields

At the same time you are determining field size and location, you can also decide on names for each field. Since you must specify field names in your RPG II programs, it is a good practice to choose names that follow the RPG II rules for forming field names. If these rules are considered at this planning stage, your RPG II programs are easier to write.

An RPG II field name can be from one to six characters long. The first character must be an alphabetic character, but the remaining characters can be any combination of alphabetic or numeric characters. Blanks and special characters are not allowed. The field names in Figure 2 follow these rules.

One other consideration is important when choosing field names: the name should be meaningful. Since field names are restricted to six characters, abbreviations are often



CODE = Record code (CM)
CUSTNO = Customer number
ST = State code
CO = County code
CITY = City code
NAME = Customer name
ADDR = Customer address (street)
CITST = Customer address (city and state)
CRLIM = Credit limit

STATUS = Status code
TRANS = Number of transactions this month
CHARGE = Current month charges
PAY = Current month payments
CREDIT = Current month credits
BAL = Balance
YTDSLS = Year-to-date sales amount
YTDNO = Year-to-date number of sales
DELETE = Delete code

Figure 2. Layout of Customer Master Record

necessary. For example, the word *address* has seven letters; it is shortened to ADDR in Figure 2. Meaningful field names contribute to better documentation, and often avoid misinterpretation or confusion while writing RPG II programs.

DOCUMENTING RECORD LAYOUT

When record layouts are documented, your RPG II programs are easier to write. Figure 2 shows the layout of a customer master record. A record layout should include the order of the fields in the record, the length of each field, and the name of each field.

Record Length

A record may contain all pre-defined fields, or space may be reserved for data to be added to the record. In either case, all records in a particular file must be the same length. In your RPG II programs you must specify *record length.* Record length is the sum of the field lengths (including reserved space).

In our example, the sum of the fields is 116 positions. However, record length is established at 128, thus reserving 12 positions for data that might be needed at a later time.

Block Length

In your RPG II programs, you must also specify information about *blocks.* A *block* is the number of records that is transferred between a disk file and the processing unit (input) or between the processing unit and a disk file (output). Although only one record at a time is available for processing by your program, one or several records may be transferred at one time. Transferring blocks of records can decrease the time required to perform a job. When records are transferred one at a time, access time is required for the disk access arm to locate each record. When several records are transferred at a time, access time is usually less.

When more than one record is transferred, the records are *blocked.* Transferring blocked records can result in more rapid processing. When only one record is transferred at a time, the records are *unblocked.*

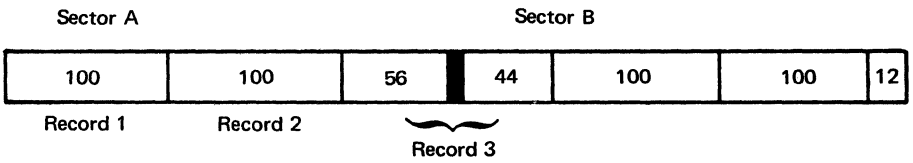
You may want to use unblocked records when an RPG II program takes a large amount of storage. In this case, there is a possible increase in total time to do the job, but your program will fit in storage.

Block length is a *multiple* of record length. For example, if your record length is 64, block length could be 256 (64 X 4 = 256). Block length is four times as large as record length. The multiple, 4 in this case, indicates the number of records you want transferred at one time.

The design of the System/3 Disk System influences block length. Recall that the smallest division of a disk is a sector, and it can contain up to 256 characters. The system transfers data in sectors, that is, multiples of 256 characters. If your record length is 128, you might have a block length of 256, indicating that you want two records transferred (128 X 2 = 256). Or you might have a block length of 512, indicating that four records are transferred (128 X 4 = 512).

For efficient blocking, you should choose a record length that is a multiple of 256 (256 X 2 = 512) or *sub-multiple* of 256. A sub-multiple is a number that divides into 256 a whole number of times. For example, 64 is a sub-multiple of 256 (256 ÷ 64 = 4).

You *can* specify a record length that is not a multiple or sub-multiple of 256. The system allows you complete flexibility in choosing a record length to fit your application and your disk storage capacity. When you use a record length which is not a multiple or sub-multiple of 256, no *disk storage* is wasted; some records will simply reside in more than one sector.

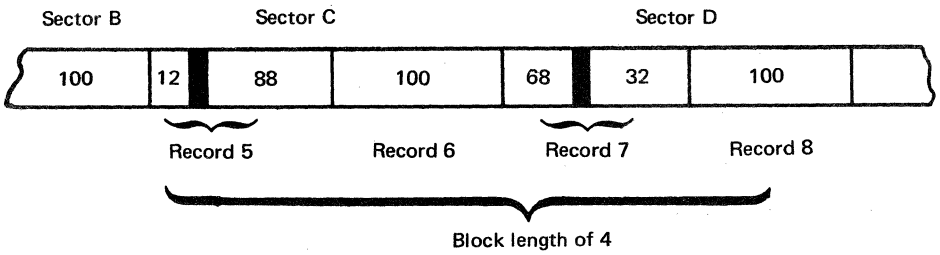


However, when you specify 100-character records as shown in the example, the computer requires more *main storage* to process these records.

You recall that the system always transfers data from disk to the computer in increments of sectors. To process record 3 therefore, two sectors must be in main storage, sector A and sector B. Fifty-six characters of record 3 reside in sector A; the remaining 44 reside in sector B. Thus, to process 100-character records with a block length of 1 requires that 512 characters (two sectors) be available in main storage.

As another example, suppose you specified 100-character records with a block length of 4. Four 100-character records *can* span three sectors.

To process your records in this case requires 768 characters (three sectors) in main storage.



REVIEW – DESIGNING RECORDS

When you design a record, consider the following questions:

1. Does the record contain all fields needed for all applications that use the file?
2. Should the record provide for a delete code?
3. Is extra space reserved for fields to be added?
4. Are the records blocked? If so, does the block length take into consideration that data is transferred in multiples of 256 characters?
5. Is the record layout well-documented?
   a. Are field location and size clearly marked?
   b. Do field names follow the RPG II rules for forming field names?

DETERMINING SIZE AND LOCATION OF A DISK FILE

Another aspect of the planning stage is (1) determining how much disk space a file requires and (2) deciding where to locate the file on the disk. These two factors must be considered together since they directly affect each other. For example, two files are already written on a disk on cylinders 8-155. A third file is to be created; it will occupy 55 cylinders. Since a disk contains 200 cylinders, this file is too large to be contained on this disk (155 + 55 = 210). The file must be written on another disk.

Determining Number of Records in a File

Let's first consider the disk space required for a file. To determine this space, you must plan how many records will be in the file at a specified time.

To determine the number of records in a file, you must consider several factors. First, you must know how many records will be in the file when it is created. If the file already exists, perhaps as a card file, use the number of records in this file as a base.

You must also know if records will be added or deleted. If additions are expected, how many records are expected, and how often will they occur? If records will be tagged for deletion, consider periodically removing them from the file. By removing records that you no longer need, you free disk space and allow more records to be added.

Only after considering these factors and the applications that use the file can you determine the number of records in the file. For example, the customer name and address file will contain 6000 records at creation time. It is estimated that each month 200 records will be added and 80 records will be deleted. It is also planned that the deletion records will be removed once a month. At the end of six months the file will contain 6720 records. (1200 records are added; 480 records are deleted).

6000	Records at creation
+1200	Records added in six months
7200	
- 480	Records deleted in six months
6720	Records in file after six months

This example points out another factor to consider. When determining the number of records in a file, consider expansion for a reasonable time into the future (at least six months). Of course, most files have deletions, and thus growth is usually slow. In a file where the number of additions and deletions are about the same, deletion records need be removed only when the disk space allowed for the file is filled.

When you create a disk file, you must give System/3 information about the size of your file. You give this information to the system through Operation Control Language (OCL) cards. (OCL is discussed in Chapter 4.) However, during the planning stage, you must calculate how much space is needed so you can decide if a file will fit on a certain disk.

Calculating Record Space

The amount of space required for a file depends upon your file organization method: sequential or indexed. If an indexed file and a sequential file contain the same number of records, the amount of space required for the records in both files is the same. However, additional space is required for the index of an indexed file.

Since the same amount of space is required for the records of a sequential file and an indexed file of the same size, record space is calculated in the same way for both files. To determine record space, you must know the number of characters in the file.

To calculate the number of characters in a file, multiply the number of records (allowing for expansion) by the length of each record. For the customer name and address file, there will be 6,720 records in the file at the end of six months. Each record contains 128 characters. Thus, the number of characters in the file is calculated as:

```
  6720      Number of records in the file
  x128      Number of characters in each record
 -----
  53760
 13440
  6720
 -----
 860,160    Total characters in the file
```

You must know how many tracks are needed. Since a track contains 24 sectors, and a sector contains 256 characters, each track can contain 6,144 characters (24 X 256 = 6144). To calculate the number of tracks the file requires, divide the number of characters in the file by 6144. In our example, this calculation is:

```
                 140        Tracks
6144 ) 860160
       6144
       -----
       24576
       24576
       -----
           0
           0
```

The calculation results in a quotient of 140 and no remainder. So 140 tracks are needed for the name and address file.

When your calculation has a remainder, always add one more track to the quotient. Otherwise, space is not reserved for the remaining characters.

Calculating Index Space

If the file is indexed, you must also determine the amount of space for the index. To find the space needed for the index, you must know the size of the index entry. Recall that an index entry is composed of a key and a disk address. Key lengths vary, depending upon the application, but disk addresses are always three characters long. Thus, the size of an entry is the key field length plus 3.

```
Entry Length = Key Field Length + 3
```

For the name and address file, the key field is customer number (CUSTNO), and it is six characters long. In this case, the index entry length is 9 (6 + 3 = 9).

Another factor affecting index space is sector length. Recall that a sector is the smallest division of a disk and can contain up to 256 characters. For System/3 an index entry must be completely contained within a sector: *an entry cannot start in one sector and end in a different sector.*

To determine the number of entries that can be written in a sector, divide 256 by the entry length. For the name and address example (entry length is 9), this calculation is:

```
                    28          Entries in a sector
Entry Length 9  ) 256
                  18
                  76
                  72
                   4          Remainder
```

Notice that the division results in a remainder of 4. Thus, 28 entries can be written in one sector. The last four positions of the sector are not used since a complete entry must be written in a sector. The twenty-ninth entry is written in the first nine positions of the next sector.

Remember, when calculating the number of index entries in a sector, *drop the remainder.*

Since index space, like record space, is specified in number of tracks, you must convert the sector space to track space. To do this, you must perform two calculations.

First divide the number of index entries in a sector into the number of records. In our example, this calculation is:

```
                      240          Sectors
Entries in a Sector 28 ) 6720      Records
                        56
                       112
                       112
                         0
```

The result of this calculation (240 in this example) gives how many sectors are needed for the index. This result must then be converted to tracks.

Since there are 24 sectors in a track, to find the number of tracks required, divide the number of sectors by 24.

```
                    10          Tracks
                24 ) 240
                     240
                       0
```

In this example, there isn't any remainder. However, if your calculation results in a remainder, add one track to your answer. Otherwise, not enough space will be reserved for the index.

Finally, for an indexed file, add the number of tracks required for the records of the file. In our example, the sum is 150 tracks.

```
140 (records) + 10 (index) = 150
```

Deciding Where to Locate the File

After you determine the amount of space the file requires, you can decide where to locate the file on the disk. Since a disk can contain several files, depending upon their size, it is good practice to document what files are on which disk.

The Disk File Layout Chart (Figure 3) is available for this purpose. The Disk Layout Chart shows space available on the fixed and removable disks. There are 406 positions (0 - 405), represented on the chart. Each position corresponds to a track. In Figure 3, notice that tracks 0 through 7 have a line through them. These tracks are reserved for system use only and are not available for data files.

As you create more files, you can refer to the chart of a particular disk to determine the amount of available space on that disk. It is helpful then to indicate the required space for each file on a Disk Layout Chart. It is also helpful to indicate the name of the file on the chart. Figure 4 shows the space and location of the name and address file using the indexed method. The calculations performed to determine the amount of disk space can be documented on the back of the chart.

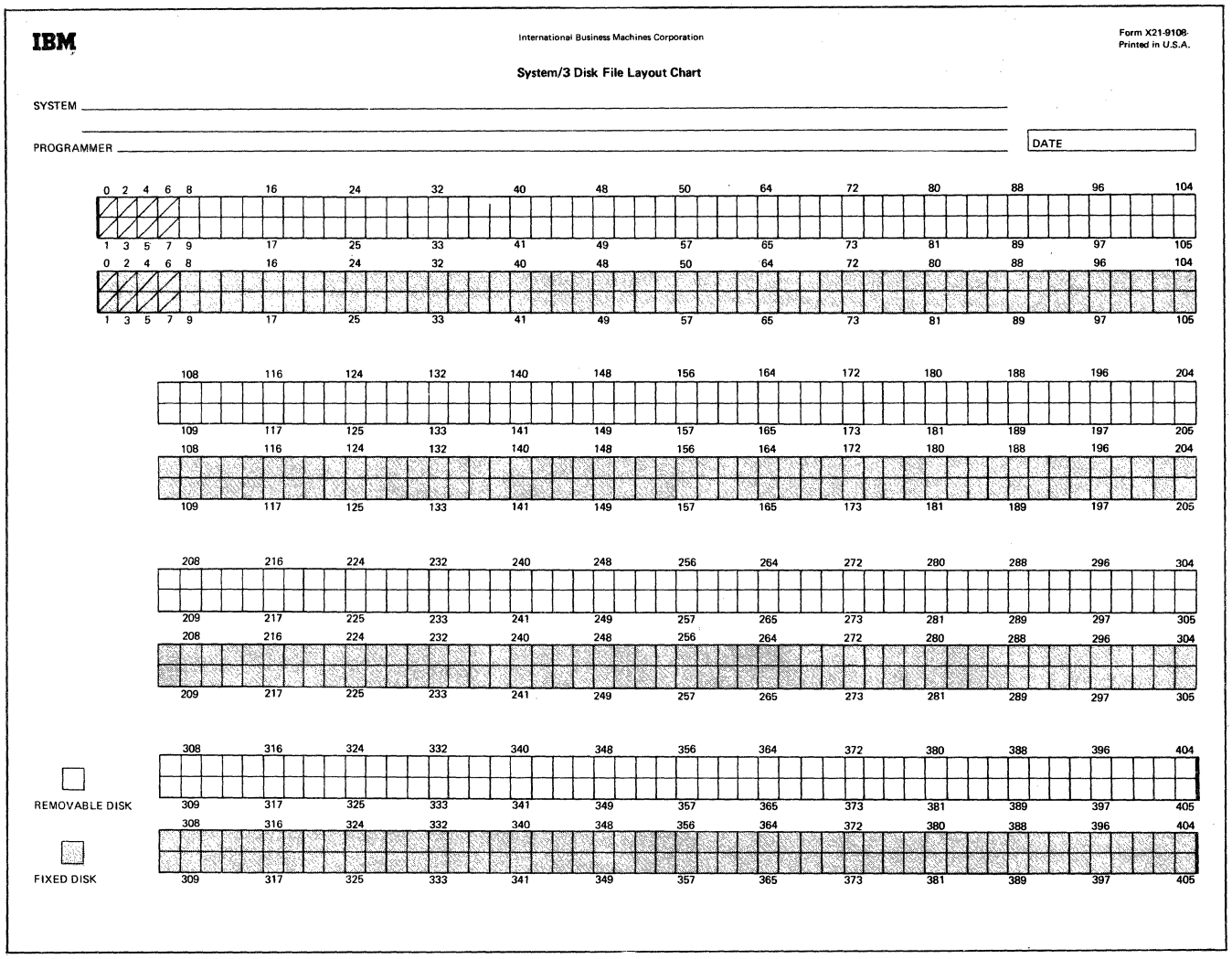


Figure 3. Disk File Layout Chart

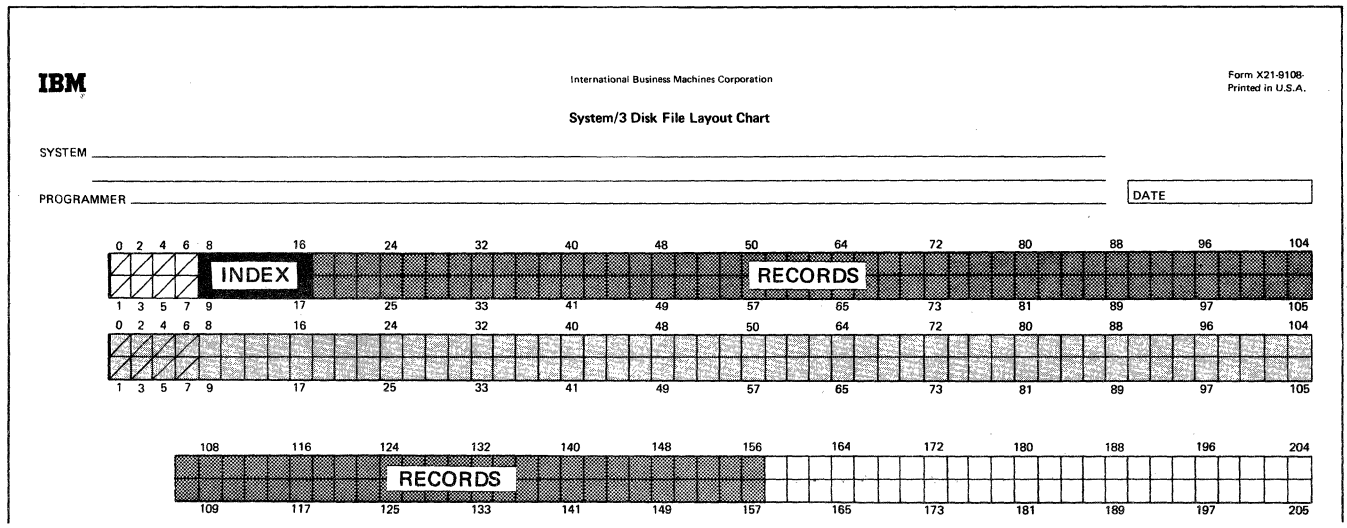


Figure 4. Disk Layout for an Indexed File

REVIEW – CALCULATING FILE SPACE

Calculation 1: Record Space

To calculate the number of tracks required for the records of a file, the following steps are necessary:

1. Multiply the number of records by record length to get total number of characters.
2. Divide total number of characters by 6,144 (the number of characters in a track) to get number of tracks.
3. Round to nearest whole number, if a remainder results.

Calculation 2: Index Space

To calculate the number of tracks required for an index the steps are:

1. Add 3 to the key field to get the length of index entries.
2. Divide 256 by the entry length (a sector contains 256 characters) to get the number of entries per sector.
3. Drop the remainder from step 2. (Complete entries must be contained in a sector.)
4. Divide the number of records by the result of step 3 to get number of sectors.
5. Divide this result by 24 (the number of sectors in a track).
6. Round to the nearest whole number, if a remainder results.

Calculation 3: Total Space for an Indexed File

Add the results of Calculation 1 and Calculation 2 to get the total number of tracks needed for an indexed file.

CHAPTER REVIEW

During the planning stage, but after the file organization method is chosen, record and disk layout are designed.

When designing your record, you should determine:

- Field size and location
- Field names
- Record length
- File name
- Block length

Remember, if you assign field names at this time that follow the RPG II rules, your RPG II program will be easier to write.

Use the Disk File Layout Chart to document:

- File size and location
- File name
- Size calculations (optional)

This chapter explains how to use the RPG II language for indexed and sequential disk files. To understand the material in this chapter, you should know how to organize, maintain, and process these two types of files (Chapter 1). The first and major part of this chapter explains how RPG II is used to:

- Create a file
- Maintain a file

Seven sample jobs illustrate these functions. The last part of the chapter contains three sample processing jobs.

To understand the sample jobs, a basic knowledge of RPG II is also necessary. Specifically you should know how to:

- Describe files
- Describe records
- Use matching records
- Use control levels
- Use mathematical operations (ADD, SUB, Z-ADD)
- Use indicators on Input, Output, and Calculation sheets (MR indicator, control level indicator, record identifying indicator, and resulting indicators).

In the sample jobs these RPG II topics are not explained, except where they relate to disk topics. If you do not fully recall and understand some of these topics, you should refer to *IBM System/3 Disk System RPG II Reference Manual,* Form C21-7504.

CREATING A DISK FILE

The RPG II language used with the System/3 Disk System can create sequential and indexed files. Records in the desired sequence are read from card (or disk) input files, and the disk records are written. Since several required RPG II specifications are common to both sequential and indexed files, these specifications are explained first, followed by two sample programs. One program explains how to create a sequential file; the other explains how to create an indexed file. The uncommon RPG II specifications, those required for a sequential file or for an indexed file, are explained before each program.

To create either type of file, you must write certain entries on the File Description sheet. Four entries are required to describe various characteristics of the disk file. These entries are coded in the following columns:

File Format	column 19
Block Length	columns 20 - 23
Record Length	columns 24 - 27
Device	columns 40 - 46

All records in a file must be the same length. Thus, an F is required in column 19; it specifies that records are the same length (fixed). Columns 20 - 27 are shaded to indicate that block length and record length vary depending on the records in your files. However, block length is always a multiple of record length. If records are unblocked (one record transferred at a time), you make the same entry in the *Block Length* and *Record Length* columns. For example, if the length of a record is 128, 128 is entered in columns 21 - 23 and in columns 25 - 27. Whenever a disk file is described, DISK is required in the *Device* columns.

File Description Specifications

Line	Form Type	Filename	File Type	File Designation	End of File	Sequence	File Format	Block Length	Record Length	L/R	Mode of Processing	Length of Key Field or of Record Address Field	Record Address Type	Type of File Organization or Additional Area	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels (S, N, or E)	Name of Label Exit	Extent Exit for DAM	Core Index	A/U	File Addition/Unordered	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind	File Condition U1-U8	N/U
3 4 5	6	7 8 9 10 11 12 13 14	15	16	17	18	19	20 21 22 23	24 25 26 27	28	29 30	31	32	33 34	35 36 37 38		39	40 41 42 43 44 45 46	47 48 49 50 51 52	53	54 55 56 57 58 59	60 61 62 63 64 65		66	67	68 69	70	71 72	73 74	
0 2	F						F											DISK												
0 3	F																													
0 4	F																													
0 5	F																													

Creating a Sequential File

To create a sequential disk file no additional disk entries are required. The four entries explained, in addition to other basic RPG II entries, are sufficient to create a sequential file. Job 1 (Figure 5) explains the coding needed to create a master name and address file from a card file.

Note: Job 1 includes the layout of the input cards, the layout of the disk records, and the coded RPG II specification sheets needed to create the name and address file. The disk entries explained previously are emphasized on each sheet. Those RPG II entries, such as defining fields, that are assumed prerequisites are not re-explained. However, some of these entries, those crucial to correctly performing the job, are discussed. This method is used in all sample jobs in this chapter.

Creating an Indexed File

To create an indexed file, four more File Description entries are required. Two entries tell the RPG II Compiler that an indexed file is to be created: an A in column 31 and an I in column 32. The I specifies an indexed file; the A specifies that a key field exists.

The other two entries describe the length and location of the key. Key length must be entered in columns 29 - 30. Recall that key length is the same length for all records in a file. The maximum key length is 29 characters. Columns 35 - 38 are used to specify the location of the first character of the key. For example, if a key field is located in columns 73 - 78 of a disk record, you would enter a 6 in column 30 and a 73 in columns 37 - 38.

Job 2 (Figure 6) shows the disk entries, and other RPG II entries, needed to create an indexed inventory file.

IBM — International Business Machines Corporation — Form X21-9092 Printed in U.S.A.

RPG CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

Date _____ Program _____ Programmer _____

Punching Instruction: Graphic / Punch

Page (1 2) Program Identification (75 76 77 78 79 80)

Control Card Specifications

Line	Form Type	Core Size to Compile	Object Output	Listing Options	Core Size to Execute	Debug	MFCM Stacking Sequence	Sterling: Input-Shillings	Sterling: Input-Pence	Sterling: Output-Shillings	Sterling: Output-Pence	Inverted Print	360/20 2501 Buffer	Number Of Print Positions	Alternate Collating Sequence	Refer to the specific System Reference Library manual for actual entries.
3 4 5	6	7 8 9	10	11	12 13 14	15	16	17	18	19	20	21	22	23 24 25	26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
0 1	H															

File Description Specifications

Line	Form Type	Filename	File Type	File Designation	End of File	Sequence	File Format	Block Length	Record Length	L/R	Mode of Processing	Length of Key Field or of Record Address Field	Record Address Type	Type of File Organization or Additional Area	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels (S, N, or E)	Name of Label Exit	Extent Exit for DAM	Core Index	File Addition/Unordered	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind	File Condition U1-U8
			I/O/U/C/D	P/S/C/R/T/D	E	A/D	F/V				A/K/I	I/D/T or 1-9												A/U	N/U			
3 4 5	6	7 8 9 10 11 12 13 14	15	16	17	18	19	20 21 22 23	24 25 26 27	28	29 30	31	32	33 34	35 36 37 38		39	40 41 42 43 44 45 46	47 48 49 50 51 52	53	54 55 56 57 58 59	60 61 62 63 64 65		66	67 68 69	70	71 72	73 74
0 2	F						F					A	I					DISK										
0 3	F																											
0 4	F																											
0 5	F																											
0 6	F																											
0 7	F																											
	F																											
	F																											

MAINTAINING A DISK FILE

In Chapter 2, four file maintenance functions were explained. These four functions apply to both sequential and indexed files. RPG II provides the capability to perform these functions. This section explains the RPG II coding required to do three of these functions: add a record, update a record, and reorganize a file. In order to explain the coding needed for these three functions, the sample jobs show the functions performed separately. However, you may write a program that performs more than one of these maintenance functions. Since the delete function is usually performed during normal processing, and it does not require new entries, this function is not explained in this section.

Adding Records to a Disk File

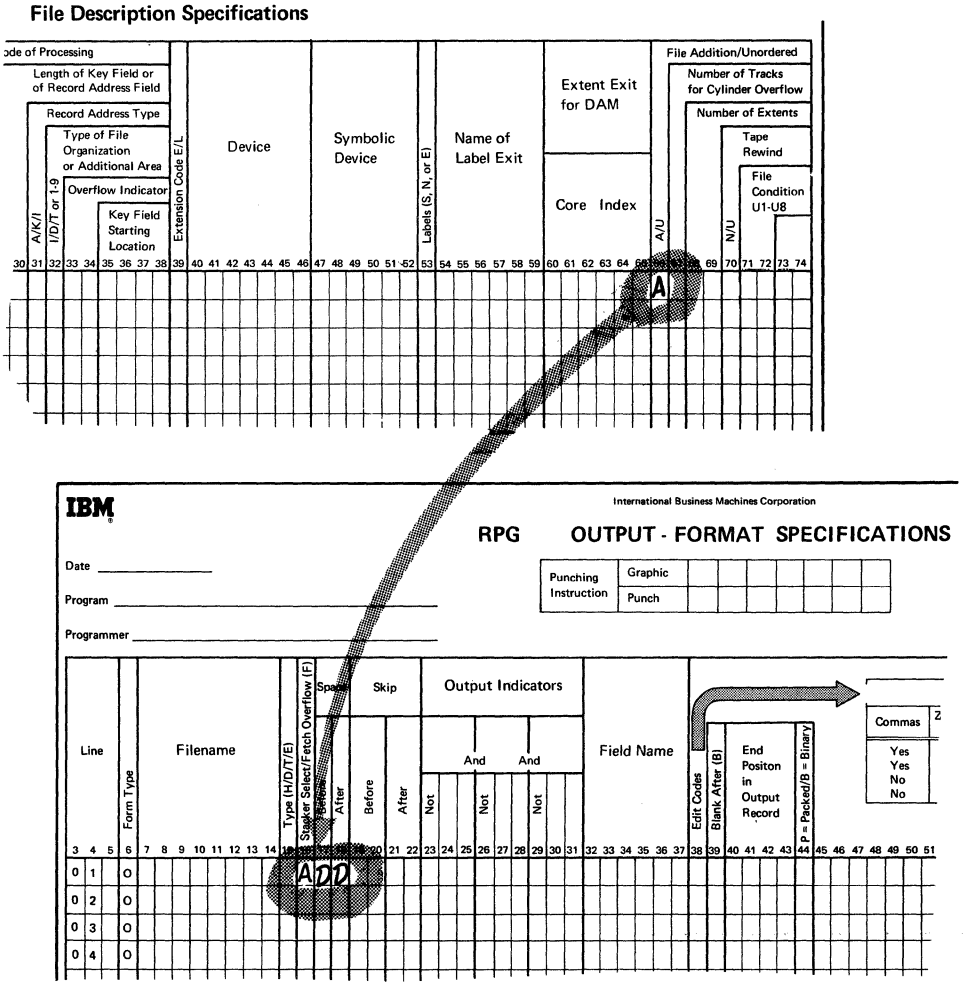
After a file is created, it is often necessary to add records to the file. Recall that adding records to a disk file means writing the records at the end of the records already in the file. To add records to an indexed or a sequential file, two more disk entries are needed, one on the File Description sheet and one on the Output-Format sheet.

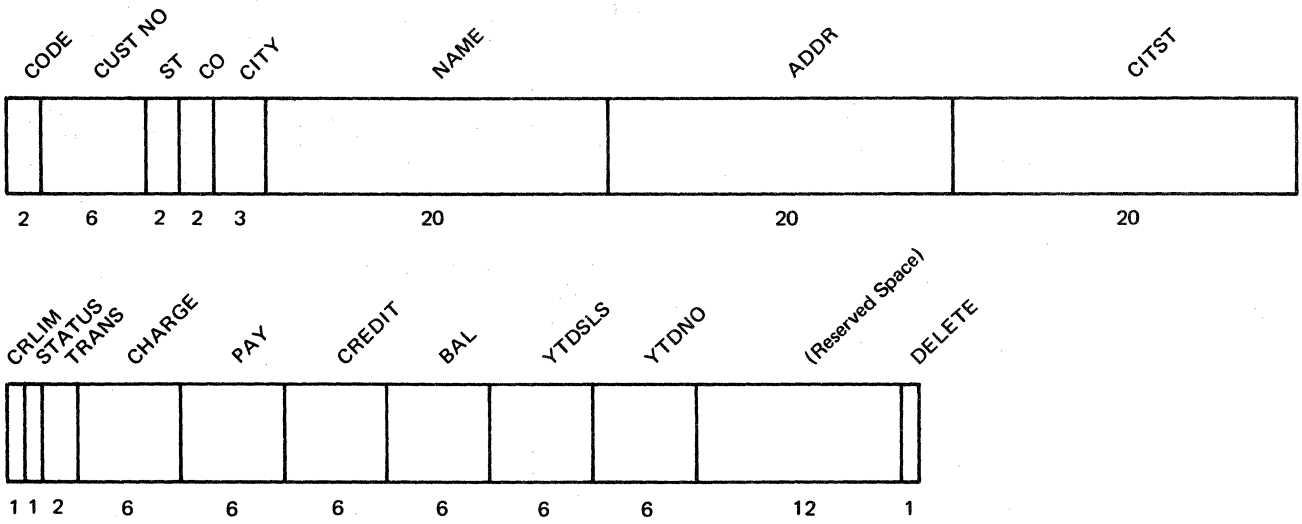
The A in column 66 on the File Description sheet tells the system that records will be added to the file described on this line. ADD tells the system that the fields defined on the following lines constitute the record to be added to the file specified in columns 7 - 14.

Adding Records to an Indexed File

Job 3 (Figure 7) shows the RPG II coding required to add records to the master inventory file (indexed) created in Job 2 (Figure 6). Note that the card layout and the disk record layout are the same for both jobs.

