60482900

CONTROL DATA CORPORATION

SORT/MERGE VERSIONS 4 AND 1 USER'S GUIDE

CDC® OPERATING SYSTEMS
NOS 1
NOS/BE 1
SCOPE 2

REVISION RECORD

Revision	Description
A (05-15-79)	Original release.

REVISION LETTERS I, O, Q, AND X ARE NOT USED

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
P. O. BOX 3492
SUNNYVALE, CALIFORNIA 94088-3492

or use Comment Sheet in the back of this manual

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

Page	Revision
Cover	–
Inside Cover	–
Title Page	–
ii	A
iii/iv	A
v thru viii	A
1-1 thru 1-4	A
2-1 thru 2-4	A
3-1 thru 3-4	A
4-1 thru 4-10	A
5-1 thru 5-7	A
6-1 thru 6-3	A
A-1 thru A-4	A
B-1	A
B-2	A
C-1 thru C-3	A
Index-1	A
Comment Sheet	A
Mailer	A
Back Cover	

Page	Revision

Page	Revision

PREFACE

This user's guide provides an introduction to the high-speed record processing facilities of Sort/Merge. It is intended for students and others unfamiliar with Control Data's Sort/Merge.

Sort/Merge is available under the following operating systems:

Sort/Merge Version 4 operates under NOS 1 for the CONTROL DATA® CYBER 170 Series, CYBER 70 Models 71, 72, 73, 74, and 6000 Series Computer Systems

Sort/Merge Version 4 operates under NOS/BE 1 for the CDC® CYBER 170 Series, CYBER 70 Models 71, 72, 73, 74, and 6000 Series Computer Systems

Sort/Merge Version 1 operates under SCOPE 2.1 for the CONTROL DATA CYBER 170 Model 176, CYBER 70 Model 76 and 7600 Computer Systems

This user's guide describes both Sort/Merge version 4 and version 1 with primary emphasis on the description of Sort/Merge 4 and the NOS operating system. Where Sort/Merge 1 differs from Sort/Merge 4, a reference is made to the Sort/Merge reference manual. The differences in specification of NOS/BE control statements are covered in appendix C.

If you are not an experienced programmer, you need not read section 5. The ability to write owncode routines, whether in COMPASS or FORTRAN Extended, is not required in order to use Sort/Merge.

Those readers who wish to find precise definitions of the various facets of Sort/Merge should refer to the Sort/Merge reference manual. This user's guide is not intended to precisely define the specific attributes of Sort/Merge, but rather to provide an introduction to its use and its application to problem solution. There should be no conflicts between this user's guide and other CDC publications. However, you should note that this user's guide presents only part of the total overview presented in the reference manuals.

If you follow the examples in this publication, you should be able to create and run simple sort programs. You will also be better prepared to use the information supplied in the reference manual. If you are not familiar with your operating system, you should consider reading the applicable user's guides and reference manuals listed below.

Publication	Publication Number
Sort/Merge Versions 4 and 1 Reference Manual	60497500
NOS Version 1 Reference Manual Volume 1	60435400
NOS Version 1 Reference Manual Volume 2	60445300
NOS Version 1 Applications Programmer's Instant Manual	60436000
NOS Version 1 Batch User's Guide	60436300
NOS Version 1 Terminal User's Instant Manual	60435800
NOS Version 1 Time-Sharing User's Guide	60436400
NOS Version 1 Time-Sharing User's Reference Manual	60435500
NOS/BE Version 1 Reference Manual	60493800
NOS/BE Version 1 User's Guide	60494000
COBOL Version 5 Reference Manual	60497100
COBOL Version 5 User's Guide	60497200
COMPASS Version 3 Reference Manual	60492600
CDC CYBER Record Manager Advanced Access Methods Version 2 Reference Manual	60499300
CDC CYBER Record Manager Advanced Access Methods Version 2 User's Guide	60499400
CDC CYBER Record Manager Basic Access Methods Version 1.5 Reference Manual	60495700

CYBER Record Manager Basic Access Methods Version 1 User's Guide	60495800
FORTRAN Extended Version 4 Reference Manual	60497800
FORTRAN Extended Version 4 User's Guide	60499700
CYBER Common Utilities Reference Manual	60495600
FORM Version 1 Reference Manual	60496200
8-Bit Subroutines Reference Manual Version 1	60495500

CDC manuals can be ordered from Control Data Corporation Literature and Distribution Services, 308 North Dale Street, St. Paul, Minnesota 55103.

CONTENTS

APPENDIXES

INDEX

FIGURES

TABLES

INTRODUCTION 1

Sorting information is part of our everyday lives. Sorting is the process of arranging information into a predefined sequence so as to enhance its value. Sorted information is easier to search. Imagine how useful a telephone directory or a dictionary would be if the information were not sorted in alphabetical order.

Or imagine trying to make use of all of the raw data collected during a nationwide census. The tremendous volume of raw data collected represents little usable information before it is sorted, totaled, and compiled into meaningful statistics. Before the introduction of data processing equipment, the time required to complete the tabulation of these statistics was awesome.

One of the first card sorting machines was devised by Herman Hollerith to help solve this problem. At the time Mr. Hollerith worked for the Census Bureau, cards containing all statistics were handwritten, hand sorted into various categories and counted, resorted into other categories and counted again, until all categories were compiled. Hollerith's basic change was to use a hand punch to punch the information into 240 separate areas of a standardized card. Each area had a specific meaning, such as age group, sex, and so on. His card reading machine had forty dial counters. Whenever a hole was encountered in the card in a specific area, the dial wired to that hole would be incremented by 1. An entire card could be read in only six passes. A card box with 26 separate compartments was attached to the card reader. Depending on which connections were made, one of the lids would open automatically to allow the reader operator to drop the card in and then close the lid.

Approximately 100 of his machines were used to tabulate the 1890 U.S. census. These machines are considered the first of the data processing machines. Though slow and tedious, they reduced the 1890 census tabulation effort from an anticipated 7 years to less than 3 years. Because these machines were hand-fed, they reached average speeds of up to 20 cards per minute.

As the years passed, the card sorting machine was improved by adding an automatic card feed. Later improvements added chutes, gates, and multiple pockets which received the cards. The importance of the card and its position within the stack grew in relation to the importance of the data contained in the card.

The basic concept of sorting cards changed to ordering all of the cards based on the content of a single card column as opposed to the individual value of each punch as used previously. By defining a field, for example, as a numeric amount contained in card columns 65 through 70, if all cards were sorted on column 70 in the first pass, column 69 in the second pass, and so on, all of the cards would be in correct numerical sequence after column 65 had been sorted. Whether the amount field would be in ascending or descending numeric order depended on the order in which the card sorter operator had stacked them after each pass. Considering that card sorters processed about 300 cards per minute by 1930, and that a sort on a 6-column field required 6 passes, the time required to sort a box of 2000 cards was about one hour.

Later improvements to the electromechanical card sorting devices allowed them to reach speeds in excess of 1000 cards per minute, but still required that each column of a field be sorted. And only after the cards were sorted could the information they contained be totaled and printed. The amount of hand labor required to sort and print a few boxes of cards was staggering by today's standards, yet the card sorter was considered a labor-saving device at the time.

The computers of the early 1950s could store the information from cards on magnetic tape, sort this information into sequences, merge these sorted sequences, and write the completely sorted information to tape for subsequent tabulation and printing. Manual labor was no longer needed to handle card decks for more than the initial input pass.

Sorting information remains one of the major uses of computers in business applications, such as credit card processing. More importance is now attached to the information the card contains than the card itself, yet in numerous applications the cards themselves are still physically sorted and returned to the customer. The electromechanical card sorter still plays a role, though a diminishing one, in present day operations. It is now cheaper to read cards, sort the information, and punch new cards rather than physically sort the input card file.

COMPUTER SORTING

The use of computers for sorting information has changed the concepts originally applied to sorting. A record is no longer considered limited to the information that can be contained in a single card. A sort run is no longer measured in terms of how many boxes of cards can be sorted in one hour.

Much work has been done in the last 25 years to improve computer sorting techniques. Many books discuss the various techniques and their applications, and yet the use of computers has not altered sorting procedures nearly as much as it has emphasized the need for speed and the ability to handle a very large number of records in one sort.

Computer sorting can still be compared, in concept, to the sorting of playing cards. The following example illustrates these concepts.

If a person is given one complete deck of playing cards and asked to put them in order, the procedure is a simple one. Most people will make an initial distribution by suit, creating four files of equal size. After that, each file can be sorted by holding the 13 cards of a suit in one hand while the cards are shifted about and placed in order. As soon as each of the four suits has been ordered, the four are stacked together and the job is completed. In sort terminology, this was accomplished by the following basic sorting methods: a distribution, an internal sort, and then a final merge of the four sorted files.

A more realistic picture of most sorting problems is created if the above problem is complicated slightly. Assume that 52 cards are taken from a stack of cards which contains four decks. The procedure outlined above might work for this case as well, but there are good reasons to doubt that it will. It is extremely unlikely that the initial distribution by suit will produce four groups of equal size. In fact, there might not even be four groups or files, and the cards selected might contain two to four identical cards. If we also add the stipulation that 15 is the maximum number of cards that can be held at one time, the solution for the second case is considerably more difficult. It can be solved by a combination of methods, but either the sequence of operations will have to be altered or the initial distribution will have to be modified.

The foregoing example illustrates well one of the basic problems of all sorting routines; as the number of records to be sorted grows, the complexity of the sort grows even faster. Unlike other aspects of our modern age, larger is not easier, nor less costly per item, when sorting records.

PURPOSE OF SORT/MERGE

The basic reason for sorting is to arrange items in order. Ordered information makes reports more meaningful. Order suggests critical relationships. Searches for information, whether by humans or by machines, are faster through ordered lists.

Sorting information into alphabetic and/or numeric order is the simplest method of classifying items. For example, many libraries use card files to aid users in finding information. Most libraries maintain a card file by book title and by author, but refer to the books by an assigned number. An additional subject card file is often available as a cross-reference aid. The books themselves are usually classified according to other sorting systems, so the assigned number is the key item in finding the book, and the number is found from the card files. It is much easier to search through the cards to find the book you want than it would be to search through all of the books. This method of assigning a number to an entity is referred to as indexing, a concept often used in sorting very large records. A computer sort of an index is much easier and faster than a sort of the large records. The time saved in sorting usually offsets the time spent in creating the index.

Reports that contain a large amount of random information or raw data are not very useful. Critical relationships can be obscured by the sheer volume of data. Sorted information offers immediate comparison. For example, it might be of interest to a large corporation to determine how wide a salary range exists for a given pay grade in various geographical locations. A simple report of all persons' records in that pay grade could run on for many pages. Sorting the records by pay grade and by region would give immediate comparison. More complex sorts based on pay grade, region, age, sex, and so forth, could offer much more information on reasons for the disparities.

To find the record for one employee manually in a large file usually requires a great deal of time. If the file were in random order, the time required to perform such a search would average one-half of the time required to scan the entire file. If the file is ordered, search time can be reduced considerably. This is as true of a computer file as it is of a hand-sorted card file. For example, to find a record in a randomly organized file of 1000 records averages the time required to search 500 records. In a sorted file of the same size, the time required would average the time to search 10 records.

SORT/MERGE

Sort/Merge is a generalized sorting and merging program available on Control Data CYBER 170, CYBER 70, 6000 Series and 7000 Series computers.

The purpose of Sort/Merge is to rearrange records in the sequence you specify. You must supply the basic information about the records you wish to sort or merge and how you wish them sorted; the Sort/Merge program will then determine optimum internal settings to achieve efficiently what you have specified and carry out that function. Many types of data can be sorted, such as 60-bit integers, 60-bit floating point numbers, and even unsigned binary integers of any length.

The data can be alphabetic names, or codes consisting of alphabetic, numeric, or other special characters. If, for some reason, the standard order of the alphabet does not suit your needs, you can specify your own order of characters.

The order into which the characters are to be arranged is called a collating sequence. Four standard collating sequences are available, or you can create your own collating sequence if you wish.

Up to 100 files can be sorted and merged into one output file. You can supply your own procedures to be executed at certain points during the sort or merge processes.

Sort/Merge is invoked with a single control statement from a terminal or a batch job. Optional parameters allow you to specify more complex operations as the need arises.

Sort/Merge also provides a set of procedures that can be called from a user program written in FORTRAN or in COMPASS assembly language. The same Sort/Merge can be called by use of the COBOL SORT verb.

MERGING

The merge has been used as the basis for many sorting routines. The principles of sorting by merging are relatively simple. A two-way tape sort was once the popular method of sorting by merging and serves well to illustrate this concept.

If a given computer has the ability to read and write records from tape and select the smaller of the records brought in, a series of random numbers can be sorted as follows:

The first pass of the sort merges two single records to create a sequence of two records.

The second pass using the output of the first pass as input, merges a pair of these two-record sequences, one from each of the two input tapes, and writes four-record strings on the output tapes.

Repeating these merging passes will eventually place all records in the file into one sequence.

Figure 1-1 shows how a set of numbers is gradually put into sequence by use of a simple two-way merge. In the first pass, the records are written out on tapes 1 and 2, each of which will then contain two-record sequences at the end of the first pass. The tapes are then rewound and the output of the first pass becomes the input for the second merge pass. The first two-record strings from tapes 1 and 2 combine to make a four-record sequence that goes onto tape 3. Then the second pair of two-record sequences is merged and stored on tape 4. The merging process for the third pass is like that for the second pass, except that the sequences read in and written out are twice as long as they were in the preceding pass; input now comes from tapes 3 and 4 and output is written onto tapes 1 and 2. Each new pass doubles the length of the input sequences. The file is sorted when the number of sequences is reduced to one. The final output will all be on a single tape. In the example, the sorted records are stored on tape 1 after four passes. More complex examples are given in an appendix of the Sort/Merge reference manual.

In a simple merge such as the one shown in figure 1-1, the number of merging passes is determined by the number of records to be sorted and the order of the merge. A two-way merge without an internal sort will develop a sequence of 2^n records in n merging passes, or 8 items in three passes, 16 in four, 32 in five, and 1024 in ten passes. As the number of records to be sorted increases, such as might be expected in data processing applications, the number of passes becomes quite large.

SORT/MERGE AND THE OPERATING SYSTEM

Sort/Merge operates as a separate product, much like a compiler, under the NOS and NOS/BE operating systems. Sort and merge capabilities are available through control statements or procedure calls. A control statement sort or merge uses the SORTMRG control statement and its optional parameters and a directive file containing more parameters.

Characteristics of the input and output data files must be specified with the CYBER Record Manager FILE control statement. Certain default file names can be used, such as INPUT, OUTPUT, and PUNCH, without the need for a FILE statement, but it is better programming practice to always use the FILE statement.

Sort and merge require that a value be specified for the maximum record length, even for CYBER Record Manager record types that do not require this specification. This value can be specified either as the MRL or the FL parameter on the FILE control statement. MRL means maximum record length and FL means fixed length; both specify the record length as a decimal number of 6-bit characters.

CYBER RECORD MANAGER

Sort/Merge uses the file handling capabilities provided by CYBER Record Manager. These same file handling capabilities are also used by COBOL and FORTRAN, and are available to users of a number of other products such as COMPASS assembly language and the data management products.

CYBER Record Manager provides a variety of file structures. The standardized formats of these file structures enhance the file interchange capabilities of programs using CYBER Record Manager. For example, standard formats make it easier for the programmer to use files created by a FORTRAN program and sorted by Sort/Merge as input to a COBOL program. CYBER Record Manager requires that a FILE control statement be included in the input file of any program that uses CYBER Record Manager. The FILE control statement is used to describe the file to the operating system. When used in conjunction with any product that uses CYBER Record Manager, the FILE control statement can change the file processing that takes place during execution. As a result of its use, tape files from other computers or operating systems can be read, or it can be used to write tape files in formats not normally produced by the operating system.

Direct calls to CYBER Record Manager, the FILE control statement, file creation capabilities and formats, and other features are explained in the CYBER Record Manager publications listed in the preface.

The examples in this publication are not intended to teach the use of CYBER Record Manager. The examples include only those CYBER Record Manager statements required to obtain the desired results of the sort job to be run.

A full description of the interaction of Sort/Merge and CYBER Record Manager appears in an appendix of the Sort/Merge reference manual.

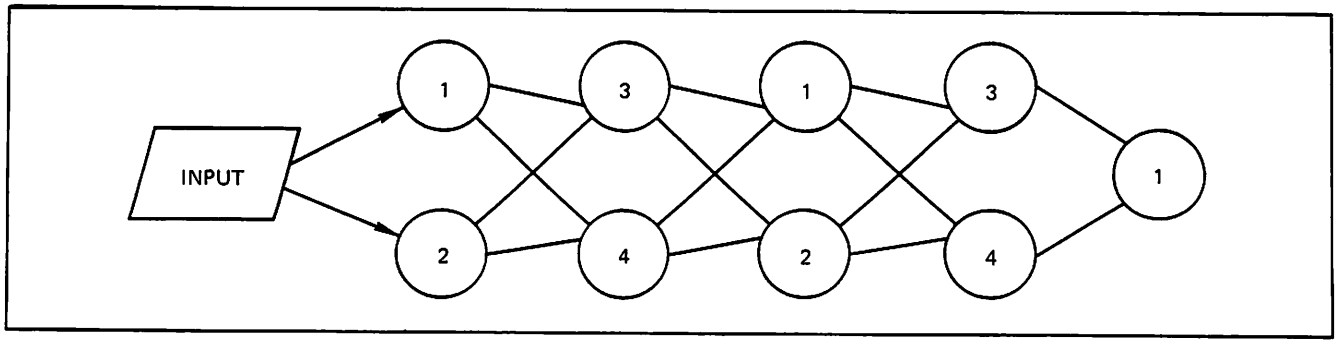


Figure 1-1. Simple Two-Way Tape Merge

EQUIPMENT USED FOR DATA ENTRY

Many storage mediums are available to Sort/Merge users. Before the data can be manipulated, however, it must first be entered into the computer. Punched cards are probably still the most common method of preparing input to the computer. Card punches are easily available, inexpensive, and relatively reliable. Off-line terminals such as key-to-tape and key-to-disk systems are rapidly gaining in popularity. On-line data entry systems are also growing in number. Whatever the entry medium, most files are stored on magnetic tape or disk after they have entered the computer.

