SPERRY

PUBLICATIONS UPDATE

Operating System/3 (OS/3)

Independent Sort/Merge

User Guide/Programmer Reference

UP-8819 Rev. 2-A

This Library Memo announces the release and availability of Updating Package A to "SPERRY® Operating System/3 (OS/3) Independent Sort/Merge User Guide/Programmer Reference", UP-8819 Rev. 2.

The independent sort/merge program available to System 80 allows Operating System/3 (OS/3) users to produce a tailored output file from existing input data files.

This manual describes the sort/merge program and is intended for the programmer with a basic knowledge of data processing, but with little programming experience.

This update documents changes to the following:

- Program restrictions and considerations
- Table 1–2, Comparison of Transfer Rates for Magnetic Tape Devices

Copies of Updating Package A are now available for requisitioning. Either the updating package only or the complete manual with the updating package may be requisitioned by your local Sperry representative. To receive only the updating package, order UP-8819 Rev. 2–A. To receive the complete manual, order UP-8819 Rev. 2.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ, and MZ	Mailing Lists B00, B02, B03, 28U, and 29U (Package A to UP-8819 Rev. 2, 7 pages plus Memo)	Library Memo for UP-8819 Rev. 2–A
		RELEASE DATE: January, 1985

SPERRY

PUBLICATIONS REVISION

Operating System/3 (OS/3)

Independent Sort/Merge

User Guide/Programmer Reference

UP-8819 Rev. 2

This Library Memo announces the release and availability of "SPERRY® Operating System/3 (OS/3) Independent Sort/Merge User Guide/Programmer Reference", UP-8819 Rev. 2.

This manual supersedes, for System 80 users, UP-8819 Rev. 1. The text is essentially the same except for changes made for Release 8.2 which include the following:

- Support of 8416, 8418, 8430, 8433, and 8470 disk subsystems.
- Addition of an UPSI byte that is set when a warning message is generated indicating Sort/Merge has located less than the number of files specified by the user.

All other changes are corrections or expanded descriptions applicable to features present in Independent Sort/Merge prior to Release 8.2.

Destruction Notice: If you are going to OS/3 release 8.2, use this revision and destroy all previous copies. If you are not going to OS/3 release 8.2, retain the copy you are now using and store this revision for future use.

Copies of UP-8819 Rev. 1 and UP-8819 Rev. 1–A will be available for 6 months after the release of 8.2. Should you need additional copies of this edition, you should order within 90 days of the release of 8.2. When ordering the previous edition of a manual, be sure to identify the exact revision and update packages desired and indicate that they are needed to support an earlier release.

Additional copies may be ordered by your local Sperry representative.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ and MZ	Mailing Lists B00, B02, B03, 28U, and 29U (Cover and 132 pages)	Library Memo for UP-8819 Rev. 2
		RELEASE DATE: February, 1984

UNISYS

OS/3 Independent Sort/Merge

Programming Guide

Relative to Release
Level 9.0

Priced Item

August 1987

Printed in U S America
UP–8819 Rev. 2

UNISYS

OS/3 Independent Sort/Merge

Programming Guide

Previous Title: OS/3 Independent Sort/Merge
User Guide/Programmer Reference

Relative to Release
Level 9.0

Priced Item

August 1987

Printed in U S America
UP–8819 Rev. 2

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THE DOCUMENT. Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed Program Product License or Agreement to purchase or lease equipment. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such License or Agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, indirect, special or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

FASTRAND, ✦SPERRY, SPERRY✦UNIVAC, SPERRY, SPERRY UNIVAC, UNISCOPE, UNISERVO, UNIS, UNIVAC, and ✦ are registered trademarks of Unisys Corporation. ESCORT, PAGEWRITER, PIXIE, PC/IT, PC/HT, PC/microIT, SPERRYLINK, and USERNET are additional trademarks of Unisys Corporation. MAPPER is a registered trademark and service mark of Unisys Corporation. CUSTOMCARE is a service mark of Unisys Corporation.

PAGE STATUS SUMMARY

ISSUE: Update B – UP-8819 Rev. 2
RELEASE LEVEL: 9.0 Forward

Part/Section	Page Number	Update Level
Cover		B
Title Page/Disclaimer		B*
PSS	1	B
Preface	1, 2	Orig.
Contents	1 thru 4	Orig.
1	1 thru 3 4 5, 6 7 8 thru 11	Orig. A Orig. A Orig.
2	1, 2	Orig.
3	1 thru 50	Orig.
4	1 thru 15	Orig.
Appendix A	1 thru 5	Orig.
Appendix B	1 thru 11	Orig.
Appendix C	1 thru 21	Orig.
Index	1 thru 8	Orig.
User Comment Form		

*New pages

All the technical changes are denoted by an arrow (⟹) in the margin. A downward pointing arrow (⇓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (⇑) is found. A horizontal arrow (⟹) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

Preface

This manual is one of a series designed to instruct and guide the programmer in the use of the SPERRY Operating System/3 (OS/3). It specifically describes the independent sort/merge program available to System 80 users of OS/3. The intended audience is the programmer with a basic knowledge of data processing, but with little programming experience. Also helpful is an understanding of job control language (JCL). Programmers planning to use own-code routines should be familiar with basic assembler language (BAL) and data management macros. An introduction manual, the introduction to sort/merge, UP-8835 (current version), is also available. It describes the general characteristics and facilities of all the sort/merge programs available to OS/3 users.

Other current OS/3 publications referenced in this manual that are helpful when using independent sort/merge are:

- Data utilities user guide/programmer reference, UP-8834

  Describes the data utility routine.

- System service programs (SSP) user guide, UP-8841

  Describes various system utilities (e.g., librarian, linkage editor).

- General editor user guide/programmer reference, UP-9976

  Describes the OS/3 general editor (EDT).

- Job control user guide, UP-9986

  Describes the job control language used under OS/3.

- System messages programmer/operator reference, UP-8076

  Describes the errors encountered when using OS/3.

- Basic data management user guide, UP-8068

  Describes the effective use of OS/3 basic data management.

- Consolidated data management macro language user guide/programmer reference, UP-9979

  Describes the data management macroinstructions.

- Interactive services commands and facilities, UP-9972

  Describes the commands and operating procedures for workstation terminals.

The subject matter in this manual is divided into the following sections and appendixes:

- Section 1. Introduction
- Section 2. Basic Concepts
- Section 3. Sort/Merge Requirements You Supply
- Section 4. Program and Control Stream Examples
- Appendix A. Statement Conventions
- Appendix B. Standard EBCDIC and ASCII Collating Sequences
- Appendix C. Control Statement Summary

Contents

3. SORT/MERGE REQUIREMENTS YOU SUPPLY

4. PROGRAM AND CONTROL STREAM EXAMPLES

APPENDIXES

A. STATEMENT CONVENTIONS

B. STANDARD EBCDIC AND ASCII COLLATING SEQUENCES

C. CONTROL STATEMENT SUMMARY

INDEX

USER COMMENT SHEET

FIGURES

TABLES

1. Introduction

1.1. WHY YOU NEED A SORT PROGRAM

Why is it important that you have a sort capability? Well, consider the amount and types of data contained in your files, and the number of ways in which you use that data. You'll probably discover that you seldom use all of the data for every job and that the organization of the data does not always lend itself to efficient methods of processing during certain applications. In general, most files contain a collection of data records, possibly of different types, that have no relationship other than their existence in the same file. Finding records and specific types of data in your files requires a search, and searching takes time. However, less time is expended to search an ordered file than to search an unordered file, and time is directly related to processing efficiency. This is where a good sort program comes into play. It allows you to select the data you need and to organize that data according to criteria such as an employee number, customer account number, an inventory item, or whatever your particular job application requires. Remember, data is useless for the most part unless it can be related to something real, such as the type of record entries mentioned. A file properly organized and formatted for the job at hand allows the use of techniques that achieve faster searching of your files, faster determination of the presence or absence of the information needed, and faster record retrieval during job execution.

1.2. WHAT INDEPENDENT SORT/MERGE DOES FOR YOU

Independent sort/merge is essentially an easy-to-use *canned* service program. It does not need to be assembled or linked and requires only a minimum of user programming and intervention. The program is loaded and directed at run time via sort/merge control statements you include in the control stream of your job. Because sort/merge control statements and job control are used to define files, records, and functional structure of the sort to the system, you have no lengthy register address manipulations to program. You simply provide the data files, assign your devices, and define the sort or merge-only procedure you want independent sort/merge to perform.

For those who want to perform specialized functions other than those provided by the program, independent sort/merge allows you to write your routine (called an *own-code* routine). Own-code routines can be used to extend your control over the selection of external formats and disposition of output records, record sequencing, and data reduction. Own-code routines are written in basic assembler language (BAL) and must conform to the interfaces of the sort program and the conventions of OS/3. Although it supports user own-code, independent sort/merge does not allow you to indiscriminately pass control to your routines. Exiting to own-code routines is restricted to specific operational phases of the sort. The rules for, and restrictions placed on, the use of own-code routines within independent sort/merge are provided in Section 3.

Independent sort/merge assists you in producing a tailored output file from your existing input data files. You can reformat a file (rearrange records and selectively include or omit specific record types), reformat records, and summarize record fields. The types of sorts performed include full record sorts, tag sorts, and summary sorts. Specifically, independent sort/merge can:

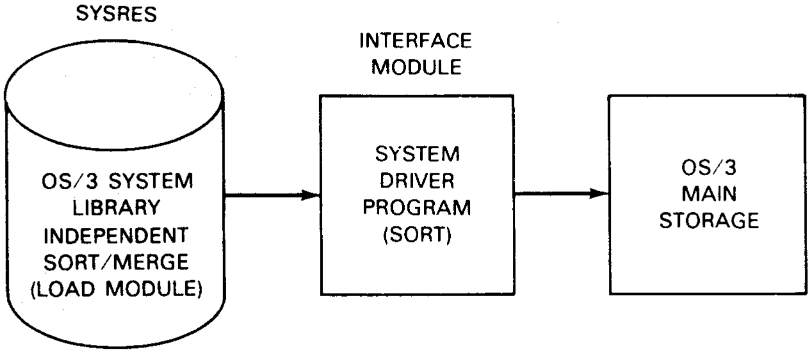
- sort records in ascending or descending sequence;
- sort fixed-length or variable-length records;
- sort blocked or unblocked records;
- sort records with noncontiguous key fields;
- recognize key fields in the following formats:
  - character
  - binary (signed and unsigned)
  - decimal (signed zoned and unsigned zoned)
  - decimal packed
  - leading and trailing sign numeric
  - overpunched leading and trailing sign numeric
  - EBCDIC data in ASCII collating sequence
  - floating point (single and double precision)
- sort two or more different characters having the same collating value (multiple character sort);
- sequence files in accordance to user-specified (alternate) collating sequence;
- perform data validity and data integrity checks during sorting; and
- perform restart procedures for tape sorts.

The output produced from your sort job is file formatted according to your instructions to the sort program. You are not, however, automatically provided a copy of the output file produced by the sort. If you want a copy of the sorted file, you can obtain it by running the appropriate data utility routine, as described in the data utilities user guide/programmer reference, UP-8834 (current version). The successful execution of your job results in a *terminated normally* message printed on your job log and a list of the total number of records included in the sort and the total number of records deleted during the sort.

1.3. CONCEPT OF MODULAR SORT STRUCTURE

In the process of describing independent sort/merge, we refer to it as being modular in structure. What do we mean by modular? Modular, as related to the sort programs, refers to the method used to package the sort/merge programs. Rather than writing separate sort programs for every conceivable type of sort, we have broken the sort/merge process into a group of interrelated, yet independent, functional subtasks. The subtasks are coded as executable routines and are provided to you as load modules residing in the system load library ($Y$LOD). Their implementation into your job is based on the structure you establish in your job stream. That is, you define the type of sort you want performed through parameterized statements in your job control stream, and the sort program will structure the sort/merge process accordingly. One advantage of modular programming is that it conserves main storage space. The sort program loads only those modules needed for the particular sort/merge phase being executed. It also aids in adapting the OS/3 independent sort/merge program to the requirements of your installation by increasing programming flexibility.

In addition to the sort modules, independent sort/merge provides a call module to interface with the system. This call module is the SORT system driver program, which resides in $Y$LOD.


SYSRES
OS/3 SYSTEM LIBRARY INDEPENDENT SORT/MERGE (LOAD MODULE)
INTERFACE MODULE
SYSTEM DRIVER PROGRAM (SORT)
OS/3 MAIN STORAGE

If you want to copy the independent sort/merge program onto your own user library file, you can do so by means of the librarian, as described in the system service programs (SSP) user guide, UP-8841 (current version). Be sure to include the system driver program (SORT) and sort load modules beginning with SM$ from $Y$LOD.

1.4. PROGRAM RESTRICTIONS AND CONSIDERATIONS

Variations in program design, capability, and implementation sometimes restrict the use of a sort program for specific applications or for specific system configurations. Therefore, consideration should be given to the following:

- All sorting is limited to storage-only, disk-only, or tape-only, single-cycle sorts.

- Input and output files can be disk, format label diskette, or tape.

- Auxiliary storage work areas can be either disk or tape, but not both, and are limited to eight disk files or six tape files.

- Volume of data sorted and merged is limited by the type and physical capacity of the tape or disk space assigned as auxiliary working storage.

- User own-code routines can be substituted for those provided; however, they must satisfy the requirements of the program and OS/3 programming conventions.

- The FILTYPE parameter is ignored when the system is generated to support only consolidated data management mode file access.

- If the system supports both consolidated data management and DTF file access, the FILTYPE parameter may be used to specify the file type as IRAM (for MIRAM), NI, or SAM.

If the FILTYPE parameter is not specified, the output file type will be the same as the input file type. Or, if an input file is not specified, the output file type will be MIRAM.

1.5. ELEMENTS AFFECTING PERFORMANCE OF A SORT PROGRAM

The careful user should be aware of elements affecting the performance of his sort program. These elements are:

- Available main storage
- Number and type of assigned auxiliary storage devices
- Record characteristics
- Input and output data file organization
- Options under which the sort program operates

Remember to be explicit in supplying instructions to your sort program and to be careful in setting up your file and record formats. This results in faster sorts that require less central processor time and reduces the number of I/O operations required. To improve program efficiency, consider these factors during record and file preparation:

- Record size
- File size
- Key or control field size
- Number of key or control fields
- Record format
- File format

As a rule, simplification reduces processing and the time needed to perform a function. By simplifying the key fields and decreasing their number and size, you decrease the number of comparisons and the length of time needed to make each comparison. Sort performance improves when input and output records are blocked. Decrease record size and you increase efficiency because a greater number of records are processed at one time for a given amount of main storage.

To improve processing speed and efficiency:

- Be generous with storage; assign more than one I/O device to the sort for auxiliary storage and more than the minimum amount of main storage.

- Simplify your file and record formats.

- Be explicit in defining your output file requirements to the sort program.