File Description Specifications





CODE = Record code (CM)
CUSTNO = Customer number
ST = State code
CO = County code
CITY = City code
NAME = Customer name
ADDR = Customer address (street)
CITST = Customer address (city and state)
CRLIM = Credit limit
STATUS = Status code
TRANS = Number of transactions this month
CHARGE = Current month charges
PAY = Current month payments
CREDIT = Current month credits
BAL = Balance
YTDSLS = Year-to-date sales amount
YTDNO = Year-to-date number of sales
DELETE = Delete code

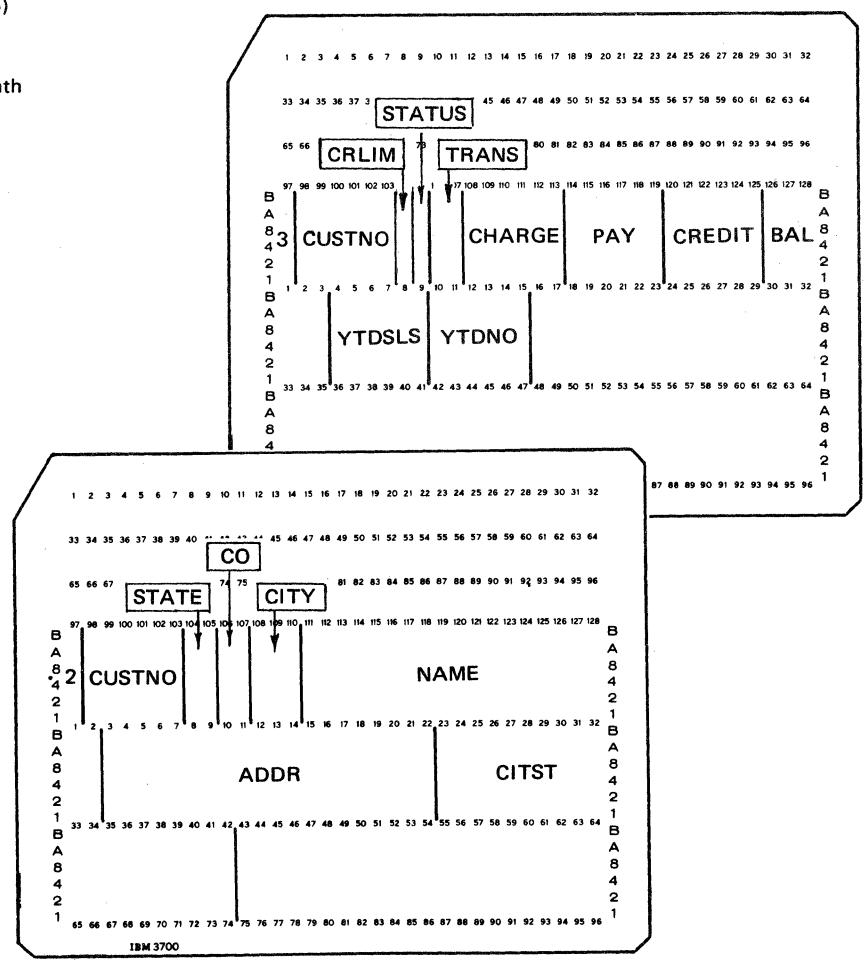


Figure 5. Job 1: Creating a Sequential Customer Master File (Part 1 of 2)

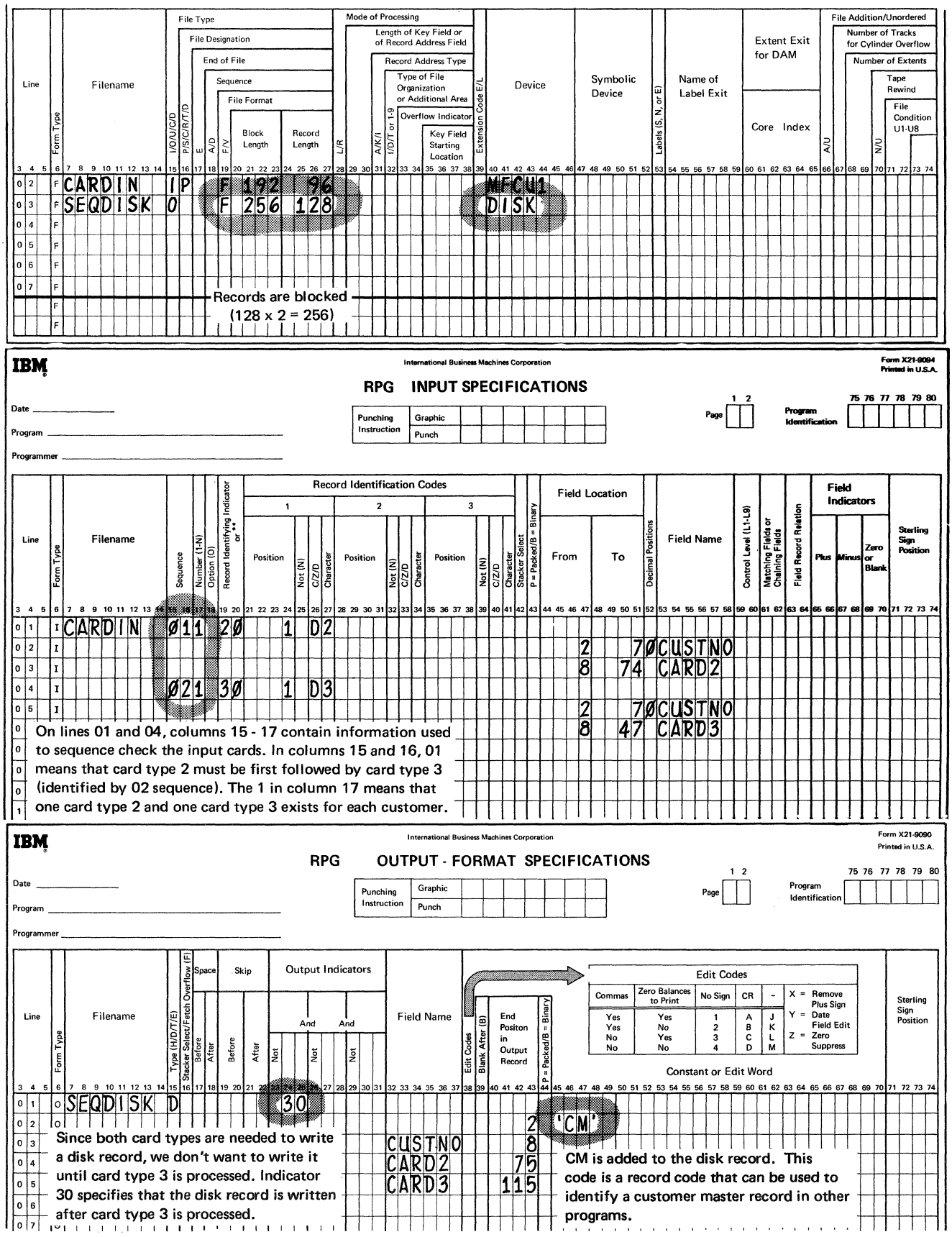
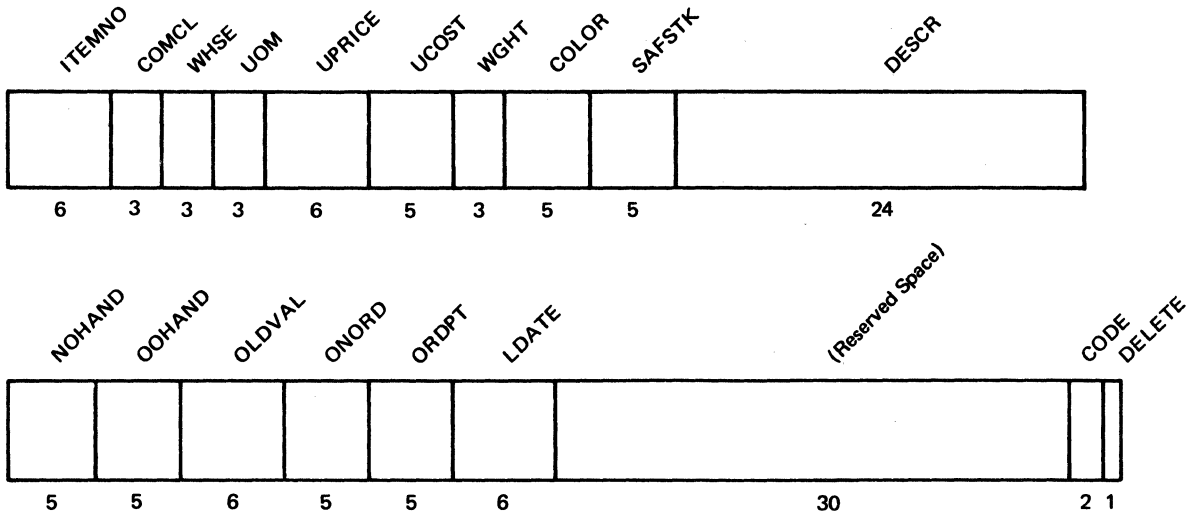
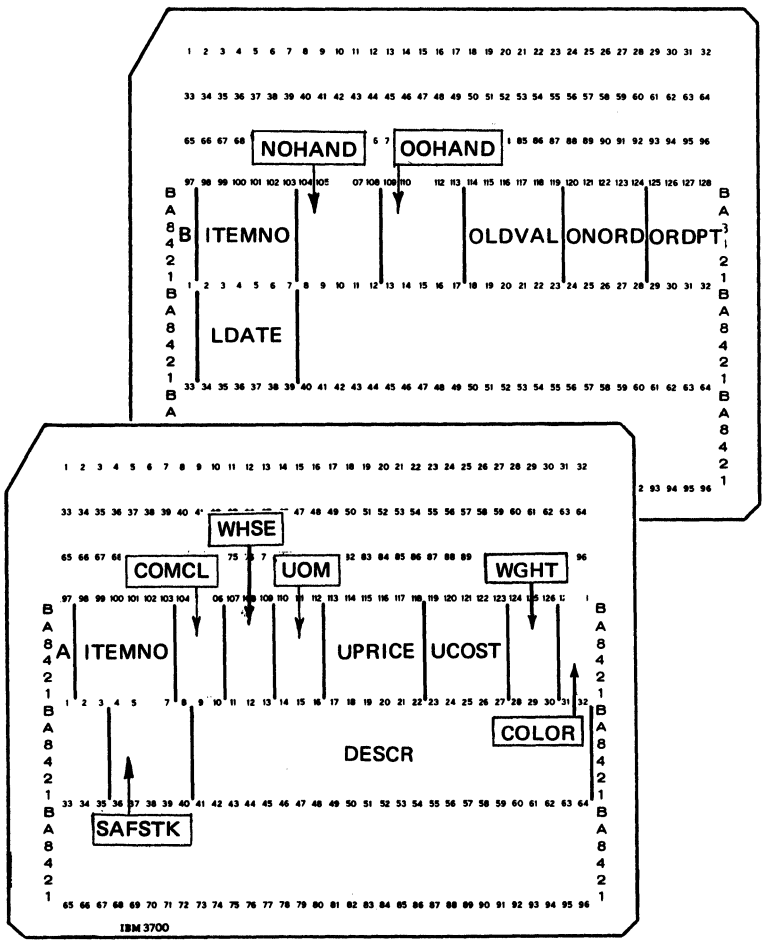


Figure 5. Job 1: Creating a Sequential Customer Master File (Part 2 of 2)



Although 95 positions are presently used, the disk record is defined as 128 positions to provide space for adding data later. Field names are the same as the input card field names, with two exceptions: CODE = MI (Master Inventory) and DELETE (position reserved for a delete code).



ITEMNO = Item number
COMCL = Commodity class
WHSE = Warehouse location
UOM = Unit of measure
UPRICE = Unit price
UCOST = Unit cost
WGHT = Weight
COLOR = Color
SAFSTK = Safety stock
NOHAND = New on hand quantity
OOHAND = Old on hand quantity
OLDVAL = Old on hand value
ONORD = Quantity on order
ORDPT = Order point
LDATE = Latest transaction date

Figure 6. Job 2: Creating an Indexed File (Part 1 of 2)

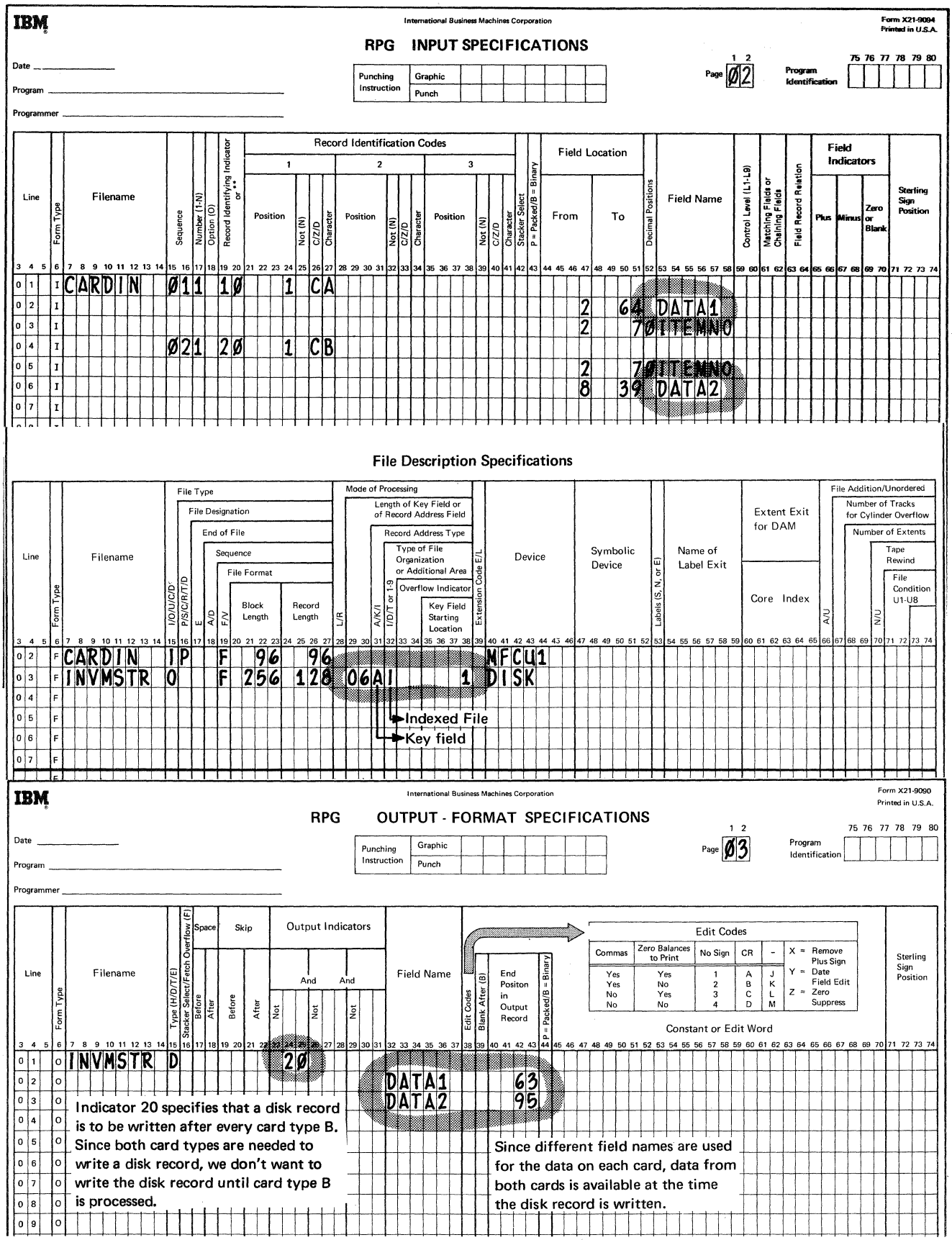
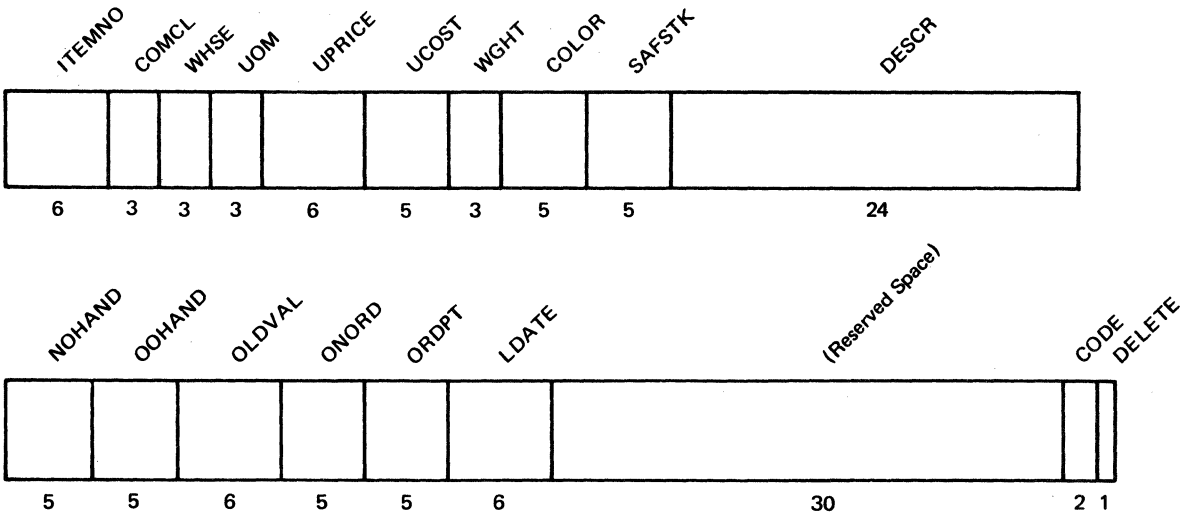
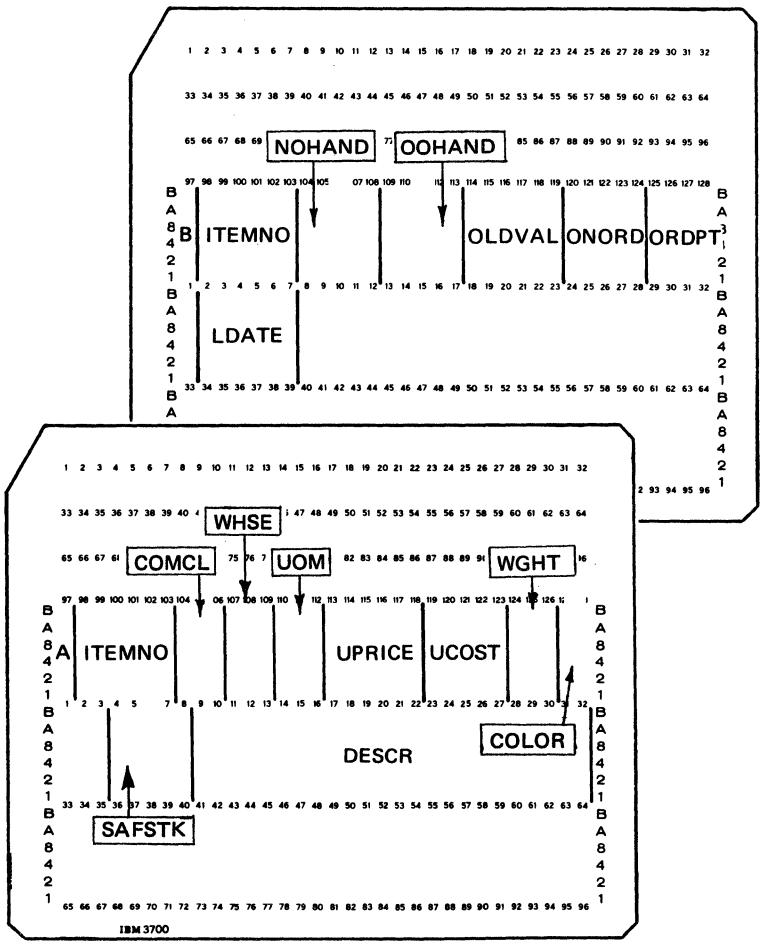


Figure 6. Job 2: Creating an Indexed File (Part 2 of 2)



Although 95 positions are presently used, the disk record is defined as 128 positions to provide space for adding data later. Field names are the same as the input card field names, with two exceptions: CODE = MI (Master Inventory) and DELETE (position reserved for a delete code).



ITEMNO = Item number
COMCL = Commodity class
WHSE = Warehouse location
UOM = Unit of measure
UPRICE = Unit price
UCOST = Unit cost
WGHT = Weight
COLOR = Color
SAFSTK = Safety stock
NOHAND = New on hand quantity
OOHAND = Old on hand quantity
OLDVAL = Old on hand value
ONORD = Quantity on order
ORDPT = Order point
LDATE = Latest transaction date

Figure 7. Job 3: Adding Records to an Indexed File (Part 1 of 2)

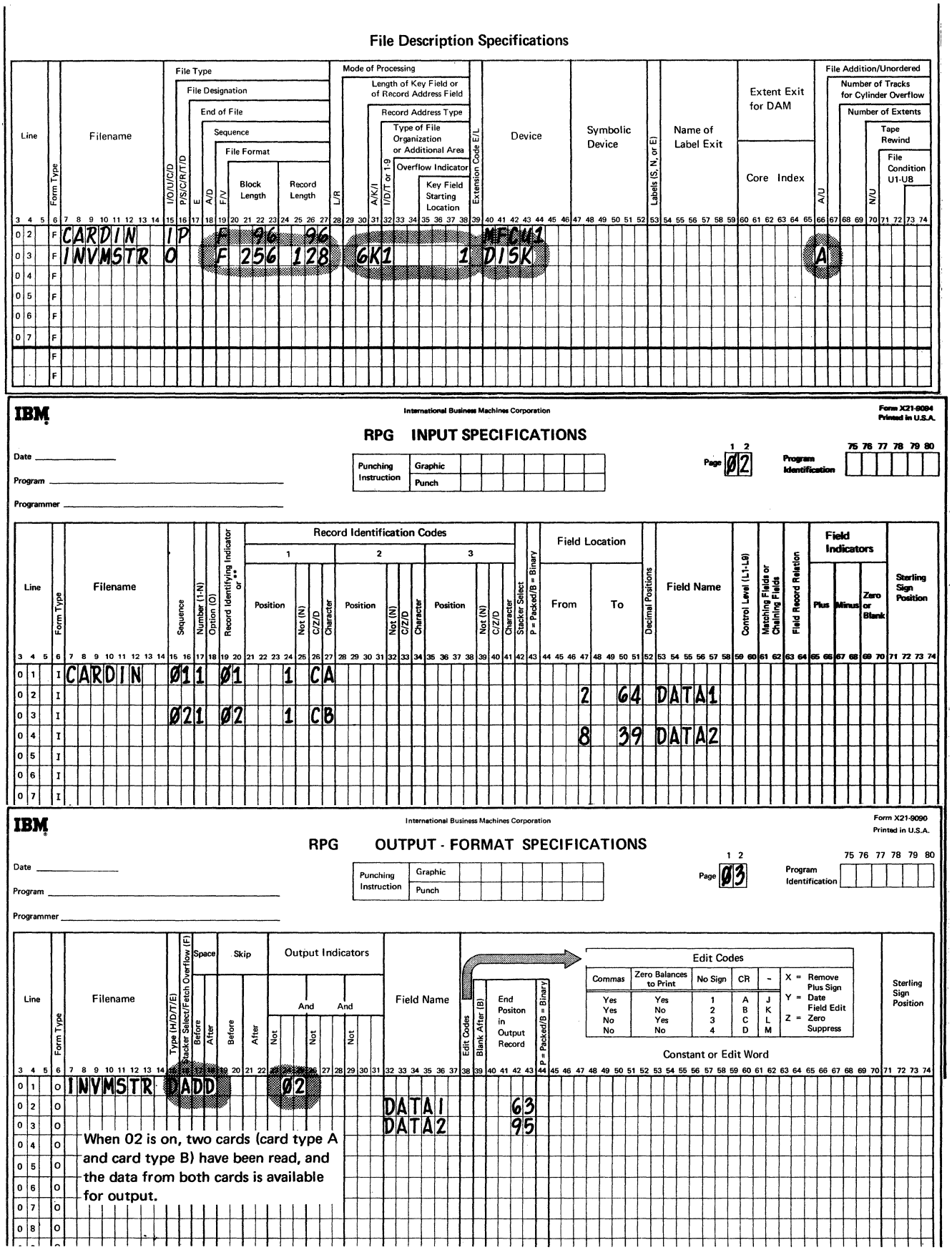


Figure 7. Job 3: Adding Records to an Indexed File (Part 2 of 2)

Adding Records to a Sequential File (Added at End of Records in the File)

Records can be added to a sequential file in the same way they are added to an indexed file: records are added at the end of the file. This method of adding records to a sequential file is good for files such as the employee payroll file explained in Chapter 2. Recall that this file was organized in ascending order according to employee number. New employees are assigned the highest available number. Thus, when records are added for each new employee, the records will be in order on the disk.

To add records to this type of sequential disk file, the four basic disk entries and the two *add* entries (A,ADD) are required.

Adding Records to a Sequential File (Merging Records Between Records in the File)

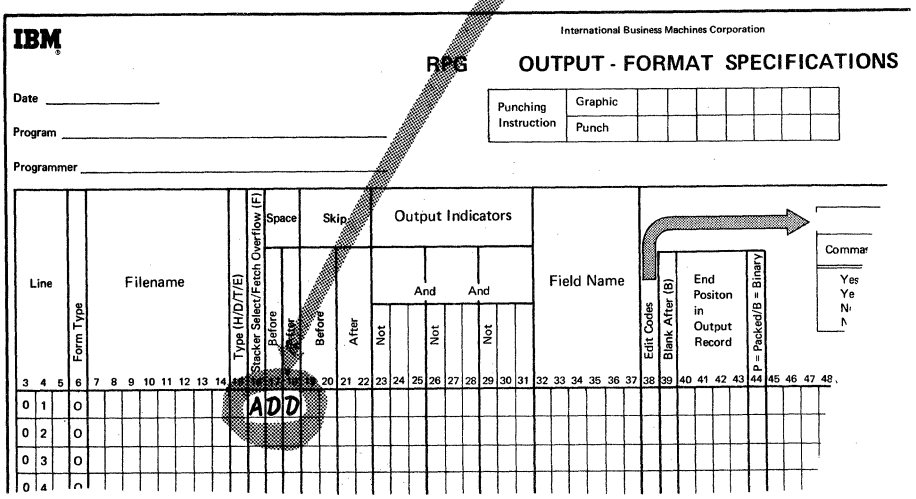
Often records must be added *between* existing records in a sequential disk file. In Chapter 2 this situation was mentioned, and you were told these additions could be handled in an RPG II program. When records must be added between existing records, a new file must be created. This file will contain the added records merged in correct order with the records from the original file.

More disk space is required for this method of adding records. Space must be available for the old file and the newly created file.

Job 4 (Figure 8) illustrates how to add records between existing records of a sequential file.

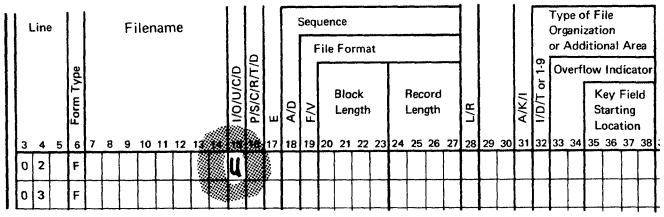
File Description Specifications

Line	Form Type	Filename	File Type	File Designation	End of File	Sequence	File Format	Block Length	Record Length	Mode of Processing	Length of Key Field or of Record Address Field	Record Address Type	Type of File Organization or Additional Area	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels (S, N, or E)	Name of Label Exit	Extent Exit for DAM	Core Index	File Addition/Unordered	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind	File Condition U1-U8
			I/O/U/C/D	P/S/C/R/T/D	E	A/D	F/V			L/R	A/K/I	I/D/T or 1-9											A/U	N/U			
3 4 5	6	7 8 9 10 11 12 13 14	15	16	17	18	19	20 21 22 23	24 25 26 27	28	29 30	31	32	33 34	35 36 37 38	39	40 41 42 43 44 45 46	47 48 49 50 51 52	53	54 55 56 57 58 59	60 61 62 63 64	65	66	67 68 69	70	71 72 73 74	
0 2	F						F										DISK						A				
0 3	F																										
0 4	F																										



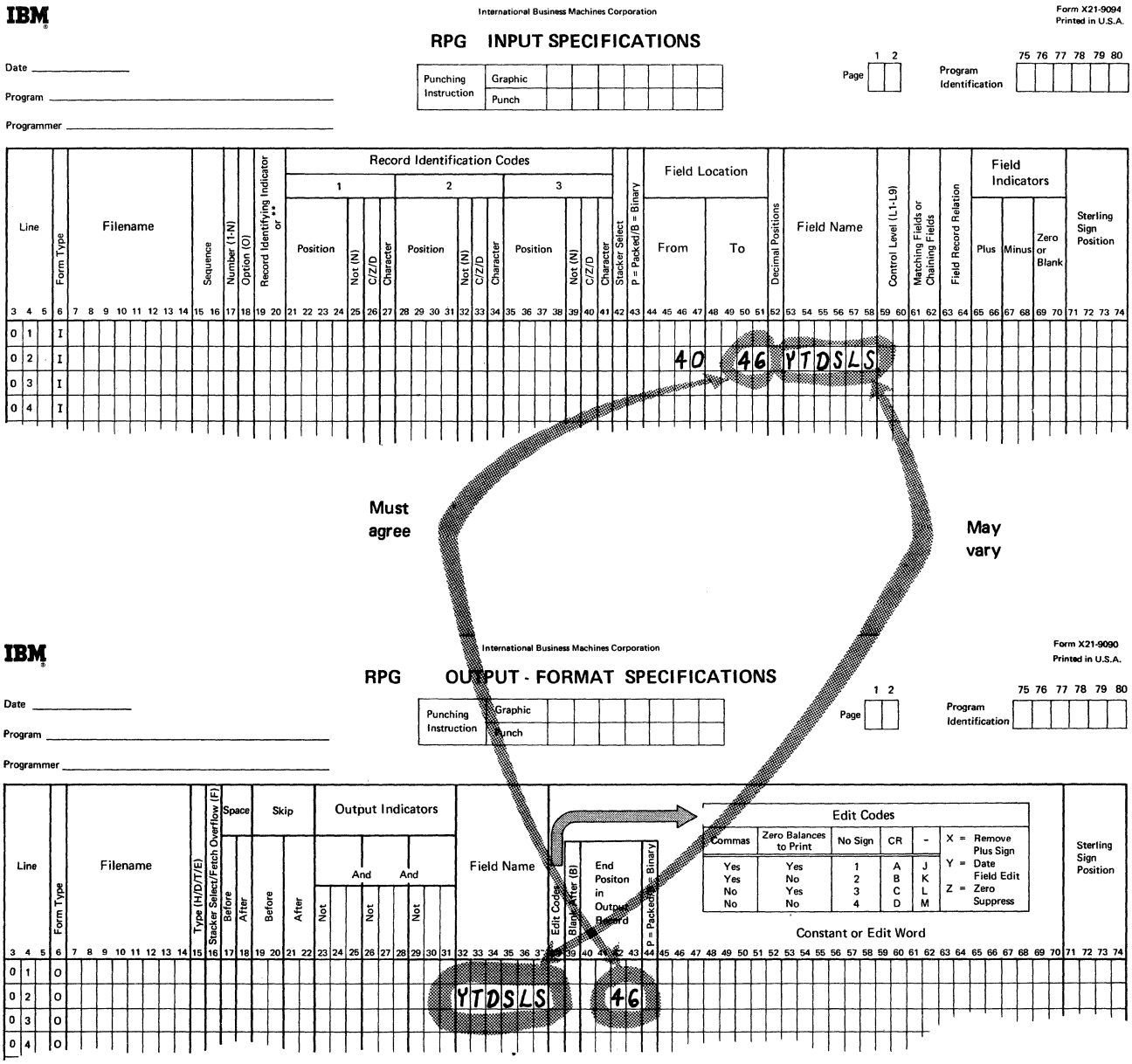
Updating Records in a Disk File

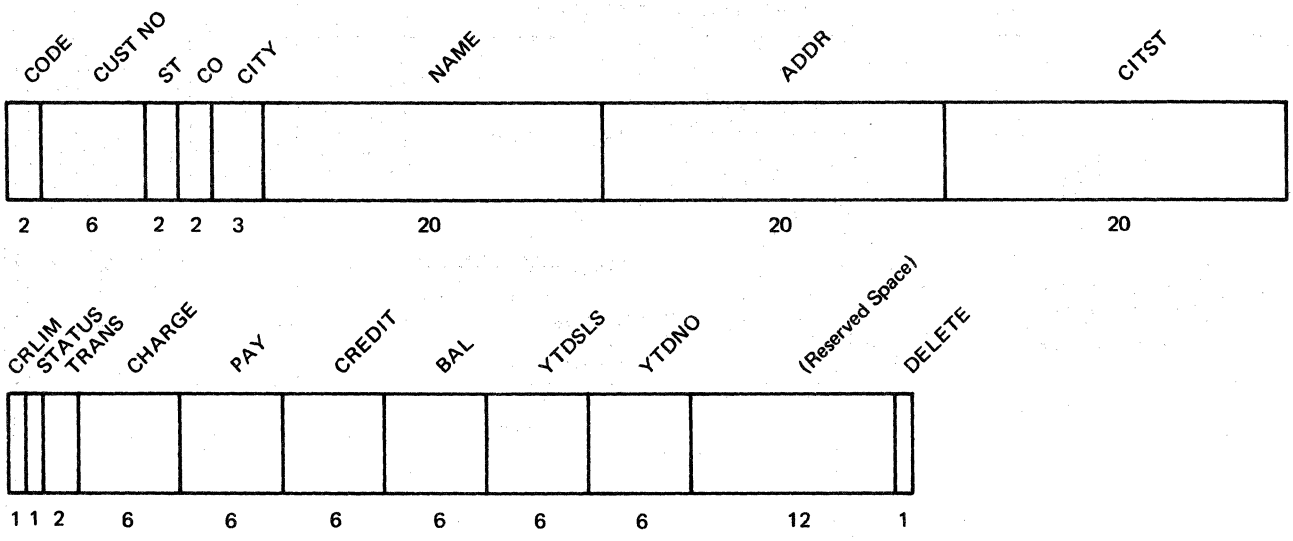
Many jobs not only require the ability to add records, but to change certain data in a record. Recall that this function is called updating. RPG II provides several ways to update both sequential and indexed files. The next four jobs illustrate four ways to update disk files. Regardless of which way it is used, one new disk entry is required on the File Description sheet.