STORAGE OF DATA

Magnetic tape offers low cost long-term storage of information. The standard 2400-foot reel can contain a large amount of information, is easily stored, and withstands most hazards of transportation. Magnetic tape is also a common method of transmitting information between computer sites, and even between computers using different character encoding.

MANIPULATION OF DATA

Records are usually sorted in memory, using as much central memory as is specified. Disk storage provides high-speed intermediate storage of records during the sort and merge processes. When processing is complete, the sorted file is usually output to disk or tape for subsequent use or for storage.

CYBER Record Manager has made the use of different files much easier for the computer user. It ensures that card, tape, and disk file formats are compatible with each other. Many magnetic tapes created on other computers, even computers from other manufacturers, can be read on CDC computers through the CRM interface. Those tapes that are not compatible with CDC computers through CRM can usually be reformatted through the use of FORM or the 8-Bit Subroutines. Moreover, with FORM, records using keys of the same length, but in different starting positions, can be reformatted to make a subsequent sort and merge possible. Additional information is available in the CYBER Record Manager AAM and BAM reference manuals and in the FORM and 8-Bit Subroutines reference manuals.

ACCURACY OF DATA

The accuracy of the data collected and prepared for input to the computer is the responsibility of the people involved in these tasks. Various methods can be employed during each phase of data handling to reduce the possibility of error.

The collection of information is very much subject to human error. When the collection mechanism is established, as many safeguards as possible should be established to reduce the opportunity for error. For example, if a record key field is established which can only be alphabetic or numeric, a simple checking routine can be written to ensure that these key fields contain only alphabetic or numeric characters. Such a routine can be used either as a separate process or incorporated into the sort process as an owncode routine.

Other safeguards can include routines to count the records both before and after processing to ensure that all records have been processed, and routines such as file label checking to ensure that the proper file is processed. There is no limit to the ingenuity that can be applied to reduce the possibility of error. One of the best methods available to users of small files is to print out the file and scan it for obvious errors.

Once the information has been entered into the computer and has been determined valid, the possibility of further error is reduced. Error checking, parity checking, and other routines can help ensure that the handling of information does not introduce errors. On the other hand, it is possible to make tremendous blunders with a computer through inaccurate specification or unintentional entry of commands. Such errors are usually easily detected and the results are obviously so far afield that they are of little danger. It is far more likely that an insidious error, such as the transposition of digits, or the misplacement of a decimal point, will cause errors more difficult to pinpoint. These are the errors that preferably will be eradicated through sufficient attention to checking the input.

It is always a good idea to spot check the input against the output to ensure that you have not inadvertently introduced errors.

INPUT PREPARATION 2

Data preparation is extremely important. Collecting and capturing data in a form compatible with all your future needs requires a great deal of forethought. To simply collect all available data into one file frequently results in a huge file which, though perhaps complete, will prove cumbersome in use. If you plan ahead and logically divide potentially large files into smaller ones, and provide a logically common field to correlate the records, you can reduce the amount of time spent in handling the files. In addition to keeping records as compact as possible, you should also give due consideration to the files you work with. File space is valuable. Files can be maintained in an efficient manner by judicious procedures based on the file content. Some files might be used more efficiently when they are combined into a larger file if their use is common to a program, whereas running some large files costs more than is saved because the extraneous information must be bypassed to find the records needed.

It has been found that the processing speeds achieved by modern computers often mask the inefficiency of a procedure. One method of planning efficient file usage is to create a log of the files used and when they are required. You might find that a different organization of the file content would reduce the volume of information to be sorted, searched, or copied. You might find that a file is often not available when you need it and that a scheduling problem could be avoided by changing part of your present procedure.

Employee files are often a result of unrestrained growth. Some such files suffer from poor maintenance. Others only suffer from poor planning. Employee files can usually be divided into smaller files based on their use; each file is interrelated to the other files by a common field such as the employee number or social security number. Parts files are often interrelated by assembly number; sales files are often interrelated by account number, and so forth. Much time can be saved in sorting and searching through files by careful planning in advance, and continuing file maintenance.

HOW IT ALL STARTED

Early attempts at collecting information for mechanized sorting were limited to the size of the punched card. For many years a record was limited to the length of one 80-column card. It was difficult to put all the information desired into such a limited length record, so considerable imagination was used to represent as much information as possible in the space available. Much of this inventiveness has been passed along to present users.

Early attempts at maintaining employee files determined that an employee name and address almost filled one punched card. As a result, name and address files were (and still are) often treated as separate files. One method of relating the address files to other employee records was to make master and secondary (or auxiliary) records. This method preceded the more modern computer methods of relating records to one another by use of a common field because a common field was of little value to card processing machines which could not recognize such fields. Attempts to maintain the master and secondary card files separately, based on a common sequence, were usually thwarted by machine malfunction and human error. Similar concepts are now used occasionally because manual handling is seldom required and machines are much more reliable than they once were. Thus, some concepts surface again when other factors permit.

RECORD DESIGN

Some characters supply no meaningful information and should not be included in a record. In the case of Social Security numbers, the two hyphens serve only a visual function and need not be included in the record. Since they traditionally occur in the same positions, they can be ignored or added later on output if necessary. Such traditional characters are rapidly disappearing; the slashes or dashes in a date are seldom entered into a record. If they are considered necessary, they can be preprinted on an output form and the meaningful digits aligned with them as part of the output formatting process. Most of us no longer expect to see such unnecessary punctuation in computer output. Reducing the size of the record speeds processing and, in turn, changes our concept of the data.

Further attempts to condense record fields led to the use of bit patterns as code to mean specific things. Some items can be represented by a single bit, such as 0 for male and 1 for female. One method of coding uses all 12 punches in a single card column to denote hair color; 11 punches denote color and the 12th signifies baldness.

EXPANDING INPUT

Many other methods of condensing information have been devised. The time to create input is costly in terms of manhours and machine availability. One method of cutting input time is to prepare standard paragraphs that can be called by a single letter designator. Other methods of coding use a table look-up procedure to create more lengthy output from short coded items. One application allows a large number of text paragraphs to be assembled to create a personal reply to correspondence. Again, efficient use of the computer depends to a large degree on the ingenuity of the users.

MAJOR AND MINOR SORT KEYS

All sort operations are based on the comparison of values assigned to the characters to be sorted. We usually speak of major and minor sort keys when we consider that the first key specified will be the most important key, and the rest will be of lesser importance. In a manner of speaking, even the major sort keys can be ranked by character value from left to right. The leftmost character dominates the sort rank.

The alignment of all characters is very important in the sort key field. All numbers are expected to be right justified within the key field. All decimal points, explicit or implied, must be aligned. All characters, other than numbers, are expected to be left-justified in the key field.

The major difference between numeric and nonnumeric data is the sign which requires that the sort order be reversed when the sign changes. When numbers go from positive to negative, or vice versa, the sort order must change. The left column of table 3-1 illustrates this change in sort order.

VARIABLE LENGTH RECORDS

Record length can vary from very short to very long; there is no length restriction on the size of records, but there are restrictions on the types of long records that can be processed by Sort/Merge. These restrictions are noted under Types of Data to be Sorted in section 3.

Records of variable length can occur in the same file; however, the sort keys must be contained in that portion of the record which contains fields common to the rest of the file. For example, if all records in a file have identical fields defined through character position 80, but some of the records are 400 characters in length, the records can be sorted only on the fields included in the first 80 characters.

Also, it is not necessary to output the remaining characters of the variable length records if they have no value to the user of that output. An owncode routine can be written to strip away the extraneous information before it is sorted by using owncode exit 1, or by reformatting the records in the file before sorting, you can drop off the extra characters before sorting the file.

If, for example, a report is desired to compare only a few fields, it could be advantageous to only output those few fields. A chart to depict age and salary for a large organization could be created from a payroll file, and only those two fields need be obtained. The remainder of the record would be superfluous. Again, an owncode routine can be written for this purpose.

In the case of variable length records, it is always preferable to specify the fixed length fields in the record before the variable length fields. It is only possible to sort records on those key fields which have a constant position in relation to the beginning of the record.

RECORDS AND FILES

Files can be organized based on how often they will be used. Some files need only be used on a monthly basis or less, whereas others might be used daily. To search through the records not needed, on a regular basis, is a wasteful process.

Installations using extremely large files often divide the workload so that a large file is only run in sections, and the total file run is completed only once per week or once per month. When the entire file is too large to run in one pass, sections are run independently. For example, an inquiry to IRS might result in the reply that names beginning with S are run only on the 17th workday of each month. In many cases where gigantic files are involved and the workload precludes individual handling, such division of jobs is common. To provide a more reactive response would require such a large computer installation that it would be cost-prohibitive.

Input is often presorted into groups prior to sorting or processing. For instance, all records can be presorted into groups based on the first letter of the name or address, and these groups can be processed on a regular calendar basis, such as A through D on Monday; E through H on Tuesday, and so forth.

In other applications, all processing might be subject to purely mechanical function limitations, such as the number of records that can be processed in a single shift, and work quotas can be established based on this number. Presorting files to establish a logical work base, based on name, region, or other logical basis defines a portion of the work load for each shift, and ensures that the entire file workload is completed in planned steps.

Sorting can be an extremely time-consuming process. Since sorting does not change or improve data, many consider it a nonproductive process. All efforts toward spending as little time possible in sorting records are worthwhile. If a sort can be avoided either by entering data in the desired order or by processing records in their present order, the sort should be avoided. If sorting is essential, it should be made as efficient as possible. Some installations have found that much sorting can be avoided or reduced, thus reducing the need for additional equipment and manpower, or even freeing present staff for other, more productive, assignments.

HOW SORT WORKS

A record is composed of fields. A field is the smallest amount of information you wish to sort. You define a field by its length and its starting position within each record. You can sort all of the records in a file based on the content of a single field. That field is then called the key field or sort key.

A file can be sorted on more than one key field, or on a combination of key fields. For example, a payroll file can be sorted on the name field, or by age (as the major key) and salary (as the minor key). In the latter case, if two or more employees are the same age, their records should appear on the output file in order by salary. The sort program compares the information in a key field with the information in the same key field of all the records in the file, and places all of the records in that file in the order you have specified. If you specify ascending sequence, the lowest number or the lowest character, according to the collating sequence, will be output first, followed by the next lowest, and so on. If you specify descending sequence, the highest number or character will be output first. If you are sorting signed numeric values, the order will change when the sign changes.

SORTED FILES

A file is said to be a sorted file when all of the records are in a defined sequence. This means that all of the records are in order as defined by at least one key field. Sorting a file on the first field, in the case of an employee file, would result in a sorted file. Later rearrangement of the file into the order of another field, such as Social Security number or age, also results in a sorted file. The only difference is that the file is sorted on another field. A file can be sorted on as many fields as there are fields in the record.

WHAT FIELDS WILL I SORT ON?

It is a good idea to consider all possible future uses of a file you are creating. Will you be sorting on all of the fields in each record? Will the file be compatible with other files already created? Will there be a need to merge this file with other files? What interrelationships exist now -- and what are other possible future needs?

You will probably sort a payroll file on name, salary, and so forth more than on an address field. In fact, it is highly unlikely that you will ever specify a street address as a sort field, and only slightly more likely that a city or state will ever be used as a sort key. On the other hand, the postal ZIP code is gaining in importance and the reduced postal rate for presorted mail makes it worthwhile to be able to sort on ZIP codes for employee mailings. Therefore, the ZIP code should be defined as a separate field when creating records if you want to take advantage of this reduced rate.

CHARACTER SETS

Four character sets are available on the Control Data computer systems described in this user's guide:

CDC 64-character set
CDC 63-character set
ASCII 64-character set
ASCII 63-character set

Only one of these character sets will be in use at your installation. It was selected when the software was installed. You cannot change it; but you must be aware of the set in use so as to determine the type of output translation you will receive. Please note the character set in use at your installation on the inside of the front cover for future reference.

To understand the meaning of a character set requires that you recognize certain aspects of computer encoding. No character enters the computer as a character, nor is it stored as a character. When you punch the letter A in a punched card, the letter is translated into the punch combination 12-1. This means that a hole is punched in rows 12 and 1 in the card column containing the letter A. When this punched card is read by the card reader, the 12-1 punch combination, not the letter A, is translated into a binary bit pattern of 000001 and sent to the computer along with other similarly translated information from the card. Graphic characters, such as the letter A, are recognized only by the card punch, terminal, and line printer. All of the rest of the computer equipment deals only with binary bit patterns.

DISPLAY CODE

Control Data computers are binary machines which use display code representation. Most CDC computers use a 6-bit binary string to represent one character. Because binary information is difficult for people to read, an additional representation of the bit string is used: the six binary digits are also recognized in groups of three, and the result indicated in an octal (base eight) numbering system. A character in display code is then a two-digit number from 00 to 77. For example, the binary value 000001 represents the letter A in the computer. In display code, the same value is expressed as 01 which represents the letter A.

For a full overview of the character sets, see appendix A; for a good explanation of the octal number system, see the NOS/BE user's guide. The binary designations 000000 through 111111 are available using the 64-character sets. The 000000 grouping is not used with the 63-character sets.

ASCII CODE

Most of the characters available under the display code character set are also available under the ASCII character set. A few special symbols are not represented in both character sets.

The ASCII (American Standard Code for Information Interchange) character set is essentially an 8-bit character set which defines only 128 of the possible 256 characters. It was developed as an industry standard, chiefly to make the interchange of information represented by different computer manufacturers' machines more compatible. All of the 128 defined ASCII characters could be represented by seven bits. However, since the Control Data character codes discussed here are 6-bit codes, all information input in ASCII format is converted to 6-bit code between the input medium and central memory -- and is again converted between memory and the output medium. ASCII input usually comes only from tape, ASCII-CRT and ASCII-TTY terminals; ASCII output is usually returned to the same devices, plus the extended character set printer. Obviously, only a subset of the 128 ASCII codes can be represented by a 6-bit character set which only allows representation of 64 characters.

Lowercase characters are treated as uppercase. The less common characters are usually ignored or treated as blanks. If the ASCII 8-bit characters must be maintained, options are available through FORM and the 8-Bit Subroutines to support the remaining ASCII characters, such as the 96-character subset, but these subsets are limited to the applications which support them.

Minor discrepancies exist between the ASCII and display code character sets; for example, when you specify a quote (") on an ASCII input device, a not equal (≠) symbol is printed on a display code printer. Other differences are noted in appendix A.

Characters in any standard CDC 6-bit format can be sorted. Data on tape can be converted to CDC 6-bit format either directly, or through use of one of the available utility programs.

For example, tape files created on an IBM computer will probably be in IBM standard EBCDIC 8-bit format. These files could be converted on the same or a similar IBM computer to ASCII 8-bit format for interchange with a CDC computer. If this is done, the ASCII format will allow the CDC computer to read and process the IBM file. The ASCII output will be limited to the CDC 63- or 64-character set in use; the remaining characters will be ignored. More likely, however, is the case where the EBCDIC 8-bit tape appears at a CDC installation and the intent is to merge the files contained on the tape with certain other files on CDC 6-bit tape. To do this requires

the use of FORM or the 8-Bit Subroutines programs, or an intricate owncode routine which must be created for this purpose. FORM is the easiest utility to learn to use, and certainly is the easiest method of converting the information to CDC format. Examples of this conversion and the exchanges possible are shown in the FORM reference manual.

The content of an EBCDIC print tape can be most easily determined by printing at least part of the tape using the 8-Bit Subroutines utility COPY8P developed for the express purpose of printing IBM format tapes. To use COPY8P requires the use of an upper/lowercase print train on a CDC 512 or 580 printer as described in the 8-Bit Subroutines reference manual.

Sorting is the process of arranging items in order. Order is defined by the person doing the sorting, though reasonable agreement exists as to the proper order of things. We know we can sort all items into any order we establish, and proceed on that basis.

Single alphabetic items can be sorted quite simply. Everyone knows the alphabet. However, if we wish to consider all letters, and do the job properly, then what is to be done with letters from other languages such as Σ and Γ, not to mention lowercase letters, special symbols, and punctuation marks?

The decimal numbers 0 through 9 can be sorted quite easily. A computer can handle this type of sorting with no problem. The octal numbers 0 through 7 are also sorted quite as easily by the computer, as are the binary numbers 0 and 1. Yet, if we encounter a value on a printout of 101101, do we really know what that value is? It could be 101 101 decimal, or 45 decimal expressed in binary, or 55 octal (which is the blank in display code), or even 2D in hexadecimal, or the minus sign in ASCII.

All data must be described exactly. Otherwise the computer will not process it correctly, nor will we be satisfied with the result. Even blanks pose a problem. Should they collate before or after letters or numbers?

Not all special characters can be processed by computer due to the limited size of the character set. Sorting the special characters is a matter of preference. The special graphics are treated differently in the various character sets and are collated in different order as well. For this reason, they are considered arbitrary characters. It is generally agreed that B should follow A, and that 3 should follow 2, but you must decide whether you wish @ to precede or follow ≠. Also keep in mind that not all special characters appear in all character sets. The predefined collating sequences of the arbitrary characters are shown in appendix A for the collating sequences available. If these predefined collating sequences are not suitable, you can create your own collating sequence.

SORT KEY DESCRIPTION

You must define every field to be used as a sort key. Sort key descriptions include the following information:

- Key length
- Starting location of key within record
- Type of data found in key field
- Sort order
- Collating sequence to be used (for character keys only)

Key field length is specified as the number of bits and bytes in the field; the default is a 6-bit character.

Starting position of a sort key field can be anywhere within a record but it must be the same for all records of all files to be sorted or merged.

Key type defines the type of data in each sort field as described below.

TYPES OF DATA TO BE SORTED

Any meaningful data that can be logically translated into a binary computer code can be sorted into the order you specify. Certain limitations are inherent in this statement. All character data to be sorted should be limited to the 63 or 64 characters available; however, data can be expressed in other forms as well, and need not be limited to 6-bit binary values, as shown below. In some cases you might wish to transform the keys into another form to achieve your purpose. Tag sorting, for example, is mentioned in section 6.

Logical Key

Unsigned binary integers of any length can be sorted. In actuality, it is unimportant to the computer how the binary integers are divided for human recognition. The binary integers are assumed to be positive values and are sorted by magnitude. This type of sorting data by actual binary value is also of importance in sorting other types of data as will be explained later. As an example, when you specify a sort of DISPLAY characters with the DISPLAY collating sequence, Sort/Merge automatically assumes logical key sorting because it is faster.