1.5.1. Main Storage Allocation

In general, the more main storage available to a sort program, the more efficient the performance. It decreases the number of I/O functions because fewer passes are needed to produce strings of sequenced data for final merging. Therefore, proper consideration given to these factors when preparing your program reduces processing time and increases program efficiency.

Independent sort/merge requires 16,000 bytes, plus sufficient main storage for the larger of either two input blocks or two output blocks. User own-code routines may require additional main storage.

When performing a merge-only operation, independent sort/merge requires 16,000 bytes, plus sufficient main storage to hold two buffers for each input file and two buffers for the output file.

An internal-only sort/merge requires sufficient main storage to hold the entire input file, plus eight bytes for each record, in addition to the preceding requirements.

Performing large volume sorts is most efficient when 50,000 to 150,000 bytes of main storage are allocated.

1.5.2. Auxiliary Storage Work Area Assignments

Work areas may be assigned as auxiliary storage on tape or disk, but not both. If disk storage is used, all work area disks must be of the same general type. It is important not to underestimate the amount of auxiliary storage required. When possible, avoid assigning the bare minimum of auxiliary storage needed; otherwise, the sort program must perform a greater number of intermediate merge passes to sequence records. This wastes time and reduces program efficiency. Because the volume of data processed varies with the quantity and type of magnetic tapes or disks assigned as auxiliary storage, selecting auxiliary storage devices with faster data transfer rates results in a faster sort. Data volume doesn't reduce sort performance.

Disk space is assigned by using standard sort work file names DM01,...,DM0n or system scratch space file names $SCR1,...,$SCRn (in consecutive order) on LFD job control statements, or by using WORK jproc calls. If one work file is allocated, the file name DM01 or $SCR1 must be assigned; if two are used, the names DM01 and DM02 or $SCR1 and $SCR2 must be assigned, and so forth. A maximum of eight disk files may be assigned to sort/merge programs. The amount of disk space requested must be sufficient to hold the entire volume of data to be sorted, plus 10 to 20 percent additional space for overhead requirements. (An additional 10 to 20 percent space should be requested if data involves variable-length records.) In addition, all disk files used in the sort operation must be the same type; i.e., mixed disk types are unacceptable. Table 1-1 contains the data capacities and access speeds of the direct access storage devices used by the sort program.

Table 1–1. Comparison of Data Capacities and Access Speeds for Direct Access Devices

Characteristics	Disk Subsystem Types							
	8416	8417	8418-92/93	8418-94/95	8419	8430	8433	8470
Maximum data capacity (8-bit bytes per disk pack)	28,958,720	118,270,000	28,958,720	57,917,440	72,396,800	100,018,280	200,036,560	491,520,000
Maximum track capacity (bytes)	10,240	15,360	10,240	10,240	12,800	13,030	13,030	24,576
Minimum cylinder access time (ms)	10	7	10	10	10	7	7	4
Average cylinder access time (ms)	30	35	27	33	33	27	27	23
Maximum cylinder access time (ms)	60	70	45	60	60	50	50	46

When tape is used, the auxiliary storage work areas use labeled or unlabeled tapes. Work files are assigned by using standard tape sort file names SM01 through SM06 (in consecutive order) on LFD job control statements. A minimum of three tape units, and a maximum of six, may be assigned. Each tape work file must be large enough to contain all of the input data; i.e., the volume of data that can be processed in a tape sort is limited to the capacity of the smallest reel of tape assigned to the sort. The speed (rate) of data transfer varies according to the tape density (number of bits recorded across the width of the tape) and tape device. Refer to Table 1–2.

Table 1-2. Comparison of Transfer Rates for Magnetic Tape Devices

Tape Density (bpi*)	Data Transfer Rate (bps**)				
	UNISERVO 10	UNISERVO 22	UNISERVO 24	UNISERVO 26	UNISERVO 28
9-track (phase encoded) 1600	40,000	120,000	200,000	120,000	200,000
9-track (NRZI) 800	20,000	60,000	100,000	60,000	100,000

* bpi = bits per inch
** bps = bits per second

1.5.3. I/O Data File Organization

Data file organization begins with record layouts. If you assume that you have a fixed number of records, a file of large records takes longer to sort than a file of smaller records. Also, larger key fields and more key fields per record increase sort time because lengthier comparisons are needed. Key fields are explained in 1.6.

Record sizes that exceed one-half track in length may require up to 100 percent more space or twice the normal space calculated by multiplying the number of records to be sorted by the record size.

1.5.4. Sort Options Affecting Performance

When using independent sort/merge, your performance time can be affected by the following options:

- NOCKSM keyword parameter

- USQ specification in the FIELDS keyword parameter

By specifying NOCKSM=D or NOCKSM=T, you suppress the calculation of a checksum word for blocks written to disk (D) or tape (T). A checksum word is used to verify the integrity of data blocks transferred from sort/merge to work files. The calculation and operation of a checksum word increases overall sort/merge operation time. Similarly, if you specify the USQ specification (FORMAT=USQ) in the FIELDS keyword parameter to indicate a user-defined collation sequence, you again increase sort/merge execution time.

1.6. STRUCTURING YOUR INPUT/OUTPUT DATA

When you first consider the problem of sorting data, you may be faced with a large volume of information that may or may not be organized into workable units. Dividing information into records, blocks, and files helps both you and the computer identify where the data is located and control the changes or manipulations you want performed. After carefully examining the nature and content of the input data and determining the record layout and block size that best suits your needs, you must indicate, via your control stream, what size records and blocks you intend to input for processing and output after the sorting operation is completed.

Records can be divided into smaller units called *fields*. Specific fields, called *key fields* or just *keys*, are used for comparing records to arrange them in the order you want. To tell the sort program which keys to use, you must specify the size and position of the keys within records. Figure 1-1 shows key, record, and block interrelationship.

Figure 1-2 illustrates what the data contained in key fields of the first two input data record blocks might look like before the sort.

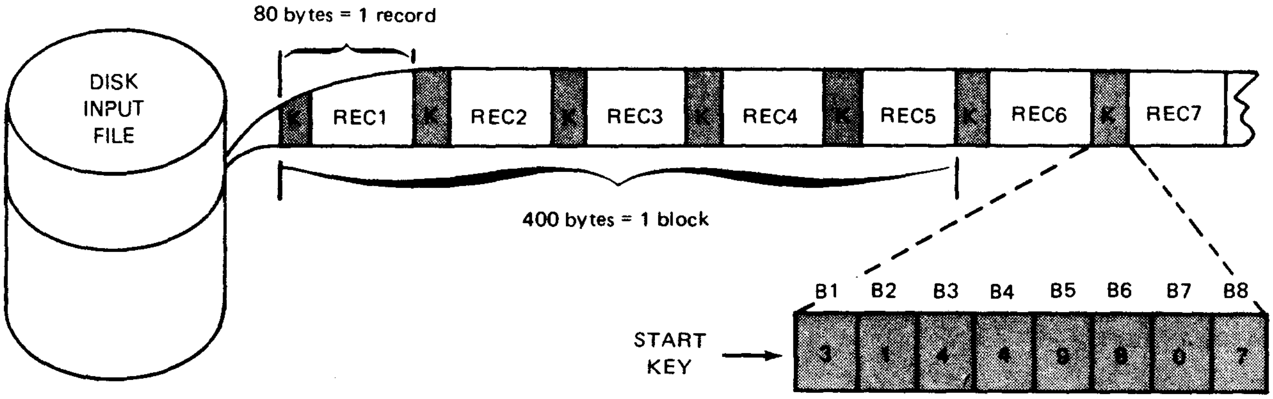


Figure 1—1. Key, Record, and Block Interrelationship

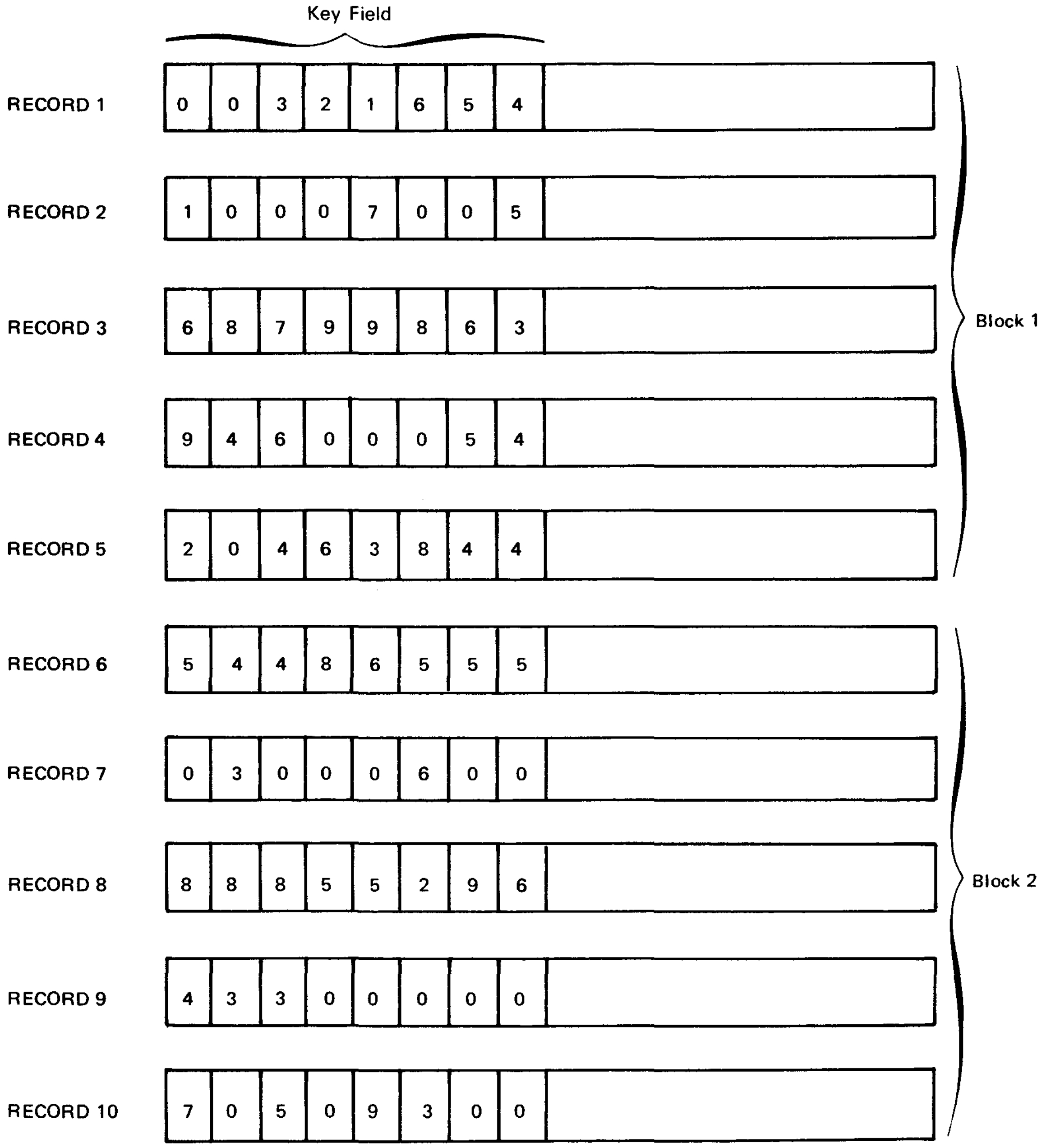
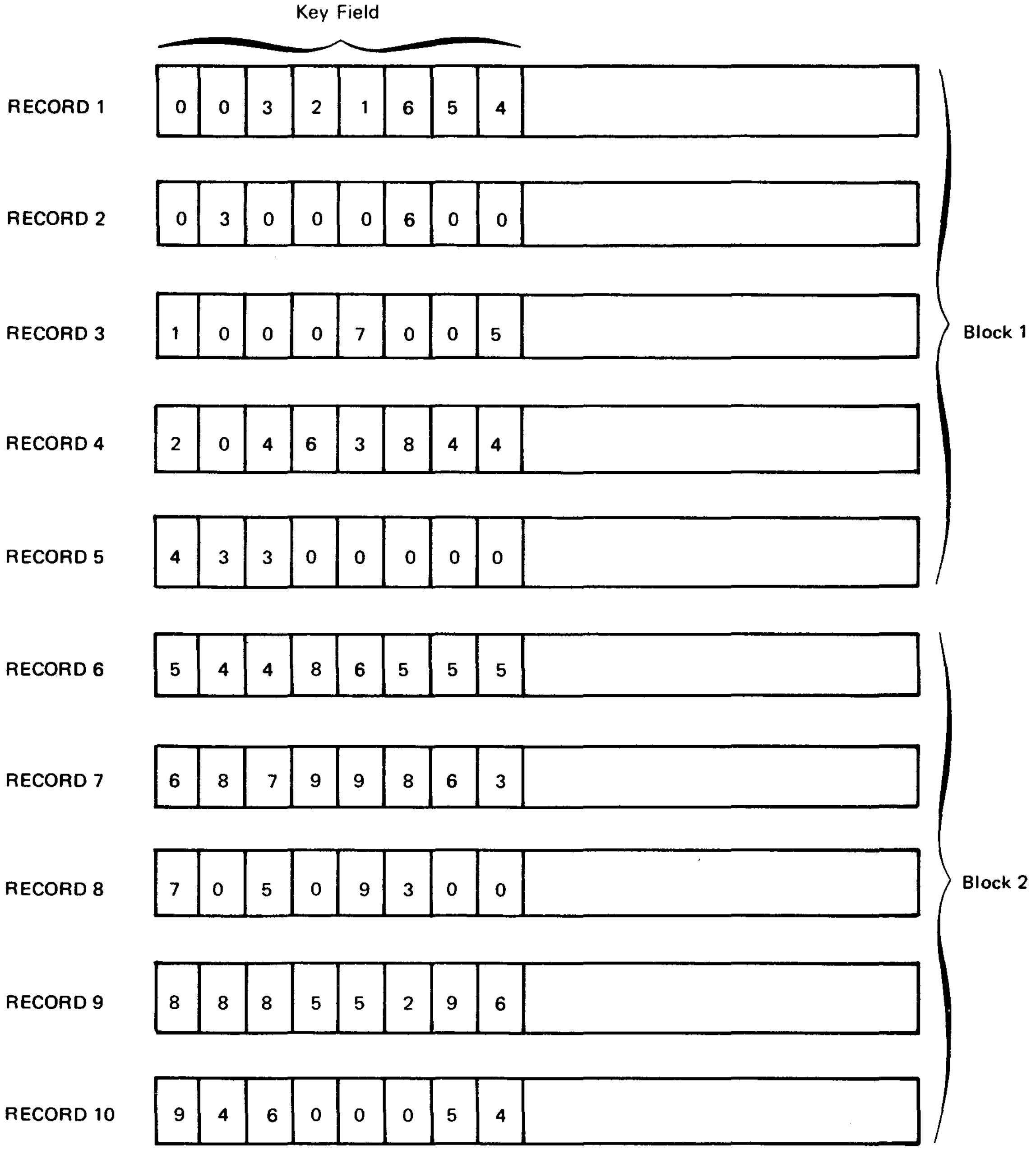


Figure 1—2. Input Data Records before Sort

Of course, your volume of data is much larger than the two 400-byte record blocks shown in Figure 1-2, but the results of sorting the records in ascending order by key fields should be as shown in Figure 1-3.