The U in column 15 indicates that the file is an update file. Since updating means getting a record from a disk file, changing some data, and then putting the record back in its original location, an update file is like a combination input/output file. For this reason, the update file must be described on both the Input and Output sheets.

Field locations must agree between the two sheets. The *To* location, columns 48 - 51 on the Input sheet, must be the same as the *End Position*, columns 40 - 43 on the Output sheet. Field names may vary depending on the type of updating being done.





CODE = Record code (CM)
CUSTNO = Customer number
ST = State code
CO = County code
CITY = City code
NAME = Customer name
ADDR = Customer address (street)
CITST = Customer address (city and state)
CRLIM = Credit limit
STATUS = Status code
TRANS = Number of transactions this month
CHARGE = Current month charges
PAY = Current month payments
CREDIT = Current month credits
BAL = Balance
YTDSLS = Year-to-date sales amount
YTDNO = Year-to-date number of sales
DELETE = Delete code

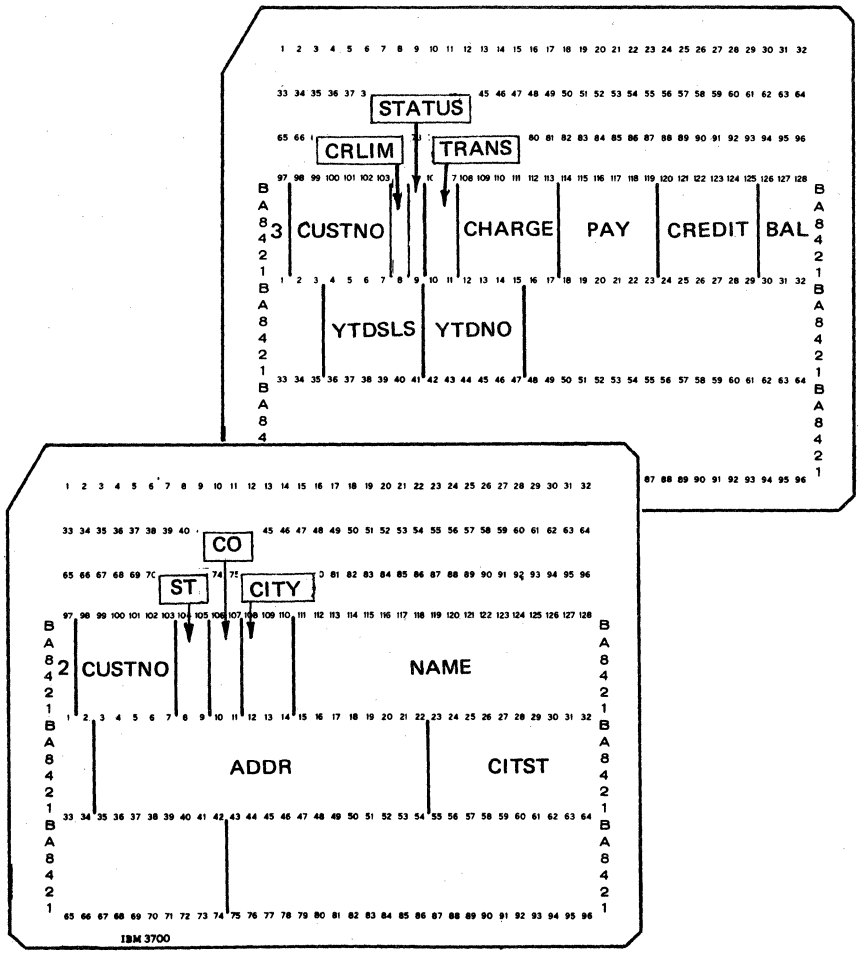


Figure 8. Job 4: Adding Records to (Re-creating) a Sequential Customer Master File (Part 1 of 2)

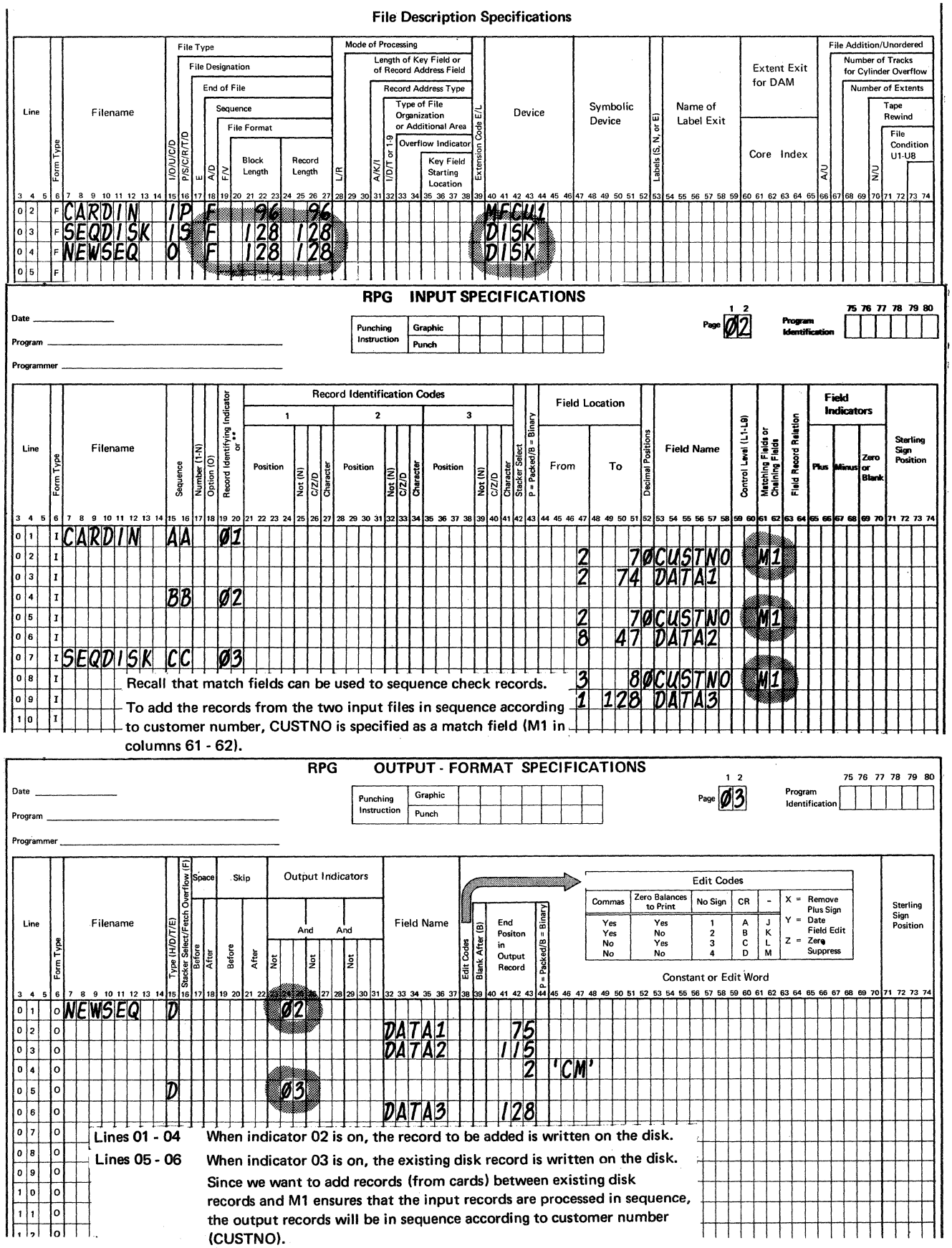


Figure 8. Job 4: Adding Records to (Re-creating) a Sequential Customer Master File (Part 2 of 2)

Updating All Records in a File

All or part of the records in a file may be updated. Let's discuss updating all records first.

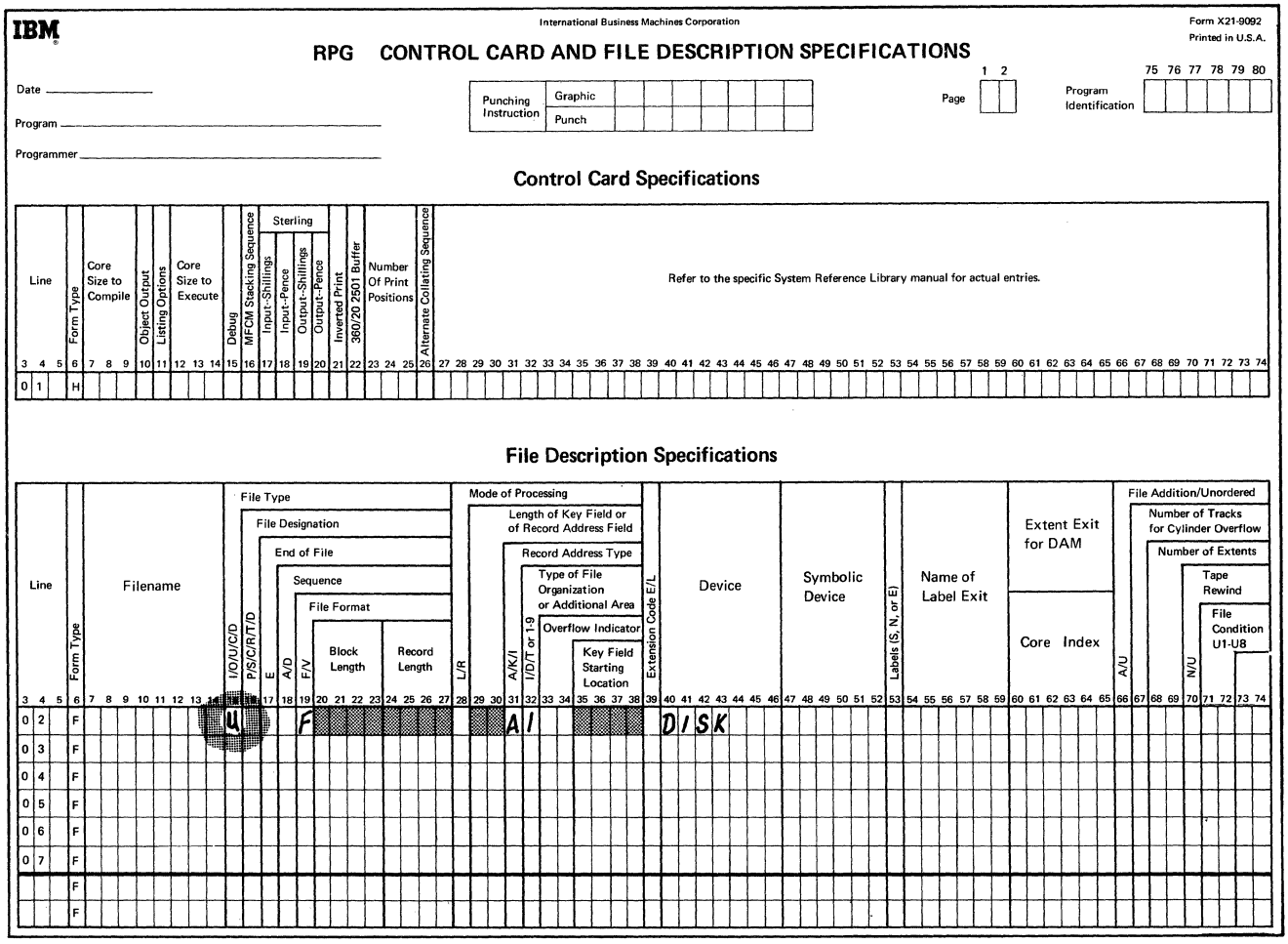
All records may need updating for a variety of reasons. For example, you might add a new field to all records. If the record were originally designed with expansion in mind, the field can be added with the update function. For example, a customer file was created without zip code; then zip code was required. It can be added as an update field. In this case an input file (probably cards) is processed against the master disk file which is specified as an update file.

In another case you might want to change some data in all the records as a result of calculations, not as the result of an input file. For example, payroll records contain year-to-date information. After all reports and W-2 forms are completed at the end of the year, the year-to-date fields must be set to zeros, so accumulations can begin for the new year. In this type of job, you have only one file, the update file. Job 5 illustrates this type of updating.

Updating All Records in a Sequential File: We discussed consecutive processing of sequential files in Chapter 2. Recall that with consecutive processing all records are read in order, one after the other. To update each record in a sequential file, the four basic disk entries plus U in column 15 are needed on the File Description sheet.

Job 5 (Figure 9) shows the RPG II coding required to update all records of a sequential customer credit file. Two fields in each customer's record contain year-to-date sales data. These two fields indicate the activity of each customer. Throughout the year this data is accumulated. At the end of the year the two fields are set to zero, so the accumulation can begin for the next year. Only one file, the update file, is used in this job since updating occurs as a result of calculations.

Updating All Records in an Indexed File: All records in an indexed file can also be updated. Recall that to process *all* records in an indexed file, the file is processed sequentially by key. Thus, to update all records in an indexed file, the entries required to describe an indexed file plus U in column 15 are the disk entries needed on the File Description sheet.


IBM

International Business Machines Corporation

Form X21-9092
Printed in U.S.A.

RPG CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

Date _____
Program _____
Programmer _____

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Control Card Specifications

Line	Form Type	Core Size to Compile	Object Output	Listing Options	Core Size to Execute	Debug	MFCM Stacking Sequence	Sterling: Input-Shillings	Sterling: Input-Pence	Sterling: Output-Shillings	Sterling: Output-Pence	Inverted Print	360/20 2501 Buffer	Number Of Print Positions	Alternate Collating Sequence	Refer to the specific System Reference Library manual for actual entries.
3 4 5	6	7 8 9	10	11	12 13 14	15	16	17	18	19	20	21	22	23 24 25	26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
0 1	H															

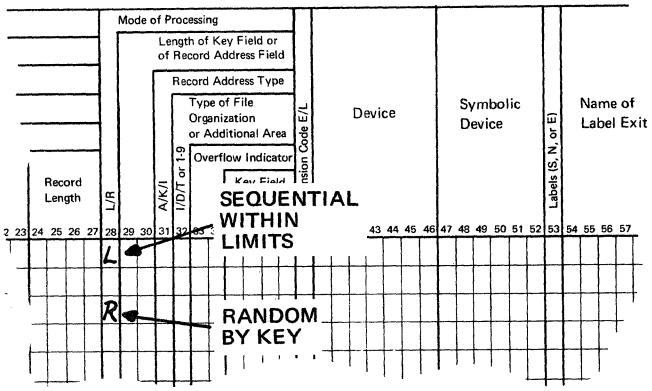
File Description Specifications

Line	Form Type	Filename	I/O/U/C/D	P/S/C/R/T/D	E	A/D	F/V	Block Length	Record Length	L/R	A/K/I	I/D/T or 1-9	Type of File Organization or Additional Area	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels (S, N, or E)	Name of Label Exit	Extent Exit for DAM	Core Index	A/U	Number of Tracks for Cylinder Overflow	Number of Extents	N/U	Tape Rewind	File Condition U1-U8
3 4 5	6	7 8 9 10 11 12 13 14	15	16	17	18	19	20 21 22 23	24 25 26 27	28	29 30	31	32	33 34	35 36 37 38	39	40 41 42 43 44 45 46	47 48 49 50 51 52	53	54 55 56 57 58 59	60 61 62 63 64 65		66	67 68 69	70	71 72	73 74	
0 2	F		U				F					A	I				DISK											
0 3	F																										
0 4	F																										
0 5	F																										
0 6	F																										
0 7	F																										
	F																										
	F																										

Updating Some Records in an Indexed File

An indexed file can also be updated sequentially within limits or randomly by key. In addition to those disk entries required for an update indexed file, one other entry is needed on the File Description sheet. This entry specifies how a file is processed. For indexed files, a file can be processed sequentially by key, sequentially within limits, or randomly by key. The entry is made in column 28.

File Description Specifications



For indexed files, if column 28 is left blank, the file is processed sequentially by key. If any of your files must be processed sequentially within limits, refer to the *Disk System RPG II Reference Manual.*

Updating Records Randomly by Key:

When only some records in an indexed file must be updated by input file (transaction), the indexed file can be updated randomly by key. Indexed files are usually updated randomly by key when the input file is unordered or activity is low. Recall that low activity means only a few records of a master file are affected. Regardless of whether the input records are in order or not, random processing by key provides the ability to update certain, not all, records.

When an indexed file is updated randomly by key, input records are *chained* to update records. Chaining means matching the control field from the input record with the key in the index.

When records are updated randomly by key, the CHAIN operation code must be used. To use CHAIN, you must make five entries on the Calculation sheet.

IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date _____

Program _____

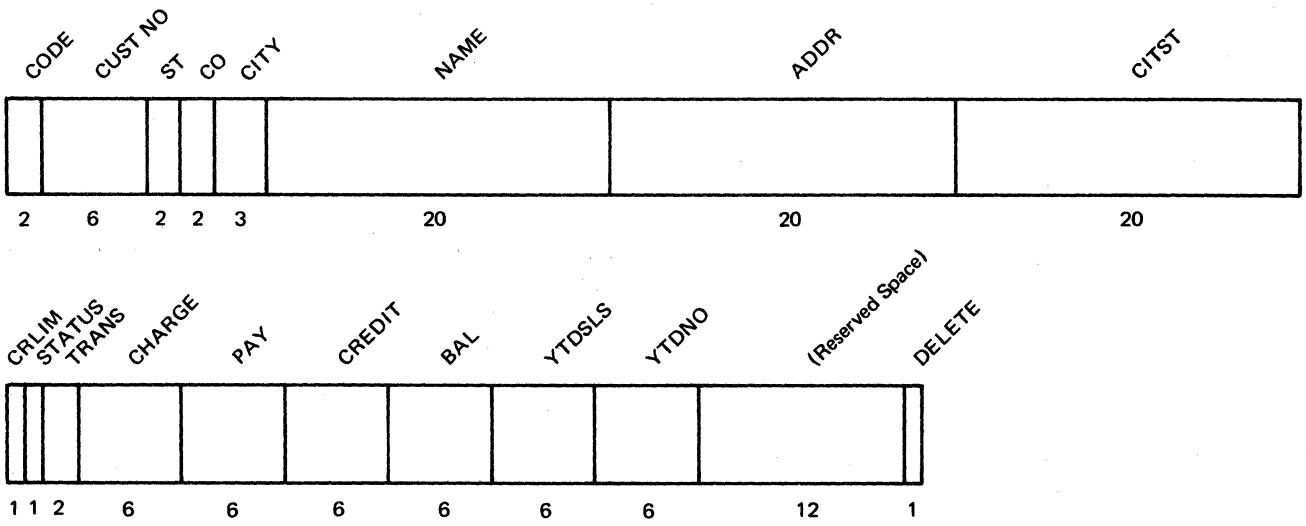
Programmer _____

Punching Instruction: Graphic / Punch

Page

Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators: And / Not	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic (Plus, Minus, Zero); Compare (High 1>2, Low 1<2, Equal 1=2); Lookup (Table (Factor 2) is High, Low, Equal)	Comments
01	C											
02	C				CHAIN							
03	C											
04	C											
05	C											
06	C											
07	C											
08	C											
09	C											
10	C											
11	C											
12	C											
13	C											
14	C											
15	C											



CODE = Record code (CM)
CUSTNO = Customer number
ST = State code
CO = County code
CITY = City code
NAME = Customer name
ADDR = Customer address (street)
CITST = Customer address (city and state)
CRLIM = Credit limit

STATUS = Status code
TRANS = Number of transactions this month
CHARGE = Current month charges
PAY = Current month payments
CREDIT = Current month credits
BAL = Balance
YTDSLS = Year-to-date sales amount
YTDNO = Year-to-date number of sales
DELETE = Delete code

IBM

International Business Machines Corporation

Form X21-9092
Printed in U.S.A.

RPG CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

Date _____
Program _____
Programmer _____

Punching Instruction: Graphic / Punch

Page 01

Program Identification

Control Card Specifications

Line	Form Type	Core Size to Compile	Object Output	Listing Options	Core Size to Execute	Debug	MFCM Stacking Sequence	Sterling: Input-Shillings	Sterling: Input-Pence	Sterling: Output-Shillings	Sterling: Output-Pence	Inverted Print	360/20 2501 Buffer	Number Of Print Positions	Alternate Collating Sequence	Refer to the specific System Reference Library manual for actual entries.
3 4 5	6	7 8 9	10	11	12 13 14	15	16	17	18	19	20	21	22	23 24 25	26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
0 1	H															

File Description Specifications

Line	Form Type	Filename	File Type	File Designation	End of File	Sequence	File Format	Block Length	Record Length	L/R	Mode of Processing	Length of Key Field or of Record Address Field	Record Address Type	Type of File Organization or Additional Area	I/O/T or 1-9	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels (S, N, or E)	Name of Label Exit	Extent Exit for DAM	Core Index	A/U	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind	N/U	File Condition U1-U8	File Addition/Unordered
3 4 5	6	7 8 9 10 11 12 13 14	I/O/U/C/D	P/S/C/R/T/D	E	A/D	F/V	20 21 22 23	24 25 26 27	28	29 30	31	32	33 34 35 36 37 38				39	40 41 42 43 44 45 46	47 48 49 50 51 52	53	54 55 56 57 58 59	60 61 62 63 64 65		66	67 68 69	70	71 72	73 74		
0 2	F	SEQD15K U	I				F	256	128										D15K											
0 3	F																													
0 4	F																													
0 5	F																													
0 6	F																													

Figure 9. Job 5: Updating Records in a Sequential File (Part 1 of 2)

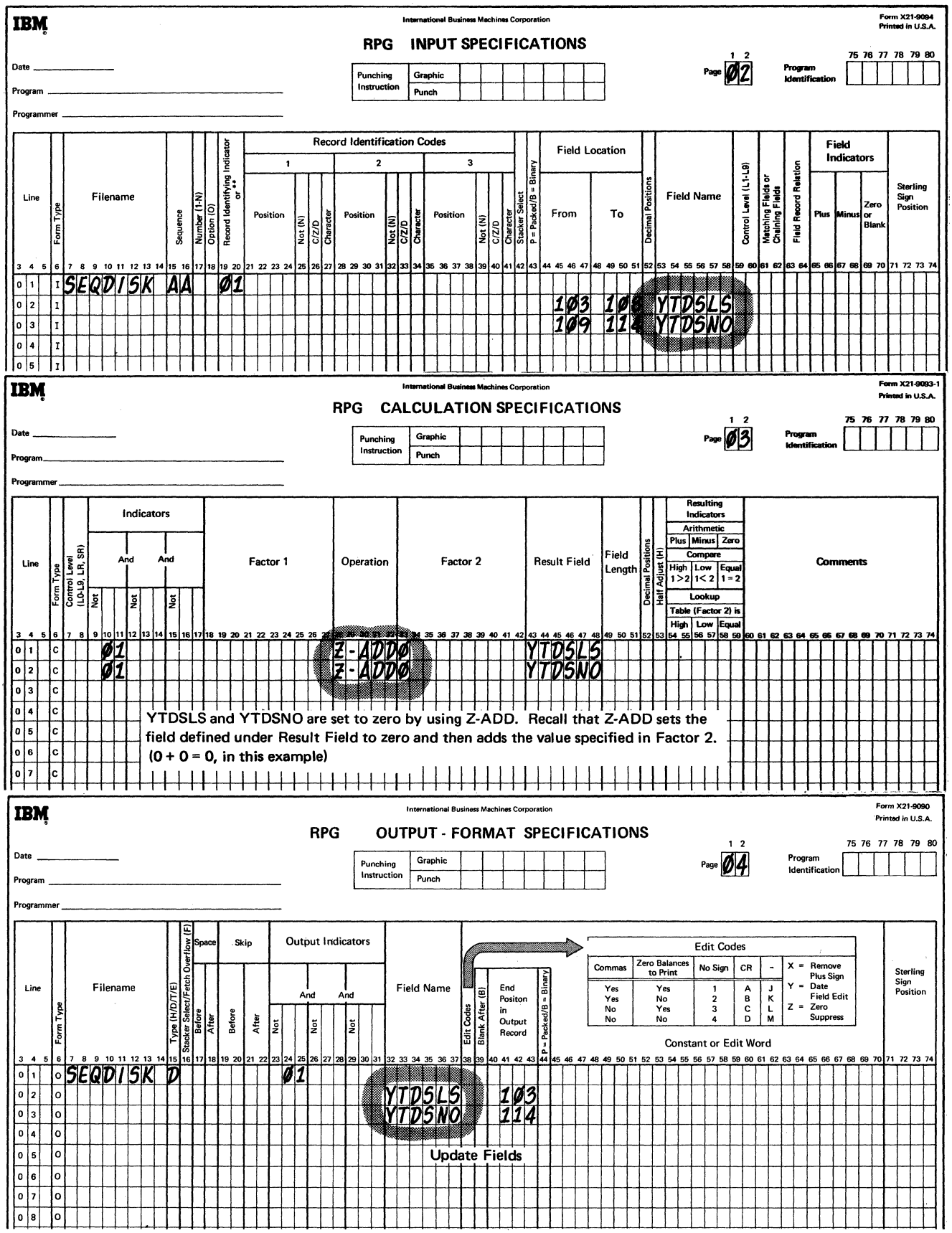


Figure 9. Job 5: Updating Records in a Sequential File (Part 2 of 2)

Under Factor 1 you must enter the name of the control field in the input file. CHAIN is entered in the Operation columns. In Factor 2 you must enter the name of the update file. These three entries are needed for the system to locate the update record. The control field from the input file is used to find the index entry that contains the same field (key). Since this entry contains the disk address of the update record, the desired record can be located and read into storage. The record can then be updated.

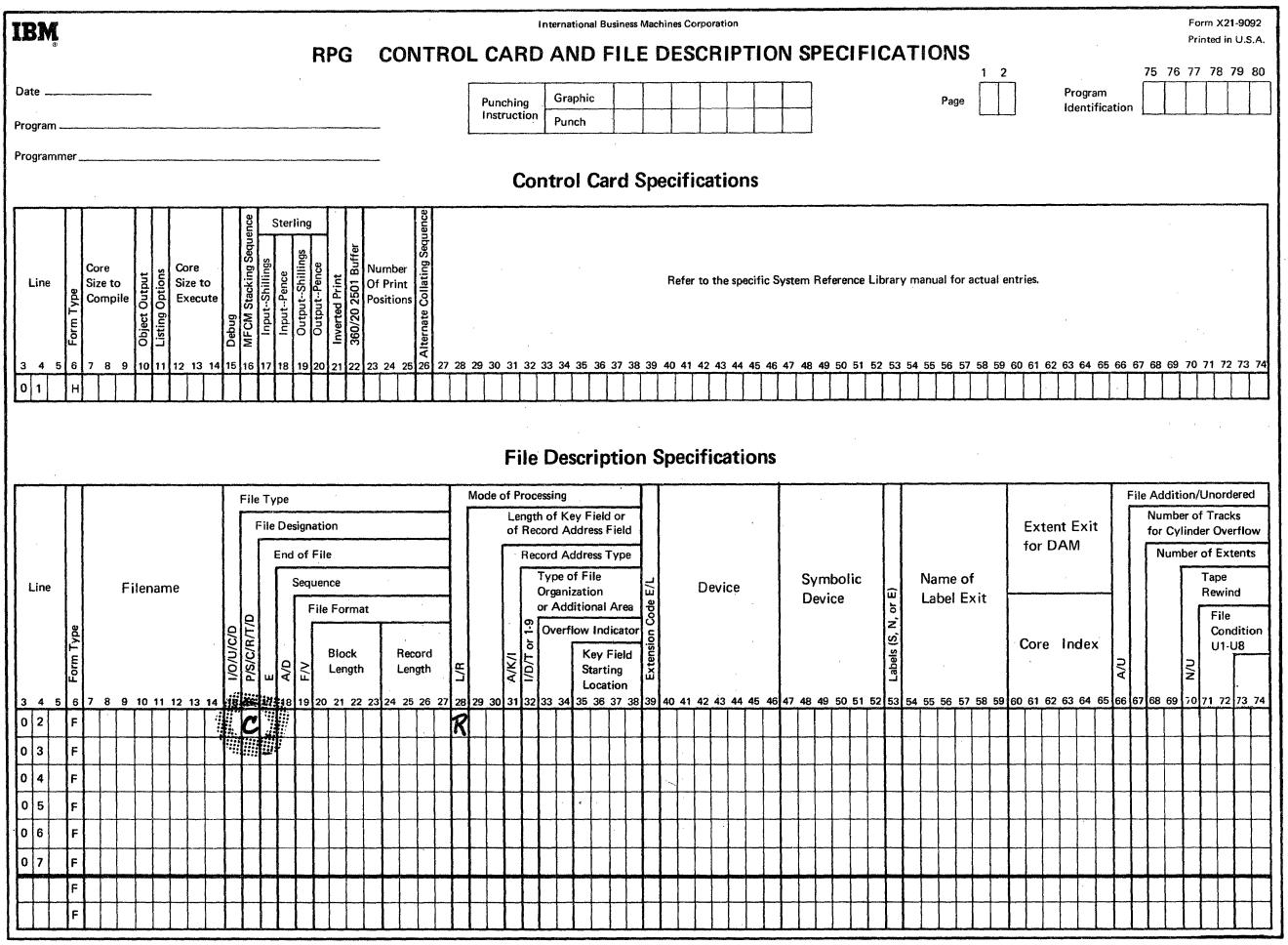
The other two entries are needed in case the desired record is *not* located. In the first two Resulting Indicator columns (54 - 57), enter the same indicator. The third Resulting Indicator columns (58 - 59) must be blank. When a record is not found in the update file, the specified indicator turns on.

A record may be updated at detail time or total time. That is, updating can be done after each transaction record is read or after a number of transaction records are read. Thus, columns 9 - 17 are used to specify conditioning indicators.

When an update file is *chained,* another entry is required on the File Description sheet.

The C in column 16 identifies a chained file. (An R in column 28 is needed since the file will be processed randomly by key.)

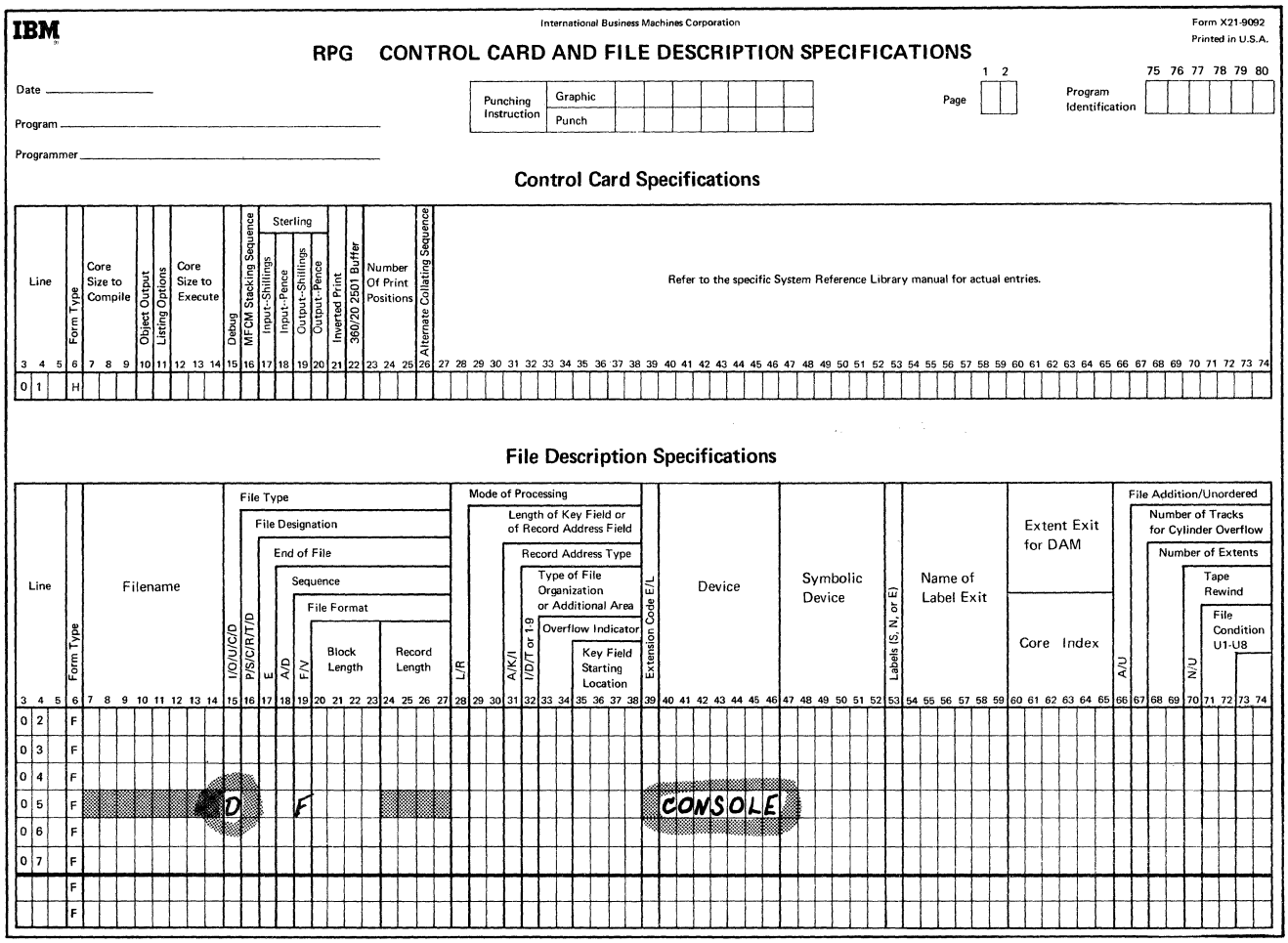
Job 6 (Figure 10) illustrates how the CHAIN operation code is used to update a master inventory indexed file, and at the same time print a back order report. The back order report lists every transaction item (by customer) where quantity shipped is less than quantity ordered.