Integer Key

Any 60-bit integer can be sorted in this manner. The CDC computer word size is 60 bits, which is large enough for most uses. The 60 bits you choose from the record to be sorted need not start or end on a character or word boundary. Any consecutive 60-bit value from any portion of the record can be chosen. It is considered better programming practice to restrict these 60-bit values to word boundaries. This permits easier field specification and promotes compatibility with other products that might be used to handle the record.

Display

Most data in character form is written in display code; it can be any number of 6-bit characters. Display code is usually sorted by a predefined collating sequence as defined in appendix A. Display code is the standard CDC character code. The DISPLAY key should always be specified when character data is used.

Signed Numeric Data

Numbers that require a sign to show they represent a plus or minus value, such as debit or credit amounts in an accounting file, are referred to as signed numeric values. The sign, plus or minus, can appear in one of four places. When the sign is plus it is often omitted because all values are assumed to be positive unless otherwise specified. The sign is often an overpunch representation over the last (rightmost) digit in the field because previous accounting systems used this method. Other accounting systems used the sign as an overpunch over the first (leftmost) digit, though this method was never as popular. If the sign is not carried as an overpunch, it will appear either as the first or last character in the numeric field. When the sign is specified separately as the last character, all values must carry the sign. In any case, the sign position must be consistent throughout the file to be sorted.

Sorting signed numeric data requires that the order, ascending or descending, change at the point the sign changes. If descending order was specified, positive values are output in descending order. At the point the values become negative, the numbers are output in ascending order.

All signed numeric values must be stored internally as display code. A numeric digit with an overpunch is represented in display code as the character it becomes as shown in table 3-1.

TABLE 3-1. SIGN OVERPUNCH CODES

Sign of Field	Value of Digit	Key Punch Code from Digit Punch and Sign Overpunch	Equivalent Display Code Character
+	9	12-9	I
+	8	12-8	H
+	7	12-7	G
+	6	12-6	F
+	5	12-5	E
+	4	12-4	D
+	3	12-3	C
+	2	12-2	B
+	1	12-1	A
+	0	12-0	<
-	0	11-0	v
-	1	11-1	J
-	2	11-2	K
-	3	11-3	L
-	4	11-4	M
-	5	11-5	N
-	6	11-6	O
-	7	11-7	P
-	8	11-8	Q
-	9	11-9	R

The easiest format of signed numeric data to sort is one which uses the leading sign as a separate character in a constant position in a value with leading zeros, such as -00153.62. Plus signs must be specified.

When sorting signed numeric data, you must specify additional parameters in the FIELD directive. If the sign is an overpunch, you must specify SIGN and LEADING or TRAILING to identify the location of the overpunch. If the sign is not an overpunch, you must also specify SEPARATE. For example, if a separate sign character is used as shown above, the last three parameters of the FIELD directive must be . . . ,SIGN,LEADING,SEPARATE.

Note that positive integers, display code numbers without a sign, can be mixed with integers that have a sign overpunch in an input file.

Float

If you are dealing with floating point numbers, you can specify FLOAT. Any 60-bit normalized or unnormalized floating point numbers can be sorted. They can start at any bit position within the record. They are sorted only by numeric value. Any floating point number written under the CDC NOS or NOS/BE operating system by a binary write can be sorted. See the Sort/Merge reference manual for the permissible range.

INTBCD

The use of INTBCD is limited to specific collating purposes. The internal BCD character code was common to some CDC 3000 Series computers, and has fallen into disuse in larger computers. One reason for this continued capability is that it allows upward mobility of data from smaller CDC computers. INTBCD should not be specified as a key type unless you have a real need to use it and understand exactly what you intend to do with it.

COLLATING SEQUENCE

Just as the character set determines what binary equivalent is assigned to the graphic character input to the computer, the collating sequence determines the order of each binary equivalent to one another. Thus, the collating sequence determines the precedence given to each character, whether it is in graphic or binary form. The collating sequence usually specifies that B will follow A and that 2 will follow 1, and so forth.

Changing the collating sequence specification changes the order of the items output from a sort.

The collating sequence applies only to character data, not to numeric data. The collating sequence can be any of the following:

ASCII6

COBOL6

DISPLAY

INTBCD

OWN

The collating sequence can be specified as one of the existing four collating sequences, or you can specify your own collating sequence if you wish. The collating sequence chosen need not correspond to the character set used in coding the data. The character set in use determines the translation between the 6-bit binary value and one of the letters, digits, and special characters available as graphics. You cannot change the character set. The collating sequence, on the other hand, determines the precedence given to each character already translated, when the key is sorted. You can select the collating sequence.

If you do not specify a collating sequence, a default collating sequence will be assumed. For example, if the character set in use at your installation is the CDC character set, the default collating sequence will be COBOL6. If the character set in use is the ASCII set, the default collating sequence will be ASCII6.

If the file that is being sorted is subsequently to be merged or compared with another file, it is essential that the two files are arranged according to identical collating sequences.

Selecting a Collating Sequence

The DISPLAY collating sequence usually is the default collating sequence because the DISPLAY character set is native to the CDC CYBER computers. This makes its use much more efficient than COBOL6 or ASCII6. However, the DISPLAY collating sequence orders blanks and special characters after alphabetic and numeric characters. For this reason it is not the best collating sequence for most directory sequence applications.

If you compare the collating sequences in appendix A, you will notice that the ASCII6 collating sequence orders numbers ahead of letters, while COBOL6 orders letters ahead of numbers. This difference in collating sequences can at times be made to work to your advantage.

When selecting the collating sequence of a large data file, there are a number of factors that should be carefully considered. If there is a possibility the file will be used on other computers, the 8-bit ASCII code is a more frequently used code for information interchange. Therefore, it could be advantageous to select the ASCII6 collating sequence now. When the file is converted, it will be in the desired sequence without requiring a sort as part of the conversion process.

One overriding consideration in selecting a collating sequence is the sequence of any other file with which the data is to be processed. When data is to be used with another file, it must be in the same sequence as that file. Advance planning to ensure the greatest degree of compatibility between files that might be used together can significantly reduce processing time.

Importance of Blanks

To further demonstrate the differences among the collating sequences, consider the following three names:

```
JOHNS AMOS
JOHNSON CLIFFORD
JOHNSTON ALFRED
```

When these names are input as shown, with a blank between the last and first names, they will sort differently depending on the collating sequence chosen. The COBOL6 and ASCII6 collating sequences will order them as shown above.

The DISPLAY collating sequence will arrange them as:

```
JOHNSON CLIFFORD
JOHNSTON ALFRED
JOHNS AMOS
```

The INTBCD collating sequence will arrange them as:

```
JOHNSON CLIFFORD
JOHNS AMOS
JOHNSTON ALFRED
```

The Sort/Merge reference manual describes how you can create your own collating sequence.

ALTERNATE SPECIFICATION OF KEY TYPES

There can be advantages in specifying a different key type than the one you would expect to use under some circumstances. Sort/Merge processes integer keys and logical keys faster than floating point or character coded keys. Thus, when possible, it can be to your advantage to specify the key type which allows the faster sort. When you have reached the level of expertise to want the fastest possible sort, see the Sort/Merge reference manual for additional information on this subject.

SORT ORDER

Most readers will agree that the normal order of the alphabet is A through Z and the normal order of numbers is 0 through 9. This order is often called ascending order because it goes from the item of lowest precedence to the item of highest precedence. To reverse this order of precedence results in descending order.

Student grades are often given based on a scale of 0 through 100 or on the scale A through F. In the case of numbers, 100 is the best possible mark, whereas in the case of letter grades, A is the best possible mark. When sorting a file of student grades, in order for the best marks to appear at the top of the list, letter grades should be sorted in ascending order and number grades in descending order.

Sort/Merge allows you to specify the order you desire, ascending or descending, for each key field to be sorted.

USING MERGE

A merge is simply the process of putting together two or more files that are in the same sequence based on the same key. A merge run results in one new file containing all of the records from the input files in the same sequence as they were supplied.

Merge is most commonly used to combine existing sorted files. Whenever new data is to be added to existing files, a merge is the most efficient method to use. (A merge operation is almost as fast as a copy operation.) Merges are also most useful in combining related sets of data, such as address files with payroll files for employee reports and tax reports.

If two files to be merged are not in the same sequence, one of them must be sorted to match the other prior to the merge. If files that are not in the same sequence are merged, the entire resulting file will require sorting at some future time. The time required to sort all of the data compared to the time to sort and merge only part of the data is the valid comparison. The larger the file, the more time the sort takes.

The frequency with which files are merged depends on the application. An airline ticket handling file might merge updates to the file on very short notice, if not totally interactively. Certain legal applications are based on daily postings, such as the title insurance for real property. All transactions must be entered into the master file by the start of the next business day. A merge of the previous day's activity provides the best answer to such a problem. A sort of the entire file is not necessary.

A large university might keep computer files on its students that would only need to be updated once per quarter or semester. State and Federal income tax files are usually updated only once per year. The United States census is only conducted every 10 years.

The basic process of a merge operation is to arrange data from two or more ordered lists by interleaving the records from the ordered lists.

In its simplest form, a merge is the process of taking the next sequential item from one of two ordered lists, moving that item to the output file, and repeating the process. The term merge assumes that the input data has been presorted.

The purpose of a merge is to create a single ordered output list.

MERGING DURING A SORT

The process of merging is often performed as a part of the process of sorting. Consider for instance that the amount of central memory available for sorting can only contain 1000 records while the file to be sorted contains more than 2000 records. The records cannot all fit into central memory and be sorted in a single process. Thus, the input is sorted into separate strings of a length that will fit into the available central memory. These strings are usually stored on a disk file for subsequent merging with other sorted strings. A replacement technique that is used during the formation of these strings allows them to contain more records than can be contained in central memory, but in this case there will be at least two strings formed. These strings are then merged to form the final output file of the sort.

MERGE ORDER

When only two strings are created, the merge order is two. If three strings are created, the merge order is three. There is a practical limit to the number of strings that can be merged efficiently. This number varies according to several different factors; it is usually between 2 and 64. The dominant factor in the selection of a merge order is the amount of central memory available; however, it can also be affected by the total number of strings to be merged and the amount of disk activity within the system.

If the number of sorted strings does not exceed the merge order, all the strings can be merged and written to the output file with no intermediate merge. If the number of sorted strings is greater than the merge order, several strings must be merged to form longer strings. This procedure is repeated until the number of strings is less than or equal to the merge order, at which time the strings are merged onto the output file.

Merge order setting is applicable only to disk sorts. The merge order for tape sorts is determined by the number of tape drives available. The default merge order is computed by Sort/Merge for each sort based on algorithms which yield optimum performance for most users. Sort/Merge users who frequently execute very similar sorts of very large files might try to improve Sort/Merge performance by selecting and specifying a merge order; however, it is more likely that an inefficient merge order will be chosen that will degrade Sort/Merge performance. It is worthwhile to compare merge order settings only when all other variables are unchanged.

The merge order formulas appear in appendix F of the Sort/Merge reference manual.

Control statement sorts are also known as directive sorts. Either term is acceptable. Either term differentiates this type of sort from the FORTRAN Extended calls to Sort/Merge, the macro calls to Sort/Merge, and the COBOL SORT verb. The sort directives specify exactly what is to occur during the execution of the Sort/Merge run. The directives are closely related to the file to be sorted. They do not interface with the operating system. As a result, the directives need not be changed if you run the Sort/Merge job on the NOS or the NOS/BE operating system.

SORTMRG STATEMENT

The SORTMRG control statement, on the other hand, functions in the same manner as a compiler call. It is the only Sort/Merge control statement. It is an operating system call, which results in the compilation of a Sort/Merge program based on the parameters specified, or upon the default values assumed. It should be noted that some default values, such as merge order, are not preselected but are actually computed to determine the most efficient setting.

FILE STATEMENT

Because Sort/Merge performs all input and output through CYBER Record Manager, a CYBER Record Manager FILE statement must be provided for every output file to be processed by a directive sort or merge. A FILE statement is not needed for the file named INPUT. A full description of the required FILE statement appears in the Sort/Merge reference manual. The FILE statement should not be confused with the FILE directive.

Sort/Merge also requires, for internal use, that a value for the maximum record length be set. This value can be specified either by the MRL or the FL parameters on the FILE control statement, or by the MRL parameter on the OWNCODE directive. If the value is specified more than once, the largest value specified is used by Sort/Merge as the value for all files.

SORT/MERGE DIRECTIVES

The Sort/Merge directives can fill all columns from 1 through 72 of a punched card or an input line. Directives can be continued on the next line by starting the continuation line with a comma. The number of continuation lines is not limited.

Comments can be entered by placing an asterisk in column 1. Comments can be placed anywhere in the Sort/Merge input deck. They are printed when they are encountered during processing but they do not in any way affect processing.

A number of special characters are reserved by the Sort/Merge program as field or parameter separators for directives. Terminators (characters which end a directive) are not required and are ignored. Additional restrictions are placed on separators when used with certain directives; these are explained with the directives which restrict their use.

Use of the comma is recommended in all cases where a choice of separators is allowed. Only the comma is used as a separator in the following directive descriptions. Blanks occurring before and after separators are ignored by the system except in the case of the SEQUENCE directive. Refer to the Sort/Merge reference manual for a complete list of the restrictions on their use.

The Sort/Merge directives can be specified in any order, except that END must be the last directive in the sequence.

SORT

If you wish to sort records, the SORT directive must be specified. The SORT directive usually has no parameters. In the case of a tape sort, a parameter specifying the tape sort is required. For Sort/Merge 1, an optional parameter allows you to specify the amount of large central memory to be used as a buffer area.

MERGE

The MERGE directive specifies merge-only processing. Either SORT or MERGE can be specified, not both. MERGE has no parameters.

FIELD

The key field on which you wish to sort must be defined and named in the FIELD directive. Each field is given a name and is identified by starting position, length, and key type. At least one FIELD directive must be specified; up to 100 fields are allowed.

Additional parameters are available to specify a separate + or - sign for numeric data in display code, or as an overpunch. These parameters are described in section 3 and in the Sort/Merge reference manual.

BYTESIZE

The size of the bytes referenced in the FIELD directive for each job is predefined by the BYTESIZE directive. If BYTESIZE is omitted, the default of 6 bits per byte is assumed. BYTESIZE does not permit the use of 8-bit bytes for character input. It is provided only as a convenience for specifying fields by byte and bit positions.

You can specify the starting position of the sort key by byte, by bit, or a combination of the two. If you specify by byte, the number of bytes from the beginning of the

record starts with 1. The same is true with bits; however, you must preface the number of bits with a period. Or, you can combine bytes and bits by separating them with a period, such as 4.4, which means byte 4, bit 4 (the 22nd bit in the record if BYTESIZE is 6--this could also be specified by .22).

characters	T	H	I	S
binary	010100	001000	001001	010011
				↑
				bit 22

For key types other than DISPLAY, you might wish to specify words instead of bytes. One method of doing this is to specify BYTESIZE equal to 60.

KEY

The KEY directive is required to specify the order and collating sequence of the sort keys. As the FIELD directive identifies the field to be sorted and gives it a name, the KEY directive specifies the order and collating sequence for the named key field.

A different sort order and collating sequence can be specified for each key field identified by keyname. Up to 100 keys can be specified.

You can specify order, ascending or descending, by A or D. If you do not specify order, ascending order is assumed.

The collating sequence can be one of the standard collating sequences: ASCII6, COBOL6, DISPLAY, or INTBCD; or your own collating sequence, which you must define by use of the SEQUENCE directive. In any case, you will only need to select a collating sequence if you have defined the key type as INTBCD or DISPLAY on the FIELD directive.

SEQUENCE

With this directive you can specify your own collating sequence or change the default collating sequence to a different standard collating sequence.

If you choose to specify your own collating sequence, you can specify it in characters, octal values, or both. If you use characters, they will refer to the binary equivalents defined by the character set named in the FIELD directive. Octal values refer to the binary value in the character positions, regardless of the character set specified in the FIELD directive. Octal values are expressed as two-digit values; single digit values are assumed to be characters. A blank is only allowed to represent a blank in the SEQUENCE directives. The special characters used as separators in the SEQUENCE directive can only be specified as octal values in the new collating sequence. See the Sort/Merge reference manual for details.

OPTIONS

The OPTIONS directive allows you to exercise more direct control over some parts of the Sort/Merge run. User options can be divided into three categories: order, dumps, and optimization.

Order

The RETAIN option allows you some control over sorting records with identical sort keys. When this parameter is specified, records with identical sort keys are output in the same order as they were read, as specified by the FILE directive. When this parameter is omitted, records with identical sort keys are sequenced arbitrarily. RETAIN does not work with MERGE runs.

The VERIFY option, when specified, causes the output file to be checked for correct sequencing. If the order of records on the output file is incorrect, the job stops and the output file is lost.

VERIFY is most often used to verify the order of records from merged input files or records inserted through owncode exits (exits 3, 4, 5). There is nothing to gain by specifying VERIFY for a sort run with no owncodes. Owncode is described in section 5.

When VERIFY is not specified, a sequence error during a merge run with no records inserted results in only an error message; all records are written to the output file, but they need not be in order.

There can be an advantage to not specifying VERIFY when you want to insert records, such as page headers, out of sequence through use of owncode (exits 3 and 4); no checking takes place and the records are inserted as specified. Merge cannot use owncode exits 1 and 2.

Dumps

The VOLDUMP is a carryover from the days of tape sorting, when it could be expected that some tapes would be bad, and when power failures or machine malfunctions were commonplace events. A checkpoint at the end of each tape volume reduced reprocessing when such problems occurred. A VOLDUMP now means that a checkpoint will be taken at the end-of-volume on input and at the new-volume condition on output whether on tape or disk.

Checkpoint dumps can also be specified at specific intervals such as after every 1000 records. The concept of a checkpoint dump is a good idea, yet in actual practice it is better to use other methods to achieve the same result. Checkpointing requires that a dump be taken and stored at regular intervals and that system resources be tied up to manage this process. In practice, it is faster to break up a large sort into smaller sort files because overall sorting time is reduced and system overhead is lower.

The option DUMP(nn) can be used to specify that a dump be taken every time the specified decimal number of records is read from the input file or written to the output file. DUMP can also be specified without the specific number appended, in which case a checkpoint is taken after each group of 50 000 records by default.