RECORD 1	0	0	3	2	1	6	5	4
RECORD 2	0	3	0	0	0	6	0	0
RECORD 3	1	0	0	0	7	0	0	5
RECORD 4	2	0	4	6	3	8	4	4
RECORD 5	4	3	3	0	0	0	0	0
RECORD 6	5	4	4	8	6	5	5	5
RECORD 7	6	8	7	9	9	8	6	3
RECORD 8	7	0	5	0	9	3	0	0
RECORD 9	8	8	8	5	5	2	9	6
RECORD 10	9	4	6	0	0	0	5	4

Figure 1—3. Data Records after Sort

1.7. RUNNING YOUR SORT JOB FROM A WORKSTATION

OS/3 provides you with the capability of running independent sort/merge interactively. This means two things:

1. You can build a control stream to execute the sort program at a workstation, as opposed to punching it on cards or writing it to a diskette.

2. You can initiate the running of the control stream from the workstation, as opposed to asking the system operator to run your job for you.

The easiest way to build a job control stream from a workstation is by using the general editor. This allows you to key in your control stream statements and have them stored on a library file. Then at some later time you can initiate the running of the program by keying in the RV system command. (An example of this procedure is provided in 4.3.)

If you are not familiar with job control, use the job control dialog for assistance. The job control dialog is an interactive facility of OS/3 that allows you to describe your job's requirements to it in English, in response to a series of questions, and then produces as its output the job control stream needed by OS/3 to run your job.

The control stream produced by the job control dialog is virtually identical to the control stream that you would have to produce if you were running your job in a batch environment. Only now, you do not have to be concerned with the intricacies of the job control language. The job control dialog eliminates this requirement on your part.

After you have answered all the questions presented to you by the job control dialog, it builds a control stream and stores it in a permanent library file for you. You can then initiate its running by simply keying in the appropriate system RUN command or, if you'd rather, you can change the contents of the control stream by using the general editor.

The procedures for activating the general editor are detailed in the general editor user guide/programmer reference, UP-9976 (current version). The procedures for activating the job control dialog and initializing the running of a job are detailed in the job control user guide, UP-9986 (current version).

Although the sample job control streams in this manual are shown on cards, the rules for preparing your sort control statements and specifications also apply to entries keyed in at a workstation.

Note that if a job has been initiated from a workstation, all messages will be displayed on the workstation rather than the system console.

2. Basic Concepts

2.1. GENERAL

Independent sort/merge, which we'll call simply *sort/merge*, is a self-contained processor that assists you in sorting and merging, or just merging, data files. You initiate it by writing a job control stream including some sort/merge control statements and job control statements that you will learn in the following sections.

Operating in a minimum system configuration, sort/merge reads your data files, sorts and merges the data according to your specifications, then writes the data to your output file. It does this with almost no user program intervention, if you supply the data files and specify the sort/merge procedures you want performed.

In addition to simplifying your sort/merge job execution, sort/merge allows you to write own-code routines to perform specialized functions that it doesn't provide or that you want to handle differently.

To use own-code routines, specify to sort/merge:

- the name of your routine;
- the approximate size of its load module; and
- the phase of the sort/merge operation from which it is to be called.

An exit code contained in the control stream automatically calls your routine to perform its function at the appropriate time. Own-code routines, their associated exit codes, and their functions are explained in greater detail in 3.3.

A good example of routines you might want to perform differently would be I/O file handling routines. You can program your own input or output file processing routines or let sort/merge handle one file while your own-code routine handles the other. Thus, you can have both convenience and flexibility when you use sort/merge.

2.2. SOFTWARE FRAMEWORK

Independent sort/merge consists of four separate operational phases, normally executed in sequence:

1. Sort initilization and assignment (Phase 1)

   This phase initializes the sort process by reading sort control statements from the job control stream. It validates the content and syntax of these statements, and examines your parameters to determine the type of sort function to be performed (tape, disk, internal sort, or merge-only). It then structures the sort/merge processor to perform only the sort functions you specify.

2. Data input and internal sort (Phase 2)

   This phase initiates the input routine and performs internal sort operations.

   - Input routine

     The input routine opens input files, validates file labels, and reads data records, one at a time, before passing them to the internal sort routine. The input routine supplied by sort/merge can be replaced by one of your own. If you use your own, you must identify it to sort/merge via a user exit code. (See 3.3.)

   - Internal sort routine

     The internal sort routine produces strings of sequenced data that are written as intermediate files to tape or disk.

3. Preliminary merge (Phase 3)

   In this phase, the data strings produced by the internal sort routine are continuously merged, producing longer and longer sequenced data strings, until only one final merge is needed to create a single string (final output string).

4. Final merge (Phase 4)

   This phase merges all strings written to the intermediate files into one sequenced string, and passes it to the sort/merge output routine or your own output routine. If you provide the output routine, you must identify the exit code required to transfer control to your routine. (See 3.3.)

In cases where the input file is biased (partially sequenced) or small enough so that one final merge produces the required output file, sort/merge bypasses the preliminary merge and proceeds to the final merge and output phase. In a merge-only operation, control proceeds directly from initialization and assignment to preliminary merge, where records are read into main storage, merged, and written to the output file.

3. Sort/Merge Requirements You Supply

3.1. GENERAL

The sort/merge requirements you supply are simple and direct. They consist of job control statements and a set of sort/merge control statements. The amount of detail involved in writing the job control stream depends upon the complexity of the sort. In essence, your sort/merge control statements tailor the available sort/merge modules to suit the requirements of a particular sort operation (disk, tape, default, etc).

By answering these three questions concerning the sort/merge operation, you can construct the specifications needed for the sort:

1. How is the sort to be performed?

2. What does the sort act upon?

3. Which file is the sort using?

The control statement, SORT, answers the first question via your parameters that supply the information needed to sort the records. You can answer question 2 by writing the RECORD sort control statement. This supplies information describing the record size and formats used in the sort. Input and output files are defined by using the INPFIL and OUTFIL control statements. To indicate the end of the sort control stream, you use the END control statement.

Before we discuss the functions of each sort control statement, let's step through the flowchart of a typical sort program (Figure 3–1).

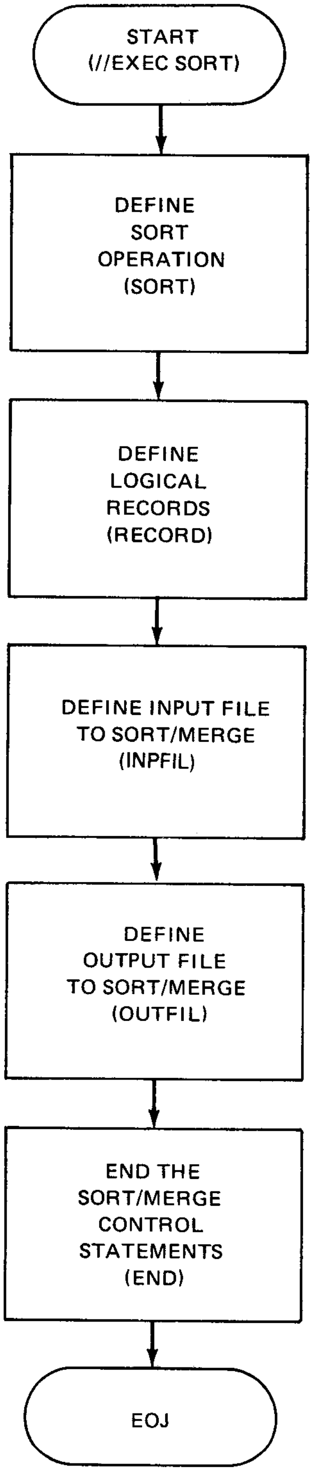


Figure 3—1. Disk Sort Program Flowchart

3.1.1. Job Control Stream

In order to schedule your program and allocate the system resources to it, you must assign a name to the job so that the system can distinguish it from other jobs. The job control statement that identifies the job and signifies the beginning of control information for the job is the JOB statement. Figure 3-2 shows the entire job control stream required for our independent disk sort program. The control stream is explained in detail in the paragraphs that follow:

```
      1        10     16
     ----------------------------------------------------------------
 1. |// JOB SRTEXMPL,,7ØØØ,9ØØØ
 2. |// DVC 2Ø // LFD PRNTR
 3. |// DVC 5Ø // VOL DSPØ28 // LBL INFILE  // LFD SORTIN1
 4. |// DVC 5Ø // VOL DSPØ28 // LBL OUTFILE // LFD SORTOUT,,INIT
 5. |// DVC 5Ø // VOL DSPØ28 // EXT ST,C,,CYL,5
 6. |// LBL $SCR1 // LFD DMØ1
 7. |// EXEC SORT
 8. |/$
 9. | SORT FIELDS=(1,8,CH),DISC=1
10. | RECORD LENGTH=(8Ø),TYPE=F
11. | INPFIL BLKSIZE=4ØØ
12. | OUTFIL BLKSIZE=4ØØ
13. | END
14. |/*
15. |/&
16. |// FIN
```

Figure 3—2. Disk Sort Coding

- Line 1

  In the first line of the sample control stream in Figure 3-2, SRTEXMPL is the 8-character alphanumeric name of your job. The double comma indicates that the job priority parameter is omitted. Because it is omitted, the system assumes normal (N) priority. The numbers 7000 and 9000 are hexadecimal values (equivalent to 28,672 and 36,864 in decimal) that represent the minimum number of main storage bytes (including job prologue) required to execute the largest job step of this job and the maximum number of main storage bytes requested but not required to execute the largest job step of this job.

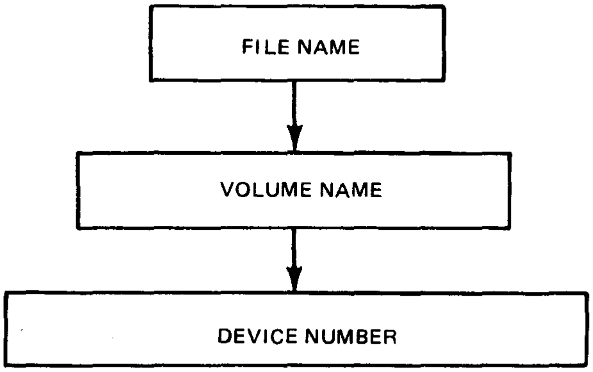
- Line 2

In order to process incoming information, the system needs hardware devices to handle the processing, and you must assign devices to various routines in your program. A device assignment set consists of at least two or as many as five job control statements; i.e., the DVC and LFD statements or the DVC, VOL, EXT, LBL, and LFD statements. Each device assignment set begins with a DVC statement that assigns a logical unit number. For specific I/O device numbers, check the list of device types and features in the job control user guide, UP-9986 (current version).

The first device usually assigned is a printer. It is needed to print messages for operator action or information. The printer must be assigned the standard name PRNTR on the LFD statement (line 2).

- Lines 3-6

Your next series of job control statements (lines 3 through 6) follow a pattern in assigning input, output, and sort work files. The pattern of specifications for each file is the file name within a volume name on a specific device.



- Lines 3-5

Your first DVC statement after the printer device assignment set assigns device number 50 to your input file named INFILE (line 3). The second DVC statement assigns the same device to your output file (line 4). Looking at the next DVC statement (line 5), notice that the same device is assigned for sort work file $SCR1. Because our input files are very low volume, this is possible; however, under normal circumstances for larger input volume, you should assign one disk device for each sort work file and another for input and output files. The sort operates more efficiently when one work file is assigned per disk. The name $SCR1 is for a temporary work file. Next you must identify the disk volume to be used. The VOL statement supplies volume serial numbers that uniquely identify tape or disk volumes (lines 3, 4, and 5). The name you assign to your input and output file volume is the alphanumeric name DSP028 (lines 3 and 4). For the sort work file volume name, you specify the same, DSP028 (line 5).

To provide disk space for sort work files and to designate information needed to create new files or extend existing disk files, you specify the EXT job control statement on the device assignment sets for each sort work file. Each EXT statement applies to the first volume specified on the immediately preceding VOL statement (line 5). Notice there is no EXT statement for either input or output files because these files already exist (lines 3 and 4). The input file was created by the data utility program that used your card input data, and the output file needed an EXT only on the first run. ST indicates that your work file is accessed via the system access technique (SAT). The C allocates contiguous space for the extent; a comma indicates omission of an optional parameter; CYL specifies that space must be allocated in cylinder; and the 5 indicates the number of cylinders allocated for the file.

Data management needs to know the file identifiers you designate for your program. Only one LBL statement is allowed per device assignment set. You specify the disk sort program's input file identifier as INFILE (line 3), the output file identifier as OUTFILE (line 4), and the sort work file identifier as $SCR1.

To associate the file information in the job control stream with the data management file definition, you must specify the standard label names SORTINn and SORTOUT on the LFD job control statement for each file (lines 3 and 4). Thus your first two LFD statements in the job control stream would specify the name SORTIN1 for the input file and SORTOUT for the output file. If more than one input file is being processed, the label names for the files must be assigned in sequence (SORTIN1 for the first file, SORTIN2 for the second file, etc). The number of input files sort/merge can process depends on the type of operation being performed (sort/merge or merge-only). For sort/merge operations, sort/merge can process up to nine tape or disk files (SORTIN1 to SORTIN9). For merge-only operations, sort/merge can process up to 16 tape or disk files (SORTIN1 to SORTIN9 for the first nine files and SORTINA to SORTING for the last seven files).

- Line 6

The INIT parameter on the LFD statement for the output file indicates that you want to start writing at the beginning of the file, overlaying its previous contents. The LFD statements for sort work files must specify the file names DM01 through DM08 or $SCR1 through $SCR8, in consecutive order, beginning with DM01 or $SCR1. Thus, the third LFD statement specifies the name DM01 (line 6).

An easier way of assigning work areas on disk would be to use WORK job control procedure (jproc) calls. A WORK jproc automatically generates a device assignment set allocating system scratch space as a work area. The format for the WORK jproc call needed for our program is // WORK1. This statement takes the place of lines 5 and 6. The WORK jproc, used without parameters, allocates 4000 256-byte blocks of scratch space on your system resident device (SYSRES) or the volume containing your system run library ($Y$RUN). You can increase the amount of work space and specify the use of other disk volumes through optional parameters. For more information about the WORK jproc, see the job control user guide, UP-9986 (current version).

- Line 7

  The EXEC statement calls the first sort/merge module into main storage.

- Line 8

  The /$ indicates the start-of-data.

- Lines 9-13

  These are your parameters for the sort parameter table being structured for your disk sort program.