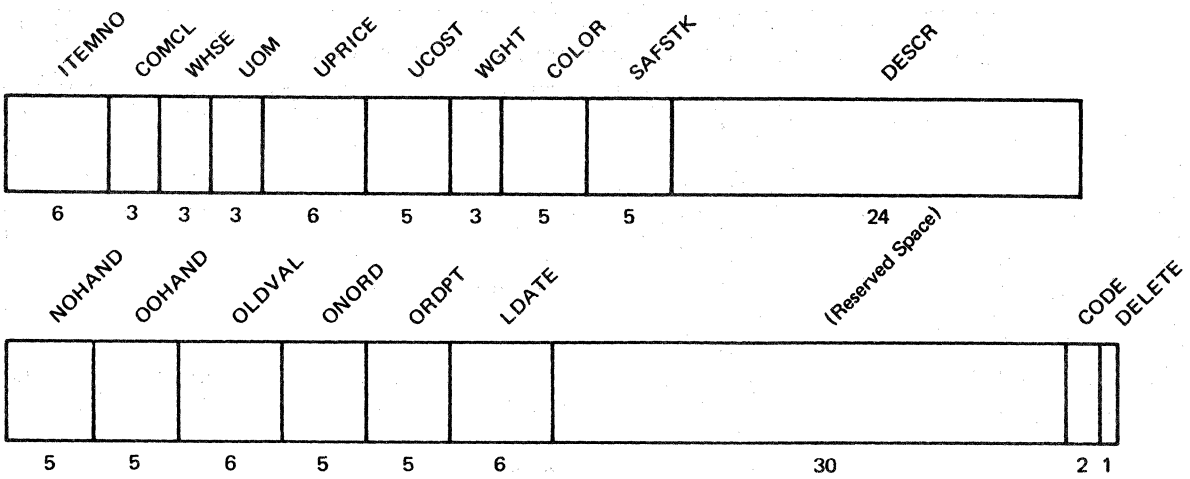


Updating Records Using the IBM 5471 Printer-Keyboard

The IBM 5471 Printer-Keyboard is an input/output device. It prints data from storage, accepts data into storage, or both prints and accepts data. The device can be used to print a field. For example, in Job 6, we could have printed all item numbers for which no disk records were found.

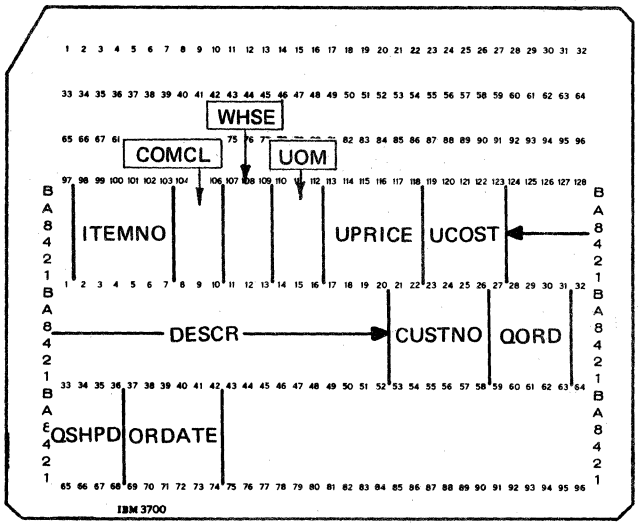
When fields are printed by the Printer-Keyboard, we say they are *displayed.* To use RPG II to display fields, the fields must be called on a *display file* and described on the File Description sheet. Two new entries are needed on the sheet.





Although 95 positions are presently used, the disk record is defined as 128 positions to provide space for adding data later. Field names are the same as the input card field names, with two exceptions: CODE = MI (Master Inventory) and DELETE (position reserved for a delete code).

ITEMNO = Item number
COMCL = Commodity class
WHSE = Warehouse location
UOM = Unit of measure
UPRICE = Unit price
UCOST = Unit cost
DESCR = Description
CUSTNO = Customer number
QORD = Quantity ordered
QSHPD = Quantity shipped
ORDATE = Order date



BACK ORDER LISTING

CUSTOMER	ITEM	SHIPPED	ORDERED	BACK ORD.	DATE
292165	A22446	100	300	200	3/17/69
430879	A40911	400	500	100	3/18/69
144367	B70128	60	125	65	3/17/69
590552	C80111	350	150	100	3/19/69

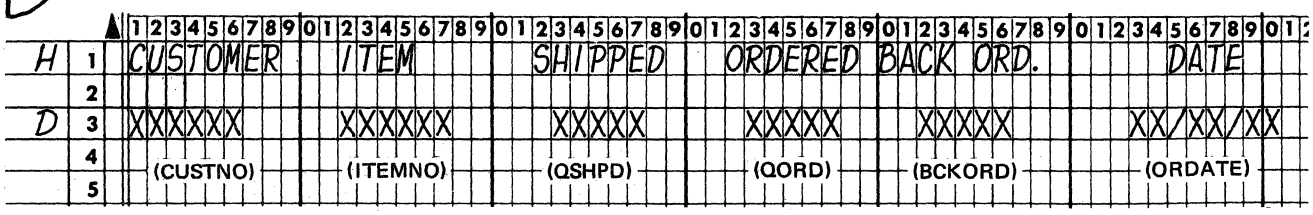
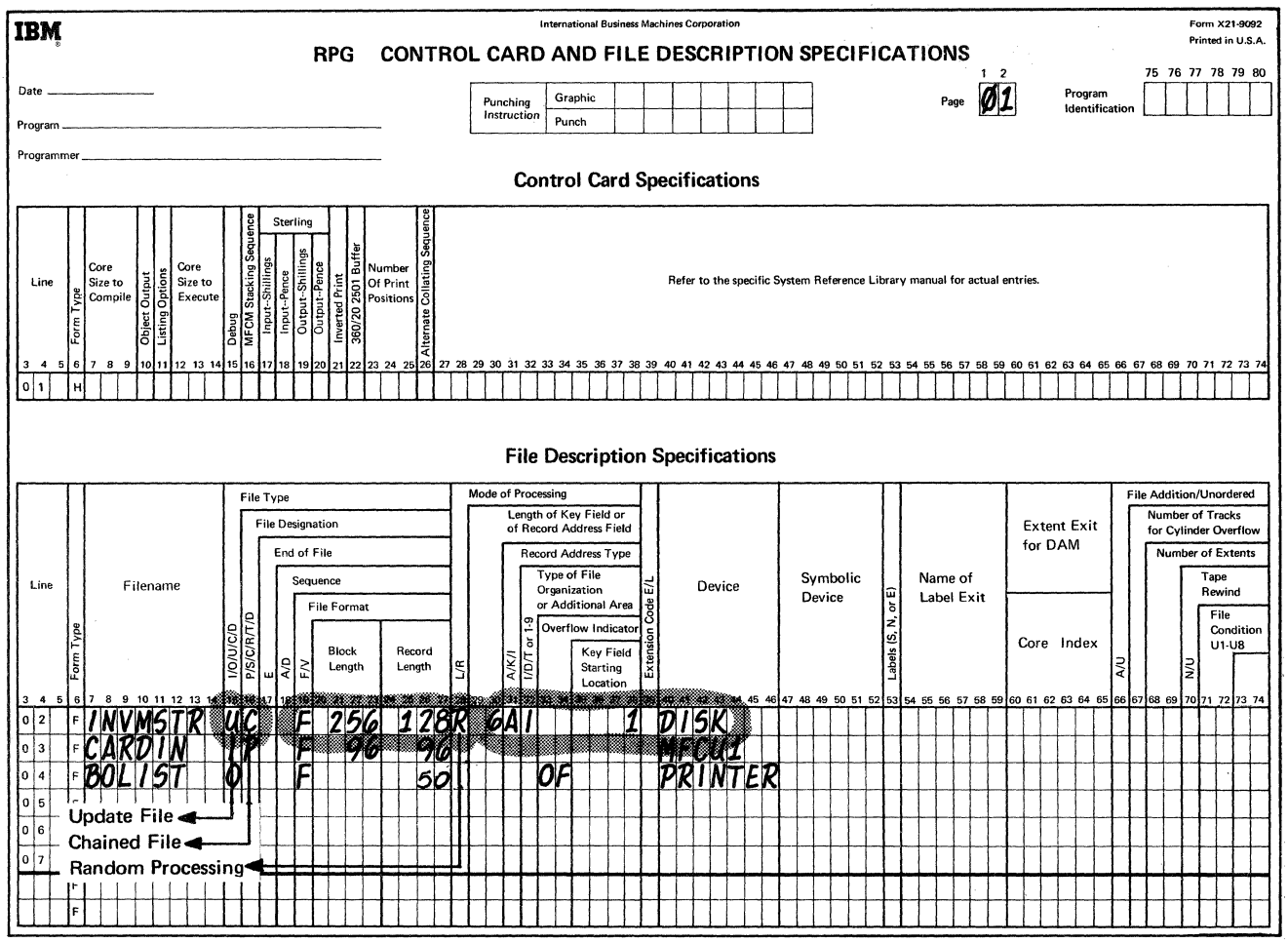


Figure 10. Job 6: Updating a Master Index File Using CHAIN (Part 1 of 3)



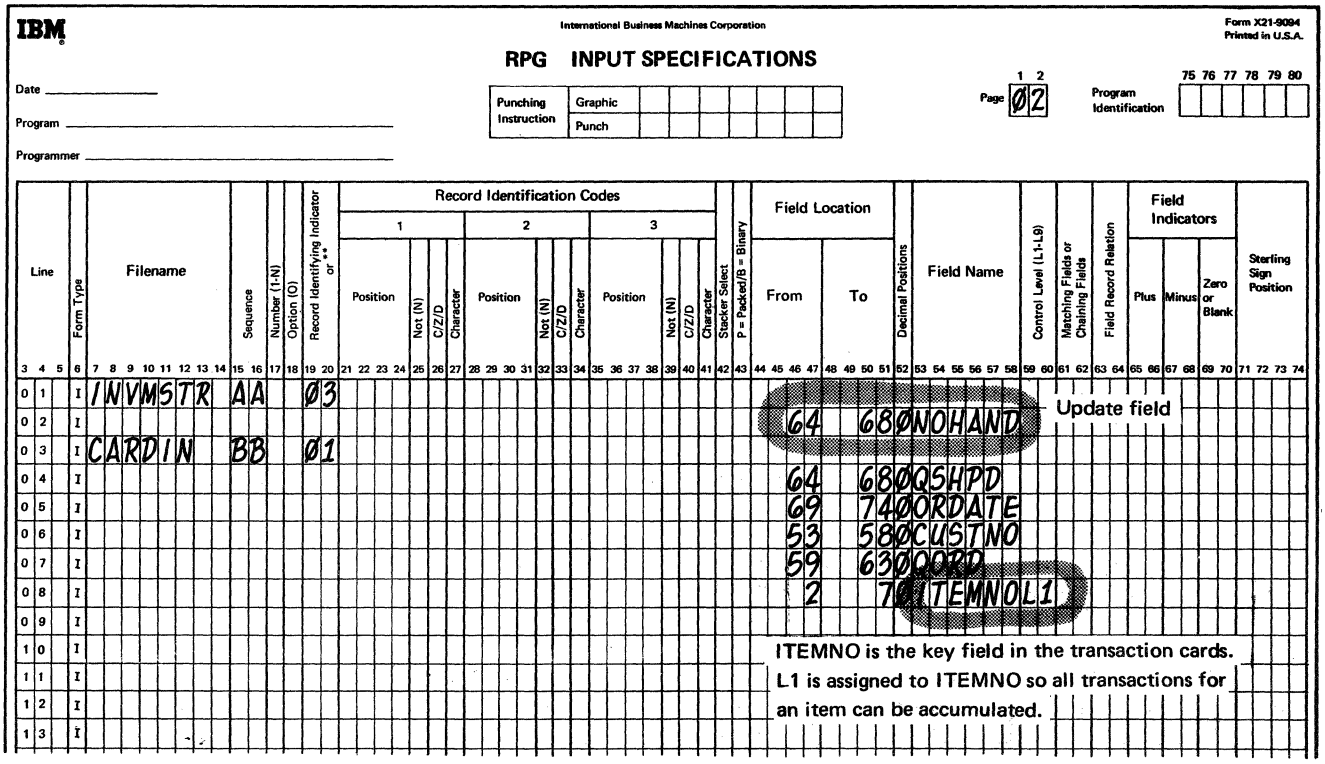


Figure 10. Job 6: Updating a Master Index File Using CHAIN (Part 2 of 3)

RPG CALCULATION SPECIFICATIONS

Date _____ Program _____ Programmer _____

Punching Instruction: Graphic / Punch

Page 03 Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators: And / And (Not)	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic (Plus / Minus / Zero)	Resulting Indicators: Compare (High 1>2 / Low 1<2 / Equal 1=2)	Resulting Indicators: Lookup (Table (Factor 2) is High / Low / Equal)	Comments
01	C		01	QORD	SUB	QSHPD	BCKORD	50					50	
02	C		01	QSHPD	ADD	TOTSHP	TOTSHP	60						
03	C	L1		ITEMNO	CHAIN	INVMSTR							9999	
04	C	L1	N99	NOHAND	SUB	TOTSHP	NOHAND							
05	C	L1			Z-ADD	0	TOTSHP							

Line 01 Quantity shipped is subtracted from quantity ordered for each transaction card. The remainder is stored in a 5-position field called BCKORD. If quantity shipped is less than quantity ordered, indicator 50 turns on.

Line 02 Quantity shipped is accumulated.

Line 03 When L1 turns on (ITEMNO changes), the transaction card's key field chains to the update file. If a like key is found, indicator 99 stays off. Indicator 99 turns on only if no key is found in the update file.

Line 04 When ITEMNO changes and a like key is found (L1 is on and 99 is off), the on hand quantity is updated.

Line 05 After the on hand quantity is updated, TOTSHP is set to zero, so accumulations for the next item will be correct. In this program Z-ADD does the same function as Blank After does on the Output sheet. However, TOTSHP does not appear on the Output sheet, so the field must be set to zero on the Calculation sheet.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date _____ Program _____ Programmer _____

Punching Instruction: Graphic / Punch

Page 04 Program Identification

Line	Form Type	Filename	Type (H/D/T/E)	Stacker Select/Fetch Overflow (F)	Space: Before	Space: After	Skip: Before	Skip: After	Output Indicators: And / And (Not)	Field Name	Edit Codes	Blank After (B)	End Position in Output Record	P = Packed/B = Binary	Constant or Edit Word
01	O	BOLIST	H		1				1P						
02	O		OR						OF						
03	O												8		'CUSTOMER'
04	O												15		'ITEM'
05	O												28		'SHIPPED'
06	O												38		'ORDERED'
07	O												48		'BACK ORD.'
08	O												58		'DATE'
09	O		D	1					50						
10	O									CUSTNO			6		
11	O									ITEMNO			17		
12	O									QSHPD			27		
13	O									QORD			37		
14	O									BCKORD			46		
15	O									ORDATE			60		
	O	INVMSTR	T						L1N99						
	O									NOHAND			68		

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Sterling Sign Position

09 - 15 When indicator 50 is on, a line is printed. This line reflects those items where the quantity shipped is less than the quantity ordered, that is, the back orders.

16 - 17 On hand quantity is written on the disk in its original location. The field was updated as a result of calculations.

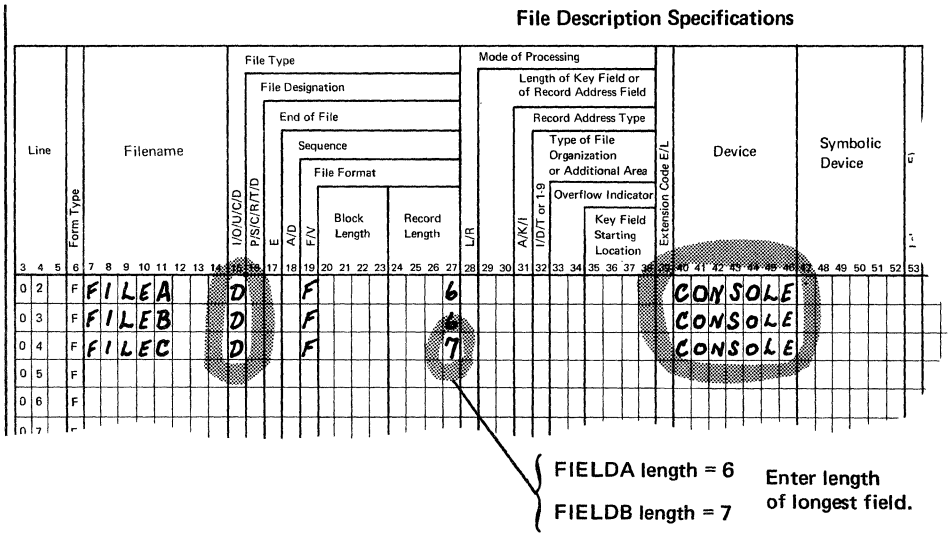
Figure 10. Job 6: Updating a Master Index File Using CHAIN (Part 3 of 3)

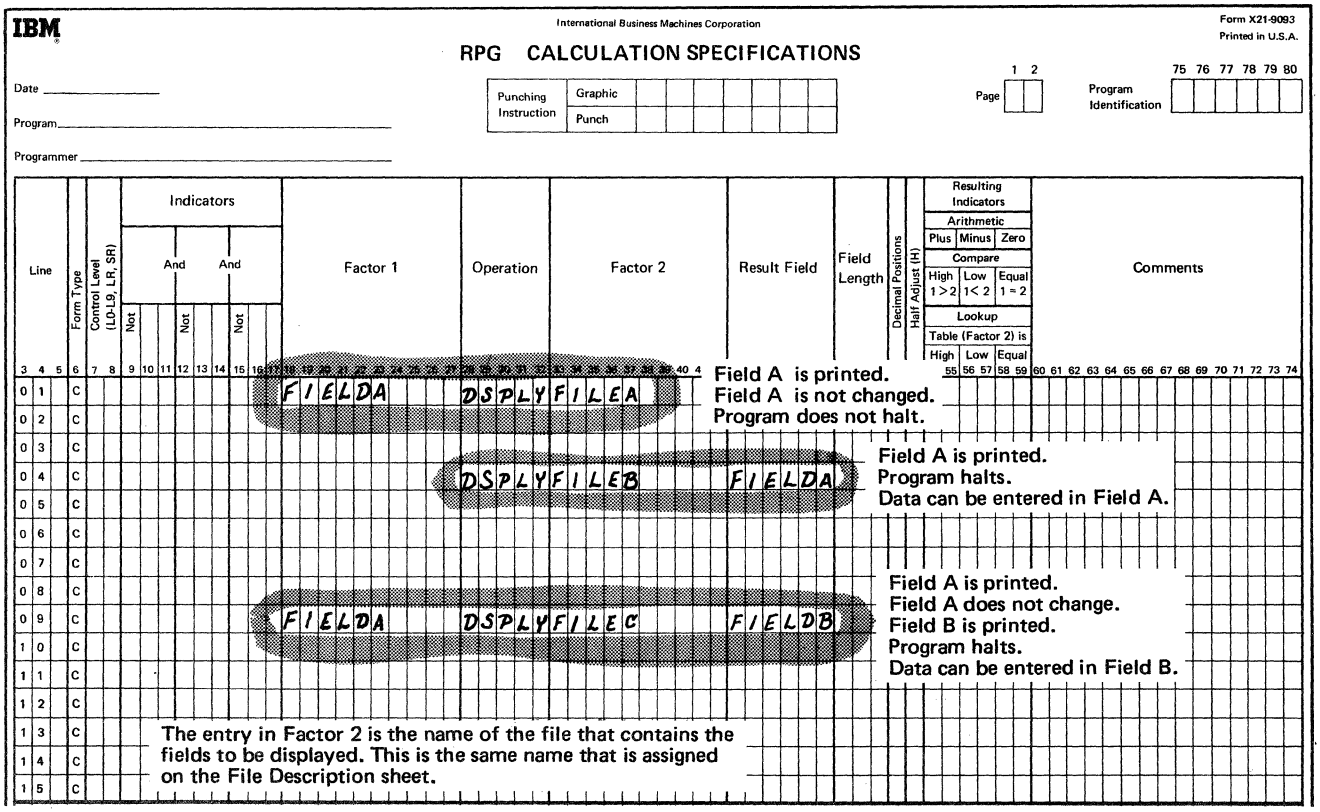
D identifies the file as a display file; CONSOLE is the required device name for the Printer-Keyboard.

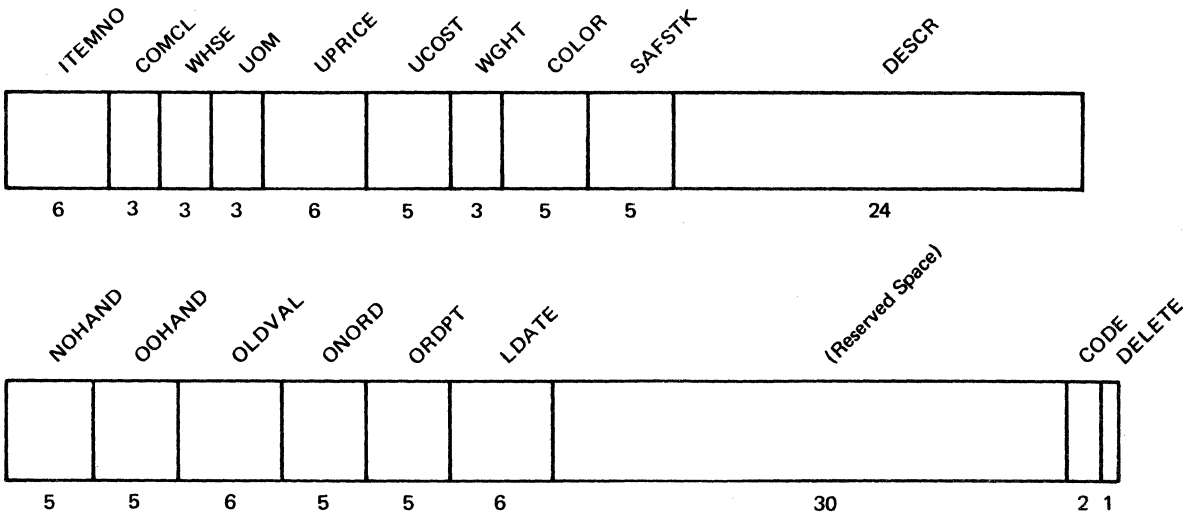
Since a file is defined, F is required in column 19, and record length is required in columns 24 - 27.

When fields are displayed, the DSPLY operation code must be used. Fields can be displayed three ways using DSPLY. These three methods and their corresponding File Description entries are:

Job 7 (Figure 11) shows how a display file can be used during an update job to print the item numbers for which no master record is found. Note that this job is the same as Job 6 (Figure 10) with the addition of the Printer-Keyboard.

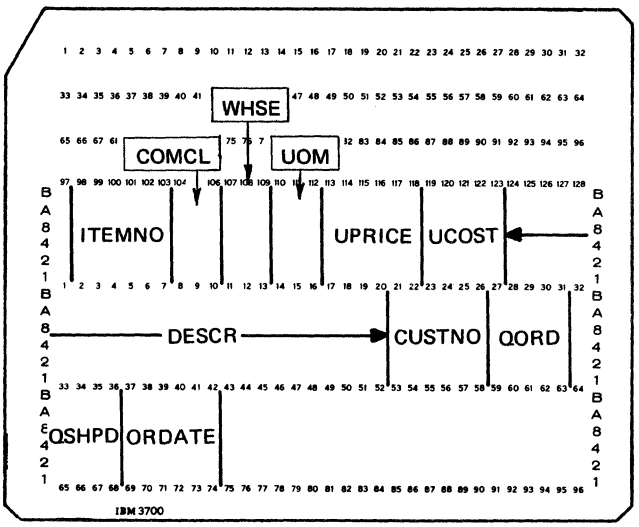






Although 95 positions are presently used, the disk record is defined as 128 positions to provide space for adding data later. Field names are the same as the input card field names, with two exceptions: CODE = MI (Master Inventory) and DELETE (position reserved for a delete code).

ITEMNO = Item number
COMCL = Commodity class
WHSE = Warehouse location
UOM = Unit of measure
UPRICE = Unit price
UCOST = Unit cost
DESCR = Description
CUSTNO = Customer number
QORD = Quantity ordered
QSHPD = Quantity shipped
ORDATE = Order date



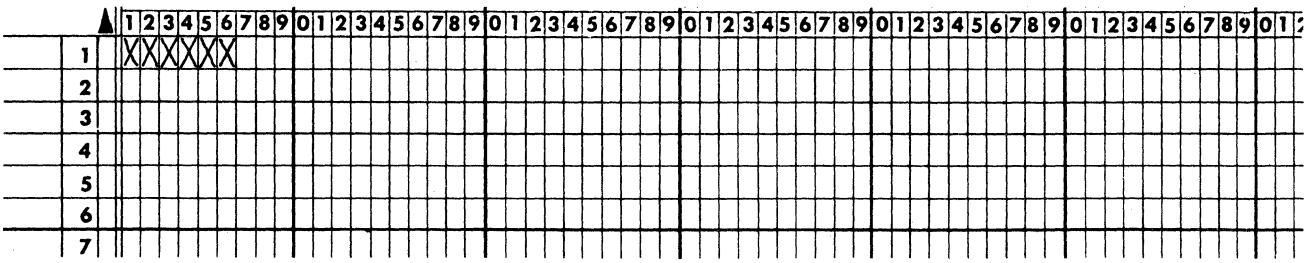
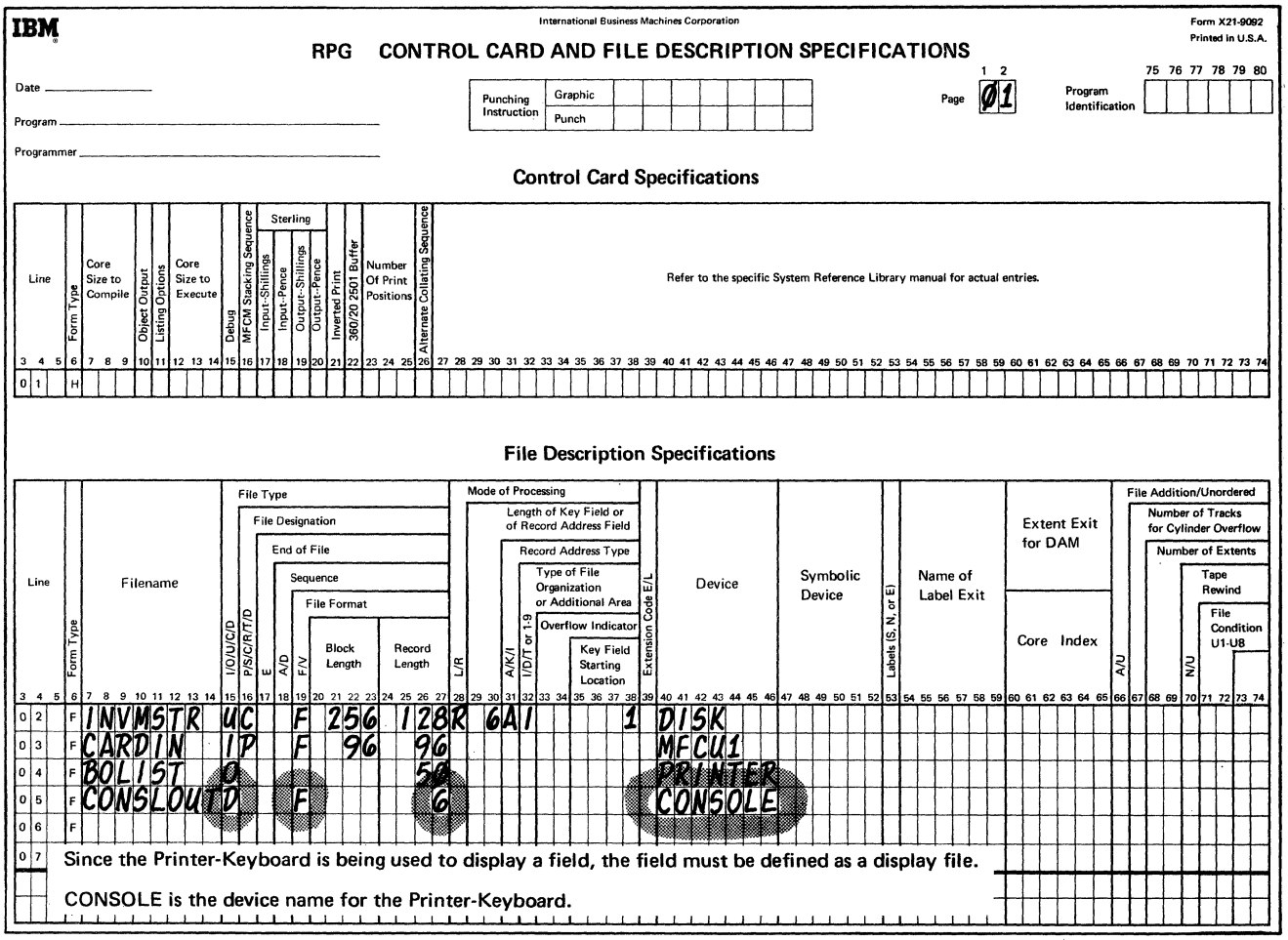


Figure 11. Job 7: Updating an Indexed File and Using the Printer-Keyboard (Part 1 of 3)



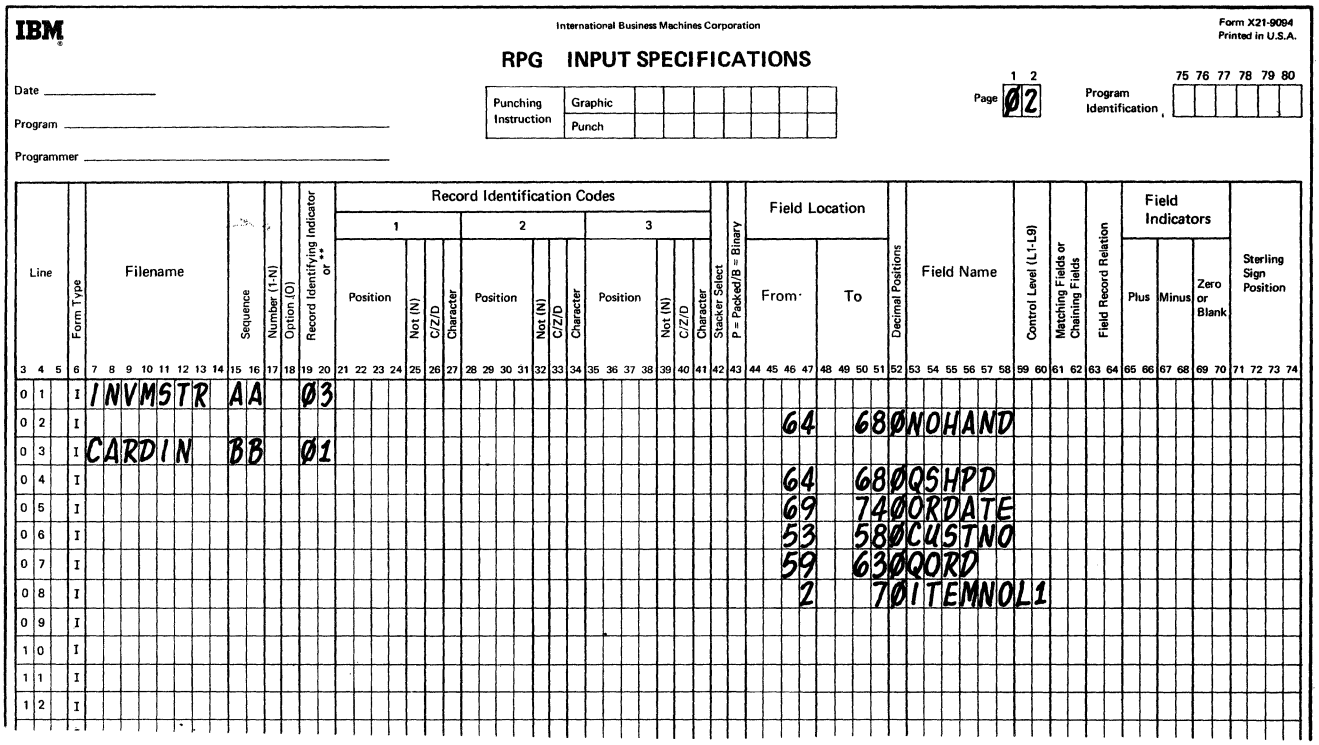
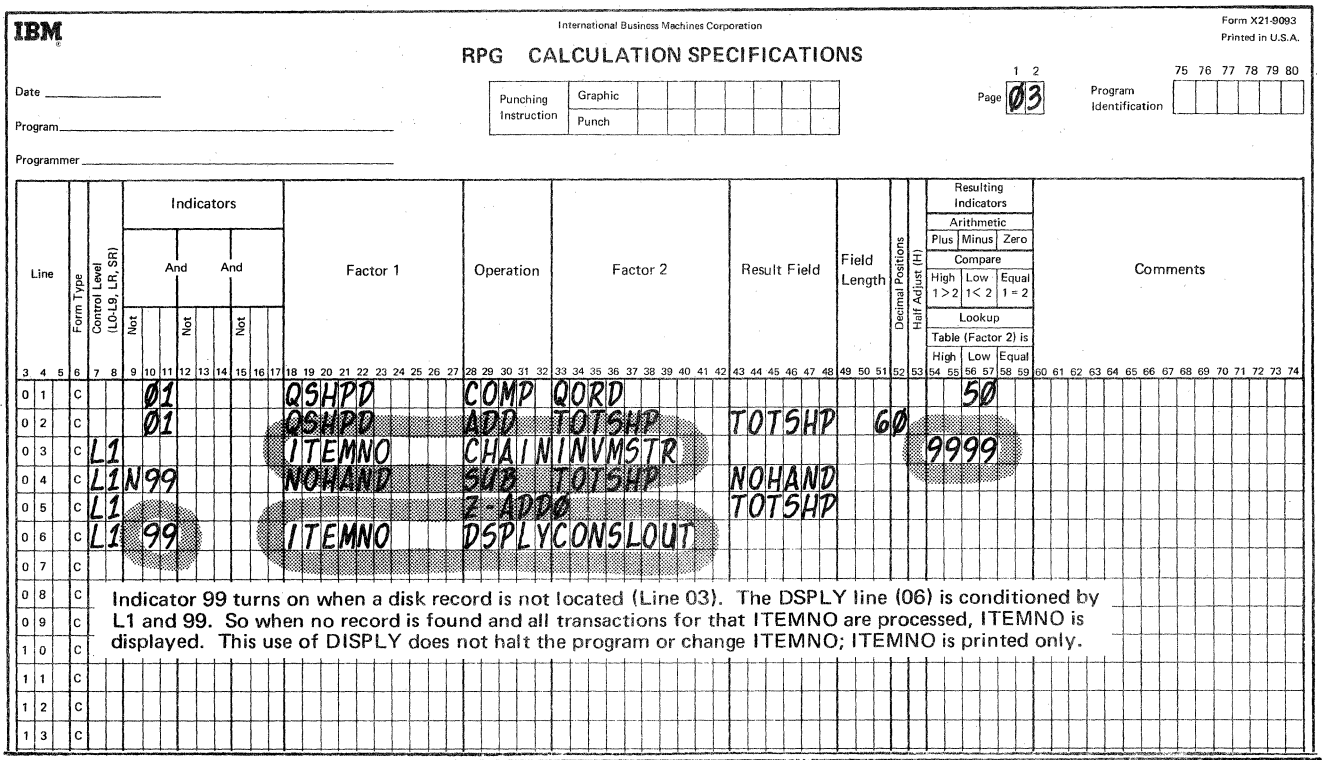


Figure 11. Job 7: Updating an Indexed File and Using the Printer-Keyboard (Part 2 of 3)



IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date _____

Program _____

Programmer _____

Punching Instruction: Graphic / Punch

Page 03

Program Identification

Line	Form Type	Control Level (LO-L9, LR, SR)	Indicators: And / And — Not	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic (Plus / Minus / Zero); Compare (High 1>2 / Low 1<2 / Equal 1=2); Lookup Table (Factor 2) is (High / Low / Equal)	Comments
01	C		01	QSHPD	COMP	QORD					50	
02	C		01	QSHPD	ADD	TOTSHP	TOTSHP	60				
03	C	L1		ITEMNO	CHAIN	INVMSTR					9999	
04	C	L1	N99	NOHAND	SUB	TOTSHP	NOHAND					
05	C	L1			Z-ADD	0	TOTSHP					
06	C	L1	99	ITEMNO	DSPLY	CONSLOUT						
07	C											
08	C											
09	C											
10	C											
11	C											
12	C											
13	C											

Indicator 99 turns on when a disk record is not located (Line 03). The DSPLY line (06) is conditioned by L1 and 99. So when no record is found and all transactions for that ITEMNO are processed, ITEMNO is displayed. This use of DISPLY does not halt the program or change ITEMNO; ITEMNO is printed only.

IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date _____

Program _____

Programmer _____

Punching Instruction: Graphic / Punch

Page 04

Program Identification

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Stacker Select/Fetch Overflow (F)	Space: Before	Space: After	Skip: Before	Skip: After	Output Indicators: And / And — Not	Field Name	Edit Codes	Blank After (B)	End Position in Output Record	P = Packed/B = Binary	Constant or Edit Word	Sterling Sign Position
01	O	BOLIST	H		1				1P							
02	O		OR						OF							
03	O												8		'CUSTOMER'	
04	O												15		'ITEM'	
05	O												28		'SHIPPED'	
06	O												38		'ORDERED'	
07	O												48		'DATE'	
08	O		D	1					50							
09	O									CUSTNO			6			
10	O									ITEMNO			17			
11	O									QSHPD	Z		27			
12	O									QORD	Z		37			
13	O									ORDATEY			50			
14	O	INVMSTR	T						LIN99							
15	O									NOHAND			68			
	O															
	O															
	O															
	O															
	O															

Figure 11. Job 7: Updating an Indexed File and Using the Printer-Keyboard (Part 3 of 3)

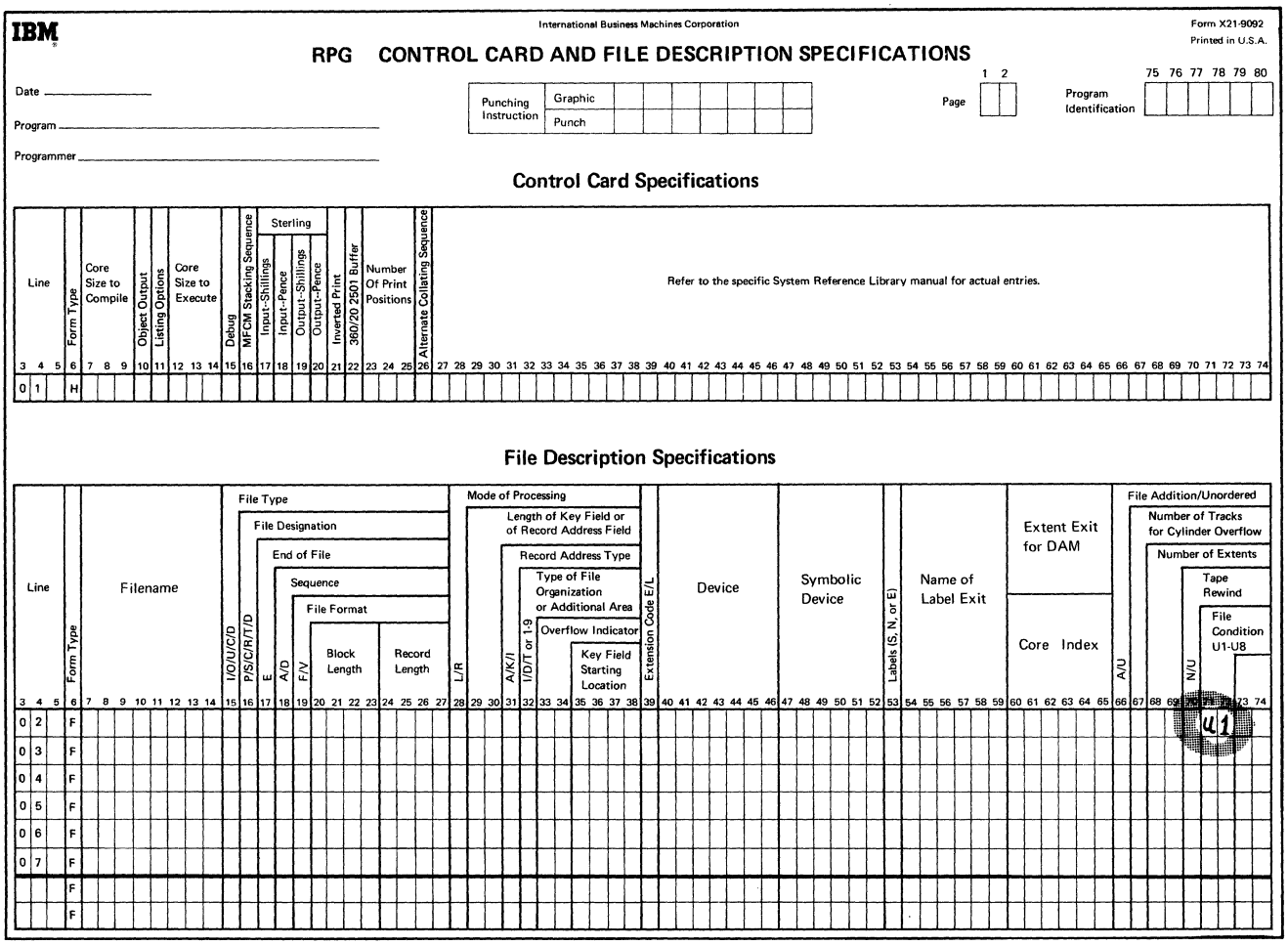
USING EXTERNAL INDICATORS WITH DISK FILES

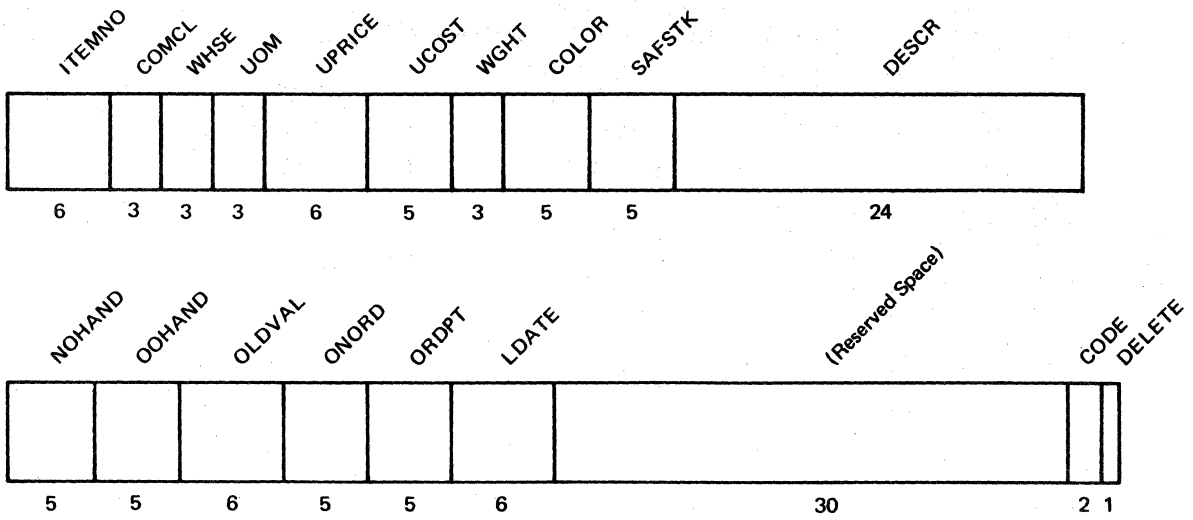
You can condition a file outside an RPG II program by using an *external indicator.* The indicator is called external because it is set on or off prior to running a job. It is set by an OCL card and *cannot* be changed during the job. The program has no control over it. Chapter 4, Operation Control Language (OCL), explains how to set external indicators.

RPG II provides eight external indicators, specified as U1-U8. External indicators are entered in columns 71-72 on the File Description sheet.

External indicators are used to condition when a file is used. For instance, a job may require two output files one time and one file another time. Instead of writing two different programs (one program having one file, the other two), you can condition a file by an external indicator. When the indicator (U1-U8) is *on*, the file is *used*; when it is *off*, the file is *not used.*

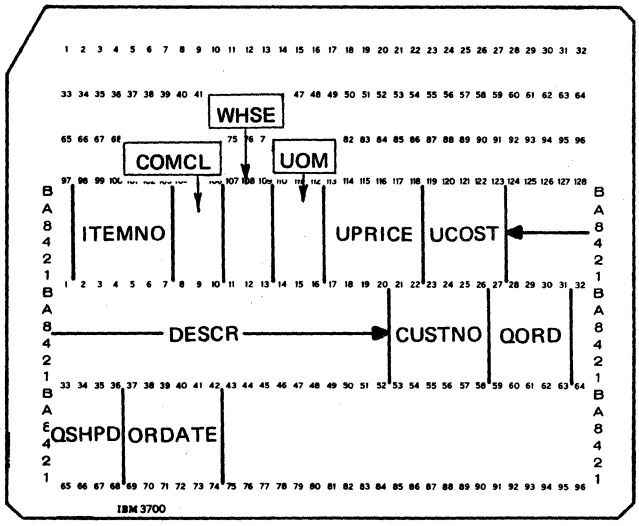
In addition to using these indicators to condition a file, you may use them to condition calculation and output operations. Job 8 (Figure 12) shows the use of an external indicator that conditions a Printer-Keyboard file, conditions calculations, and conditions output.





Although 95 positions are presently used, the disk record is defined as 128 positions to provide space for adding data later. Field names are the same as the input card field names, with two exceptions: CODE = MI (Master Inventory) and DELETE (position reserved for a delete code).

ITEMNO = Item number
COMCL = Commodity class
WHSE = Warehouse location
UOM = Unit of measure
UPRICE = Unit price
UCOST = Unit cost
DESCR = Description
CUSTNO = Customer number
QORD = Quantity ordered
QSHPD = Quantity shipped
ORDATE = Order date



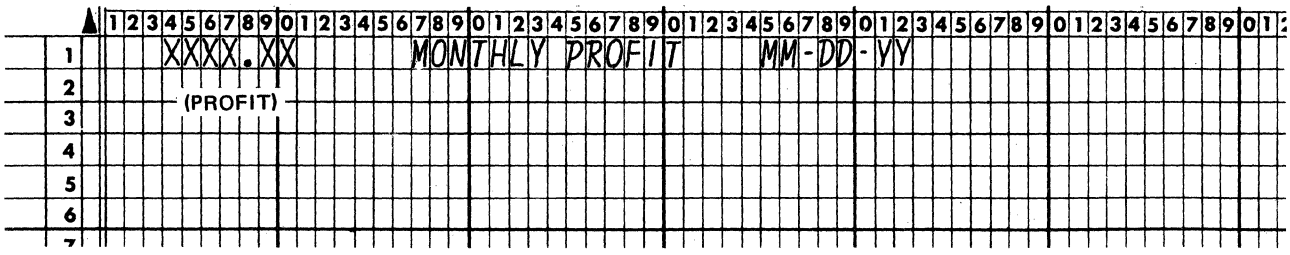


Figure 12. Job 8: Using the Printer-Keyboard as an Output File (Part 1 of 3)

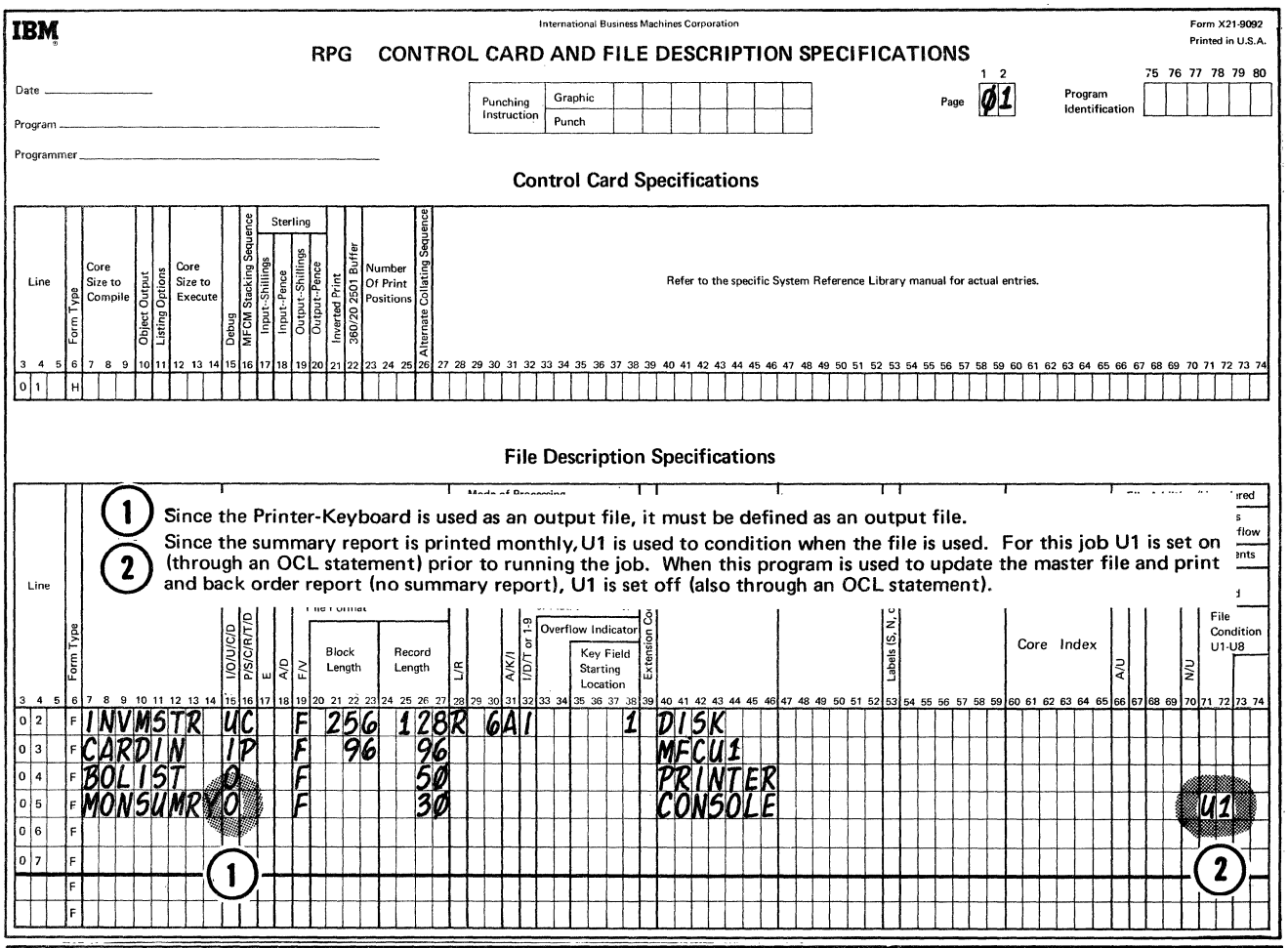
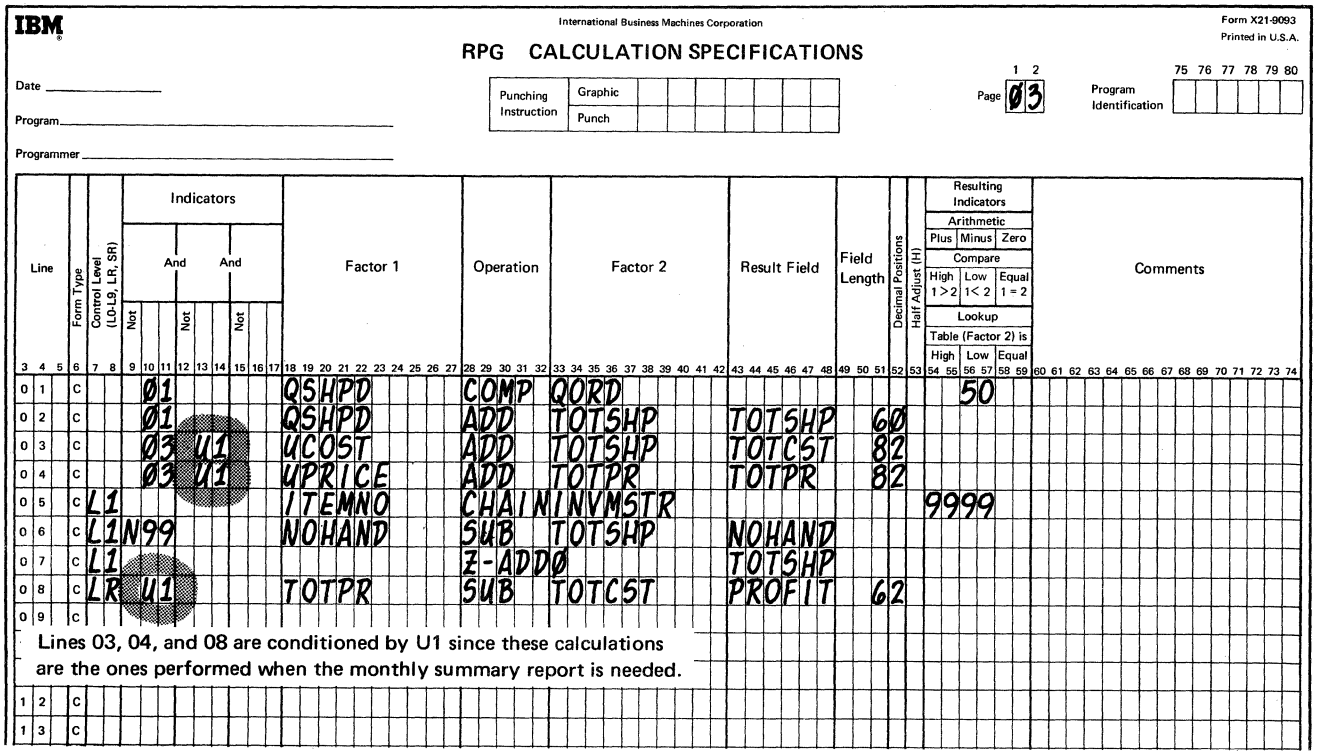


Figure 12. Job 8: Using the Printer-Keyboard as an Output File (Part 2 of 3)



IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date _____

Program _____

Programmer _____

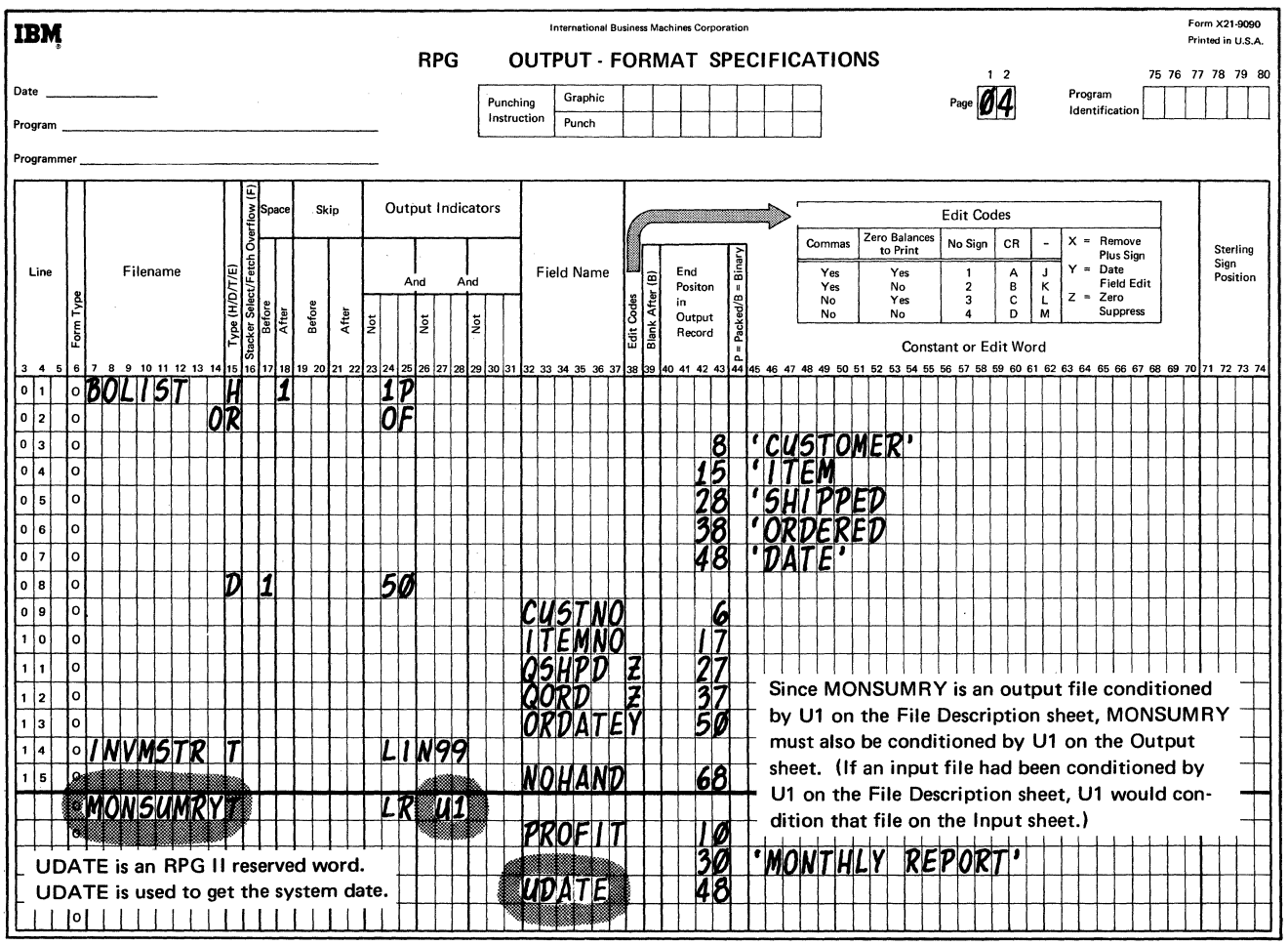
Punching Instruction: Graphic / Punch

Page 03

Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators: And / And — Not	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic (Plus / Minus / Zero); Compare (High 1>2 / Low 1<2 / Equal 1=2); Lookup; Table (Factor 2) is (High / Low / Equal)	Comments
01	C		01	QSHPD	COMP QORD						50	
02	C		01	QSHPD	ADD TOTSHP		TOTSHP	60				
03	C		03 U1	UCOST	ADD TOTSHP		TOTCST	82				
04	C		03 U1	UPRICE	ADD TOTPR		TOTPR	82				
05	C	L1		ITEMNO	CHAIN INVMSTR						9999	
06	C	L1N99		NOHAND	SUB TOTSHP		NOHAND					
07	C	L1			Z-ADD0		TOTSHP					
08	C	LR U1		TOTPR	SUB TOTCST		PROFIT	62				
09	C											
12	C											
13	C											

Lines 03, 04, and 08 are conditioned by U1 since these calculations are the ones performed when the monthly summary report is needed.



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date _____

Program _____

Programmer _____

Punching Instruction: Graphic / Punch

Page 04

Program Identification

Line	Form Type	Filename	Type (H/D/T/E)	Stacker Select/Fetch Overflow (F): Space Before / Space After	Skip Before / Skip After	Output Indicators: And / And — Not	Field Name	Edit Codes	Blank After (B)	End Position in Output Record	P = Packed/B = Binary	Constant or Edit Word
01	O	BOLIST	H	1		1P						
02	O		OR			OF						
03	O									8		'CUSTOMER'
04	O									15		'ITEM
05	O									28		'SHIPPED
06	O									38		'ORDERED
07	O									48		'DATE'
08	O		D	1		50						
09	O						CUSTNO			6		
10	O						ITEMNO			17		
11	O						QSHPD Z			27		
12	O						QORD Z			37		
13	O						ORDATEY			50		
14	O	INVMSTR	T			L1N99						
15	O	MONSUMRY				LR U1	NOHAND			68		
	O						PROFIT			10		
	O						UDATE			30		'MONTHLY REPORT'
	O									48		

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

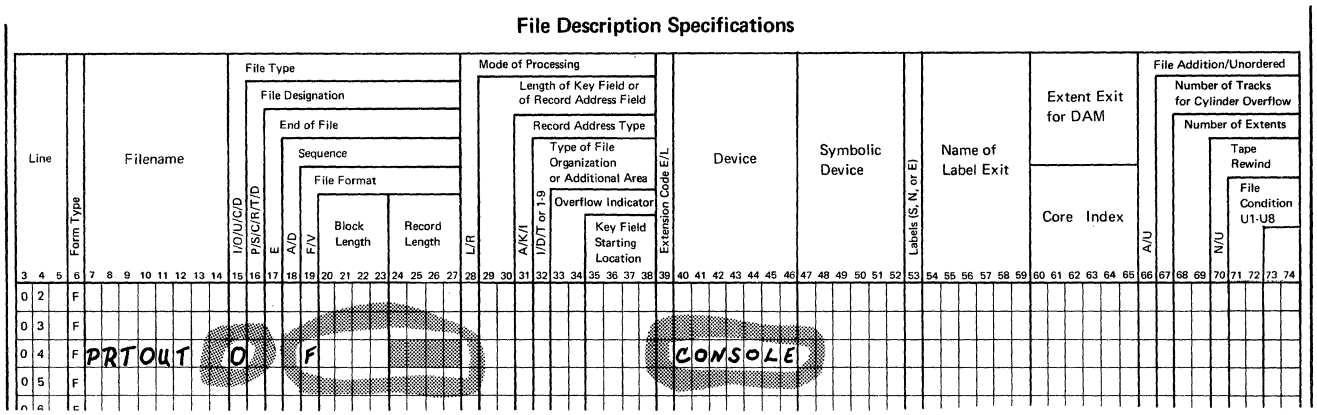
Sterling Sign Position

Since MONSUMRY is an output file conditioned by U1 on the File Description sheet, MONSUMRY must also be conditioned by U1 on the Output sheet. (If an input file had been conditioned by U1 on the File Description sheet, U1 would condition that file on the Input sheet.)

UDATE is an RPG II reserved word.
UDATE is used to get the system date.

Figure 12. Job 8: Using the Printer-Keyboard as an Output File (Part 3 of 3)

Besides being used to display fields, the Printer-Keyboard can be used as a second printer. It is so used when the printer is being used. When the Printer-Keyboard is used as a second printer, *the file must be described as an output file.*



In Job 8 (Figure 12) the master inventory file is used again. The previous jobs updated the file and printed a back order listing. Depending on the number of transactions, this job may be performed every week or every day.

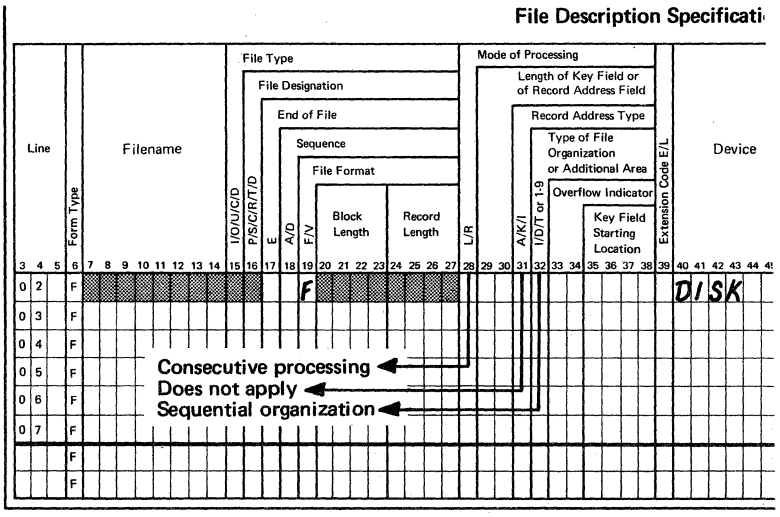
At the end of the month, however, along with updating and printing the back order report, a summary report of that month's profit is needed. Profit is calculated by accumulating unit cost and unit price for each item, and then subtracting the total cost from the total price.

The Printer-Keyboard is used to print profit; thus it is an output file. For this month-end run, U1 must be turned on by an OCL card prior to running the job. For the next run, the summary report is not needed. Thus, U1 must be turned off for that run. External indicators are used, then, to regulate when certain functions are performed.

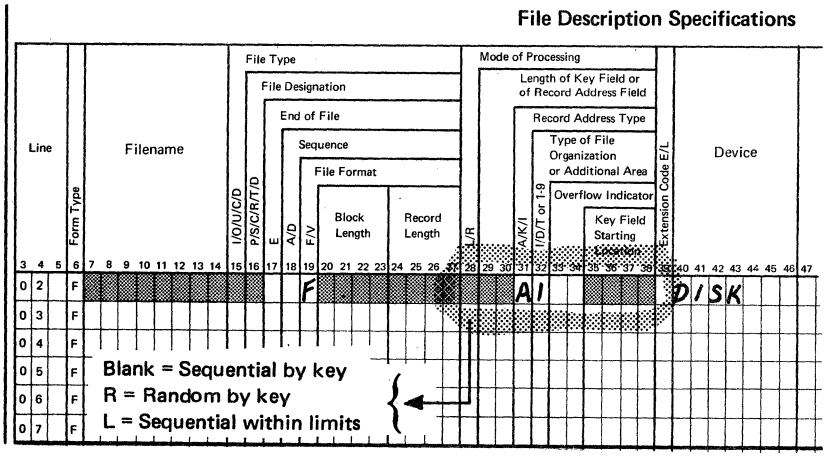
REVIEW: RPG II ENTRIES FOR DISK FILES

When a sequential file is processed consecutively, all records are read one after the other in the order they were created. For any job where a sequential file is used, the same disk entries are required on the File Description sheet.