The NODUMP option specifies that no checkpoint dumps will be taken.

Optimization

Three options are available to you in the form of directive parameters: two influence the sorting process itself, and one can be used to change the merge order.

The COMPARE and EXTRACT options are mutually exclusive within a sort. The key comparison technique runs faster but requires more central processor time; the key extraction technique runs slower but requires less central processor time. If neither option is chosen, Sort/Merge will choose the better technique according to its own algorithm. This algorithm usually yields the better choice, but should you wish to test the validity of the algorithm for a particular sort run, you can make the run twice, using both options, to satisfy your curiosity. On sufficiently large sort runs, an occasional check of both options is warranted to determine that time is not being wasted.

The ORDER option allows you to specify the merge order used during a particular sort run. Merge order determines the number of merge buffers used during the intermediate merge phase to merge sorted strings into longer sorted strings, as described in section 3.

Merge order cannot be specified for COBOL sorts, nor can merge order be specified for a tape sort since the number of available tape drives determines the merge order.

Sort/Merge computes the most efficient merge order based on formulas which should yield the most efficient performance. Though it is possible that a user change in merge order could produce a faster sort, it is more likely that a user change to the merge order would degrade Sort/Merge performance.

EQUATE

The EQUATE directive allows you to declare one or more characters in a collating sequence equal to another character in the collating sequence. For example, leading blanks do not collate the same as leading zeros in numeric data. This can be a problem, depending on the collating sequence selected. To avoid this problem, you can equate the blank to zero. However, equating the blank to zero does not move the sign in signed numeric data (see section 5). To equate the blank to zero, you can specify either the characters or their display code octal equivalents as shown below:

```
EQUATE,DISPLAY( ,0)
```

or

```
EQUATE,DISPLAY(55,33)
```

Refer to the Sort/Merge reference manual before using the EQUATE directive.

OWNCODE

If you have specified the owncode capabilities as described in section 5, you must use the OWNCODE directive to specify legal entry point names to relocatable owncode exit routines. The OWNCODE directive should not be confused with the SORTMRG control statement parameter OWN which is also required when owncode binaries are to be input on a file name other than INPUT. Owncode is described in section 5.

FILE

In addition to the FILE control statement required by CYBER Record Manager, you must also supply a FILE directive to specify all of the input and output files to be used during a sort run. For a merge run, you need only specify the merge input files and the output file; for a sort run, you can specify either INPUT or SORT and OUTPUT. You must specify the name of the file as that given on the FILE control statement. You can also specify the system action to be performed when processing is complete, such as close and rewind or unload the file.

A local (or a NOS indirect access permanent file) can be sorted upon itself; that is, the output filename and the input filename can be the same.

The order in which the input files are specified determines the order in which they will be read. If you have specified the RETAIN option, records with duplicate or equal keys will be output in the order they were read.

TAPE

If you wish to use the tape version of Sort/Merge, you must use the TAPE directive to specify the names of the intermediate merge files. You must also specify a tape parameter, either POLYPHASE or BALANCED, in the SORT directive.

Tape sorting should be avoided if possible. If a tape sort cannot be avoided, refer to the Sort/Merge reference manual appendix on tape sorting. The difference between BALANCED and POLYPHASE is explained there. POLYPHASE is usually the faster tape sort unless more than eight tape drives are available. When 10 or more drives are available, a BALANCED sort might be faster.

A simple tape sort is described in section 1. Do not confuse a disk sort, using tape input and tape output, with a tape sort. A disk sort stores the sorted strings on disk as the intermediate storage medium. A tape sort stores the sorted strings on tape. A tape sort is only chosen when insufficient disk facilities are available.

JOB EXAMPLES

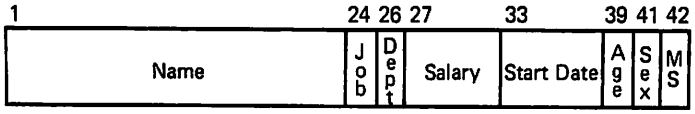
After you have created or selected the file you wish to sort, you will need to write the specifications which will direct the sort process. These specifications, called the sort directives, affect only the sort process. Sort directives do not change even when the sort is run on a different CDC computer, or under either the NOS or NOS/BE operating system. The control statements, however, are different for the two operating systems. Only the NOS operating system control statements are given in the following examples. The NOS/BE control statements are given in appendix C.

If you would like to practice running sort programs, you might wish to create a file of punched cards similar to that used in the examples which follow. Practice of this nature can sharpen your ability to work with your operating system as well as with Sort/Merge.

The record format shown in figure 4-1 is used for the practice examples in this section. The input deck created in this format is shown in figure 4-2.

Figure 4-2 gives the punched card input created for practice with the following examples. These punched cards are intentionally formatted in a manner similar to the examples contained in section 4 of the Sort/Merge reference manual.

The sort directives shown in figure 4-3 will put the preceding records in alphabetical order by employee name.



Name 23-character display coded field.

Job 2-character display coded field containing a job grade identifier:

2	employee
4	foreman
6	supervisor
8	manager
10	general manager
12	director

Dept 1-character display coded field containing a department identifer:

A	production department
B	shipping department
C	personnel department
D	accounting department
E	sales department

Salary 6-character display coded field with leading zeros.

Start Date 6-character display coded field in the format mmddyy:

mm	decimal number of month
dd	decimal day of month
yy	last two digits of calendar year

Age 2-character display coded field.

Sex 1-character display coded field.

M	male
F	female

MS 1-character display coded field containing marital status:

M	married
S	single
D	divorced

Figure 4-1. Record Format

These directives can appear in any order except that END must appear last. The SORT directive declares that the sort process is selected. The FILE directive declares that input is to come from the card reader since the card reader is the default input source. Output is to the file NEW instead of OUTPUT, because the file must be rewound and copied using COPYSBF (copy shifted binary file) as shown in figure 4-4. If COPYSBF is not used, the first character of each print line will be taken for the printer carriage control character and thus disappear. In addition to losing the first character of each line of output, the output will appear in a strange format since many of the first characters of each output line will cause multiple line skips or skip to a new page. COPYSBF causes the first character of each line to be blank which, as a printer carriage control character, causes the printer to space to the next line (single space). This example shows output going to the file NEW which is rewound and output after being shifted one position by COPYSBF.

The sort key field identified by the FIELD directive is NAME starting in column 1, and extending 23 characters in length, in display code. The key on which the file will be sorted is the field NAME as identified in the FIELD directive; output is to be in ascending order, and in display code collating sequence.

Some directives require the use of certain characters that would otherwise be allowed as general delimiters. These characters must then be used only for their specified functions. All of the exceptions are noted in the Sort/Merge reference manual; exceptions are noted here only when encountered in the examples.

In addition to the sort directives, you will also need to supply the operating system control statements. The NOS control statements shown in figure 4-4 were prepared to sort a deck of punched cards (figure 4-2) using the sort directives given in figure 4-3. The NOS/BE control statements are given in appendix C.

```
BOER,GEORGE                2 A000795122374 28MS
WANG,LISA                  2 D000585112777 23FS
DURAND,HELEN               6 A001635090168 36FD
MILLER,FLORENCE            2 A000610050378 22FS
POPOV,IVAN                 2 E000665070977 23MS
MARTIN,RICHARD             4 A000921121573 38MM
SOKOL,DONALD               8 A002108120768 46MM
JONES,CHERYL               2 D000685082775 24FM
LOPEZ,COSME                2 A000367C030776 22MM
CHANG,ROBERT               2 A000684020176 25MS
JOHNSON,ANNABELLE          6 C001472040369 26FM
NEWMAN,ANDREW              2 A000710070177 24MM
WILSON,DOUGLAS             2 E000910120573 36MD
DUBOIS,ANDRE               2 A000625031578 21MS
DAVIS,ROBERT               2 E001075010373 35MS
TAYLOR,JENNIFER            2 A000625101177 22FS
MULDER,HENK                4 B000911040174 41MM
BERNARD,JOHN               2 A000625011577 19MS
WILLIAMS,BENEDICT          6 A001721081672 39MD
MEYER,WILLIAM              4 A000875030176 32MS
SMITH,JOHN                 2 A000810091873 48MM
PETIT,ARNOLD               2 A000625022178 20MS
JOHNSON,ARMAND             4 E001410102072 32MS
BROWN,JAMES                2 A000700091176 23MS
ANDERSON,TIMOTHY           2 A000J700052575 26MM
LI,WANG                    2 A000710011077 31MS
CARLSON,JACK               8 E001975083C74 36MD
SMITH,ROBERTA              4 A000J75011227 27FS
COHEN,JOSEPH               2 A000875040474 35MD
SCHULZ,CHARLES             2 A000625040178 18MS
SMITH,MARGERY              4 E001135021575 28FS
KIM,LEE                    2 A000610112277 19FS
BAKKER,JOACHIM             2 A000710011176 24MS
IVANOV,LEONARD             2 A000723081576 32MM
GOMEZ,LINDA                2 C000635042177 21FS
WILLIAMS,ROBERT            2 A000790031576 31MS
GARCIA,ARTHUR              2 B000738022976 34MM
JONES,FRANCES              2 900075E110374 27FS
PETROV,GEORGE              2 C000750063076 29MD
FISCHER,DAVID              2 A000625060178 21MS
```

Figure 4-2. Input Records

```
SORT
FILE,INPUT=INPUT,OUTPUT=NEW
FIELD,NAME(1,23,DISPLAY)
KEY,NAME(A,DISPLAY)
END
```

Figure 4-3. Sort Directives

For the NOS operating system you will need to supply the typical job, user, and charge control statements, as shown in figure 4-4. In addition, you must supply a CYBER Record Manager FILE control statement in order to change the default block type and record type, and specify the length of the longest record, for the output file. If you do not supply this information on a FILE control statement, the job will either terminate with a fatal error or yield unusable output. The reason for this problem is that the CYBER Record Manager default for the file NEW is BT=I,RT=W. Since NEW is to be listed on the printer, the CYBER Record Manager FILE statement specifying FILE(NEW,BT=C,RT=Z,FL=80) is required.

The block types and record types are explained in the CYBER Record Manager publications listed in the preface.

```
jobcard.
USER(usernum,passwrd)
CHARGE(accounting information)
FILE(NEW,BT=C,RT=Z,FL=80)
SORTMRG.
REWIND,NEW.
COPYSBF,NEW,OUTPUT.
7/8/9 multipunched in column 1

  sort directives (Figure 4-3)

7/8/9 multipunched in column 1

  input records (Figure 4-2)

6/7/8/9 multipunched in column 1
```

Figure 4-4. NOS Control Statements

Because the input default is for card format, there is no need to specify the FILE control statement for INPUT in these examples.

The SORTMRG control statement is all that is needed to call Sort/Merge. All of the necessary values are selected by default or computed based on other parameters supplied.

The output is rewound and all records shifted as a result of the COPYSBF statement before being printed.

The resulting output should appear as shown in figure 4-5. The header line states 7C DIRECTIVES which means that Sort/Merge version 4 directives were used. Users of directives in the Sort/Merge version 3 format will find 6C DIRECTIVES in the header. The remainder of the header line shows that Sort/Merge version 4.6 installed at level 495 as run on March 29, 1979 at 2.06 P.M. The sort directives specified for this run are listed. No other information appears on page 1. A horizontal line in the figure indicates a page break.

Page 2 (and on) gives the printed sorted output. The last page of output header gives the job name assigned by the operating system as it appeared on the banner page, the date of the run, additional information about the computer installation, the computer serial number and the operating system used.

The dayfile indicates a successful run by not indicating any errors. The job control statements are listed in the order they were executed and the Sort/Merge statistics are set off by asterisks. The key comparison technique was selected by the program; no records were inserted or deleted on input or on output; 40 records were input and 40 records were output. The merge order determined by the Sort/Merge program for this job was 12. Accounting information is printed to indicate the time and resources used. Depending on the installation, this information is also used to bill the group or project named in the CHARGE statement.

The sorted output shown in figure 4-5 was cataloged as a permanent file for future use as input to other practice jobs. The SAVE,NEW. card in the job control statement section causes the file NEW to be saved as a NOS indirect access permanent file as shown in figure 4-6.

```
   7 C   D I R E C T I V E S    SORT/MERGE 4.6 L495  79/03/29. 14.06.50.      PAGE      1


1  SORT
2  FILE,INPUT=INPUT,OUTPUT=NEW
3  FIELD,NAME(1,23,DISPLAY)
4  KEY,NAME(A,DISPLAY)
5  END

-----------------------


ANDERSON,TIMOTHY          2 A0007000525752 6MM
BAKKER,JOACHIM            2 A0007100111762 4MS
BERNARD,JOHN              2 A0006250115771 9MS
BOER,GEORGE               2 A0007951223742 8MS
BROWN,JAMES               2 A0007000911762 3MS
CARLSON,JACK              8 E0019750830743 6MD
CHANG,ROBERT              2 A0006840201762 5MS
COHEN,JOSEPH              2 A0008750404743 5MD
DAVIS,ROBERT              2 E0010750103733 5MS
DUBOIS,ANDRE              2 A0006250315782 1MS
DURAND,HELEN              6 A0016350901683 6FD
FISCHER,DAVID             2 A0006250601782 1MS
GARCIA,ARTHUR             2 B0007380229763 4MM
GOMEZ,LINDA               2 C0006350421772 1FS
IVANOV,LEONARD            2 A0007230815763 2MM
JOHNSON,ANNABELLE         6 C0014720403692 8FM

                    .
                    .
                    .


SMITH,ROBERTA             4 A0007501122752 7FS
SOKOL,DONALD              8 A0021081207684 6MM
TAYLOR,JENNIFER           2 A0006251011772 2FS
WANG,LISA                 2 D0005851127772 3FS
WILLIAMS,BENEDICT         6 A0017210816723 9MD
WILLIAMS,ROBERT           2 A0007900315763 1MS
WILSON,DOUGLAS            2 E0009101205733 6MD

-----------------------

 ACLYAFA. 79/03/29.(22) SVL SN614 NOS

14.06.45.EXNRC.
14.06.45.UCCR, 6125,       0.052KCDS.
14.06.45.USER (usernum,passwrd)
14.06.45.CHARGE (accounting information)
14.06.45.FILE(NEW,BT=C,RT=Z,FL=80)
14.06.47.SORTMRG.
14.06.51.***KEY COMPARISON USED
14.06.52.  ** INSERTIONS DURING INPUT    *********0
14.06.52.  ** DELETIONS DURING INPUT     *********0
14.06.52.  ** TOTAL RECORDS SORTED       ********40
14.06.52.  ** INSERTIONS DURING OUTPUT   *********0
14.06.52.  ** DELETIONS DURING OUTPUT    *********0
14.06.52.  ** TOTAL RECORDS OUTPUT       ********40
14.06.52.  ** MERGE ORDER USED           ********12
14.06.53.  **END SORT RUN
14.06.53.REWIND,NEW.
14.06.53.COPYSBF,NEW,OUTPUT.
14.06.53.  END OF INFORMATION ENCOUNTERED.
14.06.53.UEAD,        0.002KUNS.
14.06.53.UEPF,        0.005KUNS.
14.06.53.UEMS,        0.564KUNS.
14.06.53.UECP,        0.367SECS.
14.06.53.AESR,        2.634UNTS.
14.07.05.UCLP, 6122,       0.256KLNS.
```

Figure 4-5. Sort Output by Name

```
EXNRC.
USER(usernum,passwrd)
CHARGE(accounting information)
FILE(NEW,BT=C,RT=Z,FL=80)
SORTMRG.
REWIND,NEW.
COPYSBF,NEW,OUTPUT.
SAVE,NEW.
7/8/9 multipunched in column 1

  sort directives

7/8/9 multipunched in column 1

  input records

6/7/8/9 multipunched in column 1
```

Figure 4-6. Creating a NOS Permanent File

Users of NOS permanent files will encounter both direct and indirect access permanent files. Their use is quite different. For purposes of these examples, only indirect access permanent files are used. Large files might be better served by using direct access permanent files. The choice of type of NOS permanent file is described in the NOS Time-Sharing user's guide.

By saving your input file as a permanent file, you can access it time after time with a single control statement. In this instance, it is very simple to input the card deck created for practice examples (shown in figure 4-2), make the file permanent with the SAVE,NEW control statement, and refer to the permanent file in subsequent jobs by using the GET,NEW control statement. You should purge the permanent file when you no longer need it.

Figures 4-7 and 4-8 are quite similar to the examples given in section 4 of the Sort/Merge reference manual. Figure 4-7 illustrates a sort of the practice file based on department. All of the names are sorted alphabetically; identical names are further sorted on salary.

Some sort directives are arbitrarily continued on the next line to fit the format of the example. These directives are normally up to 72 characters in length.

Figure 4-8 illustrates the ease with which you can alter the order of the departments by specifying your own collating sequence for the sorting of the departments.

One method of creating a seniority list of employees is illustrated in figure 4-9. Assuming this list will be used to determine the order of employee layoffs should the need arise, the last hired should be at the top of the list. The major sort key is the employee start date, beginning with the most recent. The secondary key is set by age, starting with the youngest. Thus, if two or more employees started on the same date, those who are younger will be listed first.