- Lines 14-16

  These lines indicate the end-of-data, the end of the job stream, and the end of the card reader operation, respectively.

For details about job control and its language, see the job control user guide, UP-9986 (current version).

Figure 3-3 shows the job control stream required to execute your disk sort program. This control stream can also be built and executed from a workstation. (See 1.7.)

3.2. CONTROL STATEMENTS

Sort/merge control statements are issued from the control stream and provide the information needed to sort and merge records in your input files. Sort/merge control statements perform the following functions:

- They define the sort/merge to be performed.
- They describe your records, input and output files, and key fields.
- They specify the own-code routines you may have used during program execution.

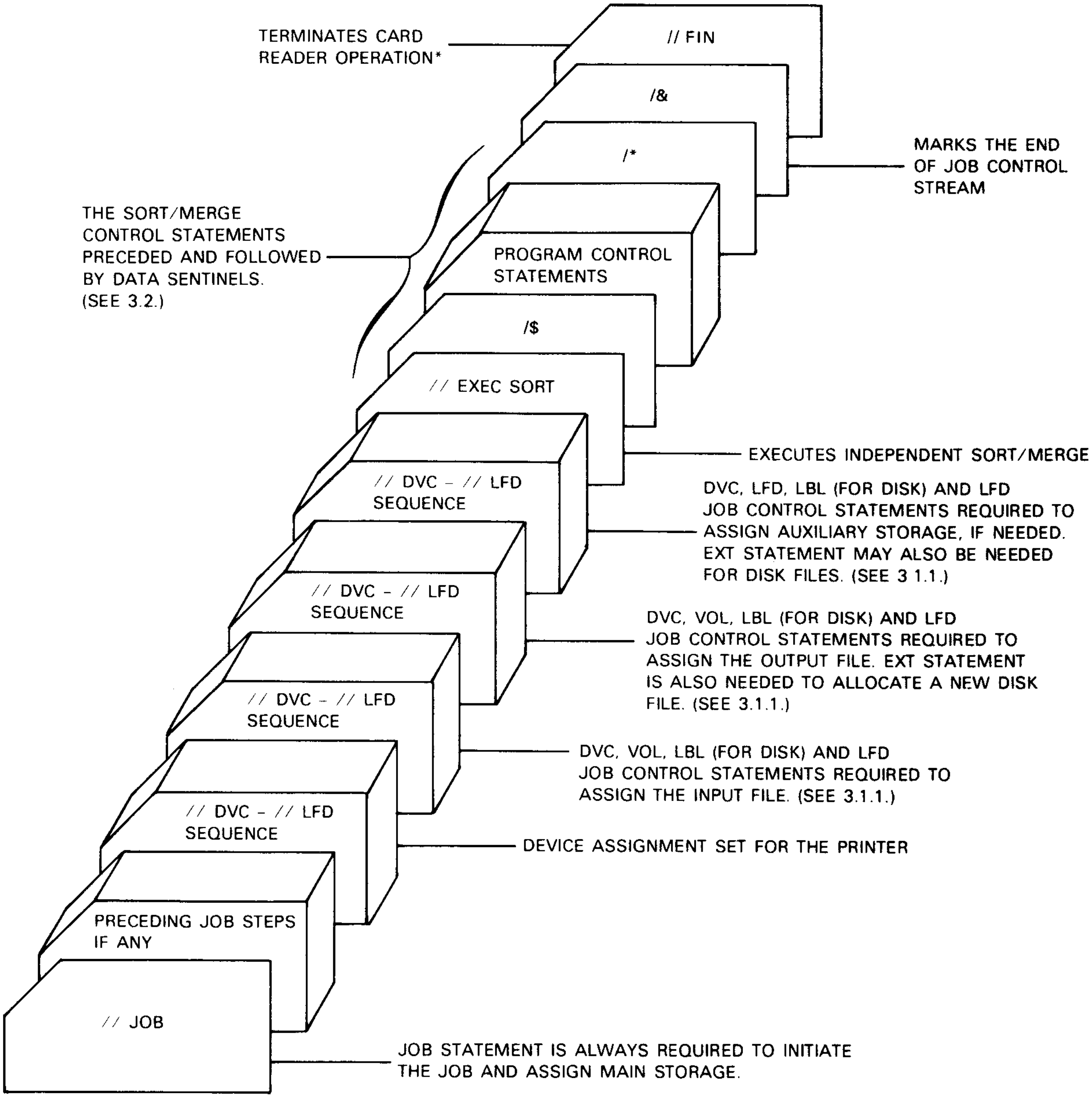
There are eight sort/merge control statements:

SORT

Defines the sort key fields, sorting sequence, auxiliary storage, and the number and size of the input files.

MERGE

Defines a merge-only job.


TERMINATES CARD READER OPERATION*
// FIN
/&
/*
MARKS THE END OF JOB CONTROL STREAM
THE SORT/MERGE CONTROL STATEMENTS PRECEDED AND FOLLOWED BY DATA SENTINELS. (SEE 3.2.)
PROGRAM CONTROL STATEMENTS
/$
// EXEC SORT
EXECUTES INDEPENDENT SORT/MERGE
// DVC - // LFD SEQUENCE
DVC, LFD, LBL (FOR DISK) AND LFD JOB CONTROL STATEMENTS REQUIRED TO ASSIGN AUXILIARY STORAGE, IF NEEDED. EXT STATEMENT MAY ALSO BE NEEDED FOR DISK FILES. (SEE 3 1.1.)
// DVC - // LFD SEQUENCE
DVC, VOL, LBL (FOR DISK) AND LFD JOB CONTROL STATEMENTS REQUIRED TO ASSIGN THE OUTPUT FILE. EXT STATEMENT IS ALSO NEEDED TO ALLOCATE A NEW DISK FILE. (SEE 3.1.1.)
// DVC - // LFD SEQUENCE
DVC, VOL, LBL (FOR DISK) AND LFD JOB CONTROL STATEMENTS REQUIRED TO ASSIGN THE INPUT FILE. (SEE 3.1.1.)
// DVC - // LFD SEQUENCE
DEVICE ASSIGNMENT SET FOR THE PRINTER
PRECEDING JOB STEPS IF ANY
// JOB
JOB STATEMENT IS ALWAYS REQUIRED TO INITIATE THE JOB AND ASSIGN MAIN STORAGE.

*Not applicable if job control stream was entered from a workstation.

Figure 3—3. Typical Sort/Merge Job Control Stream

RECORD

Defines the records to be sorted or merged.

INPFIL

Defines the input file to the sort/merge processor and specifies the procedures for opening and closing input tape files.

OUTFIL

Defines the output file to the sort/merge processor and specifies the procedures for opening and closing tapes.

OPTION

Specifies additional options and information to the sort/merge program.

MODS

Required when you include user routines in a sort/merge application. It defines your program routines with related user own-code exits. Also allows you to perform automatic data reduction of your files through the use of the system-supplied data reduction routine (DELETE).

END

Indicates that the last control statement of a related group of sort/merge control statements has been read. This is an optional control statement.

After reading the detailed discussion in the following paragraphs, you will have a basic understanding of how each control statement functions. For later quick reference, Appendix C provides a summary of the sort/merge control statements, including formats and brief descriptions of the keyword parameters.

3.2.1. Defining a Sort Operation

The SORT control statement defines a sort operation to independent sort/merge. All parameters are optional, but specifying your exact requirements will increase program efficiency. The SORT control statement defines:

- sort key fields and their sorting sequence;
- the type and number of auxiliary storage devices needed;
- the approximate number of logical records in the input file being sorted; and
- the total number of input files entered into the sort.

The format of the SORT control statement is:

```
LABEL      △OPERATION△    OPERAND
-------------------------------------------------------------------------------------
           SORT           FIELDS=⎧⎛[strt-pos-1][,lgth-1][,form-1][,seq-1]          ⎞⎫
                                 ⎪⎜[,...,[strt-pos-n][,lgth-n][,form-n]            ⎟⎪
                                 ⎪⎝  [,seq-n]]                                     ⎠⎪
                                 ⎨⎛[strt-pos-1][,lgth-1][,seq-1]                   ⎞⎬
                                 ⎪⎜[,...,[strt-pos-n][,lgth-n][,seq-n]]            ⎟⎪
                                 ⎩⎝  ,FORMAT=code:                                 ⎠⎭

                          ⎡,COPY=⎧ALL                                        ⎫⎤ See
                          ⎣      ⎩input-file-number.output-file-number       ⎭⎦ note.

                          ⎡,⎧DISC⎫=number⎤
                          ⎢ ⎨TAPE⎬       ⎥
                          ⎣ ⎩WORK⎭       ⎦

                          ⎡,FILE=⎧number⎫⎤
                          ⎣      ⎩1     ⎭⎦

                          ⎡,NOCKSM=⎧D⎫⎤
                          ⎣        ⎩T⎭⎦

                          [,SIZE=number]

                          ⎡,SORTP=output-file-number,⎤  See note.
                          ⎣       input-file-number  ⎦

                          ⎡,⎧CHPT ⎫⎤  See note.
                          ⎣ ⎩CHKPT⎭⎦
```

NOTE:

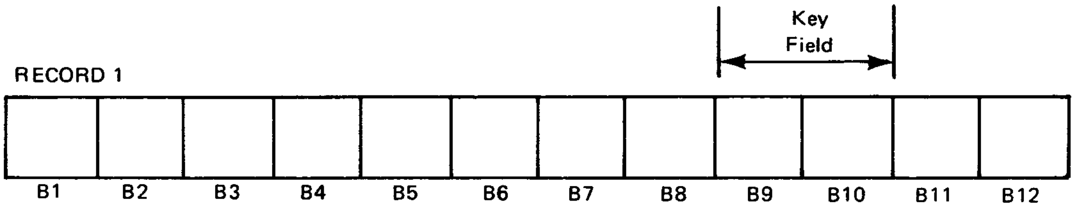
The COPY, SORTP, and CHPT/CHKPT keyword parameters are provided and accepted for compatibility with other systems; no action is performed by OS/3 sort/merge.

- Specifying key fields (FIELDS)

The FIELDS keyword parameter may be used to specify up to 12 key fields. The order in which you specify the key fields is considered by sort/merge as the order of significance. The first key field defined is the major sorting field, the second specified is the first minor sorting field, and so on.

There are two formats for the FIELDS parameter. One format has four subparameters to indicate the starting position, length, data format, and sequence for each key field. The other format has three of the same subparameters plus the FORMAT subparameter. The data format may vary for each key field or it may be the same for all key fields. If you omit the FIELDS parameter, one key field is assumed, beginning at byte 1, the same length as the record up to a maximum of 256 bytes, with character format and ascending sequence. If you specify FIELDS but omit any of the subparameters, you must retain their associated commas, except for trailing commas.

The *strt-pos* subparameter is a decimal number specifying the starting point of a key field relative to the beginning of the record. All key fields except binary key fields must start on a full-byte boundary. For example, specifying 9 as *strt-pos* indicates that the most significant byte of the key field begins at byte 9 of the record.



LEGEND:

B = Byte

The byte numbering method used by independent sort/merge is compatible with other systems.

Binary key fields may start on a bit boundary, i.e., a specific bit within a specific byte of a record. In this case, you specify *strt-pos* in *byte-bit format.* Bits are numbered from 0 to 7. As an example, assume that key field 1 starts at bit 2 of byte 9 in the record. You would specify 9.2 for the *strt-pos-1* subparameter.

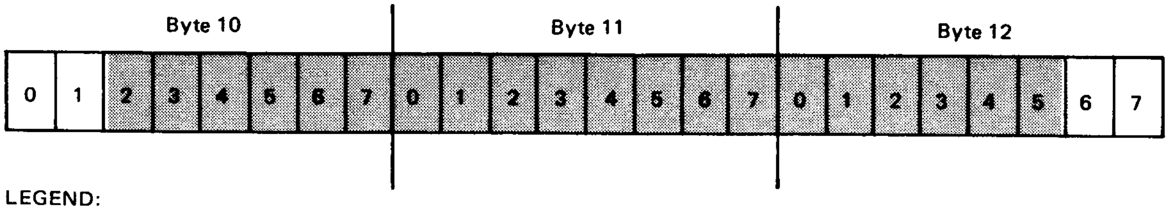
The *lgth* subparameter is a decimal number specifying the key field length in full bytes following any of these formats:

n

n.

n.0

When using binary key fields, specify key field length in byte-bit format. The number of bits specified must not exceed seven. For example, a key field length of six bits would be written as 0.6; that is, we have a key field that is six bits long. If the key field extends from bit 2 of byte 10 through bit 5 of byte 12, the *length* subparameter would be specified as 2.4.



LEGEND:

▨ Key field length

The *form* subparameter is a code consisting of two or three alphabetic characters specifying the key field's data format. If you omit this subparameter, the format is assumed to be character (CH). This subparameter is used when the data format varies for each key field. If all key fields have the same data format, you can use the *FORMAT=code* subparameter. In this case, the same codes used for the *form* subparameter are permissible; however, you must not specify the *form* subparameter when using the FORMAT subparameter. The format codes and their maximum allowable field lengths are shown in Table 3-1.

Table 3—1. Data Format Codes

Format Code	Description	Allowable Field Length
AC	EBCDIC data in ASCII collating sequence	1 - 256 bytes
ASL	ASCII numeric data leading sign	2 - 256 bytes
AST	ASCII numeric data trailing sign	2 - 256 bytes
BI	Unsigned binary	1 bit to 256 bytes
CH	Character (EBCDIC or ASCII)	1 - 256 bytes
CLO	Numeric data overpunched leading sign	1 - 256 bytes
CSL	Leading sign numeric	2 - 256 bytes
CST	Trailing sign numeric	2 - 256 bytes
CTO	Numeric data overpunched trailing sign	1 - 256 bytes
FI	Fixed-point integer	1 - 256 bytes
FL	Floating point	1 - 256 bytes
MC	Multiple character, user-specified collating sequence	1 - 256 bytes
PD	Packed decimal	1 - 32 bytes
USQ	Character, user-specified collating sequence	1 - 256 bytes
ZD	Zoned decimal	1 - 32 bytes

The *seq* subparameter specifies the sorting sequence of the key field: A for ascending order and D for descending order. If omitted, ascending sequence is assumed.

The following coding illustrates FIELDS specifications. Line 1 shows that the first key field begins at byte 1 of the record, is four bytes long, has a character format, and is to be sorted in ascending sequence. The second key field begins at byte 10 of the record and is 12 bytes long, has a binary format, and is to be sorted in ascending sequence.

Line 2 is basically the same as line 1 except that the format of both key fields is the same. Therefore, rather than defining them separately in the FIELDS parameter, they are jointly defined by means of the FORMAT parameter. The sequence subparameters are omitted, indicating that the default is to be applied. Remember that a comma must be coded in place of a missing subparameter except after the last subparameter.