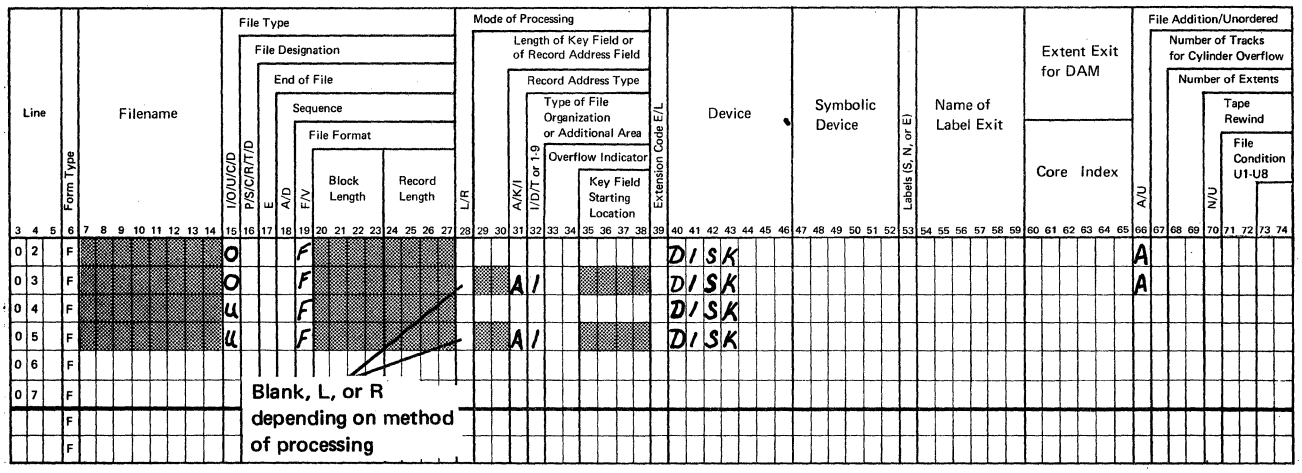
Columns 28, 31, and 32 identify file organization and processing method. For sequential files processed consecutively, these three columns are always blank.



For indexed files the same entries are required, plus five others to describe the index.



If a job involves additions to a file or updating, the same entries are required for both sequential and indexed files.



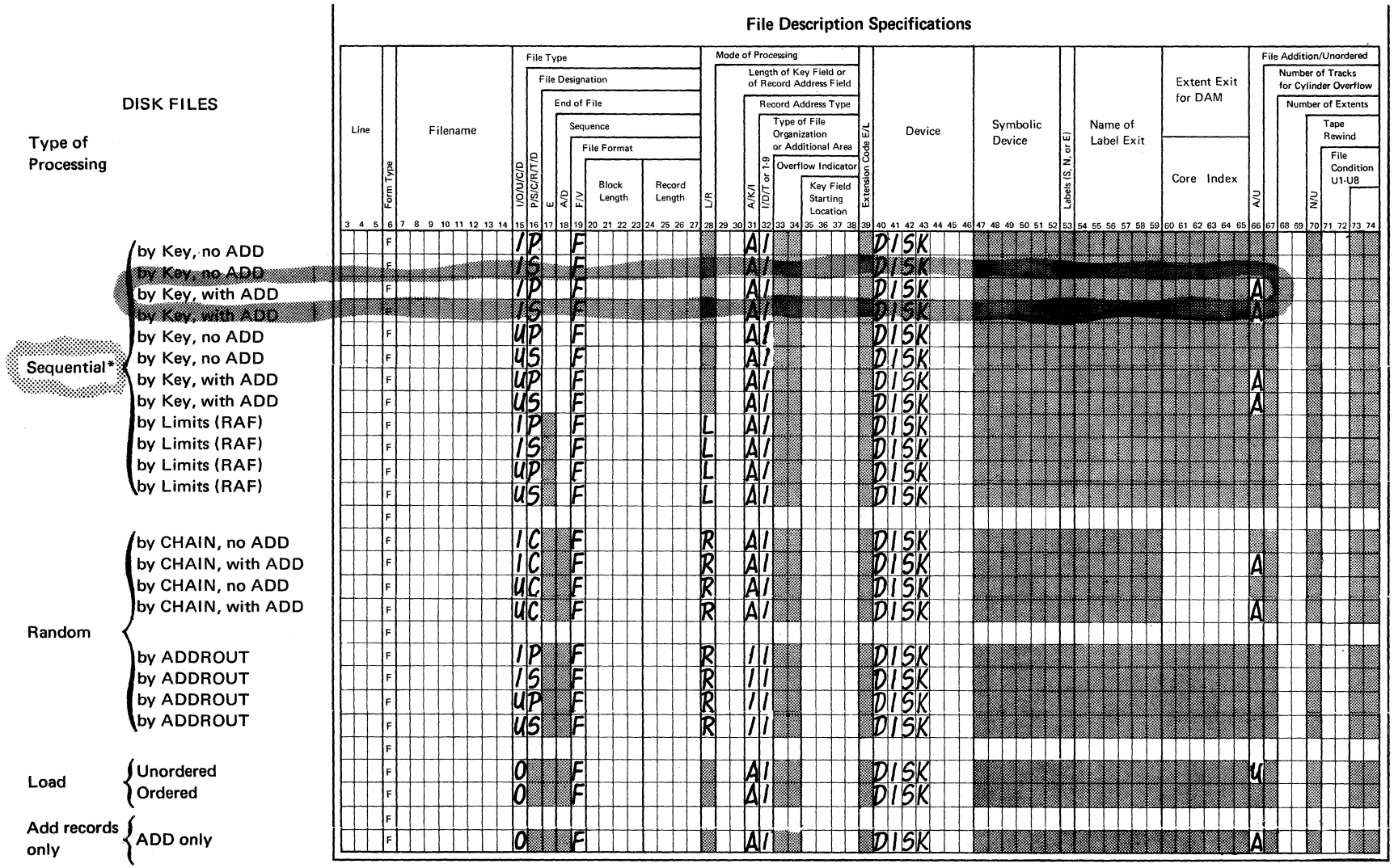
The first part of this chapter explained the RPG II entries used to perform such functions as adding records, updating records, and creating files. These functions and their corresponding entries were explained separately; however, a combination of these functions can be performed in one job. For example, during one job, inventory records are updated and added. As a result of shipping, the items-on-hand quantity (in master record) is decreased by the quantity shipped; this is an update functions. When a new item is added to the inventory, a master record containing information for this item must be added to the master file. This is an add function: a new record is added.

The *Disk System RPG II Reference Manual* contains several pages of File Description sheets that show various combinations of entries. You can use these sheets to determine the entries required to do a certain function or combination of functions. Figure 13 shows processing methods for indexed disk files. For example, the third line shows the disk entries needed to add records to an indexed file that is processed sequentially by key.

THREE SAMPLE PROCESSING JOBS

The next three jobs (Figures 14, 15, and 16) show typical processing of disk files. Job 9 (Figure 14) is an example of detail printing from a sequential file. Name and address labels are printed from a name and address file. The labels are used to mail brochures and catalogs to customers.

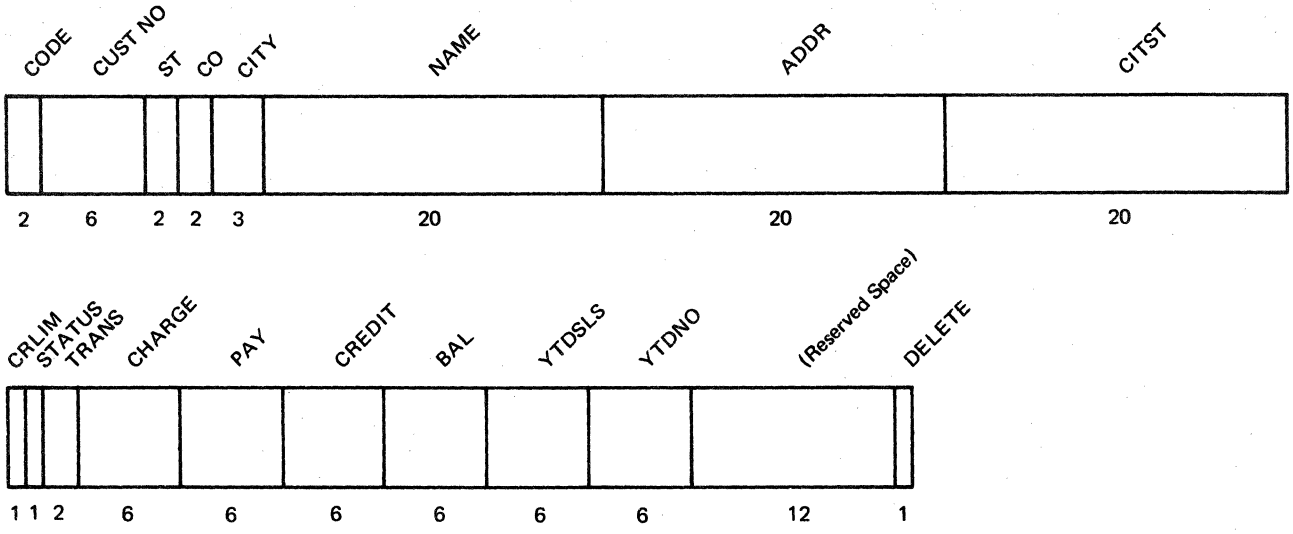
Job 10 (Figure 15) shows how matching records are used with a sequential file, and Job 11 (Figure 16) shows how matching records are used with an indexed file.



* Sequential processing by key or limits must use the file index, which is always arranged in ascending sequence. When an indexed file is processed record by record from beginning to end, the file is processed through the index using the sequential by key method.

Note: Shaded columns do not need to be coded. Unshaded columns must or may be coded.

Figure 13. Processing Methods for Indexed Disk Files



CODE	=	Record code (CM)	ADDR	=	Customer address (street)	PAY	=	Current month payments
CUSTNO	=	Customer number	CITST	=	Customer address (city and state)	CREDIT	=	Current month credits
ST	=	State code	CRLIM	=	Credit limit	BAL	=	Balance
CO	=	County code	STATUS	=	Status code	YTDSLS	=	Year-to-date sales amount
CITY	=	City code	TRANS	=	Number of transactions this month	YTDNO	=	Year-to-date number of sales
NAME	=	Customer name	CHARGE	=	Current month charges	DELETE	=	Delete code

IBM

International Business Machines Corporation

Form X21-9092
Printed in U.S.A.

RPG CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

Date _____
Program _____
Programmer _____

Punching Instruction: Graphic / Punch

Page 01

Program Identification

Control Card Specifications

Line	Form Type	Core Size to Compile	Object Output	Listing Options	Core Size to Execute	Debug	MFCM Stacking Sequence	Sterling: Input-Shillings	Sterling: Input-Pence	Sterling: Output-Shillings	Sterling: Output-Pence	Inverted Print	360/20 2501 Buffer	Number Of Print Positions	Alternate Collating Sequence	Refer to the specific System Reference Library manual for actual entries.
01	H															

File Description Specifications

Line	Form Type	Filename	I/O/U/C/D	P/S/C/R/T/D	E	A/D	F/V	Block Length	Record Length	L/R	A/K/I	I/D/T or 1-9	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels (S, N, or El)	Name of Label Exit	Extent Exit for DAM	Core Index	A/U	N/U	Tape Rewind	File Condition U1-U8
02	F	SEQDISK	I	P			F	256	128						DISK									
03	F	LISTING	O				F	20	20						PRINTER									
04	F																							
05	F																							
06	F																							
07	F																							
	F																							
	F																							

Consecutive processing
Not applicable
Sequential organization

Columns 28, 31, and 32 are always blank when a sequential file is processed consecutively.

Figure 14. Job 9: Detail Printing from a Sequential File (Part 1 of 2)

IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date _____

Program _____

Programmer _____

Punching Instruction	Graphic								
	Punch								

Page 02

Program Identification

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Record Identification Codes 1: Position	1: Not (N)	1: C/Z/D	1: Character	2: Position	2: Not (N)	2: C/Z/D	2: Character	3: Position	3: Not (N)	3: C/Z/D	3: Character	Stacker Select	P = Packed/B = Binary	Field Location: From	Field Location: To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators: Plus	Field Indicators: Minus	Field Indicators: Zero or Blank	Sterling Sign Position
01	I	SEQDISK	AA			01																									
02	I																				16	35		NAME							
03	I																				36	55		ADDR							
04	I																				56	75		CITST							
05	I																														
06	I																														
07	I																														
08	I																														
09	I																														
10	I																														
11	I																														
12	I																														
13	I																														

IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date _____

Program _____

Programmer _____

Punching Instruction	Graphic								
	Punch								

Page 03

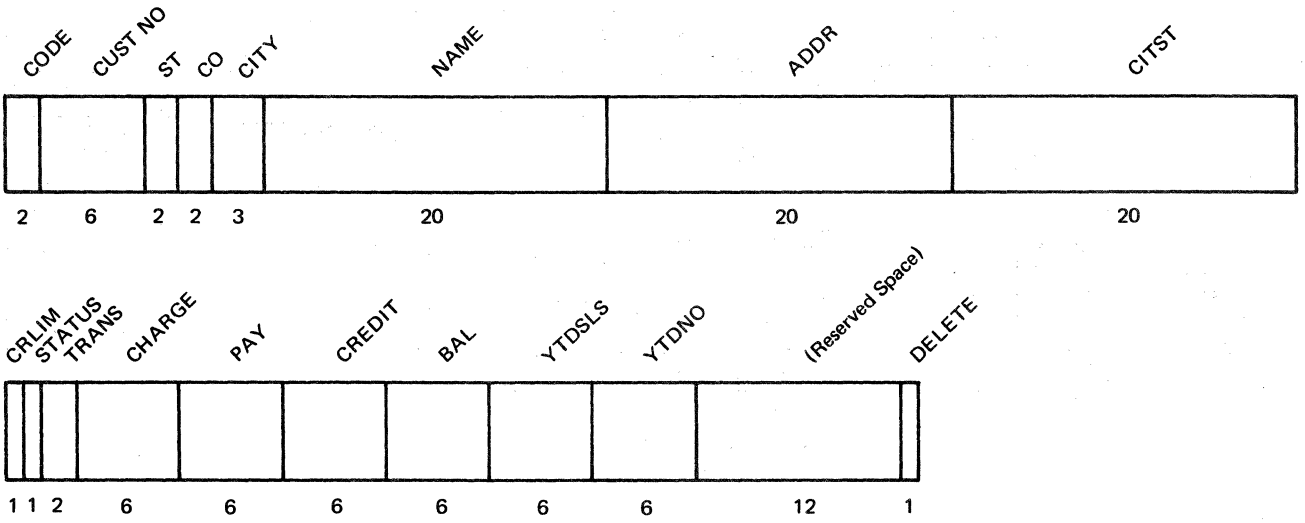
Program Identification

Edit Codes: Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

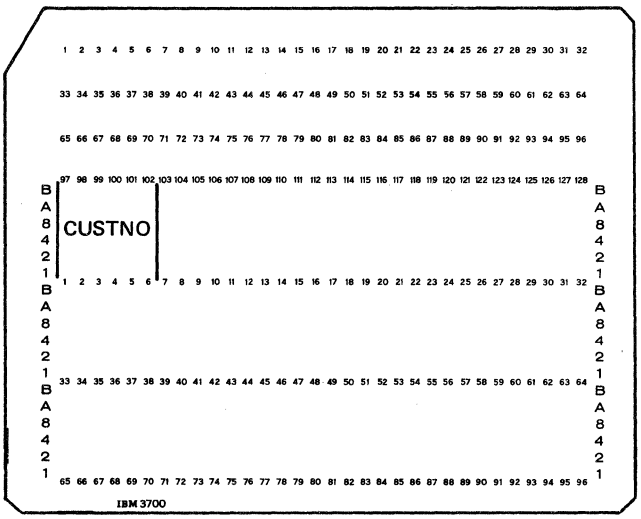
Line	Form Type	Filename	Type (H/D/T/E)	Stacker Select/Fetch Overflow (F)	Space: Before	Space: After	Skip: Before	Skip: After	Output Indicators: Not	And:	Not	And:	Not	Field Name	Edit Codes	Blank After (B)	End Position in Output Record	P = Packed/B = Binary	Constant or Edit Word	Sterling Sign Position
01	O	LISTING	D						01											
02	O													NAME			20			
03	O		D	2					01											
04	O													ADDR			20			
05	O		D	22					01											
06	O													CITST			20			
07	O																			
08	O																			
09	O																			
10	O																			
11	O																			
12	O																			
13	O																			
14	O																			
15	O																			
	O																			
	O																			
	O																			
	O																			
	O																			

Figure 14. Job 9: Detail Printing from a Sequential File (Part 2 of 2)



CODE = Record code (CM)
CUSTNO = Customer number
ST = State code
CO = County code
CITY = City code
NAME = Customer name
ADDR = Customer address (street)
CITST = Customer address (city and state)
CRLIM = Credit limit

STATUS = Status code
TRANS = Number of transactions this month
CHARGE = Current month charges
PAY = Current month payments
CREDIT = Current month credits
BAL = Balance
YTDSLS = Year-to-date sales amount
YTDNO = Year-to-date number of sales
DELETE = Delete code



A previous program checked the master customer file for customers whose credit was below the credit limit. The output from this file was a deck of cards with the customer numbers punched in columns 1-6. This program will print a list of those customers with poor credit.

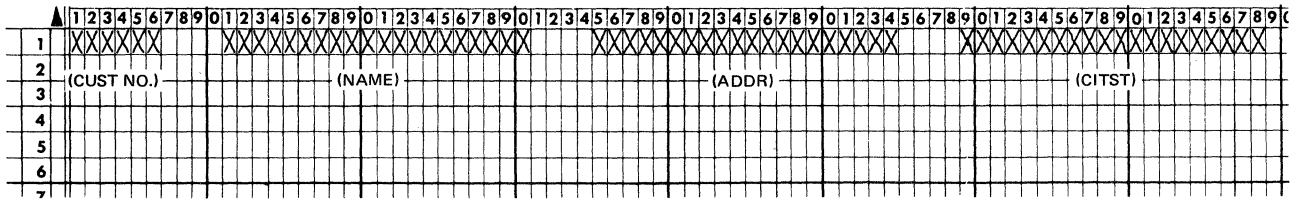


Figure 15. Job 10: Using Matching Records with a Sequential File (Part 1 of 2)

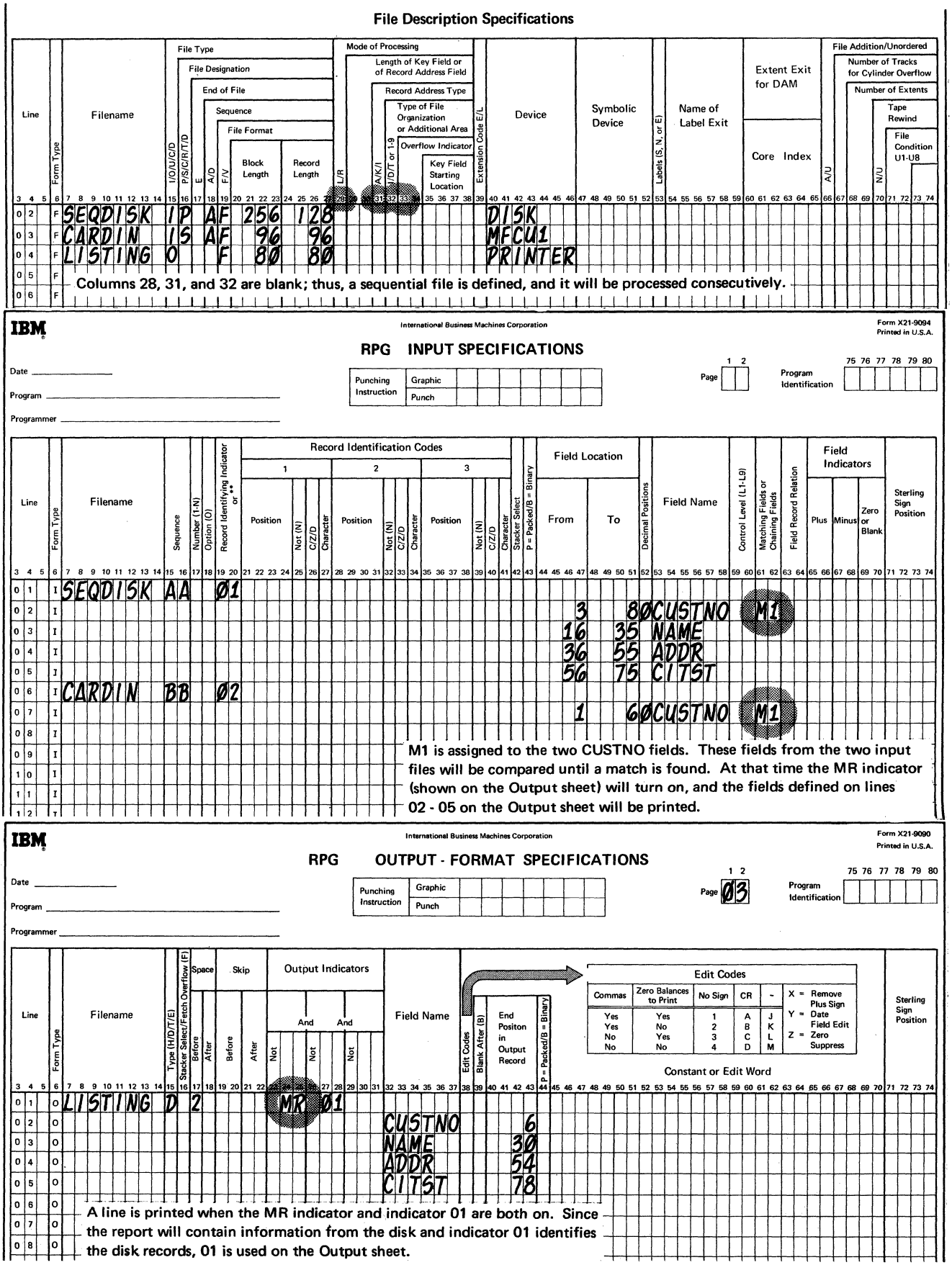
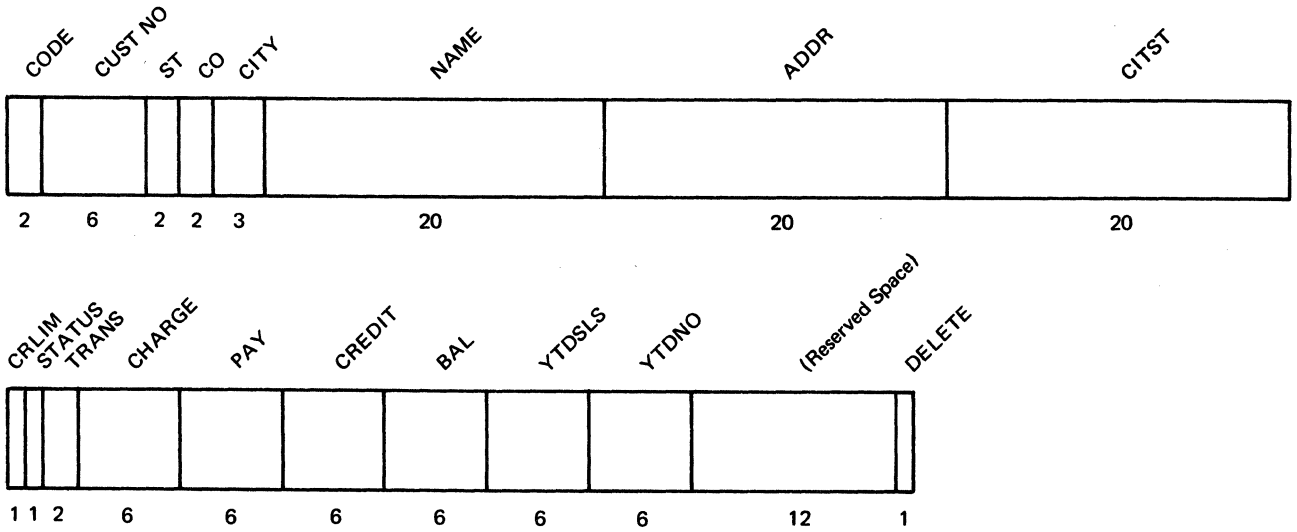
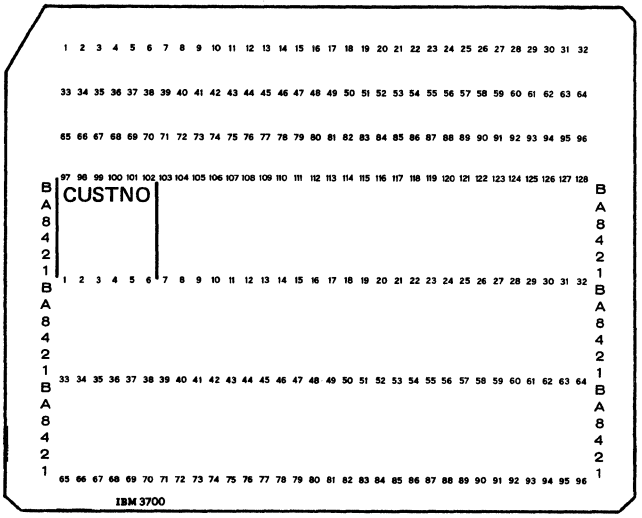


Figure 15. Job 10: Using Matching Records with a Sequential File (Part 2 of 2)



CODE = Record code (CM)
CUSTNO = Customer number
ST = State code
CO = County code
CITY = City code
NAME = Customer name
ADDR = Customer address (street)
CITST = Customer address (city and state)
CRLIM = Credit limit
STATUS = Status code
TRANS = Number of transactions this month
CHARGE = Current month charges
PAY = Current month payments
CREDIT = Current month credits
BAL = Balance
YTDSLS = Year-to-date sales amount
YTDNO = Year-to-date number of sales
DELETE = Delete code



This job does the same function as Job 10; however, the file is indexed.

```
288142    INTERNATIONAL PARTS      16 EASTERN PARKWAY    N.Y., N.Y.
298763    CONTINENTAL WHOLESALE    118 14TH STREET       WESTFIELD, IOWA
310041    AUTO SERVICE INC.        123 N. BROADWAY       N.Y., N.Y.
410787    FURNISHINGS INC.         206 LYRIC AVE.        LANSING, MICH.
```

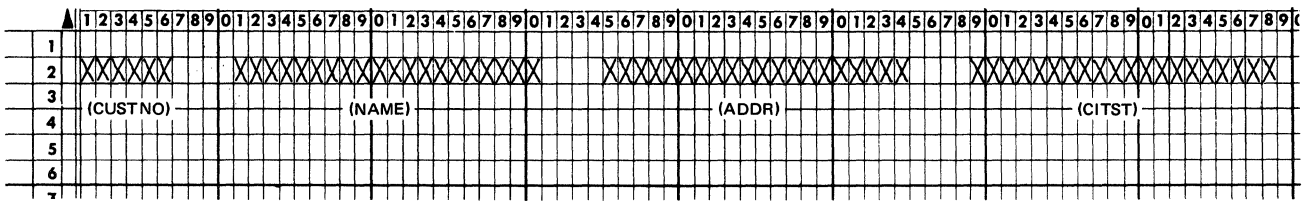


Figure 16. Job 11: Using Matching Records with an Indexed File (Part 1 of 2)

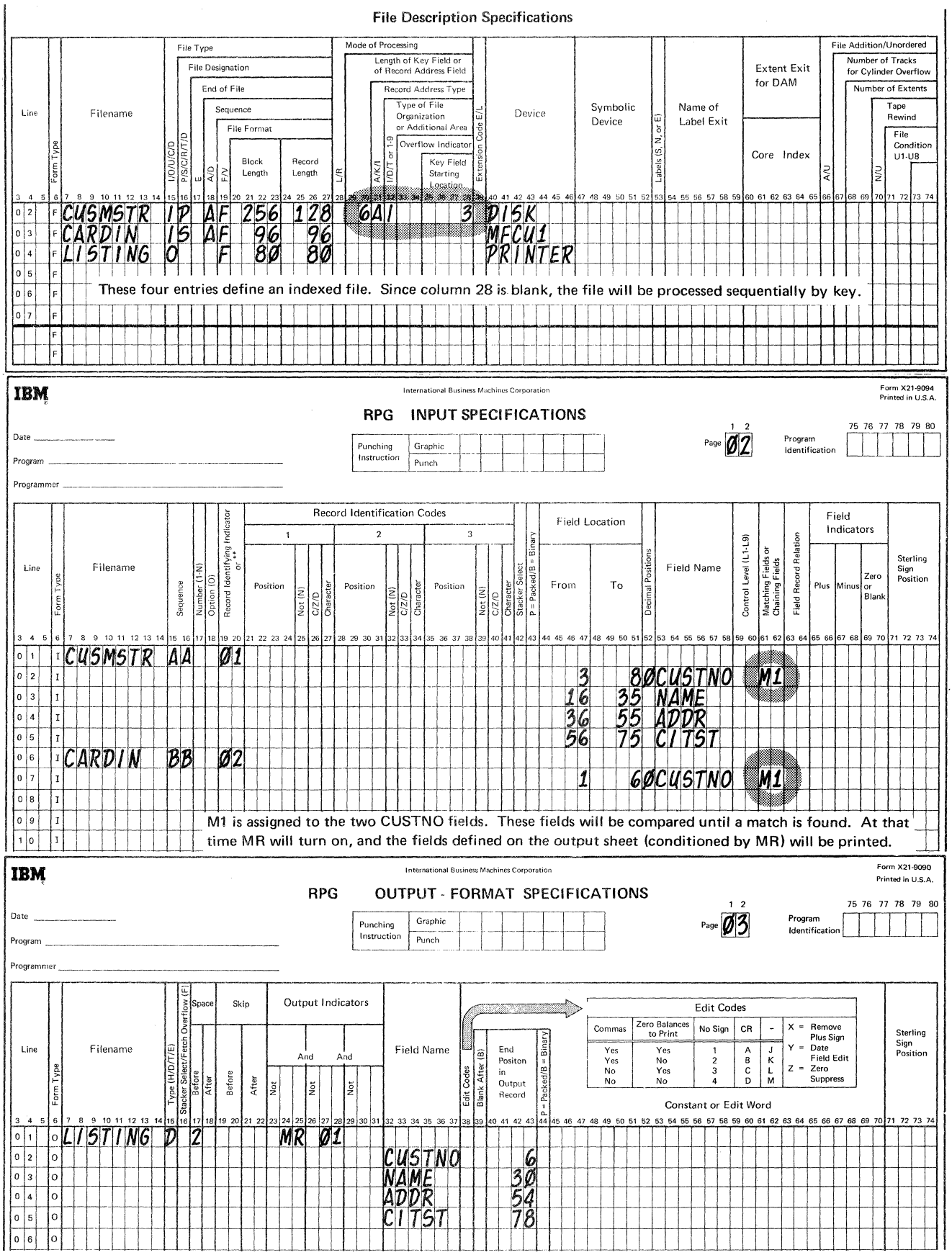


Figure 16. Job 11: Using Matching Records with an Indexed File (Part 2 of 2)

For every program you run Operation Control Language (OCL) statements are needed. You must supply OCL statements to the system through the MFCU (cards) or the Printer-Keyboard (typed).

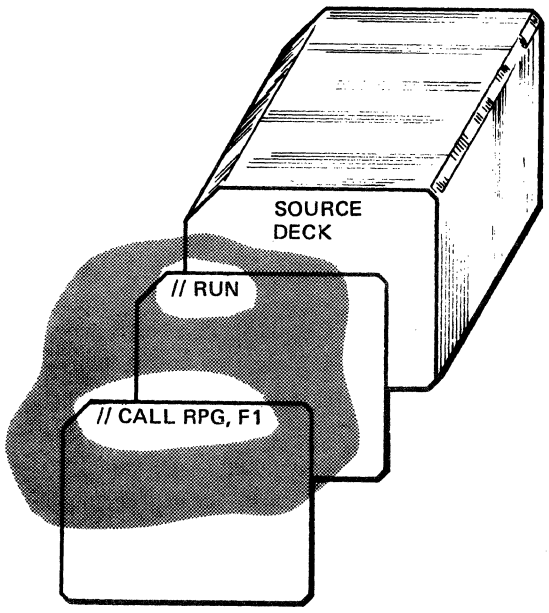
This chapter describes the statements needed to:

- Compile an RPG II program.
- Process a card file.
- Process a disk file.
- Process two disk files.
- Create a disk file.
- Process a disk file that uses external indicators.
- Perform several jobs, one after the other.