The most notable aspect of this example is the manner in which the sort on the start date should be specified. The date consists of three separate parts. The most important part is the year, second is the month, and last is the day of the month. This sort format must be used to correctly sort the dates. The date is divided into three keys in this example. Each part is sorted as a separate field. This is not really necessary; the date could be sorted on two keys,

```
1  SORT
2  FILE,INPUT=INPUT,OUTPUT=NEW
3  FIELD,NAME(1,23,DISPLAY),DEPT(26,1,
4  ,DISPLAY),SALARY(27,6,DISPLAY)
5  KEY,DEPT(A,DISPLAY),NAME(A,DISPLAY),
6  ,SALARY(D,DISPLAY)
7  END
```

```
ANDERSON,TIMOTHY          2  A0L07000525752GMM
BAKKER,JOACHIM            2  A0J071001117624MS
BERNARD,JOHN              2  A0G062501157719MS
BOER,GEORGE               2  A0C0795122374Z8MS
BROWN,JAMES               2  A0C07000911762JMS
CHANG,ROBERT              2  A0G06840201762SMS
COHEN,JOSEPH              2  A0G087504047435MD
DUBOIS,ANDRE              2  A0006250315782IMS
DURAND,HELEN              6  A0C16350901683GFD
FISCHER,DAVID             2  A0006256017821MS
IVANOV,LEONARD            2  A0007230815763ZMM
KIM,LEE                   2  A0C061011227719FS
LI,WANG                   2  A0L071001107731MS
LOPEZ,COSME               2  A0006700307762ZMM
MARTIN,RICHARD            4  A0J092112157338MM
MEYER,WILLIAM             4  A0008750301763ZMS
MILLER,FLORENCE           2  A0G06100503782ZFS
NEWMAN,ANDREW             2  A0J071007017724MM
PETIT,ARNOLD              2  A0006250221782GMS
SCHULZ,CHARLES            2  A0006250401781ZMS
SMITH,JOHN                2  A0C08100918734ZMM
SMITH,ROBERTA             4  A0C075011227527FS
SOKOL,DONALD              8  A0G21081207626ZMM
TAYLOR,JENNIFER           2  A0C062510117722FS
WILLIAMS,BENEDICT         6  A0017210816723ZMD
WILLIAMS,ROBERT           2  A0G07900315763IMS
GARCIA,ARTHUR             2  B0C07380229763ZMM
JONES,FRANCES             2  B0007581103742ZFS
MULDER,HENK               4  B0G09110401744IMM
GOMEZ,LINDA               2  C0G06350421772IFS
JOHNSON,ANNABELLE         5  C0G14720403692ZFM
PETROV,GEORGE             2  C0J075006307629MD
JONES,CHERY_              2  D0G06850827762IFM
WANG,LISA                 2  D0005851127772JFS
CARLSON,JACK              8  E0J19750830743GMD
DAVIS,ROBERT              2  E0G107501037335MS
JOHNSON,ARMAND            4  E0C141010207232MS
POPOV,IVAN                2  E0C066507097723MS
SMITH,MARGERY             4  E0G113502157528FS
WILSON,DOUG_AS            2  E0009101205733GMD
```

Figure 4-7. Sort by Department, Name, and Salary

the first on the year and the second on the month and day together as a four-digit field.

Dates specified in the form mmddyy require specification of at least two fields, yy and mmdd. If the dates entered into the file were formatted as yymmdd, only one sort key field would be needed.

You might wish to note that another example, illustrated in figure 4-10, shows that specifying the FIELD as DISPLAY code and the KEY as INTBCD results in no significant change to the alphabetic order of employee names. This is because the alphabet runs in the same order in both character sets and collating sequences. Such specification on a sort key field which contained letters, digits, and special characters would emphasize the differences between the DISPLAY and INTBCD collating sequences. The collating sequences are given in appendix A.

```
1  SORT
2  FILE,INPUT=NEW,OUTPUT=NEW
3  FIELD,DEPT(26,1,DISPLAY),SALARY(27,6,
4  ,DISPLAY),AGE(39,2,DISPLAY)
5  KEY,DEPT(A,OWN),SALARY(D,DISPLAY),
6  ,AGE(D,DISPLAY)
7  SEQUENCE,OWN(E,C,O,B,A)
8  END
```

```
CARLSON,JACK                8 E00197508307436MD
JOHNSON,ARMAND              4 E00141010207232MS
SMITH,MARGERY               4 E00113502157528FS
DAVIS,ROBERT                2 E00107501037335MS
WILSON,DOUGLAS              2 E00091012057336MD
POPOV,IVAN                  2 E00066507097723MS
JOHNSON,ANNABELLE           6 C00147204036928FM
PETROV,GEORGE               2 C00075006307629MD
GOMEZ,LINDA                 2 C00063504217721FS
JONES,CHERYL                2 D00068508277624FM
WANG,LISA                   2 D00058511277723FS
MULDER,HENK                 4 B00091104017441MM
JONES,FRANCES               2 B00075811037427FS
GARCIA,ARTHUR               2 B00073802297634MM
SOKOL,DONALD                8 A00210812076846MM
WILLIAMS,BENEDICT           6 A00172108167239MD
DURAND,HELEN                6 A00163509016836FD
MARTIN,RICHARD              4 A00092112157338MM
COHEN,JOSEPH                2 A00087504047435MD
MEYER,WILLIAM               4 A00087503017632MS
SMITH,JOHN                  2 A00081009187348MM
BOER,GEORGE                 2 A00079512237428MS
WILLIAMS,ROBERT             2 A00079003157631MS
SMITH,ROBERTA               4 A00075011227527FS
IVANOV,LEONARD              2 A00072308157632MM
LI,WANG                     2 A00071001107731MS
NEWMAN,ANDREW               2 A00071007017724MM
BAKKER,JOACHIM              2 A00071001117624MS
ANDERSON,TIMOTHY            2 A00070005257526MM
BROWN,JAMES                 2 A00070009117623MS
CHANG,ROBERT                2 A00068402017625MS
LOPEZ,COSME                 2 A00067003077622MM
TAYLOR,JENNIFER             2 A00062510117722FS
DUBOIS,ANDRE                2 A00062503157821MS
FISCHER,DAVID               2 A00062506017821MS
PETIT,ARNOLD                2 A00062502217820MS
BERNARD,JOHN                2 A00062501157719MS
SCHULZ,CHARLES              2 A00062504017818MS
MILLER,FLORENCE             2 A00061005037822FS
KIM,LEE                     2 A00061011227719FS
```

Figure 4-8. User Sequence Sort by Department, Salary, and Age

```
 1  SORT
 2  FILE,INPUT=NEW,OUTPUT=NEW
 3  FIELD,START1(37,2,DISPLAY),
 4  ,START2(33,2,DISPLAY),
 5  ,START3(35,2,DISPLAY),
 6  ,AGE(39,2,DISPLAY)
 7  KEY,START1(D,DISPLAY),
 8  ,START2(D,DISPLAY),START3(D,DISPLAY),
 9  ,AGE(A,DISPLAY)
10  END
```

```
FISCHER,DAVID               2 A00062506017821MS
MILLER,FLORENCE             2 A00061005037822FS
SCHULZ,CHARLES              2 A00062504017818MS
DUBOIS,ANDRE                2 A00062503157821MS
PETIT,ARNOLD                2 A00062502217820MS
WANG,LISA                   2 D00058511277723FS
KIM,LEE                     2 A00061011227719FS
TAYLOR,JENNIFER             2 A00062510117722FS
POPOV,IVAN                  2 E00066507097723MS
NEWMAN,ANDREW               2 A00071007017724MM
GOMEZ,LINDA                 2 C00063504217721FS
BERNARD,JOHN                2 A00062501157719MS
LI,WANG                     2 A00071001107731MS
BROWN,JAMES                 2 A00070009117623MS
JONES,CHERYL                2 D00068508277624FM
IVANOV,LEONARD              2 A00072308157632MM
PETROV,GEORGE               2 C00075006307629MD
WILLIAMS,ROBERT             2 A00079003157631MS
LOPEZ,COSME                 2 A00067003077622MM
MEYER,WILLIAM               4 A00087503017632MS
GARCIA,ARTHUR               2 B00073802297634MM
CHANG,ROBERT                2 A00068402017625MS
BAKKER,JOACHIM              2 A00071001117624MS
SMITH,ROBERTA               4 A00075011227527FS
ANDERSON,TIMOTHY            2 A00070005257526MM
SMITH,MARGERY               4 E00113502157528FS
BOER,GEORGE                 2 A00079512237428MS
JONES,FRANCES               2 B00075811037427FS
CARLSON,JACK                8 E00197508307436MD
COHEN,JOSEPH                2 A00087504047435MD
MULDER,HENK                 4 B00091104017441MM
MARTIN,RICHARD              4 A00092112157338MM
WILSON,DOUGLAS              2 E00091012057336MD
SMITH,JOHN                  2 A00081009187348MM
DAVIS,ROBERT                2 E00107501037335MS
JOHNSON,ARMAND              4 E00141010207232MS
WILLIAMS,BENEDICT           6 A00172108167239MD
JOHNSON,ANNABELLE           6 C00147204036928FM
SOKOL,DONALD                8 A00210812076846MM
DURAND,HELEN                6 A00163509016836FD
```

Figure 4-9. Sort for Seniority List

COMBINING DISSIMILAR FILES USING FORM

One basic premise of sort files is that the key fields be identical in length and starting position. A sort can only be specified on these parameters.

In order to combine files, it is necessary that at least one record field in each file match. If for example, you wish to combine two large sorted files, such as university student grade files, and different fields have been used for similar information, you can use FORM to modify the records in one of the files to match the records in the other file. The FORM reference manual gives an example of such use.

In the following example, figure 4-11, the output file which has been sorted on the name field only will be reformatted using FORM to illustrate the report formatting capabilities of FORM. The FORM directives used are shown with the formatted output. FORM can be used to expand the records by placing a row of blanks between the key fields as shown here, or it can be used to change the layout of the keyfields within each record so as to match the format of other records if you wish to merge files which are composed of identical key fields but in different formats.

For some statistical reports only a small portion of a large record needs to be extracted for sorting in order to

```
1  SORT
2  FILE,INPUT=NEW,OUTPUT=MINE
3  FIELD,NAME(1,23,DISPLAY)
4  KEY,NAME(A,INTBCD)
5  END
```

```
ANDERSON,TIMOTHY          2 A00070005257526MM
BAKKER,JOACHIM            2 A00071001117624MS
BERNARD,JOHN              2 A00062501157719MS
BOER,GEORGE               2 A00079512237428MS
BROWN,JAMES               2 A00070009117623MS
CARLSON,JACK              8 E00197508307436MD
CHANG,ROBERT              2 A00068402017625MS
COHEN,JOSEPH              2 A00087504047435MD
DAVIS,ROBERT              2 E00107501037335MS
DUBOIS,ANDRE              2 A00062503157821MS
DURAND,HELEN              6 A00163509016836FD
FISCHER,DAVID             2 A00062506017821MS
GARCIA,ARTHUR             2 B00073802297634MM
GOMEZ,LINDA               2 C00063504217721FS
IVANOV,LEONARD            2 A00072308157632MM
JOHNSON,ANNABELLE         6 C00147204036928FM
JOHNSON,ARMAND            4 E00141010207232MS
JONES,CHERYL              2 D00068508277624FM
JONES,FRANCES             2 B00075811037427FS
KIM,LEE                   2 A00061011227719FS
LI,WANG                   2 A00071001107731MS
LOPEZ,COSME               2 A00067003077622MM
MARTIN,RICHARD            4 A00092112157338MM
MEYER,WILLIAM             4 A00087503017632MS
MILLER,FLORENCE           2 A00061005037822FS
MULDER,HENK               4 B00091104017441MM
NEWMAN,ANDREW             2 A00071007017724MM
PETIT,ARNOLD              2 A00062502217820MS
PETROV,GEORGE             2 C00075006307629MD
POPOV,IVAN                2 E00066507097723MS
SCHULZ,CHARLES            2 A00062504017818MS
SMITH,JOHN                2 A00081009187348MM
SMITH,MARGERY             4 E00113502157528FS
SMITH,ROBERTA             4 A00075011227527FS
SOKOL,DONALD              8 A00210812076846MM
TAYLOR,JENNIFER           2 A00062510117722FS
WANG,LISA                 2 D00058511277723FS
WILLIAMS,BENEDICT         5 A00172108167239MD
WILLIAMS,ROBERT           2 A00079003157631MS
WILSON,DOUGLAS            2 E00091012057336MD
```

Figure 4-10. Sort Using INTBCD Collating Sequence

produce the desired results. For example, one such case could be used where only three short fields from each very long record are to be obtained, sorted, totaled, or analyzed. This information could be arranged in order by doing a normal sort of the file and then doing further work by using another program thus ensuring the desired order of the output. Such a sort alone would require a large amount of time if the file itself were large. Another method is available which could be used to avoid the handling of such lengthy records.

Such might be the case where a student name, student number, and grade point average are to be sorted based on grade point above a certain level to determine the dean's list, and below a certain level to determine first those on probation and second, those to be dropped because of low grades. If the list is to be prepared from the master file of all student records, such records could easily be quite large.

```
INP(NEW)

OUT(FORMAT,3GO=X)

REF(FORMAT,X2_,X4=24X3,N4=27N6,$ $,

N6=33N6,$ $,X3=39X2,X2=41X1,X2=42X1)


SUMMARY

 INPUT FILE - NEW        40 RECORDS READ

OUTPUT FILE - FORMAT     40 RECORDS WRITTEN

END OF RUN.
```

```
ANDERSON,TIMOTHY          2 A 0700 052575 26 M M
BAKKER,JOACHIM            2 A 0710 011176 24 M S
BERNARD,JOHN              2 A 0625 011577 19 M S
BOER,GEORGE               2 A 0795 122374 28 M S
BROWN,JAMES               2 A 0700 091176 23 M S
CHANG,ROBERT              2 A 0684 020176 25 M S
COHEN,JOSEPH              2 A 0875 040474 35 M D
DUBOIS,ANDRE              2 A 0625 031578 21 M S
DURAND,HELEN              6 A 1635 090168 36 F D
FISCHER,DAVID             2 A 0625 060178 21 M S
IVANOV,LEONARD            2 A 0723 081576 32 M M
KIM,LEE                   2 A 0610 112277 19 F S
LI,WANG                   2 A 0710 011077 31 M S
LOPEZ,COSME               2 A 0670 030776 22 M M
MARTIN,RICHARD            4 A 0921 121573 38 M M
MEYER,WILLIAM             4 A 0875 030176 32 M S
MILLER,FLORENCE           2 A 0610 050378 22 F S
NEWMAN,ANDREW             2 A 0710 070177 24 M M
PETIT,ARNOLD              2 A 0625 022178 20 M S
SCHULZ,CHARLES            2 A 0625 040178 18 M S
SMITH,JOHN                2 A 0810 091873 48 M M
SMITH,ROBERTA             4 A 0750 112275 27 F S
SOKOL,DONALD              8 A 2108 120768 46 M M
TAYLOR,JENNIFER           2 A 0625 101177 22 F S
WILLIAMS,BENEDICT         6 A 1721 081672 39 M D
WILLIAMS,ROBERT           2 A 0790 031576 31 M S
GARCIA,ARTHUR             2 B 0738 022976 34 M M
JONES,FRANCES             2 B 0758 110374 27 F S
MULDER,HENK               4 B 0911 040174 41 M M
GOMEZ,LINDA               2 C 0635 042177 21 F S
JOHNSON,ANNABELLE         6 C 1472 040369 28 F M
PETROV,GEORGE             2 C 0750 060376 29 M D
JONES,CHERYL              2 D 0685 082776 24 F M
WANG,LISA                 2 D 0585 112777 23 F S
CARLSON,JACK              8 E 1970 083074 36 M D
DAVIS,ROBERT              2 E 1075 010373 35 M S
JOHNSON,ARMAND            4 E 1410 102072 32 M S
POPOV,IVAN                2 E 0665 070977 23 M S
SMITH,MARGERY             4 E 1135 021575 28 F S
WILSON,DOUGLAS            2 E 0910 120573 36 M D
```

Figure 4-11. Reformatting Records Using FORM

The easiest method would probably be to use FORM to extract the desired fields (such as student name, student number, and grades) from the master file to create a shorter sort file. Then a sort of the new file could be used to order all other records ranked by grade level, using an owncode exit to discard all of the passing records of students who had not made the dean's list.

The sorted file output in order of top grades could be resorted for an alphabetical dean's list, and the lower grades resorted again in student number order for the mailing of required probation notices. Other examples of FORM usage are illustrated in the FORM reference manual.

This section will be of interest chiefly to experienced programmers. Sort/Merge provides a record handling capability which allows you to supply your own code for a specific purpose. There are six exits available to gain control of the sort or merge in progress to enter a routine you have written. No owncode routines are supplied by the Sort/Merge program. When your owncode routine has achieved its purpose, a jump is made to Sort/Merge processing at the exit point according to the address in your entry point.

The first part of this section requires your ability to write programs in the COMPASS assembly language. Although owncode routines are not required for Sort/Merge execution, they provide the capability for you to insert, substitute, modify, or delete input and output records.

Users of owncode when using the FORTRAN calls described later in this section can write their own subroutines in FORTRAN. COBOL users are not provided owncode exits. They can, however, use the INPUT PROCEDURE and the OUTPUT PROCEDURE to achieve similar results without leaving the COBOL program.

The Sort/Merge program provides a highly efficient sorting and merging capability through the standard Sort/Merge parameters. In order to maintain this level of efficiency, the standard parameters limit the operations that can be specified. Additional operations that you might wish to perform on the records or files probably can be done in conjunction with the sort or merge owncode exits to access the record or file at the opportune point in processing.

The reasons for wanting to use owncode can be as varied as the owncode exits will allow. Many of the problems you might encounter in attempting to sort a file are alleviated by proper use of the sort directives. Specification of a different collating sequence or character set, or changing the sequence and/or equating characters to one another will solve many problems. Most of the remaining problems which cannot be addressed by the directives can be resolved through use of the owncode exits. These allow you to exit sort or merge at an appropriate point to enter an owncode subroutine in COMPASS, or to enter a FORTRAN owncode subroutine through a FORTRAN call.

The owncode exits do not supply the code to achieve a specific purpose. You must supply your own subroutine to be used when an owncode exit is taken. Sort/Merge will deliver the record at the point in processing specified by the owncode exit number.

COMPASS OWNCODE

All COMPASS owncode subroutines must be assembled in relocatable binary form prior to the Sort/Merge run and placed in the input deck or made available from an alternate source as specified by one of the SORTMRG control statement parameter options.

Sort/Merge 4 provides six owncode exits that can be used during processing. Sort/Merge 1 provides five owncode exits. Each of the exits is numbered. Each exit serves a specific function. Each exit is available after a specific point in processing.

Owncode exits allow the examination and changing of information at four critical points during the sort process, and, in addition, allow handling of records with equal keys. Only Sort/Merge 4 allows processing of nonstandard labels.

OWNCODE EXITS 1 THROUGH 4

Owncode exits 1, 2, 3, and 4 allow you to insert, delete, or change records. Exits 5 and 6 have special uses which are explained later.

Operations include the use of exit 1 to edit input records to make sorting easier; or exit 2 to add records prior to the sort process.

After the sort, you can again change or add records through use of exits 3 or 4.