Line 3 shows three key fields with varying data formats. The first two are packed decimal and the third has a character format. All fields are to be sorted in ascending sequence by default. The WORK=3 parameter indicates that three work files (either tape or disk) are assigned to the job.

```
   1        10    16
1.          SORT  FIELDS=(1,4,CH,A,1Ø,12,BI,A)
2.          SORT  FIELDS=(1,4,,1Ø,12),FORMAT=CH
3.          SORT  FIELDS=(85,3,PD,,88,3,PD,,8,9,CH),WORK=3
```

- Assigning additional work space (DISC, TAPE, WORK)

Unless the sort is small and can be executed in main storage, it requires additional work (scratch) space to perform its operations. You can choose one of three parameters on the SORT control statement as the medium used for work area: DISC, TAPE, or WORK. DISC and TAPE parameters are used to designate those media; however, the WORK parameter can indicate the number of disk or tape files assigned to sort/merge as working storage.

After designating the medium, you must specify a decimal number indicating the maximum number of files available to sort/merge as working storage. This number must not exceed 8 for disk files or 6 for tape files. You assign disk and tape files in LFD job control statements using standard name DM01,...,DM08 or $SCR1,...,$SCR8 for disk, SM01,...,SM06 for tape.

If you omit this specification, sort/merge determines the number and type of work files assigned, from the PUBS list generated by job control when devices were assigned to your job. On the other hand, if you do not assign any work files in the job control stream, the sort defaults to an internal, main storage sort, even if you include the DISC, TAPE, or WORK parameter in the SORT control statement.

The following coding illustrates two examples of how this parameter could be specified to the previously described FIELDS parameters. In line 1, the WORK parameter indicates three work files are needed for the sort and they can be either tape or disk. In line 2, the DISC parameter indicates that three work files are required for the sort and they must be on disk and are used for work files.

```
1.  |          SORT   FIELDS=(1,4,CH,A,1Ø,12,BI,A),WORK=3
2.  |          SORT   FIELDS=(1,4,,1Ø,12),FORMAT=CH,DISC=3
```

- Specifying the number of input files (FILE)

Sort/merge needs to know the total number of input files to be sorted in each run. The FILE parameter supplies this information. Your data input files are specified as SORTIN1,...,SORTIN9 and SORTINA through SORTINF on LFD statements in the job control stream. If you have more than one input file and forget to code the FILE parameter, sort/merge will process only your first input file. The following coding indicates that two input files are to be entered into the sort.

```
1         10     16
________________________
          SORT   FILE=2
```

If sort/merge locates less than the number of files specified in the FILE parameter, a warning message is issued setting the UPSI byte to hexadecimal 40. Refer to system messages programmer/operator reference, UP-8076 (current version) to determine the nature of the warning and the corrective action to be taken. Sort/merge continues to run when a warning message is issued.

- Bypassing checksum word calculation (NOCKSM)

A checksum word is normally calculated for each data block written to work files (disk or tape). The checksum is the logical sum of all the data in the block. When the block is read, the checksum is recalculated and compared with the previous calculation to verify data integrity. A miscompare indicates a hardware problem because data integrity in reading or writing data was not maintained. You can bypass this specification by coding the NOCKSM parameter. This increases overall sort performance. D means omit disk checksum, and T means omit tape checksum. The following coding indicates that no checksum word is to be calculated for each data block written to the disk work file.

```
SORT   NOCKSM=D
```

By specifying the NOCKSM parameter, you can save a considerable amount of processing time.

- Specifying the number of records in the input file (SIZE)

Another parameter, SIZE, specifies the approximate number of records in the input file. If you use the CALCAREA parameter in the OPTION sort control statement, the SIZE parameter is required for an accurate calculation of optimum sort time and disk work space. The following coding indicates that 3500 records are contained in the input file. If you do not specify the number of records in the input file, sort/merge assumes a file size of 25,000 records. You can greatly increase sort/merge program efficiency by supplying this information.

```
SORT   SIZE=35ØØ
```

3.2.2. Defining Data Records

The RECORD sort control statement defines the type and length of the data records being sorted or merged. It also allows you to delete records from a file by character identification and byte position. The RECORD statement is not generally required for disk input files unless records are variable length or if length modifications are to be made; however, if you omit the RECORD control statement, you also must omit the INPFIL control statement. Both must be omitted or both must be present. It is required for tape input files, and when input processing is handled by a user exit routine (3.3). The RECORD sort control statement format is:

LABEL	△OPERATION△	OPERAND
	RECORD	{LENGTH=(lgth-1[,lgth-2][,lgth-3][,lgth-4] [,lgth-5]) RCSZ=bytes} [,TYPE={D **F** V}] [,BIN={bytes (min-bytes,size-1,freq-1[,...,size-n, freq-1])}] [,DEBLANK=(delete-char,byte-position)]

- Specifying record length (LENGTH and RCSZ)

The LENGTH parameter can list one to five lengths. Each length specifies definite information about fixed- or variable-length records for input, internal sort, and output phases of the sort/merge operation. *Lgth-1* specifies the decimal number of bytes in the input record for fixed-length records or the maximum input record length for variable-length records. (This length must not exceed 32,767 bytes.) *Lgth-2* gives the length (in bytes) of each record released to the internal sort phase for fixed-length records or the maximum-length record for variable-length records. If omitted, sort/merge assumes the *lgth-1* specification for this parameter. Do not specify *lgth-2* for a merge-only operation; however, you must retain its associated comma. *Lgth-3* specifies the output record length in bytes for fixed-length records or maximum output record length for variable-length records written to tape or single-partition disk output files. Output record lengths written to multipartitioned disk files are specified via the RCSZ keyword parameter in the OUTFIL control statement (3.2.4). If *lgth-3* is omitted, *lgth-2* is assumed for sort operations, and *lgth-1* for merge-only operations. *Lgth-4* is a decimal number specifying the minimum input record length in bytes for variable-length records, and *lgth-5* specifies the number of bytes in variable-length input records that appear most frequently in the input file. If you have variable-length records and omit *lgth-4* and *lgth-5,* this information is obtained from the BIN parameter. The LENGTH parameter is required for a tag sort (3.2.6.1).

The other parameter alternative is RCSZ. It is a more general specification that indicates the record length for fixed-length records or the maximum record size for variable-length records.

If input is from sequential or direct access disk files and you fail to specify either LENGTH or RCSZ parameters and also the BLKSIZE parameter on the INPFIL sort control statement, sort/merge defaults to the input record size supplied by data management.

In the following coding, line 1 illustrates the LENGTH parameter for variable-length records. The maximum input record length is 120 bytes; maximum length of each variable-length record released to the internal sort is 100 bytes; maximum length of each variable-length record written to the output file is 30; minimum input record length is 65 bytes; and the number of bytes in the most frequently appearing records of the input file is 65. Line 2 illustrates the more general specification of the RCSZ parameter, giving the number of bytes in each fixed-length record or the maximum record size for variable-length records.

```
    1        10     16
1. |         RECORD LENGTH=(12Ø,1ØØ,3Ø,65,65)
2. |         RECORD RCSZ=8Ø
```

- Specifying record type (TYPE)

The TYPE parameter specifies the type (D, F, or V) of records to be processed by sort/merge. Specifications in this keyword apply only to tape and single-partition disk files. Specifications for data record types contained in multipartitioned disk files are defined in the TYPE keyword parameter of the OUTFIL control statement (3.2.4). TYPE=D specifies that data records are ASCII, variable-length records. An F specifies fixed-length records. This type of data record is assumed by default if you omit the TYPE parameter. The V specifies variable-length records. The following coding specifies a fixed-length record format and a record size of 80 bytes.

```
RECORD TYPE=F,RCSZ=8Ø
```

- Calculating subrecord size (BIN)

To conserve main storage space and provide optimum processing speed, variable-length records are divided into fixed-length subrecords (fixed-bin sizes). The BIN parameter either specifies the size of these subrecords or supplies the information needed by sort/merge to calculate the subrecord size. The BIN parameter has two formats. In the first format, you can specify the decimal number of bytes in each bin. In the second format, you indicate the minimum number of bytes in a bin (a number large enough to accommodate all sort key fields within the record plus the 4-byte record length field), the number of bytes in the most frequently occurring record size, and a number specifying either the percentage or estimated number of the most frequently occurring records. If the number is less than 100, sort/merge assumes this specification to be a percentage. If 100 or more, the number is assumed to be an estimate of the number of the specified-size records in the file to be sorted. A maximum of six different variable-length record sizes and their frequencies may be specified. The sum of the records specified does not have to total 100 percent of the file.

Assuming that all five *lgth* subparameters of the LENGTH parameter were not specified, the following coding on line 1 specifies the number of bytes in each bin of a variable-length record. Line 2 shows information you supply to sort/merge to calculate the bin size: minimum of 30 bytes per bin, a most frequently occurring record length of 80 bytes, and approximately two hundred 80-byte records in the file to be sorted.

```
     1          10     16
    ----------------------------------
1. |            RECORD BIN=4Ø
2. |            RECORD BIN=(3Ø,8Ø,2ØØ)
```

You should code the BIN parameter if you use the RCSZ parameter or if you omit the *lgth-4* and *lgth-5* subparameters of the LENGTH keyword parameter. If BIN and LENGTH are both omitted, sort/merge calculates bin size from the *lgth* specifications of the FIELDS parameter.

- Deleting records from a file (DEBLANK)

  The DEBLANK parameter of the RECORD sort control statement allows you to delete specific records from the file by defining a specific character and identifying its byte position. The first subparameter *(delete-char)* indicates the character that, when found in the byte specified by the *byte-position* subparameter, causes the record to be deleted from the file. The second subparameter *(byte-position)* denotes the byte position of the character used as a deletion indicator. In the following coding example, the DEBLANK parameter specifies that any records with the character A in byte 4 are to be deleted.

  ```
           RECORD DEBLANK=(A,4)
  ```

3.2.3. Defining the Input File

The INPFIL control statement defines your input file to sort/merge and specifies open and close procedures for tape files. It is not required if input files are on disk or format label diskette.

```
LABEL       △OPERATION△     OPERAND
----------------------------------------------------------------------------
            INPFIL          ⎡BLKSIZE=⎧bytes                       ⎫⎤
                            ⎣        ⎩(bytes-1[,...,bytes-8])⎭⎦
                            [,BUFOFF=n]
                            [,BYPASS]
                            ⎡,CLOSE=⎧NORWD⎫⎤
                            ⎢       ⎨RWD  ⎬⎥
                            ⎢       ⎨RWI  ⎬⎥
                            ⎣       ⎩UNLD ⎭⎦
                            ⎡,DATA=⎧A⎫⎤
                            ⎣      ⎩E⎭⎦
                            [,EXIT]
                            ⎡,OPEN=⎧NORWD⎫⎤
                            ⎣      ⎩RWD  ⎭⎦
                            [,SKIPBYTE=n]
                            ⎡,VOLUME=⎧vol                   ⎫⎤   See note.
                            ⎣        ⎩(vol-1[,...,vol-8])⎭⎦
```

NOTE:

The VOLUME parameter is provided and accepted for compatibility with other systems; no action is performed by OS/3 sort/merge.

- Specifying block size (BLKSIZE)

The BLKSIZE parameter has two formats, one for sort/merge application and one for a merge-only application. The first format applies to the sort/merge operation. It specifies the number of bytes in each input file block when all input file blocks are the same length or the length of the largest input block when block size varies. If the largest block length is not specified when variable-length blocks are involved, data will be lost through truncation when the larger blocks are encountered. The following coding example illustrates the first format.

```
1       10    16
------------------------------
        INPFIL BLKSIZE=8ØØ
```

The second format is required in a merge-only operation when input files have different block sizes. The subparameters *(bytes-1[,...,bytes-8])* specify block size, in bytes, of each input file in order. For example, the first subparameter *(bytes-1)* specifies the number of bytes per block for input file 1, the second subparameter specifies the block size for input file 2, and so on. If you specify only one block size for an input file *(bytes-1* subparameter), all additional files are assumed to have blocks equal in length. The following coding example illustrates three input files, each of a different block size.

```
1        10    16
-------------------------------------
         INPFIL BLKSIZE=(8ØØ,12ØØ,16ØØ)
```

If you omit the BLKSIZE parameter and also the RCSZ keyword parameter on the RECORD control statement, sort/merge assumes that all input blocks are the size of the first block processed.

- Defining block prefix length for ASCII data (BUFOFF)

  When tape data is in ASCII code, your program needs information prefixing each block of data. This is because ASCII has a 7-bit character code and there must be a compensation between ASCII and EBCDIC character code lengths, as well as space allotted for header information. The BUFOFF (buffer offset) parameter defines the length of a block prefix when you use an ASCII data block structure. You indicate a decimal number from 0 to 99 on the BUFOFF parameter. The following coding example shows this parameter as well as the data format parameter specifying ASCII code. These two parameters are usually coded together.

```
         INPFIL BUFOFF=3Ø,DATA=A
```

- Ignoring unreadable blocks of input data (BYPASS)

  Another optional INPFIL parameter is the BYPASS parameter. It has no associated values but when you specify BYPASS, you direct the sort/merge input phase to ignore all unreadable blocks of data on the input file. Sort/merge does not keep a record of the blocks ignored. The following example shows an 800-block input file for which all unreadable data blocks are to be ignored by the sort/merge input phase.

```
         INPFIL BLKSIZE=8ØØ,BYPASS,CLOSE=NORWD
```

- Closing input tape files (CLOSE)

  There are several rewind methods for closing input tape files:

  - CLOSE=NORWD

    Does not rewind input tape file on closing.

- CLOSE=RWD

  Rewinds the input tape file to load point on closing.

- CLOSE=RWI or CLOSE=UNLD

  Rewinds with interlock on closing.

To understand when to use these parameters, consider the conditions that require their use. For example, you would want to specify NORWD if your tape contained multiple files and you were planning to run successive sorts on file 1 and file 2. You wouldn't return to the tape load point after sorting file 1 because you want to leave the tape prepositioned on file 2 for the second sort. Suppose, on the other hand, you wanted to perform two successive sorts on the same file. After the first sort at the end of the input tape or input file, the tape needs to be rewound to the beginning of the file for the second sort on a different key. This situation would require the RWD specification. The rewind with interlock (RWI) and unload (UNLD) subparameters perform identical functions; i.e., the tape is rewound with interlock, making those files inaccessible unless the operator intervenes. The RWI or UNLD is a protective procedure you might specify if you didn't want to risk writing over the tape files.