The function and the format of each statement are explained. Coding rules are also discussed.

OCL CARDS REQUIRED TO COMPILE AN RPG II PROGRAM

After your RPG II program is written and recorded in cards, it must be compiled. To compile an RPG II program two OCL cards are required.

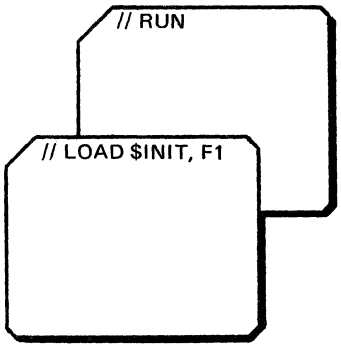


The first card, // CALL RPG, F1, tells the system to get the RPG II Compiler program from the fixed disk. The second card, // RUN, tells the system to run the compiler program. Your source deck must follow the // RUN card.

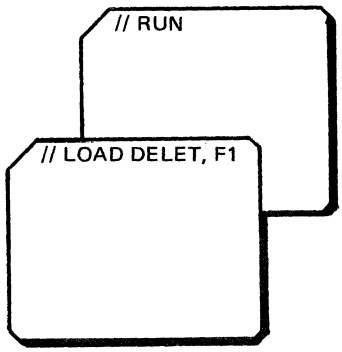
All OCL cards must contain // in columns 1 and 2, followed by at least one blank column and then a word like CALL or RUN.

OCL CARDS REQUIRED TO LOAD AND RUN SOME IBM PROGRAMS

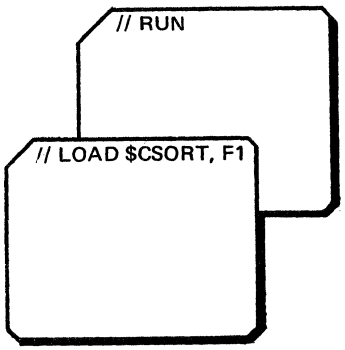
Several IBM programs require only two OCL cards, // LOAD and // RUN. The following three examples show the OCL cards needed to load and run three IBM programs.



The Disk Initialization program is loaded and run.



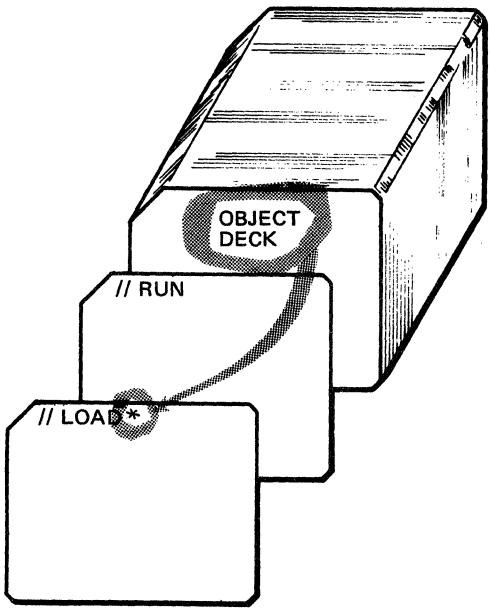
The File Delete program is loaded and run.



The MFCU Card Sort/Collate program is loaded and run.

OCL CARDS REQUIRED TO LOAD AND RUN AN OBJECT PROGRAM THAT USES CARD FILES

To run a certain job, the object program must be loaded into storage. The result of compiling a program is an object program. To load an object program that is on cards (object deck), an * must follow the word LOAD. The * tells the system that an object deck follows the RUN card. // LOAD * and // RUN are the only two OCL cards needed to load and run RPG II programs that use no disk files. For example, only these two cards are required for a program that prints data from a transaction card file.



OCL CARDS REQUIRED TO LOAD AND RUN A PROGRAM THAT USES ONE DISK FILE

To load and run an object program that uses a disk file, another OCL card is required: a FILE card. Three items of information must follow the word FILE: the name of the file, the name of the disk pack the file is on, and the location of the disk pack on the disk drive.

Before discussing the specific entries some terms need defining. OCL statements can contain two types of information: *statement identifiers* and *parameters.* A statement identifier tells one statement from another. CALL, LOAD, RUN, and FILE are statement identifiers. A parameter is additional information required with statement identifiers. Some OCL statements do not require parameters; RUN is such a statement.

A parameter is information defined by you or by IBM. For example, in the LOAD * statement, the * is defined by IBM. (The * means an object deck is being loaded from cards.) The information you define is variable information, such as the name of a file which you determine or the disk on which an IBM program is stored.

OCL has *positional parameters* and *keyword parameters.* A positional parameter contains information that must be supplied in a specific order; that is, information that occupies a specific position in an OCL statement. Such a statement is:

```
// CALL RPG, F1
```

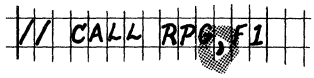
CALL is the statement identifier. RPG must be specified first, followed by F1; thus RPG and F1 are positional parameters.

Certain coding rules must be followed for positional parameters.

1. Leave at least one blank between // and the statement identifier.

2. Leave at least one blank between the statement identifier and the first parameter.

3. Use a comma to separate parameters.



Note: When several parameters are needed, they must appear in a specific order.

Keyword parameters contain an IBM defined word *(keyword)* followed by information that you supply. Keywords are required to tell one parameter from another. The FILE statement, introduced earlier, requires three keywords: NAME, PACK, and UNIT. The order in which they appear is not important.

The other information is either defined by you or IBM may offer several options. For example, to process a job with one file, such as Job 9, the keywords and the information you supply are:

NAME–(file name)

PACK–(name of disk pack)

UNIT–(location of disk pack)

In Job 9 SEQDISK, the name of the disk file, resides on a removable disk pack that is named VOL1. The keyword parameters for Job 9 are:

NAME–SEQDISK

PACK–VOL1

UNIT–R1

Both SEQDISK and VOL1 are names determined by you. Since the file resides on a removable disk, R1 is used with UNIT. R1 is one of four options defined by IBM. These options and their meanings are:

R1 = Removable disk on drive one

F1 = Fixed disk on drive one

R2 = Removable disk on drive two

F2 = Fixed disk on drive two

You must supply the code that tells on which unit your file resides.

When several keyword parameters are needed in an OCL statement, certain coding rules must be followed.

1. Leave at least one blank between // and the statement identifier.

2. Leave at least one blank between the statement identifier and the first keyword.

3. Separate the keyword and the information you supply with a hyphen (–).

4. Use a comma to separate keyword parameters.

Note: When several keyword parameters are needed, they may appear in any order.

To follow these rules and ensure your OCL statements are correct, you can use a 96-column coding sheet. The OCL statements for Job 9 (Figure 14) are coded as:

IBM

International Business Machines Corporation

96 COLUMN GENERAL PURPOSE DATA RECORDING DOCUMENT

System | Punching Instructions

Program | Graphic | Card

Programmer | Date | Punch

1 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72

```
// LOAD *
// FILE NAME=SEQDISK,PACK=VOL1,UNIT=R1
// RUN
```

OCL CARDS REQUIRED TO CREATE A DISK FILE

To create a disk file, sequential or indexed, you must tell the system the size of the file and the use of the file. To state the file size, two keywords are available: TRACKS and RECORDS. You may use one or the other, but not both.

If you use RECORDS, the system calculates the disk space required and converts it to tracks for you. The system performs the same calculations as described in Chapter 2.

If you use the TRACKS parameter, there is no need for the system to perform these calculations.

The RETAIN parameter tells the system how to classify the use of a file. A file can be classified as *scratch, temporary,* or *permanent.* A scratch file is normally used only once. When the program using it ends, all or any part of the area containing the file can be used by the next program. Data in the scratch file can be accessed until the area containing the file is used for another file.

A temporary file is usually used more than once and possibly by more than one program. The area containing a temporary file can be used by some other file under any one of the following conditions:

1. A FILE statement defining a scratch file is supplied for the temporary file. This converts the temporary file to a scratch file.

2. Another file is loaded into the *exact* area occupied by the temporary file.

3. The File Delete program is used to delete the file.

A permanent file is like a temporary file. However, the area containing a permanent file cannot be used for any other file until the File Delete program deletes the permanent file.

A file is classified as scratch, temporary or permanent when it is created. If at that time the RETAIN parameter is omitted, the file is assumed to be a temporary file.

RETAIN must be followed by a code that indicates the classification of the file. The possible codes are:

S – scratch

T – temporary

P – permanent

The OCL statements required to create the name and address file used throughout Chapter 3 are as follows.

IBM

International Business Machines Corporation

96 COLUMN GENERAL PURPOSE DATA RECORDING DOCUMENT

System | Punching Instructions

Program | Graphic | Card Fo

Programmer | Date | Punch

1 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72

```
// LOAD *
// FILE NAME-SEQDISK,PACK-VOL1,UNIT-R1,RECORDS-6720 RETAIN-P
// RUN
```

Since this file is a master file, it is classified as *permanent.* (RETAIN–P).

The TRACKS or RECORDS and RETAIN parameters tell the system how much disk space to use and how long to keep this space.

OCL CARDS REQUIRED TO LOAD AND RUN A PROGRAM THAT USES MORE THAN ONE DISK FILE

One FILE card is required for each disk file used in a job. To load and run an object program that uses two disk files, two FILE cards are required. In Chapter 3, Job 4 explained how to re-create a sequential file. Two disk files were used, one as input (the file to be re-created) and one as output (the newly created file); thus, two FILE cards are required.

IBM

International Business Machines Corporation

96 COLUMN GENERAL PURPOSE DATA RECORDING DOCUMENT

System | Punching Instructions

Program | Graphic | Card Fo

Programmer | Date | Punch

1 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72

```
// LOAD *
// FILE NAME-SEQDISK,PACK-VOL1,UNIT-R1
// FILE NAME-NEWSEQ,PACK-VOL1,UNIT-R1,RECORDS-6720,RETAIN-P
// RUN
```

OCL CARDS REQUIRED TO RUN A JOB THAT USES A DISK FILE AND EXTERNAL INDICATORS

In Chapter 3, Job 8 (Figure 12) used external indicators. You were told that these indicators are set by an OCL statement. This OCL card is // SWITCH. For Job 8, (Figure 12) the OCL cards required are:

IBM

International Business Machines Corporation

96 COLUMN GENERAL PURPOSE DATA RECORDING DOCUMENT

System		Punching Instructions
Program		Graphic
Programmer	Date	Punch

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
// LOAD *
// FILE NAME-INVMSTR,PACK-VOL1,UNIT-R1
// SWITCH 10000000
// RUN
```

In the SWITCH statement, a 1 is followed by seven zeros. Since RPG II allows up to eight external indicators to be used, the SWITCH statement must tell which of these indicators to set and how to set them.

In the SWITCH statement the eight characters correspond to the eight external indicators.

U1	U2	U3	U4	U5	U6	U7	U8
1	0	0	0	0	0	0	0
ON	OFF	OFF	. . . . .				

A 1 means set the indicator on; a 0 means set the indicator off. Job 8 uses only one external indicator (U1); thus a 1 appears in the U1 position.

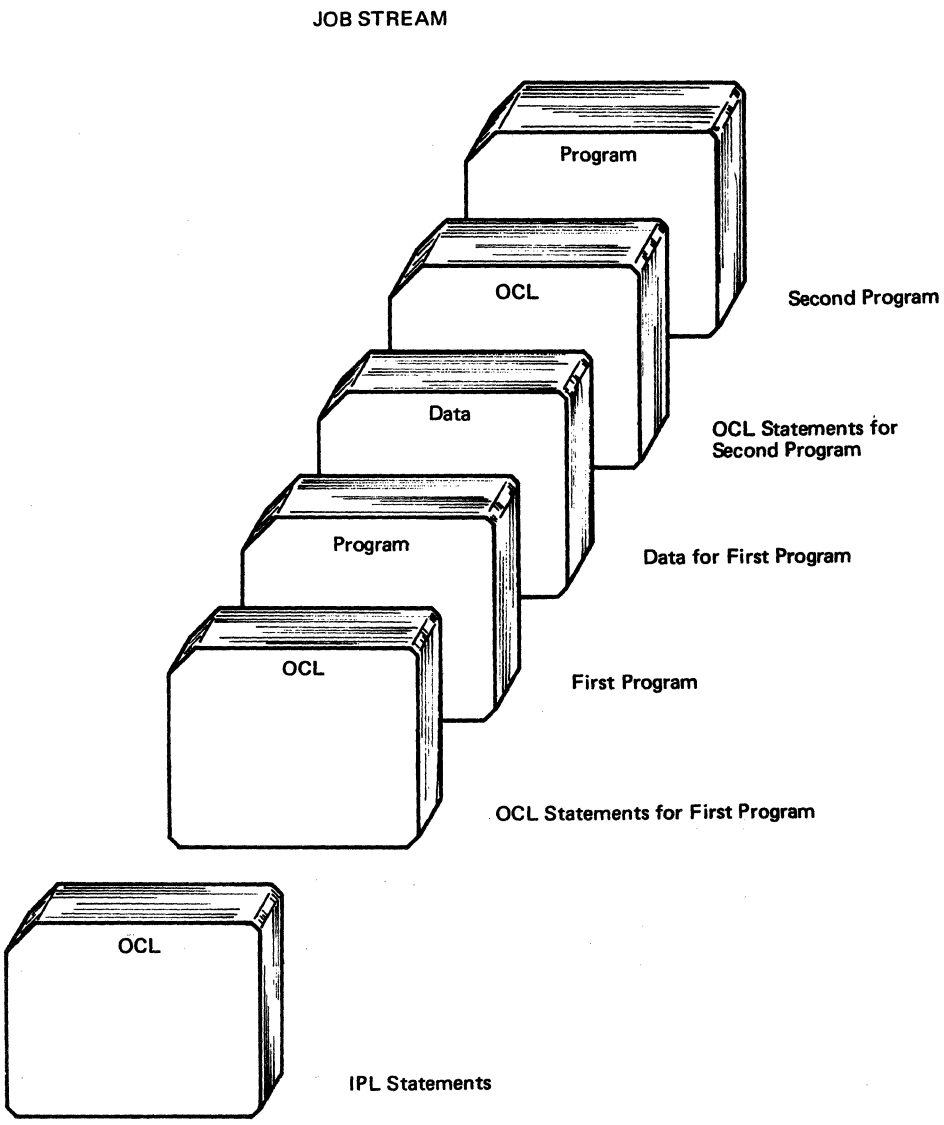
Note: The SWITCH statement can also contain an X. X means do not change the indicator status.

OCL CARDS REQUIRED TO RUN SEVERAL JOBS, ONE AFTER THE OTHER

After the system reads a set of OCL statements for a job, it loads the program for that job and runs the job. When that job is completed, another job can be run.

The OCL statements you supply for several jobs form a *job stream.*

Note that the first group of OCL statements are called *IPL statements.* IPL statements are those read by the system during Initial Program Load (IPL). IPL is an operating procedure the operator uses to initially start the system.



// DATE STATEMENT

Before the system will run any of your jobs, it must read a special OCL card, // DATE, at IPL time. The date supplied at IPL time is called the *system date.* The system date is saved by the system and is available to your RPG II programs. For example, to print a date in the heading of a report, the RPG II program can reference the system date by using the RPG II reserved words: UDATE, UDAY, UMONTH, and UYEAR.

// DATE must be followed by a date. The date can be entered in several ways:

mmddyy

mm/dd/yy

mm-dd-yy

ddmmyy

dd-mm-yy

However, you must specify an edit word to format the system date at output.

The DATE card can also be used to specify a date different than the system date. For example, at the beginning of each IPL you supply a system date reflecting that day's date. However, during that day you may want some reports, bills for example, with a different date. The OCL cards in Figure 17 illustrate how to specify a date different than the system date.

SUMMARY – OCL STATEMENTS

For every job you run, a set of OCL statements is required. Some jobs, such as compiling an RPG II program, require only two statements: CALL and RUN.

Other jobs require more statements, depending upon the requirements of the job. All jobs involving disk files must have one FILE statement for every disk file in the job.

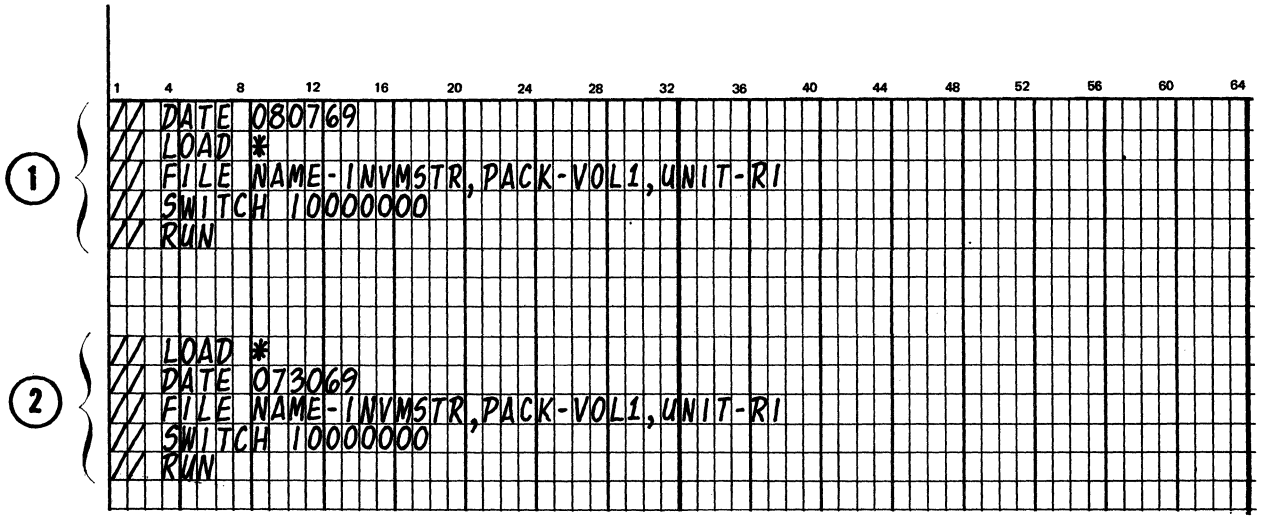
Various keywords are required in the FILE statement depending upon what the job does. To process a disk file, NAME, PACK, and UNIT are required. To create a disk file, NAME, PACK, UNIT, TRACKS or RECORDS, and RETAIN are required.

If a job uses external indicators the SWITCH statement is required. Other jobs may require DATE.

There are several important points to remember about OCL statements:

1. OCL statements contain identifiers. LOAD, CALL, RUN, and FILE are identifiers.

2. Some OCL statements contain only identifiers. RUN is such a statement.

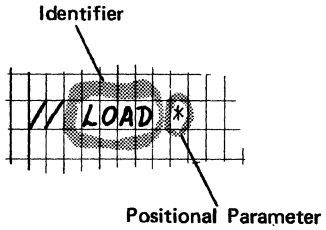


(1) 080769 is the date entered at IPL time. This date represents the current date (system date).

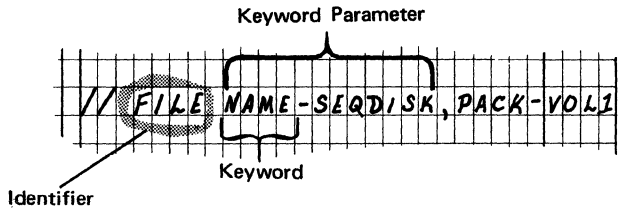
(2) This DATE card changes the date from 080769 to 073069. These OCL cards are for a program that prints a monthly profit summary, and we want a month-end date on the report. After this program is run, the system date (080769) will be available to the next program to use, unless that program's OCL cards also change the date.

Figure 17. Using the DATE Card

3. Some OCL statements contain postional parameters.

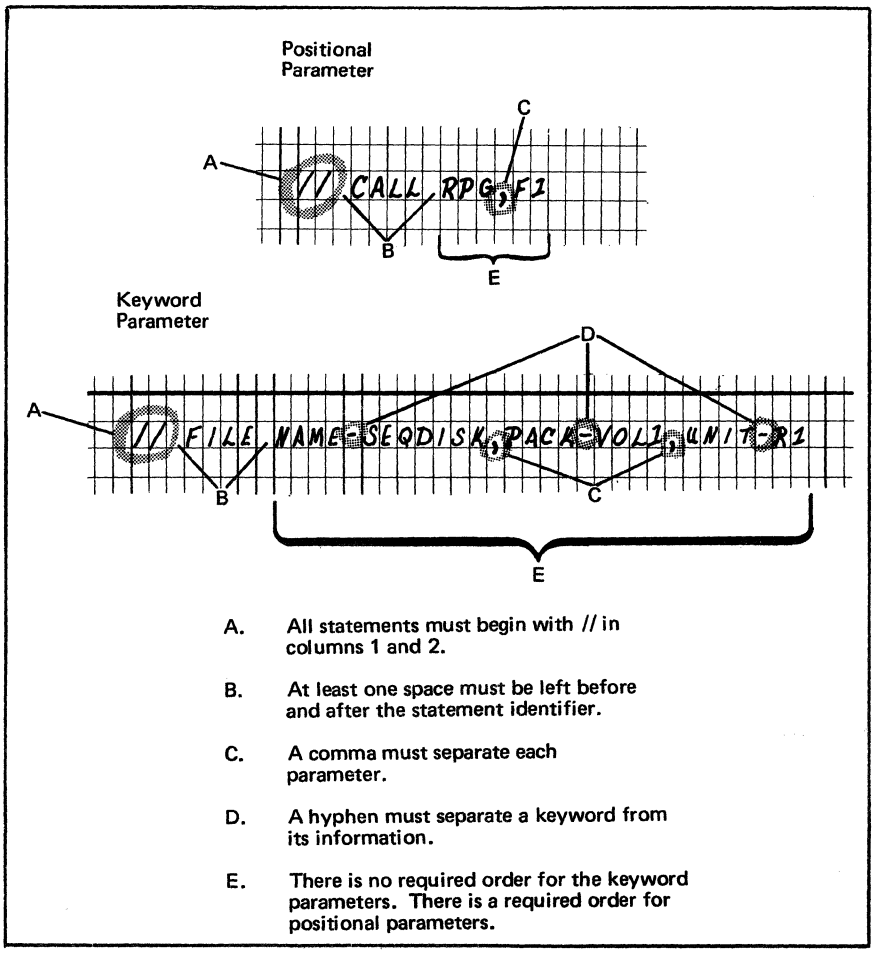


4. Some OCL statements contain keyword parameters.



5. The OCL statements, the programs, and the data for several jobs form a job stream.

6. OCL statements must be coded according to certain rules.



The *IBM System/3 Disk System Reference Manual,* Form C21-7512, contains a thorough explanation of all statements described in this chapter.

In Chapter 1, the need for sorting a disk file was discussed as a factor to consider when choosing a file organization method. Often a master file is used in various applications to produce a variety of reports. Some of these reports may require data printed in an order different than the order of the disk file. The IBM Disk Sort program is available to sort disk files. It can produce two types of output disk files that can be used as input to your RPG II programs. This chapter explains these two types of output files.

For additional information on the Disk Sort program, see the *IBM System/3 Card Sort/Collate Programmer's Guide,* Form C21-7539, and the *IBM System/3 Disk System Utility and Sort Programs Reference Manual,* Form C21-7522.

This chapter assumes you know how to:

1. Write Header lines that:
   a. Specify sum of lengths of control fields
   b. Specify sequence (ascending or descending)
2. Write Record Type lines that:
   a. Include all records
   b. Omit certain records
   c. Use the AND relationship
   d. Use the OR relationship
   e. Use the EQ, LE, NE, GT, GE, and LE relationships
   f. Specify constants in Factor 2
   g. Specify fields in Factor 2
3. Write Field lines that:
   a. Indicate the order of importance of the fields
   b. Specify type of field (normal, opposite, data, or forced)

TAG–ALONG SORT

With both the Card Sort/Collate program and the Disk Sort program, you can sort all records or only certain records. You specify which records you want sorted by describing them on Record Type lines. However, with the disk sort an additional function is provided.

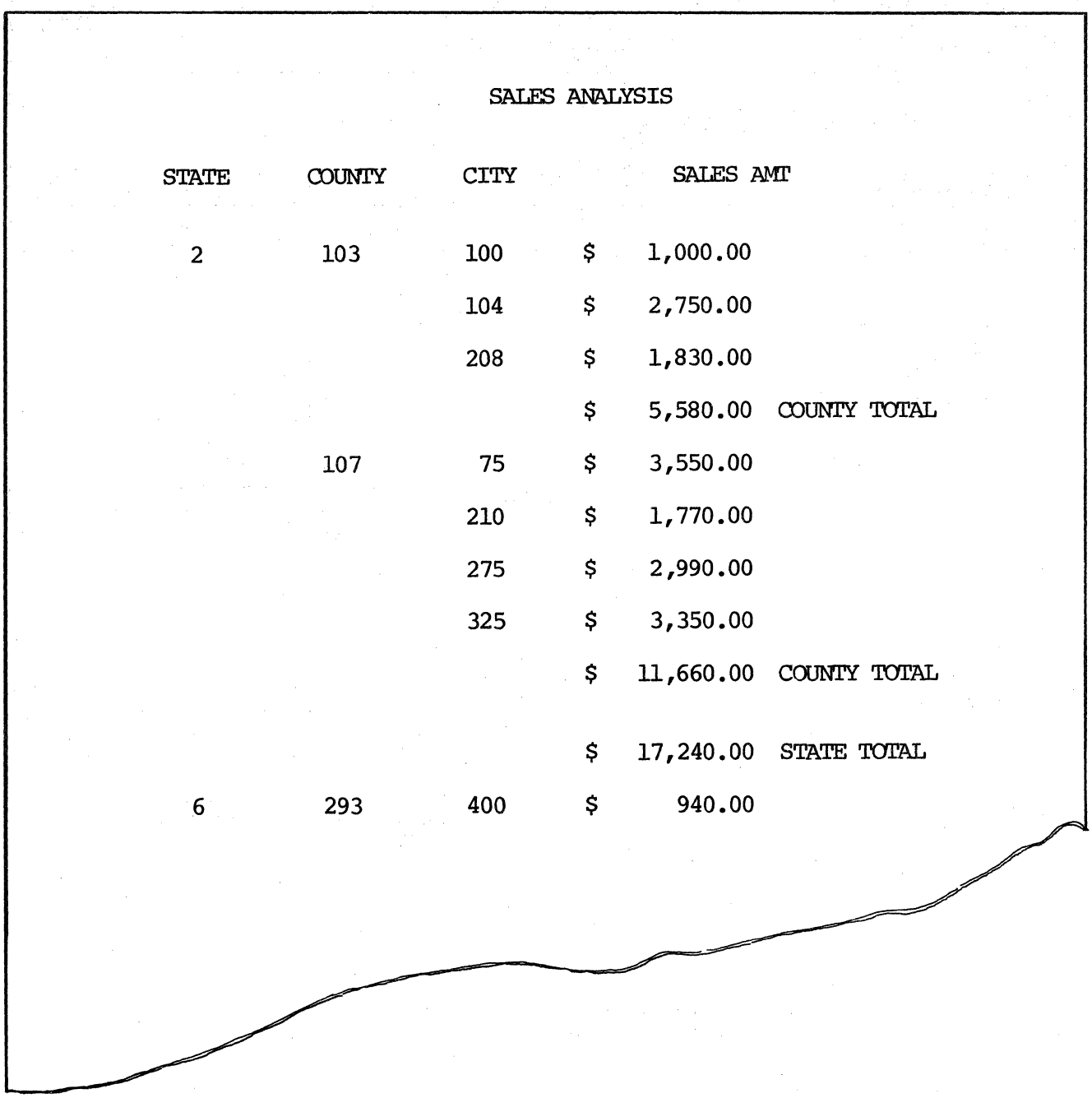
Your output records can contain *all* the data from the input records, or your output records can contain only *certain* fields. For example, the customer master file described in Chapter 2 and created in Chapter 3 will be used to produce a report called *Sales Analysis* (Figure 18).

Note that this report contains four fields: three codes (state, county, and city) and a sales amount. Since the master file is in customer number order, it must be sorted in order by state, within county, within city. So these three codes plus sales amount are the only fields needed to produce the report. The three codes are control fields. The other field, sales amount, is a *data field.* Data fields are those fields (in the output record) that *tag-along* with control fields. A tag-along sort is a job where data fields are included in the output record.

A tag-along sort can produce two types of output records. The output records can either retain or drop the control fields. The Sales Analysis report is an example of when control fields are retained. Since the control fields are printed, they must be retained in the output record. However, if you have some reports that must be in a specific order, but you do not want to print the control fields, then these fields can be dropped from the output record.

Describing Output Records that Retain Control Fields

To produce an output record that retains control fields, three new entries are required on the Sequence Specification sheet. Two entries are required on the Header line, and one entry is required on the Field lines. On the Header line, SORTR (columns 7-11) identifies the job as a tag-along sort. You must also specify the length of the output record in columns 29-32.

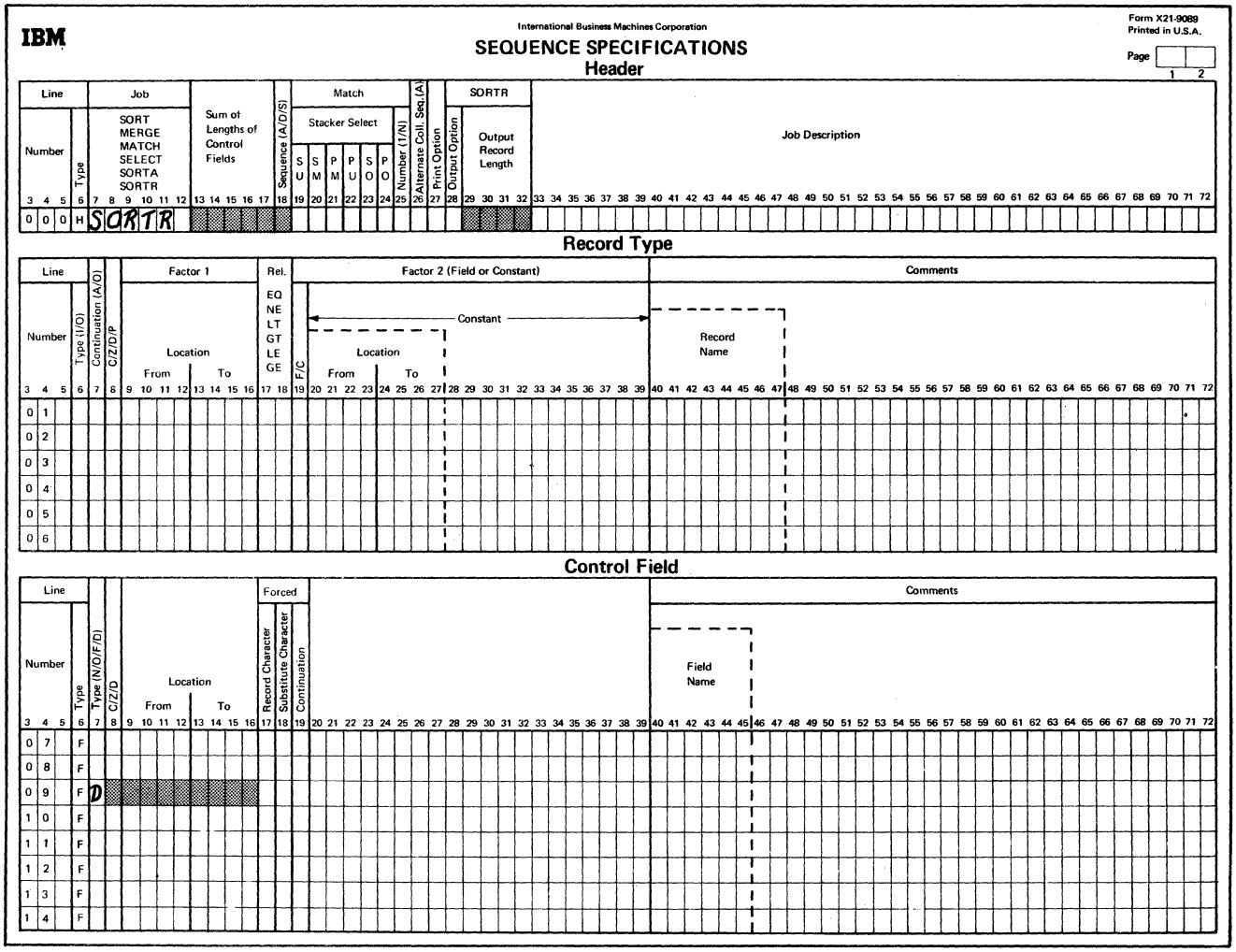


SALES ANALYSIS

STATE	COUNTY	CITY	SALES AMT	
2	103	100	$ 1,000.00	
		104	$ 2,750.00	
		208	$ 1,830.00	
			$ 5,580.00	COUNTY TOTAL
	107	75	$ 3,550.00	
		210	$ 1,770.00	
		275	$ 2,990.00	
		325	$ 3,350.00	
			$ 11,660.00	COUNTY TOTAL
			$ 17,240.00	STATE TOTAL
6	293	400	$ 940.00	

Figure 18. Sales Analysis Report

On the Field lines, you must describe your data fields. Data fields are described in the same way as control fields, with one exception. You must enter a D in column 7; the D specifies that the field described on this line is a data field. A D card must not follow N, O, or F cards for a given record type.

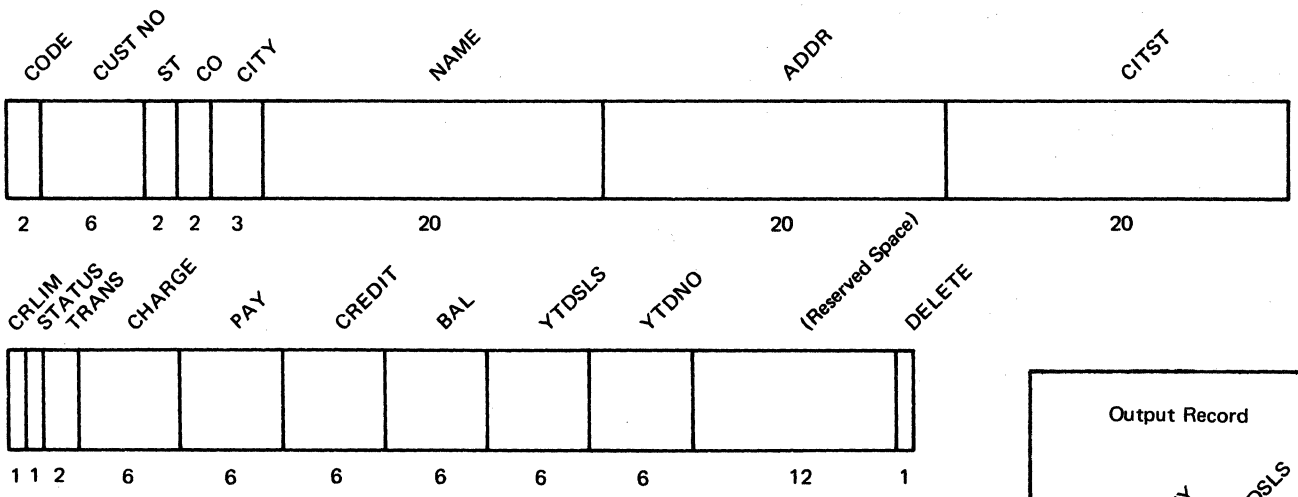


In a tag-along sort, output records are always written in the same way.

1. Control fields are placed first in the new record. On the Sequence Specifications sheet, the control field described first (most important) is placed first in the output record; the control field described second (second most important field) is placed second, and so on.

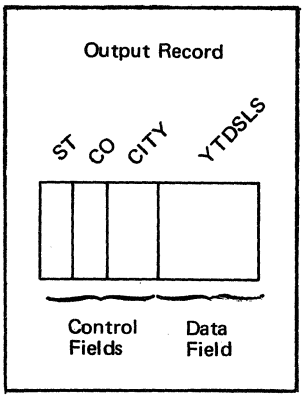
2. Data fields are placed immediately following the control fields.

Figure 19 shows the specifications needed to produce the file that will be used to print the Sales Analysis report.



CODE = Record code (CM)
CUSTNO = Customer number
ST = State code
CO = County code
CITY = City code
NAME = Customer name
ADDR = Customer address (street)
CITST = Customer address (city and state)
CRLIM = Credit limit

STATUS = Status code
TRANS = Number of transactions this month
CHARGE = Current month charges
PAY = Current month payments
CREDIT = Current month credits
BAL = Balance
YTDSLS = Year-to-date sales amount
YTDNO = Year-to-date number of sales
DELETE = Delete code



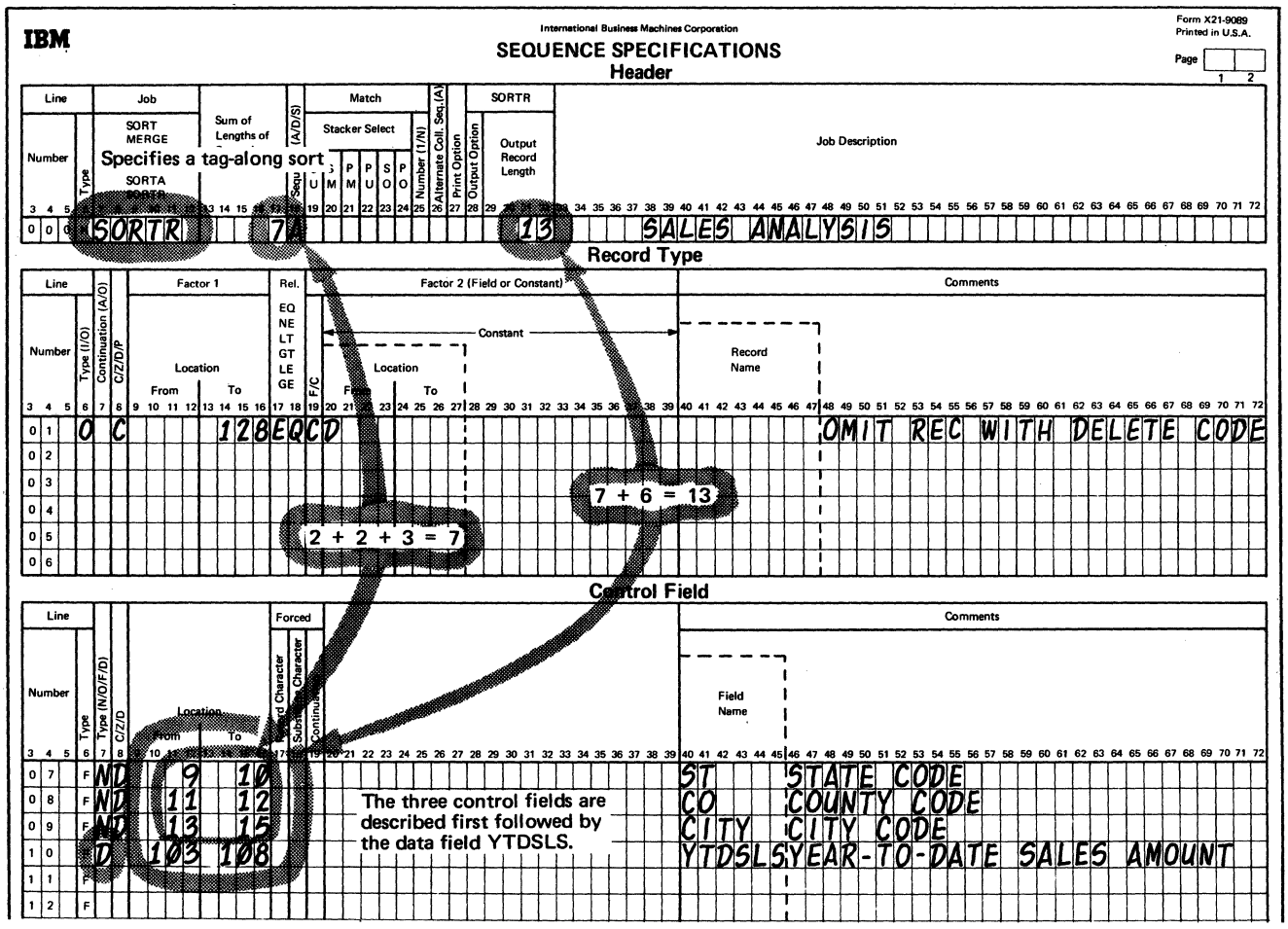
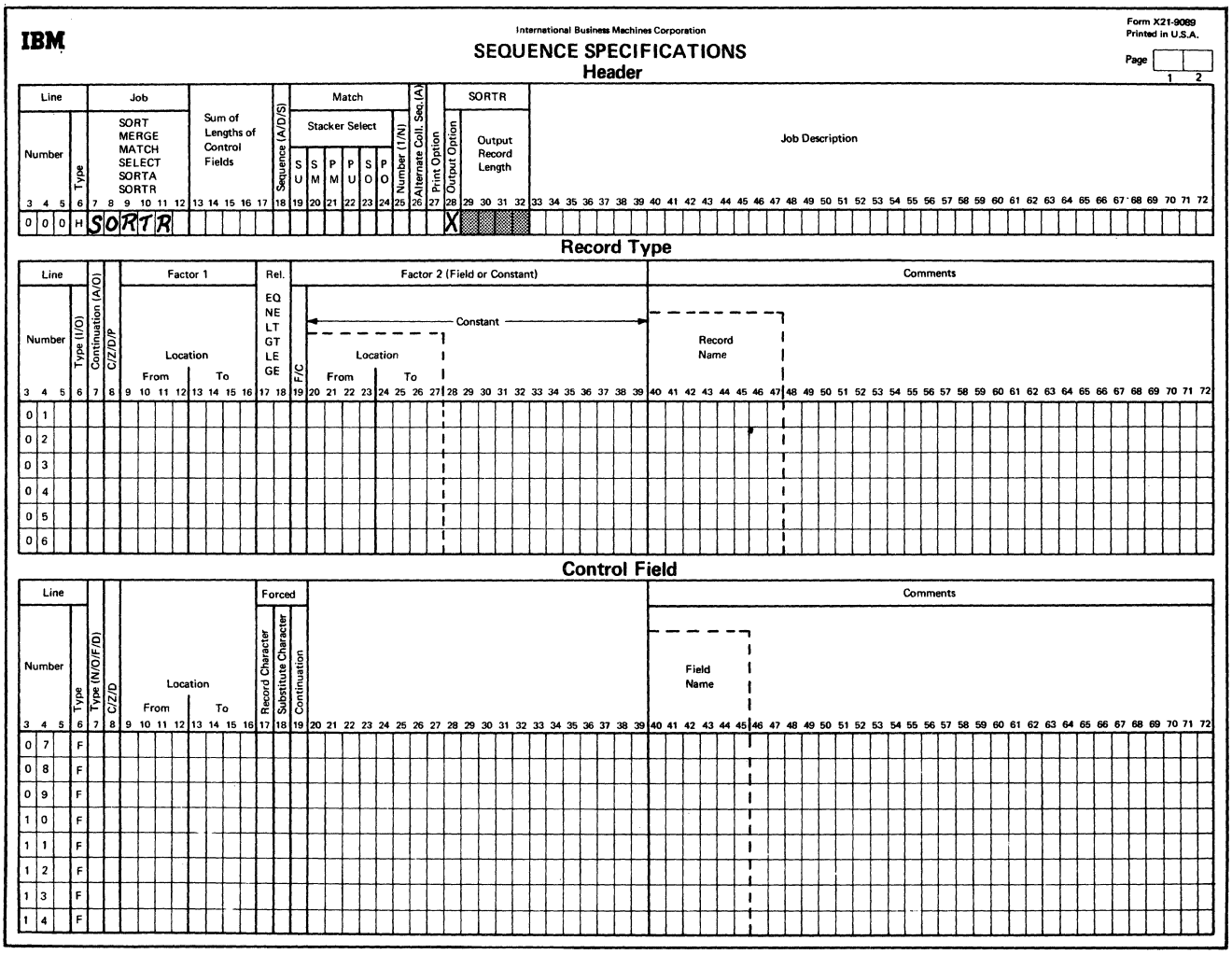


Figure 19. Tag-Along Sort Job

Describing Output Records that Drop Control Fields

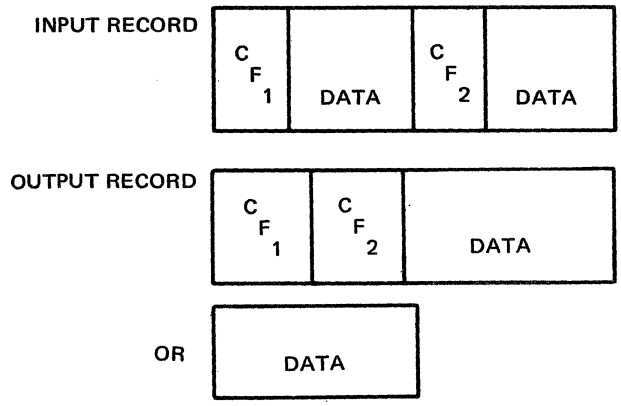
To drop control fields from the output record, an additional entry is required and one other entry must be modified on the Header line.



The X in column 28 tells the Disk Sort program to drop the control fields from the output record. Columns 29-32 are still used to indicate the length of the output record. However, you must *not* include the sum of the lengths of the control fields, since control fields will not appear in the output record.

REVIEW—TAG-ALONG SORT

A tag-along sort produces output records that retain or drop control fields (CF).



The Header line and Field lines contain entries that identify the type of output record desired.

Header Line

1. SORTR (columns 7-11) identifies a tag-along sort.
2. An X in column 28 means drop control fields; a blank means retain control fields.
3. The number in columns 29-32 specifies the length of the output record. (Do not include control field length if X is entered in column 28).

Field Lines

1. Control fields are described in order of importance.
2. Data fields (D in column 7) are described following the control fields.

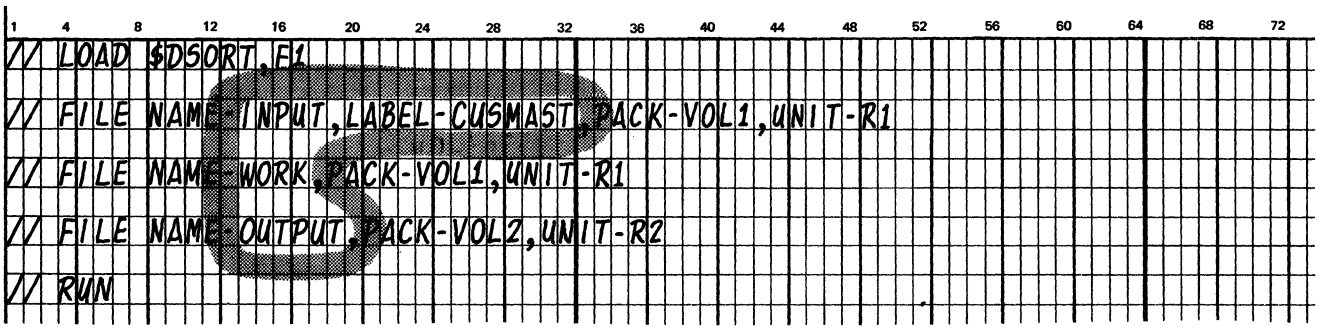
ADDROUT SORT

The other type of disk sort is ADDROUT. An ADDROUT sort produces a disk file that contains only the relative record numbers of the desired records.

To produce an ADDROUT file, SORTA is placed in columns 7-11 on the Header line. No other disk sort entries are required on the Header line. However, you must describe your control fields on Field lines. Even though an ADDROUT file contains only relative record numbers these numbers are sorted in the sequence you specify in column 18 on the Header line. The control fields are needed to produce this sequence.

OCL REQUIRED WITH DISK SORT PROGRAM

You use the Disk Sort program when you want to perform an ADDROUT or a tag-along sort job. For the Disk Sort program to do the job you specify, you must use certain OCL statements. Figure 20 illustrates and explains these statements.



// LOAD $DSORT -- This statement tells disk system management to load the disk sort program from the fixed disk. $DSORT is the IBM name for the Disk Sort program. F1 is a fixed disk.

// FILE -- As you know, every file used in a program must be defined by a // FILE statement. For a disk sort job, you must always define three files: an input file, a work file, and an output file. The input file is the file you want sorted. The work file is space on the disk that the program uses to do the sort job. The output file is the new file created from the input file as a result of the sort job.

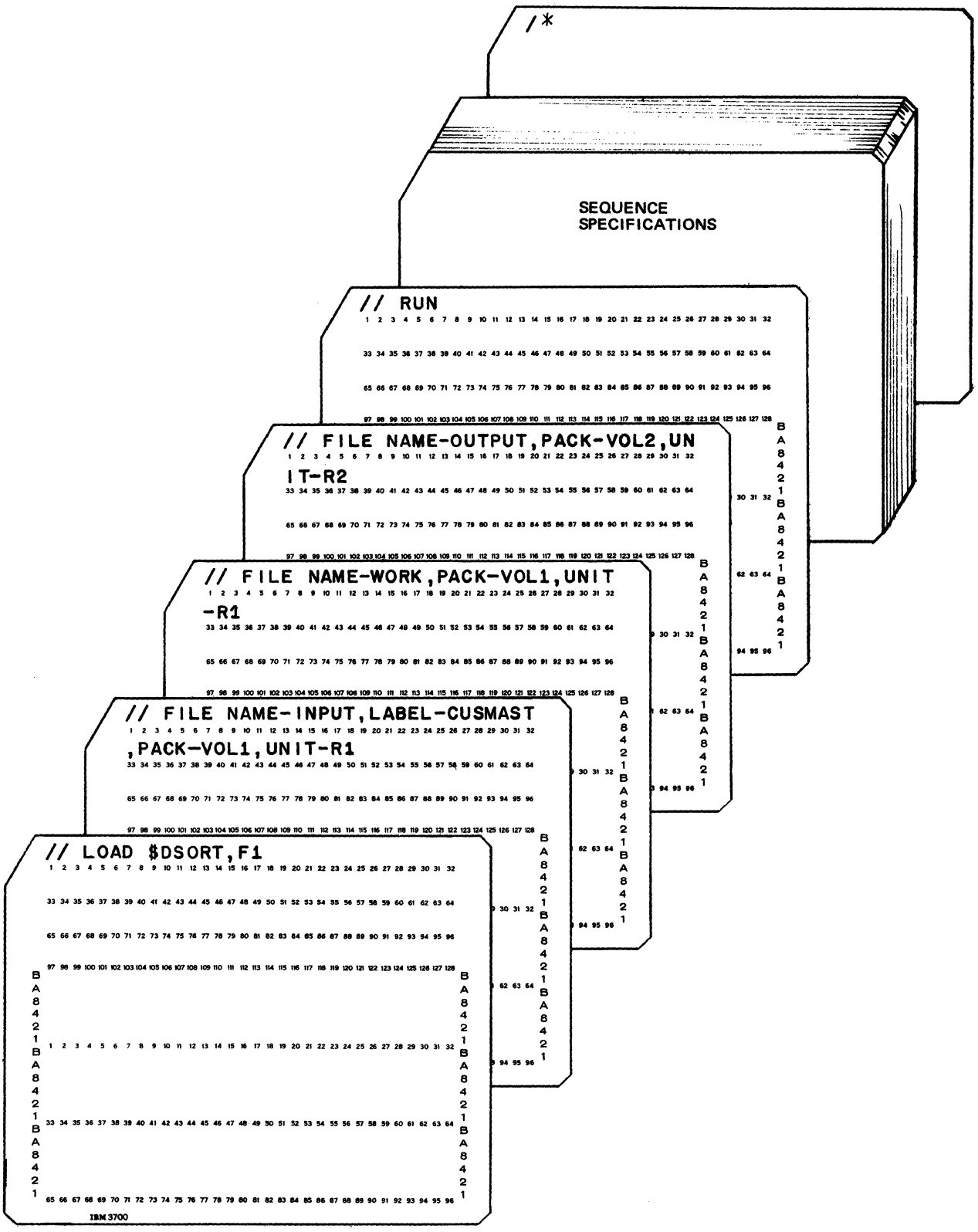
The keyword parameters for NAME are INPUT, WORK, and OUTPUT. These three words are predetermined by IBM. For the program to use a file in one of these three ways, you must correctly use these three words.

// RUN -- A RUN statement is always the last OCL statement for a job.

Figure 20. OCL Statements for a Disk Sort Job

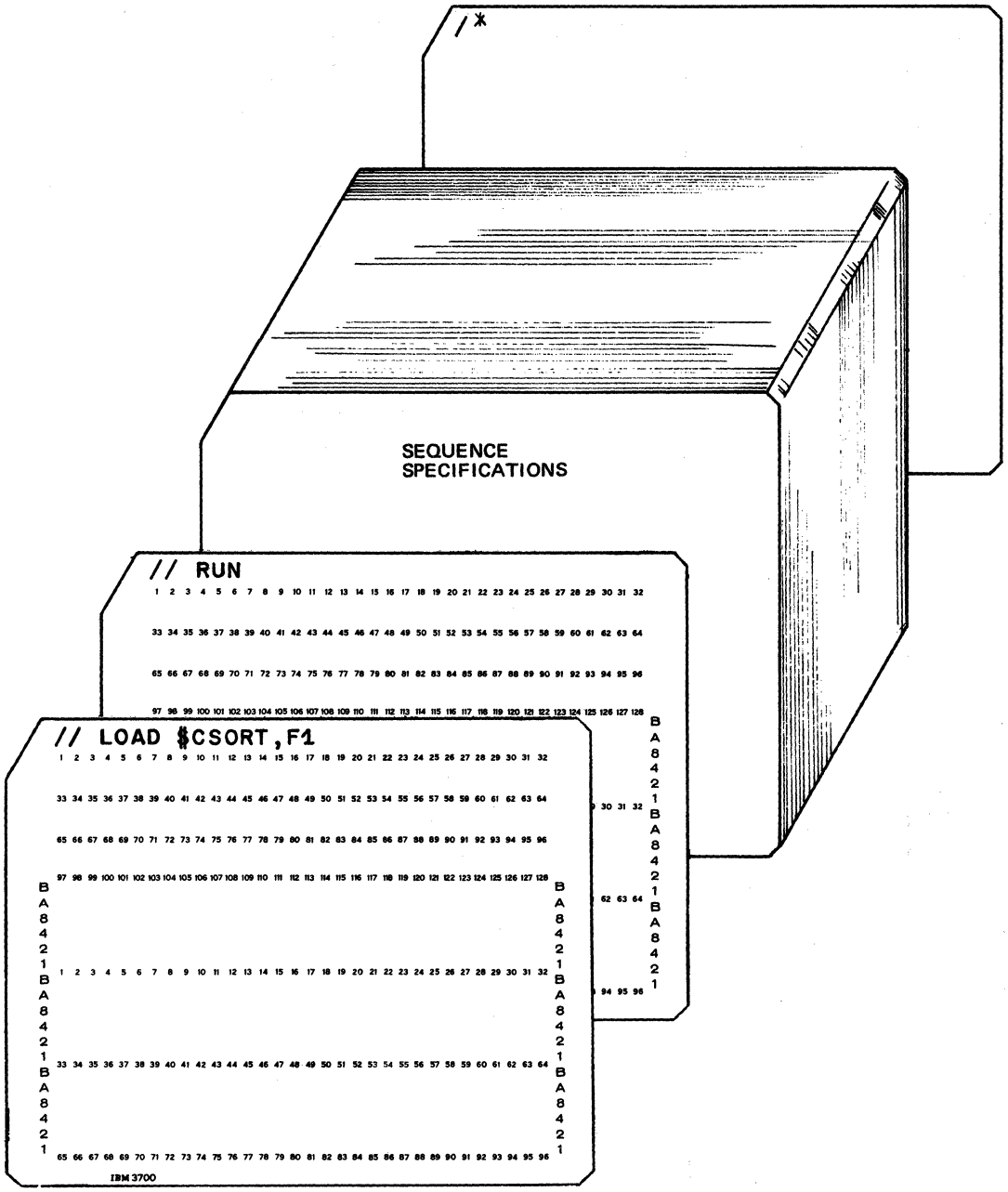
The OCL cards precede the Sequence Specification cards in the MFCU.

/* is needed to tell disk system management that the last specification card has been read. At that time, the Disk Sort program will begin the specified sort job.



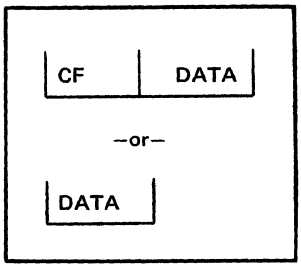
OCL REQUIRED WITH CARD SORT/COLLATE PROGRAM

You use the Card Sort/Collate program when you want to sort, merge, match or select cards. The Card Sort/Collate program resides on disk. For it to do the job you specify, you must use certain OCL cards.



The relationship of these cards to the Sequence Specifications is the same as for a disk sort.

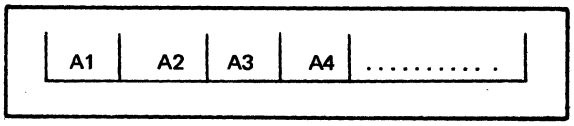
The name of the Card Sort/Collate program is $CSORT. The LOAD card must contain this name. LOAD and RUN are the only OCL cards required for a card sort. Again /* must be the last card after the Sequence Specifications. As for a disk sort, /* tells the system that all specifications have been read.



REVIEW–SORT

The Disk Sort program can produce two types of output files, tag-along and ADDROUT. The tag-along sort can produce output records with or without control fields.

The ADDROUT file contains the relative record numbers of the sorted records.



Coding considerations for tag-along and ADDROUT sorts are shown in Figures 21 and 22.

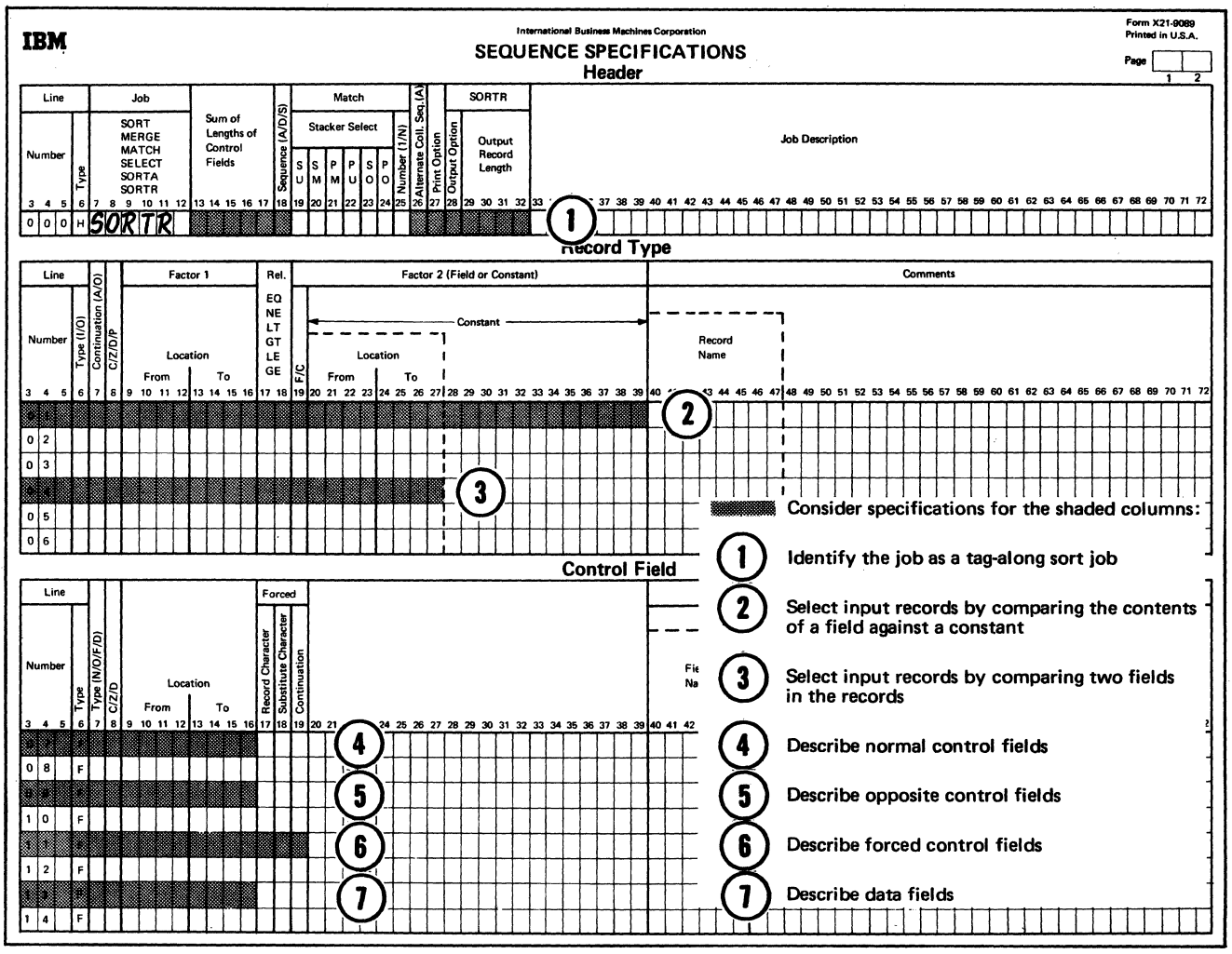


Figure 21. Column Summary (Tag-Along Sort)

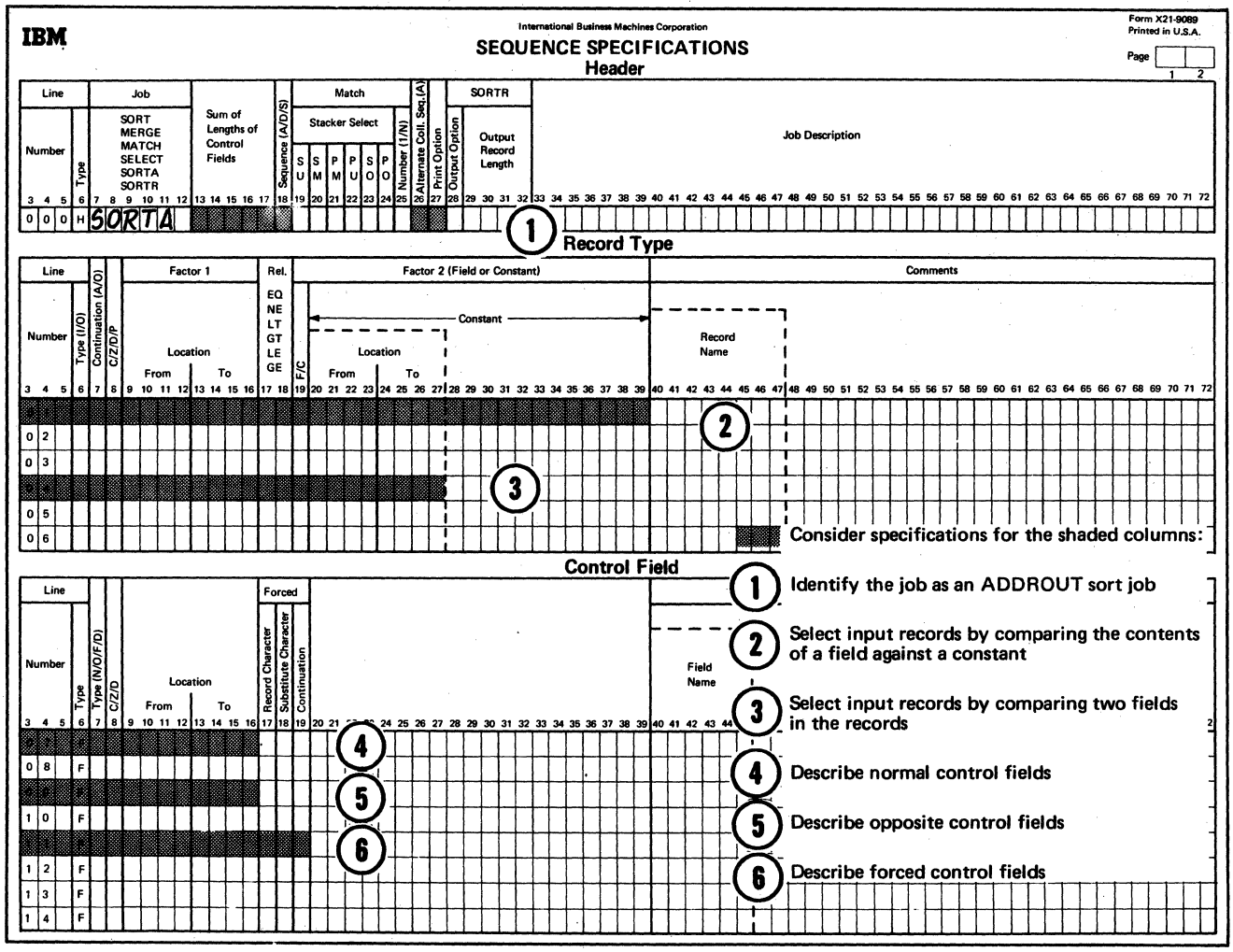


Figure 22. Column Summary (ADDROUT Sort)

READER'S COMMENT FORM

IBM System/3
Disk System Concepts
and Programming,
Programmer's Guide

Form GC21-7503-0

Your answers to the questions on this sheet will help us produce better manuals for your use. If any of your answers require comments, or if you have additional information you think would be helpful, please use the space provided. All comments and suggestions become the property of IBM.

	Yes	No
1. Is the manual easy to read?		
2. Is any of the information unclear?		
3. Is additional information needed?		
4. Is any of the information unnecessary?		
5. Did you read the Preface?		
6. Did you use the Table of Contents?		
7. Did you use the Index? *		
8. Did you take the tests? *		

* Not included in all manuals

9. How did you use the manual:

Instructor in a class _____
Student in a class _____
Reference material _____
Self-Training _____
Other (Explain) _____

Have you had previous computer or programming training? _____

What is your present job? _____

What business is your company engaged in? _____

COMMENTS

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

IBM System/3 Printed in USA GC21–7503–0

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York 10601
(USA only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)