Example

An important use of owncode exit 1 can be illustrated in the solution of a problem often encountered in sort processing. Numeric information is often prepared without leading zeros in the sort key fields. Because leading blanks collate differently than leading zeros, the sorted output will not be in correct order. As mentioned previously, the zero and the blank can be equated (by directive) in many cases to avoid this problem. However, in some cases, such as with signed numeric data, where the position of the sign is not fixed, you will have to resort to the owncode exit 1 capability to change the leading blanks to zeros in order to fix the sign position to ensure that the records will collate correctly.

The owncode example given in figure 5-1 shows one method of solving the above problem using owncode exit 1 and a COMPASS subroutine. A number of various methods can be used to reach the same goal; the method shown here is one of the more simple means of converting leading blanks to leading zeros and fixing the position of the sign.

The sorting of dates is often more complicated than expected as you might have noted in figure 4-9. Dates are more easily sorted when their format is changed to yymmdd so as to allow a sort on only one key field. This format is not a common one, though it can easily be achieved through use of an owncode table look-up routine.

Imagine trying to sort an existing file on a date field created using letters and digits such as JUL041776. Such dates can be changed (through use of a table look-up procedure with owncode exit 1) to 17760704.

```
SAMPL,T10,P3.
COMPASS(S=0)
FILE(OUT,BT=C,RT=Z,FL=80)
SORTMRG(OWN)
```

```
          IDENT  OWN
COL       EQU    3
LEN       EQU    9

          IFLT   COL,1,1
          ERR    COL MUST BE AT LEAST 1
          IFLT   LEN,1,1
          ERR    LEN MUST BE AT LEAST 1
          IFGT   LEN,10,1
          ERR    LEN MUST BE LESS THAN 11


**        OWN1 -  OWNCODE EXIT 1 PROCESSOR
*     CALLING SEQUENCE-
*         A2 = ADDRESS OF RECORD
*         X0 = 30/NWORDS, 30/NCHARACTERS
*     DOES-
*         CONVERTS THE FIELD STARTING IN COLUMN <COL> OF LENGTH <LEN>
*         FROM FORTRAN FORMATTED INTEGERS (I.E. LEADING BLANKS, OPTIONAL
*         NEGATIVE SIGN, DIGITS) TO A LOGICAL VALUE THAT
*         PRESERVES NUMERIC ORDER

*         X0 = 30/NWORDS, 30/NCHARACTERS
*         X1 = 7777777777777777700B
*         X2 = RECORD TO RIGHT OF FIELD
*         X3 = SIGN  (ALL ZEROS MEANS +, ALL ONES MEANS -)
*         X4 = WORD WITH CURRENT CHARACTER
*         X5 = CURRENT CHARACTER
*         X6 = CURRENT BINARY VALUE
*         X7 = CURRENT BINARY DIGIT
*         A2 = ADDRESS OF FIRST WORD OF RECORD
*         B5 = CURRENT CHARACTER
*         B6 = SCRATCH
*         B7 = NUMBER OF CHARACTERS LEFT IN FIELD

          ENTRY  OWN1
OWN1      BSS    1
                                COL = 1 2 ... 10 11 12 ...
T         SET    COL-1                0 1 ...  9 10 11 ...
W         SET    T/10                 0 0 ...  0  1  1 ...
C         SET    COL-1-10*W           0 1 ...  9  0  1 ...
          SA4    A2+W           GET WORD WITH FIRST CHARACTER
          SA5    A2+W+1         GET NEXT WORD
          MX3    6*C            ALIGN MASK AT START OF FIELD
          BX4    -X3*X4         EXTRACT (FIRST PART OF) FIELD
          BX5    X3*X5          EXTRACT SECOND PART OF FIELD (MAYBE)
          BX4    X4+X5.         COMBINE
          LX4    6*C            LEFT-JUSTIFY FIELD
          MX2    6*LEN
          BX2    -X2*X4         SAVE RECORD TO RIGHT OF FIELD
          MX1    -6             7777777777777777700B
          BX6    X0-X0          CLEAR RESULT REGISTER
          MX3    0              GUESS THAT SIGN IS POSITIVE
          SB7    LEN            SET NUMBER OF CHARACTERS LEFT

OWN11     ZR     B7,OWN14       IF NO MORE CHARACTERS IN FIELD
          LX4    6              RIGHT-JUSTIFY NEXT CHARACTER
          BX5    -X1*X4         EXTRACT NEXT CHARACTER
          SB5    X5
          SB7    B7-1           DECREMENT COUNT OF CHARACTERS LEFT
          SB6    1R
          EQ     B5,B6,OWN11    IF BLANK, LOOP
          SB6    1R-
          NE     B5,B6,OWN13    IF NOT -, SKIP
          MX3    60             NOTE NEGATIVE SIGN
```

Figure 5-1. COMPASS Owncode Example to Convert Leading Blanks to Zeros in Signed Numeric Data (Sheet 1 of 3)

```
OWN12   ZR      B7,OWN14        IF END OF FIELD, JUMP
        LX4     6               RIGHT-JUSTIFY NEXT CHARACTER
        BX5     -X1*X4          EXTRACT NEXT CHARACTER
        SB5     X5
        SB7     B7-1            DECREMENT COUNT OF CHARACTERS LEFT
OWN13   SB6     1R0
        LT      B5,B6,ERROR     IF CHARACTER LESS THAN "0", ERROR
        SB6     1R9
        GT      B5,B6,ERROR     IF CHARACTER GREATER THAN "9", ERROR
        SX7     10
        IX6     X6*X7
        SX7     B5-1R0
        IX6     X6+X7           RESULT = 10*RESULT + DIGIT
        EQ      OWN12           GO TRY FOR NEXT CHARACTER
OWN14   BSS     0               (HERE AT END OF FIELD)
        BX6     X3-X6           APPLY SIGN TO RESULT
        LX6     60-6*LEN        LEFT-JUSTIFY RESULT
        MX3     1
        BX6     X3-X6           TOGGLE SIGN BIT
        MX3     6*LEN
        BX6     X3*X6           STRIP EXCESS ONES (IF NEGATIVE)
        BX6     X6+X2           APPEND RECORD TO IMMEDIATE RIGHT OF FIELD
        LX6     -6*C            ALIGN FIELD
        SA4     A2+W            GET WORD FOR FIRST CHARACTER
        MX3     6*C             MASK ALIGNED AT START OF FIELD
        BX4     X3*X4           DELETE ORIGINAL FIELD
        BX2     -X3*X6          INSERT CREATED RESULT
        BX7     X4+X2           COMBINE
        SA7     A4              STORE BACK
        SA5     A2+W+1          GET NEXT WORD, JUST IN CASE
        BX5     -X3*X5
        BX2     X3*X6
        BX7     X2+X5
        SA7     A5
        EQ      OWN1            GO BACK WITH RECORD
ERROR   SA1     OWN1
        LX1     30
        SB7     X1
        JP      B7+1
        EJECT
**      OWN3 -  OWNCODE EXIT 3 PROCESSOR
*
*     CALLING SEQUENCE-
*         A2 = ADDRESS OF RECORD
*         X0 = 30/NWORDS, 30/NCHARACTERS
*     DOES-
*         CONVERTS THE FIELD STARTING IN COLUMN <COL> OF LENGTH <LEN>
*         FROM THE FORMAT CREATED BY OWN1 TO FORTRAN FORMATTED INTEGERS
*         (I.E. LEADING BLANKS, OPTIONAL NEGATIVE SIGN, DIGITS).
*
*         X0 = 30/NWORDS, 30/NCHARACTERS
*         X1 = 0.10000000001
*         X2 = RECORD TO IMMEDIATE RIGHT OF FIELD
*         X3 = SIGN
*         X4 = NUMBER AT START OF LOOP
*         X5 = NEXT NUMBER
*         X6 = CHARACTERS
*         X7 = RIGHT-JUSTIFIED CHARACTER
*         A2 = ADDRESS OF FIRST WORD OF RECORD
*         B7 = NUMBER OF CHARACTERS LEFT IN FIELD

        ENTRY   OWN3
OWN3    BSS     1
T       SET     COL-1
W       SET     T/10
C       SET     COL-1-10*W
        SA4     A2+W            GET WORD WITH FIRST CHARACTER
        SA5     A2+W+1          GET NEXT WORD, JUST IN CASE
        MX3     6*C             MASK STARTING AT FIELD
        BX4     -X3*X4          DELETE BEFORE FIELD
        BX5     X3*X5           DELETE AFTER FIELD+9
```

Figure 5-1. COMPASS Owncode Example to Convert Leading Blanks to Zeros in Signed Numeric Data (Sheet 2 of 3)

```
          BX4     X4+X5           COMBINE
          LX4     6*C             LEFT-JUSTIFY FIELD
          MX2     6*LEN
          BX2     -X2*X4          SAVE RECORD TO IMMEDIATE RIGHT OF FIELD
          MX3     1               MASK FOR SIGN BIAS
          BX3     -X4*X3          1 IFF NEGATIVE
          AX3     59              777777777777777777B IFF NEGATIVE
          LX4     1               SHIFT TO EXTENDABLE SIGN
          AX4     61-6*LEN        EXTEND TO RIGHT
          BX4     X4-X3           TAKE ABSOLUTE VALUE OF NUMBER
          SB7     LEN             NUMBER OF CHARACTERS LEFT TO SET UP
          MX6     0               CLEAR RESULT REGISTER
          SA1     =0.10000000001

OWN31     PX5     X4
          NX5                     (E.G.)  123456789.0
          FX5     X5*X1           (E.G.)  12345678.9012
          UX5     X5,B5
          LX5     X5,B5           (E.G.)  12345678
          IX7     X5+X5           (E.G.)  12345678*2
          LX5     3               (E.G.)  12345678*8
          IX7     X5+X7           (E.G.)  123456780
          AX5     3               (E.G.)  12345678
          IX7     X4-X7           (E.G.)  9
          SX7     X7+1R0          CONVERT TO DIGIT CHARACTER
          BX6     X6+X7           INSERT INTO RESULT
          LX6     -6              MAKE ROOM FOR NEXT CHARACTER
          SB7     B7-1            DECREMENT NUMBER OF CHARACTERS LEFT TO SET
          BX4     X5              (E.G.)  12345678
          NZ      X4,OWN31        IF MORE DIGITS, LOOP

          PL      X3,OWN32        IF POSITIVE, SKIP
          SX7     1R-             INSERT NEGATIVE SIGN CHARACTER
          BX6     X6+X7
          LX6     -6
          SB7     B7-1

OWN32     ZR      B7,OWN33        IF FIELD ALREADY FULL, SKIP
          SX7     1R              INSERT A BLANK CHARACTER
          BX6     X6+X7
          LX6     -6
          SB7     B7-1
          EQ      OWN32           GO TRY AGAIN

OWN33     BX6     X6+X2           COMBINE LEFT-JUSTIFIED CHARACTERS AND
                                          RECORD TO IMMEDIATE RIGHT OF FIELD
          LX6     -6*C            ALIGN FIELD WITH RECORD
          SA4     A2+W            GET WORD FOR FIRST CHARACTER
          MX3     6*C             MASK ALIGNED AT START OF FIELD
          BX4     X3*X4           DELETE CREATED FIELD
          BX2     -X3*X6          INSERT FORTRAN CHARACTERS
          BX7     X4+X2           COMBINE
          SA7     A4              STORE
          SA5     A2+W+1          GET NEXT WORD, JUST IN CASE
          BX5     -X3*X5
          BX2     X3*X6
          BX7     X2+X5
          SA7     A5
          EQ      OWN3            EXIT


          END
SORT
FILES,SORT=INPUT,OUTPUT=OUT
FIELD,K(3,9,LOGICAL)
KEY,K
OWNCODE,1=OWN1,3=OWN3
END
   (Input Records here)
```

Figure 5-1. COMPASS Owncode Example to Convert Leading Blanks to Zeros in Signed Numeric Data (Sheet 3 of 3)

After sorting all of the records on the dates (as changed for this sort run through use of owncode exit 1), an exit 3 table look-up owncode routine could be used to change the format of the output date to July 4, 1776. Even though this use of owncode appears complicated, it requires far less processing time than a separate subroutine would require to do a similar translation.

In most cases, files containing records modified through use of owncode to simplify or speed the sort process are not modified back to their original state; they are usually left in their more easy to sort format to avoid duplicate processing each time they are used. To illustrate, the date that was originally stored as JUL041776 was changed to 17760704 for sorting purposes and the file was stored in this format. When this file is updated, any new input should first be sorted using the same sort routine with its owncode table look up code before merging the new input with the existing file. Output returned to its initial state through use of owncode exit 3 does not change the format of the records stored on the sort input file.

Another variation on this change is even easier to use. When the date is converted from the original format by means of a table look-up procedure and owncode exit 1, the new format is stored as part of the same record. In this manner, you can sort the record based on the easily sorted format and output the more easily recognized format. Your record would change from JUL041776 to JUL0417760704 as a result of the owncode exit 1. Thus, you would sort this field on the last eight characters and output the date as the first nine characters.

Variations of this concept allow you to sort any type of information you might define because you can transform any data into a sortable field tag and append the tag to your original record.

Exits 1 and 2 owncode routines are not allowed in a merge-only run to ensure that the input records remain in sorted order. You can achieve the same purpose by using either exits 3 and 4 in a merge-only run or exits 1 and 2 in a sort run with supplementary merge files. If you specify an exit 1 or 2 owncode routine in a merge-only run, the exit is ignored and a nonfatal diagnostic message occurs.

OWNCODE EXIT 5

One typical use for the owncode routines is to handle duplicate or equal keys. Equal keys can be duplicate keys, or a key comprised of characters treated the same (as the result of an EQUATE directive), or created equal because of a signed numeric overpunch.

When equal sort keys are encountered in two or more records, the method of handling such records often needs to be defined. Just because the sort keys are equal does not mean that the records are identical. In many cases, you will want to be able to control the order of records with equal keys. One of the controls available to you is to output the records containing equal keys in the order they were input. This control is available by specifying RETAIN on the Sort/Merge OPTIONS directive. Owncode is not required in such a case.

Using owncode exit 5 will allow you to stop and compare records with equal keys. If the records are exact duplicates, you might wish to delete one record. In the case where two identical orders were booked in one day from one company, you would most likely wish to flag the records to remind you to check the validity of both orders. Owncode exit 5 allows you to modify or replace either or both records or to retain both records without modification. It is more difficult to delete both equal records, but it can be done by having the owncode exit 5 routine signal an owncode exit 3 routine.

At times equal record keys are considered to indicate an error, such as in a file that should only contain one entry for each customer. In such files, it would be proper to delete any duplicate records. Exit 5 can be used to identify records with equal sort keys. If the record overlap is sufficient to ensure that a record is indeed a duplicate, then deletion of the duplicate record is quite simple. You merely need to provide the address and length of the record you wish to keep in registers A2 and X0, and modify the return address.

Owncode routines can also be used to write all questionable records to an exception file for separate processing. When this is done, the questionable records are usually deleted from the file being processed. (The file should be closed in the owncode exit 4 routine.) When the exception file is corrected, it can be sorted and merged with the original file.

Another use of duplicate key processing allows you to count records. If, for example, you wanted to determine the number of items of each particular size article sold during a given period, you might specify sufficient sort keys to uniquely identify the articles and sizes you wish to tally, append a count field containing 1, and then use owncode exit 5 to add the counts for each duplicate record key found in the articles sold file. (The record containing the appended count field is retained while the other record is omitted.) The sort run will then list every item sold, in the order specified, with total sales of each item.

In cases where only the items that sold more than a certain number are of interest, further owncode output specification could be used to delete all records whose numerical tally is below that level. Thus, in this manner, you could create a base for next year's reorder levels, a list of this season's most popular items, and so forth.

OWNCODE EXIT 6

Owncode exit 6 is used for checking or verifying nonstandard labels on files. Most files will not use nonstandard labels; owncode exit 6 is not needed for these files. If your CYBER Record Manager FILE control statement specifies LT=NS,ULP=NO, you will need to refer to the Basic Access Methods reference manual information on label processing. The use of the GETL and PUTL macros described there requires a knowledge of COMPASS assembly language.

Owncode exit 6 is only available to users of Sort/Merge 4; this capability is not supported for Sort/Merge 1.

HOW OWNCODE WORKS

When you specify an owncode exit in your Sort/Merge program, register A2 contains the address of the current data record and register X0 contains the record length. In addition, during entry into owncode exit 5, registers A3 and X4 are used for the address and length of the second record of a comparison involving equal sort key data.

Transfer from Sort/Merge to the owncode routines is accomplished with a return jump (RJ) instruction which fills the entry point of the owncode routine with a return to the Sort/Merge program. To return to Sort/Merge control (and leave the owncode routine), your code must

return to the entry point of the owncode routine. This is the normal return address. You can request specific processing action by altering the return address in the entry point of the owncode routine. You will usually do this by putting the normal return address in a B register, Bn, and jumping to Bn+1, Bn+2, or Bn+3. (This operation is often described in text but not in code as NR+1, NR+2, or NR+3.) When the function you have chosen is complete, a normal return address to the entry point of the owncode routine causes a jump to Sort/Merge to continue normal processing.

The specific owncode functions available to you are described in detail in the Sort/Merge reference manual. Further details of the COMPASS instructions are presented in the COMPASS reference manual.

FORTRAN CALLS

A set of library routines is provided for calling Sort/Merge from a FORTRAN program. The use of Sort/Merge in conjunction with FORTRAN provides a record and file handling capability often overlooked by the typical programmer using FORTRAN. Calls to Sort/Merge allow the FORTRAN programmer to use the Sort/Merge owncode exits to access records and files more easily than is possible directly from FORTRAN.

Moreover, calls to Sort/Merge owncode allow the FORTRAN programmer to write subroutines in FORTRAN instead of COMPASS, and use variables instead of constants as parameters.

The FORTRAN calls require that all conventions for using FORTRAN statements be observed when using these calls. The more commonly used FORTRAN calls are listed below with their corresponding Sort/Merge directives.