If you omit the CLOSE parameter, the default is UNLD. The following example shows that no rewinding is performed upon closing the input file.

```
1           10      16
-------------------------------------------------
            INPFIL  BLKSIZE=8ØØ,BYPASS,CLOSE=NORWD
```

- ■ Specifying data format (DATA)

  You can specify two data formats: ASCII or EBCDIC. *DATA=A* indicates data recorded in ASCII; *DATA=E* indicates data recorded in EBCDIC. EBCDIC is the default assumed if you omit the DATA parameter. In the following coding, line 1 has no data format specified, so the system assumes a normal default condition of E (EBCDIC). On line 2, ASCII data format is specified.

```
1. |          INPFIL  BLKSIZE=8ØØ
2. |          INPFIL  BUFOFF=3Ø,DATA=A
```

- ■ Providing your own input routines (EXIT)

  Instead of letting sort/merge provide the input routines, sometimes you may want to supply your own routine for reading the input file. The EXIT parameter indicates that you are providing the entire input routine. EXIT has no associated value, and you may not code any other INPFIL parameters when you specify it. The following coding shows this by indicating that you want to read the input file via your own input routine.

```
            INPFIL  EXIT
```

- Opening input tape files (OPEN)

  Just as there are several rewind methods for closing input tape files, there are two rewind methods for opening input tape files:

  - OPEN=NORWD

    Specifies no rewind to load point on opening and is used when you don't want to begin processing an input file at the beginning of the tape but at some prepositional location.

  - OPEN=RWD

    Specifies rewind to load point on opening and is used when you want to begin processing at the tape load point. RWD is the assumed default if you omit the OPEN parameter.

- Indicating the first data record in a block (SKIPBYTE)

  A record block doesn't always begin with the first data record. The SKIPBYTE parameter specifies the location of the first data record in relation to the beginning of the block.

  The *n* is a decimal number you supply to indicate that the first data record is n+1 from the beginning of the block; that is, the first n bytes are to be skipped. In the following coding example, the first data record starts at byte 11.

  ```
  INPFIL SKIPBYTE=1Ø
  ```

3.2.4. Defining the Output File

The OUTFIL control statement defines output procedures to sort/merge. All parameters are optional. They:

- define block size;
- define rewind alternatives for opening and closing the output file;
- indicate if you are providing the output routine; and
- indicate if a tape mark is to be written before the first data record of each volume in the output file.

Notice that the following OUTFIL control statement format contains parameters similar to the INPFIL control statement (3.2.3).

```
LABEL    △OPERATION△    OPERAND
______________________________________________________________
         OUTFIL         [BLKSIZE=bytes]
                        [,BUFOFF=n]
                        ⎡,CLOSE=⎧NORWD⎫⎤
                        ⎢       ⎪RWD  ⎪⎥
                        ⎢       ⎨RWI  ⎬⎥
                        ⎣       ⎩UNLD ⎭⎦
                        [,EXIT]
                        ⎡,FILTYPE=⎧IRAM⎫⎤  See note.
                        ⎢         ⎨NI  ⎬⎥
                        ⎣         ⎩SAM ⎭⎦
                        [,NOTPMK]
                        ⎡,NPTN=⎧number⎫⎤  See note.
                        ⎣      ⎩1     ⎭⎦
                        ⎡,OPEN=⎧NORWD⎫⎤
                        ⎣      ⎩RWD  ⎭⎦
                        [RCSZ=bytes]
                        [SIZE=percentage] ⎫  See note.
                        [TYPE=type]       ⎭
                        [UOS=ext-percent]
```

NOTE:

The FILTYPE, NPTN, SIZE, and TYPE parameters are provided and accepted for compatibility with other systems; no action is performed by OS/3 sort/merge.

These parameters are valid for disks as well as tapes. The OUTFIL control statement is not needed if both input and output files are on disk and the output file is to have the same block size and record size as the input files. If the output file has been predefined, you should not specify the first optional parameter on the LFD job control statement, indicating the maximum number of extents in the file. In addition, if you do use the OUTFIL control statement for a previously defined output file, all file specifications must be the same as when the file was created, or an error will result.

- Specifying block size (BLKSIZE)

  The BLKSIZE parameter can specify the number of bytes in the output data block written to a tape or disk output file.

  If block size is needed and you do not specify the BLKSIZE parameter or the RCSZ parameter in any sort control statement, sort/merge assumes a block size equal to the input block. The following coding indicates that you are writing 400-byte data blocks to the output file.

```
1           10     16
_____________________________________
            OUTFIL BLKSIZE=400
```

- Defining block prefix length for ASCII data (BUFOFF)

  The BUFOFF parameter specifies the length of a block prefix for an ASCII data block structure. This buffer offset specifies a decimal number from 0 to 99, indicating a special adjustment for data written in ASCII character code. The BUFOFF example indicates an adjustment of 20 bytes for an ASCII format file.

```
            OUTFIL BUFOFF=20
```

- Closing output tape files (CLOSE)

  The CLOSE parameter specifies rewind alternatives for closing tape output files. All the specifications are identical to the CLOSE parameter specifications for the INPFIL control statement (3.2.3): NORWD indicates no rewind on closing a tape output file; RWD, rewind on closing; RWI or UNLD, rewind with interlock on closing. Similarly, the UNLD is assumed by default if you omit the CLOSE parameter on the OUTFIL control statement.

- Providing your own output routines (EXIT)

  To specify that you are providing your own output routine for writing the entire output file, write the EXIT parameter. No other parameters may be specified on the OUTFIL control statement when you specify EXIT. There is no assigned value for the EXIT parameter.

- Omitting tape marks (NOTPMK)

  If you do not want a tape mark written before the first data record of each volume in the tape output file, you can indicate this via the NOTPMK parameter. It has no associated values and is coded as follows:

  ```
  OUTFIL NOTPMK
  ```

  Omitting the NOTPMK causes a tape mark to be written before the first record of each volume in the tape output file.

- Opening tape output files (OPEN)

  To specify rewind alternatives on opening tape output files, use the OPEN parameter values identical to the OPEN parameter of the INPFIL control statement; i.e., OPEN=NORWD for no rewind to load point and OPEN=RWD for rewinding the output tape file to the load point. If you omit the OPEN parameter on OUTFIL, independent sort/merge assumes the RWD specification by default.

- Specifying output record size (RCSZ)

  In addition to the block size, you can also indicate the number of bytes in the output record.

  If you fail to specify the RCSZ parameter, sort/merge supplies the same number of bytes as the input record.

- Extending output file (UOS)

  After you've assigned disk output file space, your number of records might increase and you might find that you exceed the amount of output file space allocated. The UOS parameter solves this problem by allowing the file to be dynamically extended by data management when it becomes full. When you submitted an EXT job control statement for your output file, you specified, in the third parameter, the number of cylinders you wanted for secondary storage allocation. In the UOS parameter, you indicate what percentage of that amount you want the file extended by when it requires more space. You can specify up to 100 percent. If you want to extend your file by 100 percent, you can specify 100 or you can omit the specification because the default is 100 percent. If you want to extend your file by less than 100 percent, you must specify a percentage in the UOS parameter. Suppose, for example, you specified five cylinders in the third parameter of the EXT statement. You might want to specify 20 percent as the percentage of the amount you want the file extended by when it requires more space. To do this, you would include the following control statement:

```
1       10    16
_________________________________
        OUTFIL UOS=2Ø
```

If your file requires more space, data management will extend your file by one cylinder at a time rather than by five cylinders at a time.

3.2.5. Ending Input to Sort/Merge

The END control statement is optionally used to notify sort/merge that all sort/merge control statements have been processed and that program execution may begin. This control statement has no parameters and is coded as follows:

```
END
```

The END control statement is not to be specified when sort/merge specifications are embedded in a jproc. Otherwise, the run processor mistakenly interprets the END control statement as the END directive for the jproc.

3.2.6. Handling Special Sort/Merge Specifications

The OPTION control statement consists of optional parameters that supply sort/merge with additional information not applicable to any of the other sort/merge control statements. Parameters with built-in default conditions automatically become effective if you omit them. The OPTION control statement format follows:

LABEL	△OPERATION△	OPERAND
	OPTION	$\left[\text{ADDROUT=}\begin{Bmatrix}\text{A}\\\text{D}\end{Bmatrix}\right]$
		$\left[,\begin{Bmatrix}\text{CALCAREA}\\\text{CALCAREA=}\begin{Bmatrix}\text{NO}\\\text{YES}\end{Bmatrix}\end{Bmatrix}\right]$

(continued)

```
LABEL        △OPERATION△       OPERAND
----------------------------------------------------------------------------
             OPTION (cont)     ⎡,CSPRAM=⎧NO ⎫⎤
                               ⎣        ⎩YES⎭⎦
                               [,KEYLEN=bytes]
                               [,LABEL=(output,input-1[,....,input-n],work)]
                               ⎡,PRINT=⎧ALL     ⎫⎤
                               ⎢       ⎨CRITICAL⎬⎥
                               ⎣       ⎩NONE    ⎭⎦
                               ⎡,RESERV=⎧work-file-name                          ⎫⎤
                               ⎣        ⎩(work-file-name[,output-file-name])⎭⎦
                               [,RESTART]
                               ⎡,SHARE=⎧work-file-name                         ⎫⎤
                               ⎣       ⎩(work-file-name[,input-file-name])⎭⎦
                               [,STORAGE=bytes]
                               [,VERIFY]
                               [,ALTWK]   ⎫
                               [,DUMP]    ⎪
                               [,ERASE]   ⎪
                               [,ROUTE]   ⎬ See note.
                               [,SORTIN]  ⎪
                               [,SORTOUT] ⎪
                               [,SORTWK]  ⎭
```

NOTE:

The ALTWK, DUMP, ERASE, ROUTE, SORTIN, SORTOUT, and SORTWK parameters are provided and accepted for compatibility with other systems; no action is performed by OS/3 sort/merge.

You may specify the OPTION control statement parameters in any order. With this in mind, we plan to discuss those parameters concerning special specifications for disk access input records (ADDROUT and KEYLEN), those concerning input, output, and work files (LABEL, RESERV, SHARE, VERIFY, CALCAREA, and STORAGE), and those that affect external control (CSPRAM, PRINT, and RESTART).

3.2.6.1. Disk Access Input Records

- Sorting by direct access addresses and key fields (ADDROUT)

  The ADDROUT parameter is required when sort/merge must perform a *tag sort*. The tag sort is a method of constructing a file that contains only the direct access addresses, or the addresses and key fields, of the records in the original file. If you provide the input through an own-code routine, you must obtain the disk address of each input record and place it into the 10-byte address field of the new tag sort record. The total length of all key fields per tag sort record, including the 10-byte record address field, cannot exceed 256 bytes. A tag sort can be performed only when input is from a nonindexed file. Multiple input files cannot be tag sorted.

If you specify A on the ADDROUT parameter, the final output is only the direct access addresses of the input records. D specifies that the output file is to contain both the direct access addresses and sort key fields of each record. The following coding example illustrates the ADDROUT parameter.

```
1       10    16
---------------------------------------------
        OPTION ADDROUT=D,CALCAREA=YES,CSPRAM=YES
```

Figures 3-4, 3-5, and 3-6 show unsorted key fields from four records and the resulting records returned to your output file after the tag sort. It is not the intent to show actual record formats in these figures, but to illustrate the concept of record sorting by key fields and the outputs produced by a tag sort operation.

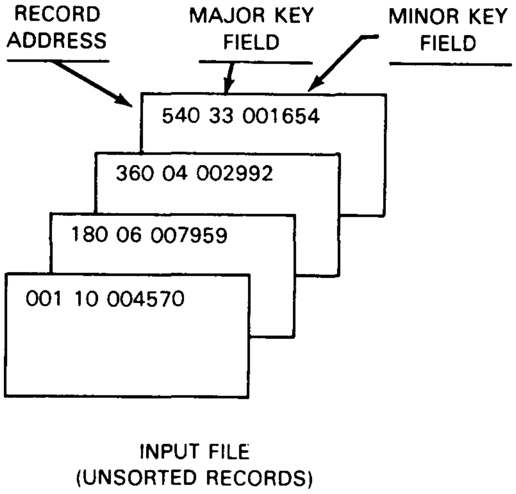


Figure 3—4. Input File, Unsorted Records (Additional Data Fields Not Shown)

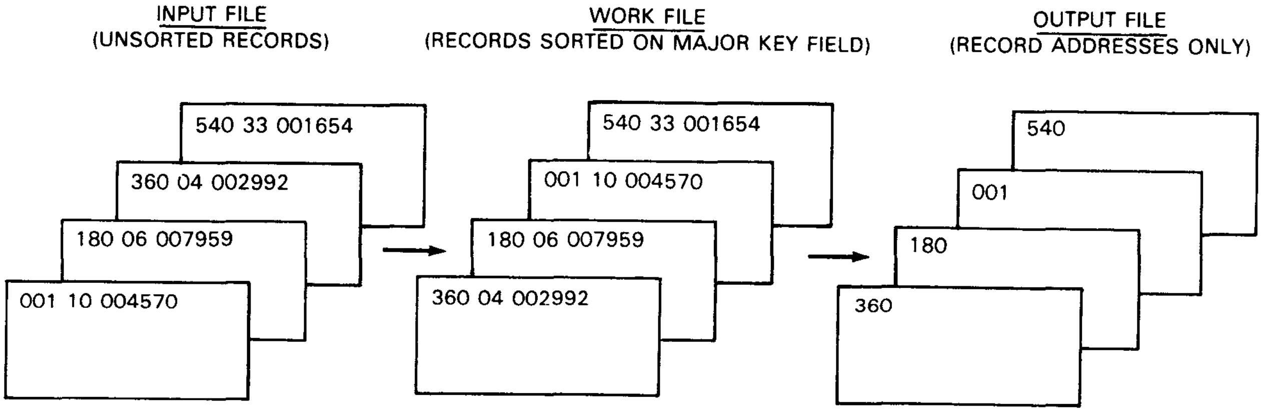


Figure 3—5. Tag-Sorted Output File when ADDROUT=A

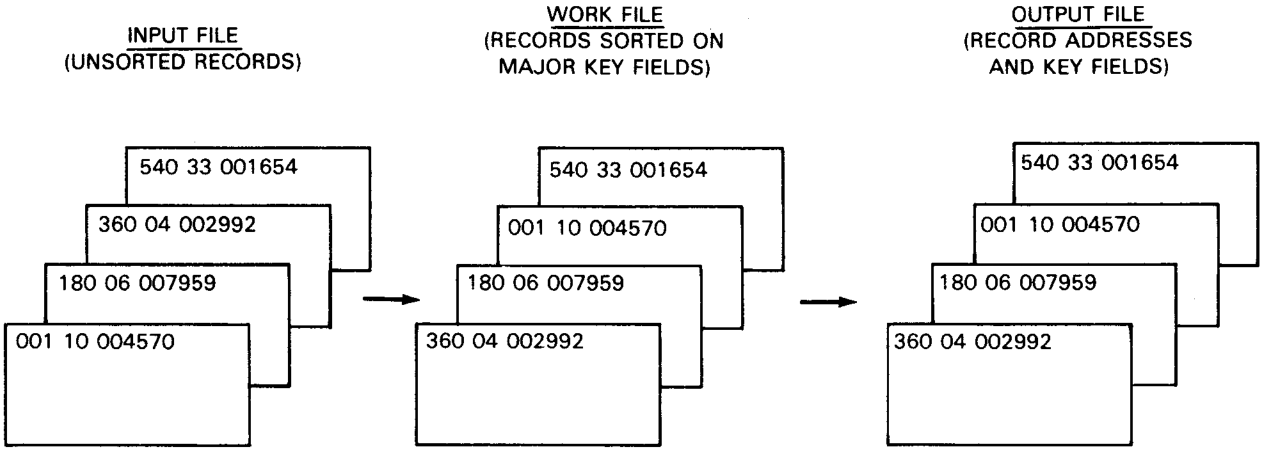


Figure 3—6. Tag-Sorted Output File when ADDROUT=D

The following restrictions apply when ADDROUT is used:

1. Output block size must be a multiple of:

   a. 10 bytes for ADDROUT=A

   b. The sum of the sort key field lengths plus 10 bytes for ADDROUT=D

2. The *lgth-2* and *lgth-3* values in the length specification of the RECORD control statement must be used. The *lgth-2* value must be 10 bytes plus the sum of the sort key field lengths. The *lgth-3* value must be:

   a. 10-bytes for ADDROUT=A

   b. 10 bytes plus the sum of the sort key field lengths (after any user modification at exit E35) for ADDROUT=D

- Specifying number of bytes in keys (KEYLEN)

Focusing now on the use of direct access for input records, note that record blocks may be preceded by a key. This key is used by data management and has an entirely different purpose from the sort key field represented on the FIELDS parameter of the SORT control statement. You use the KEYLEN parameter to specify a decimal number of bytes in each key. A sample KEYLEN parameter is shown in the following coding. If you do not specify the KEYLEN parameter, it is assumed that blocks do not have keys.

```
1        10    16
------------------------------------------------------
         OPTION STORAGE=18ØØØ,KEYLEN=1Ø,PRINT=NONE,VERIFY
```

3.2.6.2. Input, Output, and Work Files

- Specifying label types (LABEL)

  Files may have standard or nonstandard labels or may be on unlabeled tapes. The LABEL parameter specifies the label types for output, input, and work files. If files have nonstandard labels, you must process those labels yourself via the user exit codes E11 and E31 (see 3.3). The LABEL parameter specifies one of the following 1-character codes describing the label type for output, input, and work files, in that order:

N	Nonstandard labels
S	Standard labels
U	Unlabeled tapes

  You may specify a maximum of nine input files. Standard labels are assumed on all files if you omit the LABEL parameter. The following example illustrates the coding of a LABEL parameter, indicating standard labels for output, input, and work files.

```
1        10    16
_____________________________________________________________
         OPTION STORAGE=18ØØ,LABEL=(S,S,S),RESERV=(SMØ6),
               SHARE=(SMØ1)
```

- Reserving a tape unit for work file (RESERV)

  By coding the RESERV parameter, you can reserve for your output data file a tape unit assigned to the sort/merge as a scratch or work file. This allows the tape unit to function as a work file during the input and intermediate phases of the sort/merge operation and as the device for the output data file during the output phase.

  Sort/merge provides messages at the system console instructing you when to unload the scratch tape and mount the output tape. The *work-file-name* specifies the standard sort work file name (SM01,...,SM06) of the reserved tape device. The same device cannot be assigned for both the RESERV and SHARE keyword parameters. The device is associated with this name through an LFD job control statement. If you use the second format, you can also specify the *output-file-name,* and the console messages will include the name of the output file the operator is to mount.

- Using a tape unit as an input device and a work file (SHARE)

  Like the RESERV parameter, the SHARE parameter specifies the double use of one tape device by input and work files. It allows a tape unit assigned to sort/merge to be used as the input device during the input phase and as a work file during the preliminary merge and final merge phases of the sort operation.

Messages at the system console tell you when to unload an input tape and mount a scratch tape. The *work-file-name* specifies the standard sort work file name (SM01,...,SM06) of the shared tape device. The device is associated with this file name through an LFD statement in the job control stream. If you use the second format, you can use the *input-file-name* subparameter to specify the name of the input file that is to be shared, and a console message will be provided telling the operator which input tape to dismount. Remember to assign different device numbers to your files specified on the RESERV and SHARE parameters.

- Checking output blocks (VERIFY)

  For output accuracy, coding the VERIFY parameter specifies that each output block will be checked to ensure that it is written correctly when the output file is on a direct access device. The VERIFY parameter has no associated values.

- Calculating optimum working-storage area (CALCAREA)

  In a disk sort, sort/merge can calculate for you the optimum working-storage area required for efficient sorting operations based on the parameters you supply on the sort control statements. After its calculations, it displays execution information pertinent to the defined operation. It does these calculations when you specify the CALCAREA parameter. The information it supplies is the estimated sort time in minutes and the number of cylinders sort/merge requires for work space. If you specify CALCAREA=YES, the sort is executed. If you specify CALCAREA or CALCAREA=NO, optimum working-storage is calculated and execution information is displayed, but the sort is not executed. If you use the CALCAREA parameter, the SIZE parameter on the SORT statement should be specified; otherwise, the default value of 25,000 records will be used in calculating working-storage area and the result may not be accurate.

- Using less main storage (STORAGE)

  If you want the sort to use less main storage than is allocated in the job region, you can indicate that decimal number of bytes on the STORAGE parameter. Otherwise, sort/merge obtains this information from your job control statements. The following coding shows an example of the STORAGE parameter.

```
1        10    16
         OPTION STORAGE=18ØØØ,KEYLEN=1Ø,PRINT=NONE,VERIFY
```

3.2.6.3. External Control

- Including parameters at execution time (CSPRAM)

  Occasionally, you may need to include certain parameters from the job control stream at execution time. To tell sort/merge you are submitting parameters in this way, you must use the CSPRAM parameter. The keyword parameters that sort/merge can accept via the control stream at run time are BIN, DISC, NOCKSM, RSERV, RESUME, SHARE, and TAPE. You enter these keyword parameters via PARAM job control statements. There are three values to choose from on the CSPRAM parameter: NO, or YES. NO specifies that sort/merge parameters will not be accessed from the control stream. This specification is assumed by default if you omit the parameter. The YES keyword specifies that the control stream is tested for the presence of // PARAM statements. If they are present, they are read. An example of this parameter is shown in the following coding.

```
1        10    16
         OPTION ADDROUT=D,CALCAREA=YES,CSPRAM=YES
```

- Specifying printing options for error messages (PRINT)

  When sort/merge encounters errors, it provides error messages. These error messages are interpreted in the system messages programmer/operator reference, UP-8076 (current version). The way to specify the printing options for these error messages is the PRINT parameter. There are three values to choose from: ALL, CRITICAL, and NONE. If omitted, the default provided for the PRINT parameter is ALL, which specifies that all messages and control statements are written to the job log for subsequent printing. The CRITICAL specification indicates that only fatal error messages are to be written to the job log. NONE specifies that no messages are to be written to the log. The coding example for the STORAGE parameter also shows an example of this specification. Error messages that are written to the job log are displayed on the operator console.

- Restarting a tape sort (RESTART)

  When a tape sort has been interrupted, and you want to restart it at the last recovery point, you write the RESTART parameter. There are no values associated with this parameter. The system console interfaces with sort/merge by displaying messages concerning sort/merge execution status, fatal errors, possible recovery information, and directions for mounting, demounting, and labeling tapes during the sort/merge process. The recovery information supplied by the system console is the recovery point number or last cycle break executed before the sort was interrupted. You need this number to restart your job. By coding this number on a PARAM job control statement on the RESUME keyword parameter and by indicating the RESTART and CSPRAM keyword parameters on the OPTION sort control statement (3.2.6), you can restart your sort job by resubmitting the job control stream. The PARAM job control statement must immediately follow the /* statement for the sort/merge control statements.

The OPTION control statement that you must include for a tape restart is coded in line 1 as follows.

```
    1          10      16
1.  |          OPTION  RESTART,CSPRAM=YES
2.  | // PARAM RESUME=(PASS,Ø61)
```

Line 2 is an example of a PARAM statement that could indicate the recovery number you just read from the system console.

3.3. EXIT CODES

The independent sort/merge allows you to pass control during certain phases of its operation to the system-supplied DELETE data reduction routine or your own-code routines that you write in BAL.

The points where you cause control to be passed are called *exit codes.* Each exit code allows definite functions to be performed and is associated with a specific phase of the sort. The exit codes, the functions each exit code allows you to perform, and the phase associated with each exit code are listed in Table 3–2.

Table 3—2. Exit Codes (Allowable Functions and Associated Phases)

Phase	Exit Code	Function
Data input (phase 1)①	E11	Input file label processing
	E15	Input file processing: – Reading input files – Counting input records – Inserting records – Deleting records – Modifying record size – Modifying record content – Modifying control fields
	E18	Read error processing
Final merge (phase 3)②	E31	Output file label processing
	E32	Input file processing during merge-only application: – Modifying record content – Modifying control fields – Record substitution
	E35	Output file processing (Same as for E15 except applicable to output files)
	E38	Read error processing during merge-only application
	E39	Write error processing for direct access devices
All (phases 1-3)③	E65	Record sequencing
	E75	Data reduction
	E84	User-defined collation sequencing

① Specified by PH1.
② Specified by PH3.
③ Specified by PH6 for exit code E65, PH7 for E75, and PH8 for E84.

If you want to perform a particular function, you must choose the proper exit code for the function, assign it to the operational phase to which it is associated, and specify the name of the load module that contains the routine that performs the function. This is done through the use of the MODS control statement which is discussed in detail for the DELETE data reduction routine in 3.3.1 and in 3.3.2 for your own code routines.

3.3.1. Exiting to DELETE Data Reduction Routine

As noted, a MODS control statement is required to cause the sort to exit to this routine. The format of the MODS control statement for the DELETE data reduction routine is:

LABEL	△OPERATION△	OPERAND
	MODS	PH7=(DELETE,,E75)

This routine allows you to have duplicate records automatically deleted from your files during the execution of the sort. This is all that this routine does; that is, it is a stand–alone routine that does not require you to provide any own code. All that is required to use this routine is that you include this form of the MODS control statement in the control stream for your sort program. (An example of this is provided in 4.5.) If you require other functions, you must exit to a BAL own–code routine as described in 3.3.2.

3.3.2. Exiting to Your BAL Own-Code Routines

In order to activate your own-code routines (load modules), you need a MODS control statement to define exits. The MODS statement specifies the sort/merge phase in which your own-code routine load module is to be executed (*PHn*), the name (*module-name*) and approximate length (*length*) of your module, and the exit-code numbers (*exit-code*) that are to be used. If you plan to use your own routines in more than one phase, you must specify each phase individually by repeating the *PHn* parameter for each phase exiting to your own-code routines. The three exit codes which apply to all three phases (E65, E75, and E84) must be specified individually by means of an identifying code that takes the place of a phase number. Follow the first *PHn* parameter and subparameters with a comma, a continuation character coded in column 72 (if necessary), and another *PHn* parameter with its set of subparameters defining exits for that phase.

The MODS control statement format for BAL own-code routines is:

LABEL	△OPERATION△	OPERAND
	MODS	PHn=(module-name[,length],exit-code [,...,exit-code]) [,...,PHn=(module-name [,length],exit-code [,...,exit-code])]

You must always code the phase number *PHn, module-name,* and *exit-code*; however, the *length* subparameter is optional. You can choose from the following decimal numbers specifying the sort/merge phase in which your own-code routine is to execute or identifying a routine that is executed during all phases.

n	Description
1	Phase 1 (input and internal sort). Exit codes are E11, E15, and E18.
3	Phase 3 (final merge and output). Exit codes are E31, E32, E35, E38, and E39.
6	All phases (record sequencing routine). Exit code is E65.
7	All phases (data reduction routine). Exit code is E75.
8	All phases (user-defined collation sequencing). Exit code is E84.

The *module-name* subparameter may contain up to eight characters; the first character must be alphabetic. The name you specify is the name of your own-code routine's load module. Module *length* specifies the number of decimal bytes in the load module. If you omit the *length* subparameter, sort/merge obtains the length from the load module header record. *Exit-code* specifies the exit code numbers (i.e., E11, E15) listed as subparameters on the phase to which they apply. You format the MODS control statement according to the routines you want to use during sort/merge operations; for example, if you're going to provide your own input file label processing routine and input file reading routine, you format the MODS control statement to reflect the exit codes for input label (E11) and input file (E15) processing (line 1).

```
      1         10    16
 1.   |         MODS  PH1=(PHASE1,385Ø,E11,E15),
 2.   |               PH3=(PHASE3,27ØØ,E31,E35)
```

Since both exits pertain to data input (phase 1), you indicate PH1 (line 1). In addition, you specify the name of your own routine's load module (PHASE1) and the approximate number of bytes (3850) required for your load module. Line 2 illustrates the continuation of phase specifications. Here, you specify that your routine's load module named PHASE3 contains 2700 bytes and is to receive control during phase 3 of the sort/merge execution via exits 31 and 35. These exits process output file labels and read output files.

3.3.3. Using Exit Codes

Independent sort/merge can exit to your own-code routines only during the phases you specify in the MODS control statement. Control is passed to your own-code routine via a branch table and general registers. When the exit is reached, register 15 is loaded with the address of the first location of the exit routine load module, which must be the branch table. This branch table must be covered by specifying register 15 as the base register. Before your routine assumes control from sort/merge, you must save certain register contents. (See 3.3.4.) Table 3–2 helps to categorize exit codes within their related phases. The following discussion describes the functions that your own-code routines are permitted to perform.

3.3.3.1. Input File Label Processing

When you specify nonstandard labels for tape input files on the OPTION control statement, you must enter that tape input file label processing routine via exit code E11. E11 enables sort/merge to gain entry to your own-code nonstandard label processing routine.

If you omit exit 11 (you do not specify E11 on the MODS control statement) for input files that contain nonstandard or user labels, the labels are bypassed. Exit code E11 enables the input files to interface with your own-code routine.

3.3.3.2. Input File Processing

Input file processing (exit code E15) enables sort/merge to enter your own-code routine to perform any of the following functions during data input (phase 1).

- Read input files
- Count input records
- Insert records
- Delete records
- Lengthen or shorten records
- Modify record contents or control fields

When you specify E15 on your MODS control statement, exit code E15 receives control each time an input record passes to internal sort. Since your routine may perform a number of different functions on an E15 exit code, you must tell the sort what you decide to do after the exit occurs. You supply this information to the sort by placing an action code in the action word, a 4-byte area in main storage that sort/merge sets up when it detects the EXIT parameter on the INPFIL sort control statement. You may place any of the following action codes in the action word.

Action Code	Action Taken
0	Accept the record by modifying it prior to entering the internal sort or by taking no action
4	Delete the record from the sort
8	Request no return to exit code (E15 in this case) because exit use is completed
12	Create a new record and insert it into the sort

The action word is a 1-word (4-byte) entry in the parameter list, a table built by sort/merge to specify location of records and information affecting record processing (3.3.7).

3.3.3.3. Input File Read Error Processing

When you specify exit code E18 on the MODS control statement, sort/merge enters your own-code read error processing routine for your input file. You write only the BR 14 instruction to return to the sort program.