CALL SMSORT	SORT directive
CALL SMMERGE	MERGE directive
CALL SMFILE	FILE directive
CALL SMKEY	KEY directive
CALL SMSEQ	SEQUENCE directive
CALL SMEQU	EQUATE directive
CALL SMOPT	OPTIONS directive
CALL SMEND	END directive
CALL SMOWN	OWNCODE directive
CALL SMRTN	No corresponding directive

As with the Sort/Merge directives, the first call should be to either SMSORT or SMMERGE; the last call must be to SMEND, which initiates processing using the information collected by the other calls. The SMSORT and SMMERGE calls require that you specify the maximum record length, in characters, of the records to be sorted as the first parameter. You can also specify the number of CM words to be used for working storage if you wish, though the default of 22 000 octal words is usually sufficient.

The full list of FORTRAN calls and how to use them appears in the FORTRAN Extended reference manual and in the Sort/Merge reference manual.

The use of Sort/Merge owncode complements the capabilities of FORTRAN; their combined use provides both excellent record handling and computing power.

For example, consider a problem where a series of worldwide experiments over a number of years has resulted in the collection of voluminous amounts of data concerning the effects of temperature and pressure on radio waves. Though most of these readings were recorded in metric measurement (Celsius and millimeters of mercury), a great number were recorded in nonmetric values (Fahrenheit and inches of mercury). One advantage exists in that all records within each file are consistent; all files are either metric or nonmetric. In order to combine all of the data for input to a program written in FORTRAN, the data must first be converted to all metric equivalents and sorted into order based on temperature as the major sort key and pressure as the minor sort key.

Convert the nonmetric information in each nonmetric file to metric equivalents before combining all of the files. The most straightforward method of combining and sorting all of the data is to sort all files into order and then merge them.

The FORTRAN call to Sort/Merge owncode exit 1 is one of the best methods available for solving the problem of conversion. Record manipulation is handled entirely by Sort/Merge. The owncode routine is written in FORTRAN; each temperature and pressure value is converted to metric, the values in metric are sorted and output to an intermediate file in one operation. When all of the files to be converted are in metric, and all of the metric files are sorted, all intermediate files are merged to create the single input file for the processing program.

Such operations are not as easily managed if attempted only within FORTRAN.

The following problem illustrates that raw input is of little value before it has been organized into a more usable form. The problem is to determine which zones of a given real estate area are appreciating the fastest, the frequency of sales in each zone, and how much each zone has appreciated since last year. With thousands of real property sales each year spread over hundreds of zones, it becomes a task for the computer to determine such statistics.

To obtain a sorted comparison, all sales within each zone must be totaled and averaged. The averages for this year must be compared to the averages for last year in order to compute average appreciation. If averages are not available for last year, they too must be computed. Then the difference between this year's average and last year's average can be computed for each zone. The differences are sorted in descending order by zone, along with other desired information, to create a usable report.

One method of creating such a report is to use FORTRAN to create records that contain the desired information, sort these records based on the percentage of appreciation, add report headers through use of owncode, and output a formatted report.

The sort specified would probably be made through a FORTRAN call to the sort routine, and the owncode exits 3 and 4 header additions would naturally follow.

UNIQUE USES OF OWNCODE

One unique use of the Sort/Merge owncode routines is to sort text entries for an index or for a glossary. For an index, it is possible to create a row of leading dots the length of the index line, and overlay the initial portion of the line with the entry and the final portion of the line with the page reference to create a typical entry that looks like this:

Symbol generator 42

These lines can easily be sorted to create the index.

For a glossary, you might consider a standard entry which starts with the term to be defined, followed by a blank line and then the descriptive text. The final output might look like:

> DIRECTIVES-
> Instructions that supplement processing defined by the SORTMRG control statement for execution of Sort/Merge record processing.

To create a glossary that can be sorted, you will need to create each glossary entry as a single continuous record. Owncode exit 3 is then used, following the sort of the records, to format the glossary entries.

Other uses which are often overlooked include preprocessing of input such as stripping away headings in a report file for subsequent sorting, pagination of reports, combination of entries, deletion of duplicate entries, reformatting reports, adding headings, creating footnotes, controlling page depth, blanking certain fields in reports, and so on.

As a final note, if you have part of your information already sorted, this can sometimes be made to work in your favor. For example, when creating an index, page number references will be in page number order in your intermediate file. Thus, on output of multiple page references, you can first check to delete duplicates, and the remaining references will remain in numerical order if you have specified the RETAIN option throughout the run.

RECORD COMPACTION

Many variations of compaction are possible. Using owncode routines or FORM, you can extract the key fields of interest from all the records in a file, and sort only the new records. For example, you might wish to extract only the age and salary fields from a payroll file to create an age/salary profile chart.

You would probably find a number of identical short records. Rather than keep each record separately, it could be worthwhile to keep only one copy of each identical record, with a count field appended to the record to keep track of the number of times an identical record is encountered. In this manner, the identical records can be deleted, thus shortening the total length of the file. Owncode exit 5 makes this process simple because the exit is available on every duplicate key encountered. On creation of the profile chart, the amplitude of the identical entries will have to be expanded by the number of identical occurrences recorded in the appended count field.

It might occur, for example, on a civil service or military payroll profile that the number of unique salaries is quite low but the number of identical records quite high. Thus, the profile input file would be much shorter than the original file. Not only would each record be shorter, but the compacting of identical entries would substantially reduce the number of records in the file.

Time is one of the main considerations when sorting records. Sorting is generally an extremely time-consuming process. After the desired order of the records has been established and the present condition of the records is known, an efficient procedure should be established to make the sort process as efficient as possible. Establishing an efficient procedure is very important because sorting can require large amounts of computer time; moreover, most sort jobs are repeated on a regular basis. A small saving on each run, therefore, can compound to a significant saving over the course of a year.

To establish an efficient procedure requires a great deal of planning and checking. It includes all of the steps involved in creating the records and files to be sorted, the order of the output desired, the specification of the Sort/Merge directives, and even the time of day the sort is to be run. For example, if sort is to be run in a multiprogramming environment, it can be beneficial to run the sort program with certain types of jobs which will not result in much competition for the system resources. If jobs compete for resources, time can be lost through the swapping in and out of one or more of the jobs. If tremendously large sort jobs are to be run, you might consider scheduling them for the hours of low system activity to avoid such conflict or try to balance the job mix to avoid conflict.

At the point the sort run is initiated, the avenues available to you are limited in respect to time-saving opportunities. There is usually little choice as to the computer to be used, the sort program available, the form of the existing input files, and the desired order of output. The length of time required to sort any given file depends on the characteristics of the records and the characteristics of the computer to be used. Since these are usually fixed, only a few options such as available central memory and the possibility of additional hardware devices can be controlled at this point. Most time-saving methods must be considered before the records and files to be sorted are created.

TIME-SAVING DESIGN

The length of records affects the sort process. Long records require a larger amount of central memory, reducing the number of records that can be stored as a sorted string. Shorter records increase the number, thus requiring fewer passes during the sort phase. The length of the records can only be controlled during the process of designing and creating the records as noted in section 2. The size of the files to be sorted can also be reduced if they are organized so that only active records are sorted; inactive records can be relegated to another file. File space is valuable so you should be careful that inactive or useless information is either stored on another file, archived, or purged when it is no longer of use. Such a process is known as file maintenance. File maintenance includes keeping the file up to date by adding records as well as deleting or modifying them. It is also important to monitor the use of each file to determine which files are active and which should be reorganized. If the use of one file often requires the use of another file, you should consider consolidating the two files to reduce the overhead associated with handling two files.

Extremely long records require a large amount of time and memory to sort. It might be advantageous to create a tag sort environment where only the key need be sorted, thus reducing the time and central memory requirements. Also, when a long record proves difficult to read, subsequent read attempts cost a great deal of time for a long record, whereas for a short key, an added saving accrues.

If you work with large files, you will be concerned about possible machine malfunction, power failures, and other potential problems. Checkpointing files being sorted is a good idea which can work to your advantage; however, dividing large files into smaller files makes them much more manageable, and is a superior insurance of successful completion. Not only is sort time reduced, but overhead is also reduced since no time is required for a checkpoint dump nor is a device required to receive the dump. Note that the checkpoint/restart option is not available to tape sort users.

Efficient organization of records, excellent file management, working with smaller files, and a proven procedure will make the required sort jobs run faster and ensure optimum performance.

COBOL AND SORT/MERGE

The COBOL SORT verb calls the same Sort/Merge program available under the operating system. Identical input should result in identical output because there is no difference between the sort program available under COBOL and the sort program available under the operating system. The choice of which sort to specify, for COBOL programmers, depends on other factors.

For a small sort job, it probably will not be advantageous to leave COBOL to specify a sort under the operating system. You won't need to store registers, variables, and so forth. However, a COBOL SORT ties up the COBOL field length which often reduces the amount of central memory available to the sort process. When large files are involved, you might well find an advantage in leaving COBOL to take advantage of the additional field length which could improve the speed and efficiency of the sort program. This decision depends on the size of the file to be sorted and the size of the COBOL program competing for field length.

One point worthy of note is that a COBOL SORT is often specified because the COBOL programmer is not accustomed to specifying an operating system sort.

Actual specification of the COBOL SORT is different from the specification of the operating system sort, though the concepts are the same. For a complete description of the COBOL Sort/Merge facility, see the COBOL reference manual.

As a general rule, a sort which is not extremely large will run just as fast under either COBOL SORT or the operating system SORTMRG control statement. If you have serious doubts whether there will be sufficient field length under COBOL, then by all means consider using the operating system Sort/Merge program.

Another effective method of reducing field length conflict when using large files and a large COBOL program is to segment the COBOL program and put the sort into a short segment.

There are no owncode exits available to the COBOL programmer such as those available to the COMPASS and FORTRAN programmers. On the other hand, COBOL programmers can use procedures to achieve most of the same results. COBOL allows the use of INPUT PROCEDURE and OUTPUT PROCEDURE phrases to specify the procedure to be executed under system control either at the time the SORT statement is executed or after the records have been sorted. The INPUT PROCEDURE can include statements to select, create, or modify records before the sort process begins. Thus, the same functions of Sort/Merge owncode are available to the COBOL programmer though their specifications are totally different.

When using the INPUT or OUTPUT PROCEDURE, the COBOL programmer must be wary of the problems that can be created. Changing the length of the record can shift the position of the sort key fields so they no longer match the original specification or the rest of the record to be sorted. Extending the length of the records can cause some records to exceed the record length specified. Deleting records without being aware of a future need for them can prove catastrophic.

Use of the COBOL SORT is described in the COBOL reference manual and the COBOL 5 user's guide.

FORTRAN CALLS AND SORT/MERGE

FORTRAN calls allow the use and specification of owncode routines written in FORTRAN. The FORTRAN calls are almost identical with the Sort/Merge directives. They are described fully in the Sort/Merge reference manual.

A number of the arguments concerning the advantages of the COBOL SORT verb versus the operating system sort hold true for the FORTRAN calls to Sort/Merge as well. It can be advantageous to sort without leaving the FORTRAN program; however, this convenience is not without its cost in field length that otherwise might be used by Sort/Merge for more efficient operation.

One method of avoiding a field length conflict between a large FORTRAN program and large sort files is to use separate job steps with control statements. Another method is to use overlays and ensure that the sort occurs in a short overlay.

CHECKPOINT/RESTART

The use of checkpoint/restart is recommended in many applications to allow you to recover some of the work already done in case of a power failure or machine malfunction. In the case of Sort/Merge there are other options that you should consider. Checkpointing a program requires that you specify a certain point or a certain number of records after which the operating system will take a dump of the work done to that point. In case of job failure, you can return to that point in processing rather than start anew.

A checkpoint dump requires that system resources such as disk space or tape units be assigned for the dump. Such resource allocation often reduces the resources available to the Sort/Merge program.

As noted previously, reducing the size of the files you wish to sort, sorting them, and then merging them can be faster than sorting one large file. If you compare this procedure with the checkpoint/restart procedure, you will find that time is saved in sorting, and that the same insurance against machine or system malfunction is afforded.

Shorter files sort faster on a relative basis than do long files, and reducing the system overhead associated with checkpointing a file adds to the speed achieved by sorting smaller files.

Refer to the Sort/Merge reference manual for details of checkpoint/restart usage.

TAPE SORTING

The tape variant of Sort/Merge provides two forms of processing, balanced and polyphase. A tape sort is not the same as a disk sort with tape input and output. A tape sort does not require disk units. Sort/Merge is most efficient when the disk oriented version is used. Use of tape sorts is discouraged when disk facilities are available because tape is less reliable and is usually less efficient. In cases where the size of records and the size of files seems to require specification of a tape sort, you might consider dividing the large files into smaller files to be presorted and merged later rather than use the tape sort.

Other reasons for not specifying a tape sort include the need for many tape units to contain all of the scratch tapes required, and the added possibility for operator error, in addition to the almost certain occurrence of tape parity errors in a multireel environment. Also, a tape sort cannot be checkpointed. Tape sorts are explained in the Sort/Merge reference manual appendix.

Note that backup tape files are usually maintained in step sequence, so that if any one operation is unsuccessful, you can back up to the next previous step to recover. It is not uncommon to see up to four levels of such backup files being retained as insurance against loss of information on tape.

The polyphase sort is usually more efficient than the balanced sort. In a situation where you can have the opportunity to try both types of tape sort on the same or very similar files, the best way to determine the better sort is to try both and compare. The number of tape drives available best determines which tape sort is better; the balanced sort becomes preferable when you can commit eight or more tape drives to the sort.

TAG SORT

When records to be sorted are extremely large, it is sometimes quite time-consuming or impossible to sort them because few will fit into the memory available. It is often easier and better to sort such large records by creating a key which identifies the record and includes sufficient information to link the key to the record, and then sort only the keys. When they are in order, the

records can then be retrieved in sorted order from their storage locations on disk in the order of the sorted keys. This is called a tag sort.

Depending on the application, it is often advantageous to never order the records themselves on disk, but rather to order only the output.

Keys can be created in a number of ways depending on your needs. One common method of creating keys is to use an algorithm which extracts the sort key from the record along with disk address values which identify the location of the record and appends this value to the key field to be sorted. When the keys are all sorted, the records can be retrieved in sorted order from the disk containing them. This type of sort technique is usually only undertaken by systems programmers.

SUMMARY

The following points are offered for your consideration before starting any sort or merge operation:

- If you can avoid sorting information, you will probably save time.
- Sorting small files and merging sorted files is faster than sorting one large file.
- If possible, sort only the information you need, not the entire record.
- Always sort files before merging them with larger files.
- It is always better to make smaller sort runs than to checkpoint larger sort runs.
- It is usually true that the time required to sort half the records requires less than half the time needed to sort all the records. A shorter sort is usually more efficient. As the size of a sort grows, the number of records sorted per second decreases.
- Be very careful when using owncode to change record size or sort key field position before sorting the file.
- Always keep backup files. You can always go back if you keep previous information. Some installations keep up to four levels of backup files.

CHARACTER SETS A

CONTROL DATA operating systems offer the following variations of a basic character set:

CDC 64-character set

CDC 63-character set

ASCII 64-character set

ASCII 63-character set

The set in use at a particular installation was specified when the operating system was installed. You cannot change it.

Depending on another installation option, the operating system assumes an input card deck has been punched either in 026 or in 029 mode (regardless of the character set in use).

Under NOS, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of any 6/7/9 card. In addition, 026 mode can be specified by a card with 5/7/9 multipunched in column 1, and 029 mode can be specified by a card with 5/7/9 multipunched in column 1 and a 9 punched in column 2. Under NOS/BE, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of the job statement or any 7/8/9 card. The specified mode remains in effect through the end of the job unless it is reset.

Graphic character representation appearing at a terminal or printer depends on the installation character set and the terminal type. Characters shown in the CDC Graphic column of the standard character set table are applicable to BCD terminals; ASCII graphic characters are applicable to ASCII-CRT and ASCII-TTY terminals.

STANDARD COLLATING SEQUENCES

If the installation character set is the CDC character set, the collating sequence default is COBOL6. If the installation character set is ASCII, the collating sequence default is ASCII6.

COLLATION OF ARBITRARY CHARACTERS

Several graphics are not common for all codes. Where these differences in graphics occur, arbitrary assignment of collation positions and of translations between codes must be made. For example, display code data that is collated in the ASCII6 collating sequence requires assignment of specific graphics. One of these graphics is the identity character ≡ (60) in display code that is interpreted as the number character (#) in ASCII6 collating sequence in table A-2.

TABLE A-1. STANDARD CHARACTER SETS

Display Code (octal)	CDC			ASCII		
	Graphic	Hollerith Punch (026)	External BCD Code	Graphic Subset	Punch (029)	Code (octal)
00†	: (colon)††	8-2	00	: (colon)††	8-2	072
01	A	12-1	61	A	12-1	101
02	B	12-2	62	B	12-2	102
03	C	12-3	63	C	12-3	103
04	D	12-4	64	D	12-4	104
05	E	12-5	65	E	12-5	105
06	F	12-6	66	F	12-6	106
07	G	12-7	67	G	12-7	107
10	H	12-8	70	H	12-8	110
11	I	12-9	71	I	12-9	111
12	J	11-1	41	J	11-1	112
13	K	11-2	42	K	11-2	113
14	L	11-3	43	L	11-3	114
15	M	11-4	44	M	11-4	115
16	N	11-5	45	N	11-5	116
17	O	11-6	46	O	11-6	117
20	P	11-7	47	P	11-7	120
21	Q	11-8	50	Q	11-8	121
22	R	11-9	51	R	11-9	122
23	S	0-2	22	S	0-2	123
24	T	0-3	23	T	0-3	124
25	U	0-4	24	U	0-4	125
26	V	0-5	25	V	0-5	126
27	W	0-6	26	W	0-6	127
30	X	0-7	27	X	0-7	130
31	Y	0-8	30	Y	0-8	131
32	Z	0-9	31	Z	0-9	132
33	0	0	12	0	0	060
34	1	1	01	1	1	061
35	2	2	02	2	2	062
36	3	3	03	3	3	063
37	4	4	04	4	4	064
40	5	5	05	5	5	065
41	6	6	06	6	6	066
42	7	7	07	7	7	067
43	8	8	10	8	8	070
44	9	9	11	9	9	071
45	+	12	60	+	12-8-6	053
46	-	11	40	-	11	055
47	*	11-8-4	54	*	11-8-4	052
50	/	0-1	21	/	0-1	057
51	(	0-8-4	34	(	12-8-5	050
52	)	12-8-4	74	)	11-8-5	051
53	$	11-8-3	53	$	11-8-3	044
54	=	8-3	13	=	8-6	075
55	blank	no punch	20	blank	no punch	040
56	, (comma)	0-8-3	33	, (comma)	0-8-3	054
57	. (period)	12-8-3	73	. (period)	12-8-3	056
60	≡	0-8-6	36	#	8-3	043
61	[	8-7	17	[	12-8-2	133
62	]	0-8-2	32	]	11-8-2	135
63	%††	8-6	16	%††	0-8-4	045
64	≠	8-4	14	" (quote)	8-7	042
65	←	0-8-5	35	_ (underline)	0-8-5	137
66	∨	11-0 or 11-8-2†††	52	!	12-8-7 or 11-0†††	041
67	∧	0-8-7	37	&	12	046
70	↑	11-8-5	55	' (apostrophe)	8-5	047
71	↓	11-8-6	56	?	0-8-7	077
72	<	12-0 or 12-8-2†††	72	<	12-8-4 or 12-0†††	074
73	>	11-8-7	57	>	0-8-6	076
74	≤	8-5	15	@	8-4	100
75	≥	12-8-5	75	\	0-8-2	134
76	¬	12-8-6	76	^ (circumflex)	11-8-7	136
77	; (semicolon)	12-8-7	77	; (semicolon)	11-8-6	073

†Twelve zero bits at the end of a 60-bit word in a zero byte record are an end of record mark rather than two colons.

††In installations using a 63-graphic set, display code 00 has no associated graphic or card code; display code 63 is the colon (8-2 punch). The % graphic and related card codes do not exist and translations yield a blank (55_8).

†††The alternate Hollerith (026) and ASCII (029) punches are accepted for input only.

TABLE A-2. 6-BIT CHARACTER CODE COLLATING SEQUNCES

COBOL6†		DISPLAY†		INTBCD		ASCII6††	
Graphics	Display Code	Graphics	Display Code	Graphics	CDC INTBCD	Graphics	Sequence
blank	55	:†	00†	0	00	blank	00
≤	74†	A	01	1	01	!	01
%†	63	B	02	2	02	"	02
[	61	C	03	3	03	#	03
→	65	D	04	4	04	$	04
≡	60	E	05	5	05	%††	05
∧	67	F	06	6	06	&	06
↑	70	G	07	7	07	'	07
	71	H	10	8	10	(	10
>	73	I	11	9	11	)	11
≥	75	J	12	:	12	*	12
¬	76	K	13	=	13	+	13
.	57	L	14	≠	14	,	14
)	52	M	15	≤	15	-	15
;	77	N	16	%	16	.	16
+	45	O	17	[	17	/	17
$	53	P	20	+	20	0	20
*	47	Q	21	A	21	1	21
-	46	R	22	B	22	2	22
/	50	S	23	C	23	3	23
,	56	T	24	D	24	4	24
(	51	U	25	E	25	5	25
=	54	V	26	F	26	6	26
≠	64	W	27	G	27	7	27
<	72	X	30	H	30	8	30
A	01	Y	31	I	31	9	31
B	02	Z	32	<	32	:	32
C	03	0	33	.	33	;	33
D	04	1	34	)	34	<	34
E	05	2	35	≥	35	=	35
F	06	3	36	¬	36	>	36
G	07	4	37	;	37	?	37

TABLE A-2. 6-BIT CHARACTER CODE COLLATING SEQUENCE (Contd)

COBOL6 †		DISPLAY †		INTBCD		ASCII6 ††	
Graphics	Display Code	Graphics	Display Code	Graphics	CDC INTBCD Code	Graphics	Sequence
H	10	5	40	-	40	@	40
I	11	6	41	J	41	A	41
∨	66	7	42	K	42	B	42
J	12	8	43	L	43	C	43
K	13	9	44	M	44	D	44
L	14	+	45	N	45	E	45
M	15	-	46	O	46	F	46
N	16	*	47	P	47	G	47
O	17	/	50	Q	50	H	50
P	20	(	51	R	51	I	51
Q	21	)	52	∨	52	J	52
R	22	$	53	$	53	K	53
]	62	=	54	*	54	L	54
S	23	blank	55	↑	55	M	55
T	24	,	56	↓	56	N	56
U	25	.	57	>	57	O	57
V	26	≡	60	blank	60	P	60
W	27	[	61	/	61	Q	61
X	30	]	62	S	62	R	62
Y	31	%†	63†	T	63	S	63
Z	32	≠	64	U	64	T	64
:	00 †	→	65	V	65	U	65
0	33	∨	66	W	66	V	66
1	34	∧	67	X	67	W	67
2	35	↑	70	Y	70	X	70
3	36	↓	71	Z	71	Y	71
4	37	<	72	]	72	Z	72
5	40	>	73	,	73	[	73
6	41	≤	74	(	74	\	74
7	42	≥	75	→	75	]	75
8	43	¬	76	≡	76	^	76
9	44	;	77	∧	77	_	77

†Under the CDC 63-character set, there is no percent graphic; the colon is display code 63. Display Code 00 is not used.

††Under the ASCII 63-character set, there is no percent graphic; the colon collates in position 05, not position 32.

GLOSSARY B

Advanced Access Methods (AAM) -
A file manager that processes indexed sequential, direct access, and actual key file organizations, and supports the Multiple-Index Processor. (See CYBER Record Manager.)

Balanced Tape Sort -
Sort that always keeps its intermediate tapes divided into the same two groups. Sorted strings are merged from one group to another as long as possible, then the direction is reversed.

Basic Access Methods (BAM) -
A file manager that processes sequential and word addressable file organizations. (See CYBER Record Manager.)

Buffer -
An intermediate storage area used to compensate for a difference in rates of data flow, or time of event occurrence, when transmitting data between central memory and an external device during input/output operations.

Collating Sequence -
Sequence that determines precedence given to character data for sorting, merging, and comparing.

CYBER Record Manager -
A generic term relating to the common products AAM and BAM that run under the NOS and NOS/BE operating systems and that allow a variety of record types, blocking types, and file organizations to be created and accessed. The execution time input/output of COBOL 4, COBOL 5, FORTRAN Extended 4, Sort/Merge 4, ALGOL 4, and the DMS-170 products is implemented through CYBER Record Manager. Neither the input/output of the NOS and NOS/BE operating systems themselves nor any of the system utilities such as COPY or SKIPF is implemented through CYBER Record Manager. All CYBER Record Manager file processing requests ultimately pass through the operating system input/output routines.

Direct Access File -
In the context of CYBER Record Manager, a direct access file is one of the five file organizations. It is characterized by the system hashing of the unique key within each file record to distribute records randomly in blocks called home blocks of the file.

In the context of NOS permanent files, a direct access file is a file that is accessed and modified directly, as contrasted with an indirect access permanent file.

Directives -
Instructions that supplement processing defined by the SORTMRG control statement for execution of Sort/Merge record processing.

File -
A logically related set of information; the largest collection of information that can be addressed by a file name. Starts at beginning-of-information and ends at end-of-information.

FILE Control Statement -
A CYBER Record Manager control statement that contains parameters used to build the file information table for processing. Must be provided for every input or output file to be processed by a directive sort or merge. Not to be confused with the Sort/Merge FILE directive.

File Information Table (FIT) -
A table through which a user program communicates with CYBER Record Manager. All file processing executes on the basis of fields in the table. Some fields can be set by the Sort/Merge user in the FILE control statement.

Key Comparison -
Internal technique of comparing sort keys that usually requires less elapsed time and more central processing time than key extraction.

Key Extraction -
Internal technique of comparing sort keys that usually requires less central processing time and more elapsed time than key comparison.

Macro -
Sequence of source statements that are saved and then assembled when needed through a macro call. Used when Sort/Merge functions as a COMPASS subroutine for a COMPASS program or as a relocatable program generated for the COBOL SORT verb.

Merge Order -
Internal parameter governing the number of buffers used by Sort/Merge Version 4 in the intermediate merge phase.

Owncode Routine -
Closed COMPASS subroutine written by the user that provides the capability to insert, substitute, modify, or delete input and output records during Sort/Merge processing.

Polyphase Tape Sort -
Sort with only one intermediate output tape for each merge phase; however, the output tape is changed for each merge phase. A polyphase tape sort usually can sort more records than a balanced tape sort in the same amount of time and with the same number of intermediate tapes.

Random File -
In the context of CYBER Record Manager, a file with word addressable, indexed sequential, direct access, or actual key organization in which individual records can be accessed by the values of their keys.

Record -
CYBER Record Manager defines a record as a group of related characters. A record or a portion thereof is the smallest collection of information passed between CYBER Record Manager and a user program. Eight different record types exist, as defined by the RT field of the file information table.

Signed Numeric Data -
Integer data stored internally in display code. Sorts according to the magnitude and the sign of the integer the display code represents.

Sort Key -
Field of information within each record in a sort or merge input file used to determine the order in which records are written to the output file.

Sort Order -
Order for sorting keys, either ascending or descending.

Tape Sort -
Sort that has its intermediate scratch files residing on tape rather than disk. Original input file and/or final output file can reside on disk or tape.

RUNNING SORT/MERGE UNDER THE NOS/BE OPERATING SYSTEM

C

This appendix illustrates the basic differences between the NOS and the NOS/BE operating systems with respect to Sort/Merge and includes the NOS/BE control statements you will need to run the job examples given in section 4 if your installation is using the NOS/BE operating system.

As noted previously, the Sort/Merge directives need not be changed because of a change of operating system. Certain limitations apply if you are using Sort/Merge Version 1 which is required if your computer is a CYBER 170 Model 176, a CYBER 70 Model 76, or a 7600. Refer to the Sort/Merge reference manual for these limitations. Users of the remaining CYBER 170 and CYBER 70 models and the 6000 Series computers will use Sort/Merge Version 4 described in the same publication.

User programs can call Sort/Merge with COMPASS assembly language macros, the FORTRAN Extended interface routine calls, or through the COBOL language. These uses are described in detail in the Sort/Merge reference manual and in the respective language reference manuals.

CONTROL STATEMENT FORMATS

The major differences between the control statements required for the NOS and NOS/BE operating systems from the Sort/Merge user's perspective are in the areas of the job control statement including accounting information and in the use of permanent files. Other important differences are noted only where they apply to this user's guide. You should refer to the user's guide and reference manuals applicable to your operating system for all details.

ACCOUNTING INFORMATION

As you have noted in the practice examples, NOS usually requires a USER and a CHARGE control statement following the job control statement. These are used for identification and accounting purposes. If these control statements are not given, or are incorrect, the run will terminate with a message indicating the error. Procedures vary from installation to installation depending on the accounting methods in use. Interactive users at some installations are limited in the number of attempts they are allowed when signing onto the system.

NOS/BE users are often required to include their accounting information on the job control statement following the terminator. Other installations require this information on a separate ACCOUNT control statement. Security procedures usually terminate any unauthorized job. You might wish to note the accounting information required at your installation inside the front cover. You should be careful with this information because your account will be billed for all jobs run under this number.

PERMANENT FILES

Users of NOS permanent files will encounter both direct and indirect access permanent files. Their use is quite different. For purposes of these examples, only indirect access permanent files are used. Large files might be better served by using direct access permanent files. The choice of type of NOS permanent file is described in the NOS Time-Sharing user's guide. There is no NOS/BE counterpart for direct and indirect access files.

If during a NOS job, you wish to save a file for future use, such as for input to another job, a simple SAVE,filename is the minimum control statement you can specify to create an indirect access permanent file. This file will be saved for the number of days your installation allows. When you wish to use this file again in a subsequent job, all you need enter to access the file is the control statement GET,filename.

NOS/BE permanent files can be used in a similar manner. NOS/BE requires that you enter CATALOG,filename,id to make an existing local file permanent and ATTACH,filename,id to access an existing permanent file. Before you can create the permanent file, however, you must first allocate file space for it through use of the REQUEST,*PF control statement. NOS/BE also allows you to keep up to 5 cycles of one permanent file under one file name; there is no NOS counterpart for this concept.

The length of time that a NOS/BE permanent file is kept depends on the time you specify, or on the operating system default. At the installation where these jobs were run, the default retention period is set at 5 days. By running a job using the permanent file more often than every 5 days, the 5-day period is renewed. When the file is no longer needed, it is automatically purged from the system 5 days later. If you specify a longer period, you should purge the permanent file when you no longer need it.

JOB EXAMPLES

The following NOS/BE control statement examples will allow you to run the practice examples given. For more information on your operating system and the control statements available, you should consult the NOS/BE user's guide or the NOS Batch user's guide or the NOS Time-Sharing user's guide.

The NOS control statements given in figure 4-4 can be replaced with the NOS/BE control statements shown in figure C-1. Each figure caption includes the figure number of the related NOS example.

Under NOS/BE, the T parameter on the job statement specifies a time limit for the job in octal seconds. The time limit also influences the priority given the job in the input queue. Too high a limit can reduce the job priority. Too low a limit can stop the job before it completes. Refer to the NOS/BE user's guide for details.

Figure C-2 is the NOS/BE counterpart of figure 4-5. The explanations associated with figure 4-5 also apply here.

Figure C-3 illustrates how you can create a NOS/BE permanent file of the sorted output file. When creating a permanent file under NOS/BE, you must specify an ID for the file and you can specify a retention period if you wish as shown with the CATALOG statement in figure C-3. To subsequently access this permanent file requires a statement such as ATTACH,NEW,ID=ME. Both the CATALOG and the ATTACH statements allow a large number of parameters for various purposes. These parameters are described in the NOS/BE user's guide and the NOS/BE reference manual.

```
jobcard.  user and accounting information
FILE(NEW,BT=C,RT=Z,FL=80)
SORTMRG.
REWIND,NEW.
COPYSBF,NEW,OUTPUT.
7/8/9 multipunched in column 1

  sort directives

7/8/9 multipunched in column 1

  input records

6/7/8/9 multipunched in column 1
```

Figure C-1. NOS/BE Control Statements (Figure 4-4)

```
7 C  D I R E C T I V E S    SORT/MERGE 4.6 L497  03/29/79  14.19.11.      PAGE      1


1  SORT
2  FILE,INPUT=INPUT,OUTPUT=NEW
3  FIELD,NAME(1,23,DISPLAY)
4  KEY,NAME(A,DISPLAY)
5  END

-----------------------


ANDERSON,TIMOTHY          2 A000700052575264M
BAKKER,JOACHIM            2 A000710011176245S
BERNARD,JOHN              2 A000625011577195S
BOER,GEORGE               2 A000795122374245S
BROWN,JAMES               2 A000700091176235S
CARLSON,JACK              8 E001975083074365D
CHANG,ROBERT              2 A000684020176255S
COHEN,JOSEPH              2 A000875040474355D
DAVIS,ROBERT              2 E001075010373355S
DUBOIS,ANDRE              2 A000625031578215S
DURAND,HELEN              6 A001635090168365D
FISCHER,DAVID             2 A000625060178215S
GARCIA,ARTHUR             2 A000738022976345M
GOMEZ,LINDA               2 C000635042177215S
IVANOV,LEONARD            2 A000723081576325M
JOHNSON,ANNABELLE         6 C001472040369285M
JOHNSON,ARMAND            4 E001410102072325S
                    .
                    .
                    .
SMITH,ROBERTA             4 A000750112275275S
SOKOL,DONALD              8 A002108120768465M
TAYLOR,JENNIFER           2 A000625101177235S
WANG,LISA                 2 D000585112777235S
WILLIAMS,BENEDICT         6 A001721081672395D
WILLIAMS,ROBERT           2 A000790031576315S
WILSON,DOUGLAS            2 E000910120973365D
```

Figure C-2. NOS/BE Sort Output by Name (figure 4-5) (Sheet 1 of 2)

```
  MFS  NB1-  (YB74-SN108     6RSV/I2C  03/16/79
14.19.05.EXRC 16A  FROM
14.19.05.IP   (0000448 WORDS - FILE INPUT  , DC 04
14.19.05.EXRC 1,T10.      user and accounting information
14.19.09.FILE (NEW,BT=C,PT=Z,FL=80)
14.19.10.SORTMRG.
14.19.12.***KEY COMPARISON USED
14.19.15. **  INSERTIONS DURING INPUT    *********0
14.19.15. **  DELETIONS DURING INPUT    *********0
14.19.15. **  TOTAL RECORDS SORTED      ********40
14.19.15. **  INSERTIONS DURING OUTPUT   *********0
14.19.15. **  DELETIONS DURING OUTPUT   *********0
14.19.15. **  TOTAL RECORDS OUTPUT      ********40
14.19.15. **  MERGE ORDER USED          ********13
14.19.15. **END SORT RUN
14.19.15.REWIND,NEW.
14.19.15.COPYSBF,NEW,OUTPUT.
14.19.15.OP   (0000512 WORDS - FILE OUTPUT , DC 40
14.19.15.MS     3584  WORDS  (   10752 MAX USED)
14.19.15.CPA        .210 SEC.          .210 ADJ.
14.19.15.CPB        .270 SEC.          .270 ADJ.
14.19.15.IO         .347 SEC.          .347 ADJ.
14.19.15.CM       14.601 KWS.          .891 ADJ.
14.19.15.SS                           1.720
14.19.15.PP        6.055 SEC.      DATE 03/29/79
14.19.15.EJ  END OF JOB, **
```

Figure C-2. NOS/BE Sort Output by Name (figure 4-5) (Sheet 2 of 2)

```
EXRC1,T10.  accounting information
REQUEST,NEW,*PF.  <----------
FILE(NEW,BT=C,RT=Z,FL=80)
SORTMRG.
CATALOG,NEW,ID=ME,RP=10.  <----------
REWIND,NEW.
COPYSBF,NEW,OUTPUT.
7/8/9 multipunched in column 1

  sort directives

7/8/9 multipunched in column 1

  input records

6/7/8/9 multipunched in column 1
```

Figure C-3. Creating a NOS/BE Permanent File (Figure 4-6)

INDEX

COMMENT SHEET

MANUAL TITLE: Sort/Merge Versions 4 and 1 User's Guide

PUBLICATION NO.: 60482900

REVISION: A

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments on the back (please include page number references).

_____ Please reply _____ No reply necessary

FOLD FOLD

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division
P.O. BOX 3492
Sunnyvale, California 94088-3492

CUT ALONG LINE

FOLD FOLD

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.
FOLD ON DOTTED LINES AND TAPE

NAME:

COMPANY:

STREET ADDRESS:

CITY/STATE/ZIP:

TAPE TAPE