If you specify the BYPASS parameter on the INPFIL control statement and exit E18 on the MODS control statement, the E18 specification overrides the BYPASS.

3.3.3.4. Output File Label Processing

The exit-code E31 specification on your MODS control statement enables independent sort/merge to enter your own-code nonstandard label processing routine for the output file. Functionally, it corresponds with the E11 exit for input files and interfaces the output file via the ULABEL data management keyword parameter and the DMLAB imperative macroinstruction.

3.3.3.5. Output File Processing

Exit code E35 enables independent sort/merge to enter your own-code routine for output file processing during the final merge and output phase (phase 3). Any of the following functions may be used in your own-code output routine:

- Write output records
- Count output records
- Insert records
- Lengthen or shorten records
- Modify record contents or control fields

By specifying exit code E35 on your MODS control statement, you indicate that E35 should receive control each time an output record passes to final merge phase (phase 3). Like exit code E15 for input file processing, there are a number of possible functions your own-code output routine can perform. Thus, you must tell the sort what you decide to do after exit E35 occurs. To supply this information, you place action codes in the action word of exit code E35 in the exit parameter list. (The action word is a 1-word (4-byte) field identified by the parameter list; a table built by sort/merge to specify the location of records and information affecting record processing. Additional details concerning parameter list format are given in 3.3.7.) The action codes allowed are:

Action Code	Action Taken
0	No change
4	Delete the record from the sort
8	Request no return to exit
12	Insert and accept a new record for output

Action codes 4 and 8 are valid only when the EXIT parameter is specified in the OUTFIL control statement (3.2.4). If the EXIT parameter is not specified, then all of the action codes listed are valid until sort/merge passes the last record to the exit 35 routine. At this time, 8 and 12 are the only valid action codes.

After the last record is written, control is passed to the end-of-file routine. In this case, the first entry in the exit parameter table is 0 contained in a 1-word (4-byte) field which normally contains the address of the next record to be sent to the output buffer.

Exit code E35 is not valid in a merge-only application.

3.3.3.6. Write Error Processing for Direct Access Devices

There is no recovery from this type of error; however, you may supply your own-code routine to handle a direct-access-device writing error by writing an E39 exit code on your MODS control statement. When a write error occurs, sort/merge enters your own-code write error processing routine for your output file. The BR 14 instruction returns control to sort/merge.

3.3.3.7. Record Sequencing

Exit code E65 is used during phases 1, 2, and 3 for entering your own-code record sequencing routine from sort/merge. Sort/merge enters your routine each time two records are compared, to determine which will be sorted first. You decide the record sorting sequence in your routine.

The first instruction in your own-code routine must be the USING assembler directive, assigning register 15 as a base register. Your program receives the addresses of the two records to be compared in registers 11 and 12. For variable-length records, the addresses supplied are those of the first bin of each record. The 4-byte record length field is part of the first bin. You pass the result of the comparison to sort/merge via condition code settings. If the record for the address in register 11 is first, the condition code should be set to low (cc=1). If the record for the address in register 12 is first, you set the condition code to high (cc=2). If the sequence of the two records is arbitrary, you set the condition code to equal (cc=0). Control is returned to sort/merge via a branch to register 14.

3.3.3.8. Data Reduction

When sort/merge encounters records with equal keys, it normally retains both records in an arbitrary sequence. If you want to eliminate duplication in your files, you can do so by using the system-supplied DELETE routine or by using exit code E75 to enter your own-code data reduction routine. To use the system-supplied automatic data reduction routine, include the following version of the MODS control statement in your control stream.

```
MODS PH7=(DELETE,,E75)
```

When processed, this MODS statement causes sort/merge to load and execute the load module for the automatic data reduction routine called DELETE. The routine reduces data by deleting duplicate records whenever they are encountered. You can use the DELETE routine for input files that contain either fixed-length or variable-length records but not both types. In your own-code routine, each time two records with equal keys are processed, you may:

- delete one of the duplicate records;
- combine data contained in the duplicate records to create a new record; or
- use a combination of retaining, deleting, and combining duplicate records.

The first instruction in your own-code routine must be the USING assembler directive specifying register 15 as a base register. Sort/merge places the addresses of the two records with equal keys in registers 11 and 12. If one of the records is to be deleted, normally the address of the record to be retained is in register 11 and the deleted record address is in register 12, unless in your routine you overlay the address in register 11, thereby forcing the deletion of the address in register 11 and saving the address in register 12. Your program returns control to sort/merge four bytes beyond the address specified in register 14.

If you want to save the contents of both records, control must be returned to sort/merge at the address specified in register 14.

3.3.3.9. User-Defined Collation Sequencing

Exit code E84 is used whenever you want to specify an alternate collating sequence to the one supplied by sort/merge or to sort two or more different characters that have the same collating values. To determine which operation you wish to perform, E84 is used in conjunction with the character format code (USQ and MC) specified in the FIELDS keyword parameter of the SORT and MERGE control statements. Because both USQ (user-specified collating sequence) and MC (multiple character) specifications use the E84 exit code, they are mutually exclusive within a sort or merge operation. The distinction between the two is that the USQ specification for character code format requires you to provide sort/merge with two 256-byte translation tables at exit code E84 when control is passed to your own-code routine. The first table (input) must translate and collate the input record key fields, and the second table (output) must return the fields to their original format. You only require one table, the input table, when you use the MC specification. The translation table is used only for comparison purposes and not to change the actual data in the record. (See Appendix B for OS/3 EBCDIC and ASCII standard collating sequences.)

3.3.4. An Example of Exit Code Use

Figure 3–7 finishes the discussion of exit codes by illustrating the coding required to build a branch table, set up a base register, save and restore general registers, and provide a return address to sort/merge. It also shows how you can write your own input/output routine. The exit code used in this example is E15 as specified by the MODS control statement (line 113). This example modifies the record contents (line 66).

```
1        10    16
// JOB SRTEXMP5,,8ØØØ,AØØØ
// DVC 2Ø // LFD PRNTR
// WORK1   )
// WORK2   } =// ASM
// EXEC ASM)
/$
PHASE1   START Ø
         USING *,15
         B     E11                      ERROR
         B     E15
         B     E18                      ERROR
E11      EQU   *
E18      EQU   *
         CANCEL
*
*              DATA MANAGEMENT WORK AREA
*
         VTOC  CDIB=YES
         USING CD$CDIB,5
INPUT    CDIB
MYRIB    RIB   BFSZ=4ØØ,RCSZ=8Ø,IOA1=BUFF1,IOA2=BUFF2,IORG=(2),
               RCFM=FIXBLK,OPTN=YES
         DS    OH
BUFF1    DS    CL4ØØ
BUFF2    DS    CL4ØØ
SAVE     DS    1ØF
SAVE     DS    1ØF                      ROUTINE SAVE AREA
*
*              EXIT E15 ROUTINE
*
E15      EQU   *
         STM   13,6,SAVE                SAVE REGISTERS
         LR    4,15                     SET NEW BASE REGISTER FOR YOUR ROUTINE
         DROP  15                       FREE R15
         USING PHASE1,4                 SET R4 AS BASE REGISTER
TAG      BC    Ø,NEXT                   FALL THROUGH ON FIRST TIME
         OPEN  INPUT,(MYRIB)            OPEN THE INPUT FILE
         L     5,=A(INPUT)              LOAD R5 WITH CDIB ADDRESS
         TM    CD$ISVCC,L'CD$ISUCC
         BZ    IOERROR
         MVI   TAG+1,X'FØ'              ALTER BRANCH FOR NEXT ENTRY
NEXT     EQU   *
         DMINP INPUT                    GET A RECORD
         L     5,=A(INPUT)              LOAD R5 WITH CDIB ADDRESS
         TM    CD$IEOF,L'CD$IEOF
         BO    EOF
         TM    CD$ISUCC,L'CD$ISUCC
         BZ    IOERROR
         BAL   5,MOD                    MODIFY THE RECORD
         L     1,SAVE+16                LOAD PARAM LIST ADDR INTO REG 1
         ST    2,Ø(Ø,1)                 STORE THE ADDRESS OF THE RECORD
                                        IN THE PARAM LIST
         L     3,4(1)                   GET ADDR OF ACTION CODE
         MVC   Ø(4,3),INSERT            SET INSERT IN ACTION WORD
RETURN   EQU   *
         LM    13,6,SAVE                RESTORE REGISTERS
         BR    14                       RETURN TO INDEPENDENT S/M
*
```

Figure 3—7. Coding Example for Using Exit Code E15 (Part 1 of 2)

```
1        10    16
*
MOD      EQU   *
*
*
*              ROUTINE TO MODIFY THE RECORD
*
*
         MVC   8(45,2),MESSAGE         ADD MESSAGE TO RECORD
         BR    5                       RETURN
*
EOF      EQU   *                       END OF DATA ROUTINE
         L     1,SAVE+16               LOAD PARAM LIST ADDR INTO REG1
         L     3,4(1)                  GET ACTION WORD ADDR
         MVC   0(4,3),EOD              SET ACTION CODE 8 FOR END OF
*                                      EXIT ACTIVITY
         CLOSE INPUT                   CLOSE INPUT ROUTINE
         L     5,=A(INPUT)             LOAD R5 WITH CDIB ADDRESS
         TM    CD$ISUCC,L'CD$ISUCC
         BZ    IOERROR
         B     RETURN                  RETURN
*
IOERROR  EQU   *                       ERROR HANDLING ROUTINE
         B     E18                     BRANCH TO CANCEL
*
INSERT   DC    F'12'
EOD      DC    F'8'
MESSAGE  DC    CL45'THIS RECORD HAS BEEN MODIFIED THROUGH EXIT 15'
         END
/*
// WORK1
// EXEC LNKEDT
/$
 LOADM PHASE1
 INCLUDE PHASE1
/*
// DVC 50
// VOL DSP028
// LBL MYFILE1
// LFD INPUT
// DVC 50
// VOL DSP028
// EXT ,,,CYL,4
// LBL MYFILE2
// LFD SORTOUT,,INIT
// DVC RES
// EXT ST,C,,CYL,5        )
// LBL $SCR1              }=//DM01 WORK1 BLK=20000
// LFD DM01               )
// EXEC SORT,$Y$RUN
/$
 SORT FIELDS=(1,8,CH)
 RECORD RCSZ=80,TYPE=F
 INPFIL EXIT
 OUTFIL BLKSIZE=80
 MODS PH1=(PHASE1,,E15)
 END
/*
/&
// FIN
```

Figure 3—7. Coding Example for Using Exit Code E15 (Part 2 of 2)

Notice the test under mask (TM) and branch on condition (BO and BZ) instructions at lines 45-48. After DMINP macroinstruction finishes its other operations on a file, it sets control information in the CDIB that reflects the status of the file. After the end-of-file indicator CD$IEOF is tested (line 45), control branches (line 46) to the end-of-file routine EOF if the indicator is set, or to the next sequential instruction if the indicator is not set. The next instruction (line 47) tests the successful function indicator CD$ISUCC and sets the condition accordingly. The branch instruction at line 48 then causes a branch to the error routine IOERROR if the CD$ISUCC is not set; otherwise, control passes to the instruction at line 49.

Data management requires I/O buffers to be half-word aligned (line 23). If you want to make your program device independent, your I/O buffer areas must be in multiples of 256 bytes, or, in this case, 512 bytes instead of the 400 bytes shown in the example (lines 24 and 25).

Since your routines are associated with phases of sort/merge, all routines for a particular phase must be linked together as one load module. Exit code E15 that we are using in this coding example, as well as exit codes E11 and E18, belong to the data input and internal sort phase (phase 1). Thus, to access exit code E15, we must code the branch table for the phase 1 exits in the order shown (lines 9-11). At execution time, your sort control statements have told sort/merge that:

- your key field starts in byte 1, extends eight bytes, and has a character data format (line 109);
- your records are fixed type and 80 bytes long (line 110);
- you intend to use a phase 1 exit code (E15) for your own-code input processing routine (line 113);
- your output block size is 80 bytes (line 112);
- the load module name for your input routine is PHASE1 (line 113); and
- you are supplying a routine to modify input records (lines 56-63 and line 111).

The first step you must take in your own-code routine is to save those registers used by sort/merge (line 32). You set up a new base register for your own-code routine (line 35), open the input file, and read records (lines 37-43). To modify records, you branch out to your record modification routine (line 49) and return inline to load the parameter list address in register 1 and store the address of the modified record in the parameter list (lines 50 and 51). This record modification information is needed by sort/merge to continue its succeeding phases and give the sorted record results you want. Therefore, you must move the address of the changed record to the parameter list. After this move, the parameter list contains the changed record address and, thus, the information needed by sort/merge in the first full word addressed by register 1.

You must then tell sort/merge how to create a new record and to insert it into the sort. First, you get the address of the action code (line 53) and place it in register 3. Then, you insert the action code, a DC assembler directive indicating a full-word constant with the number 12 (line 83), into the parameter list at the second full-word position (line 54). When the end of input file is reached and all record inserts and modifications have been made, you change your action code in the parameter table by getting the action word address (line 71) and setting the action code to 8 (line 72). You close the input file (line 74).

The final step in coding is to restore the registers used by independent sort/merge (line 56) and return to sort/merge (line 57). The end-of-file routine (line 69) is labeled EOF, the address of your routine handling the end-of-data condition.

Certain registers do play an important role in implementing the transfer from sort/merge to your own-code routines. As we examine the use and function of these registers in the following discussion, refer frequently to Figure 3–7 to understand how registers help implement the linkage.

3.3.5. General Purpose Registers

Four general purpose registers play important roles in enabling sort/merge to communicate with your own-code routines and to provide linkage between its modules and your routine. These registers are 1, 13, 14, and 15.

In cases where several functions may be performed by your routine during a particular sort/merge phase, sort/merge requires an action code from your routine to tell it what to do with a record or how to handle the situation at hand. Your parameter list is the place where sort/merge receives this action code, but first it needs the address of the parameter list. Sort/merge places the parameter list address in register 1. The possible action code response your routine must make depends upon the exit-code function being performed. Action codes for the various exit codes used in sort/merge are described in 3.3.3.2. The format for your own-code parameter list is discussed in 3.3.7.

In your own-code routine, you use registers for base registers and movement of addresses. The contents of any registers you use during execution of your own-code routine must be saved in a save area and restored to their original values before returning control to sort/merge. This save area must be 18 full words (72 bytes) long, full-word aligned, and defined by a DS assembler directive in your program. Sort/merge places the address of a save area in register 13.

Before sort/merge enters your own-code routine via the exit code, it must save the address of the next sequential instruction in its module. This address is known as the return address. Sort/merge places its return address in register 14. At its conclusion, your own-code routine must then branch back to sort/merge via register 14.

When exiting to a user routine, sort/merge loads register 15 with the address of the exit routine. The appropriate exit routine is then entered via the branch table (Table 3–3), which is required at the beginning of each exit load module.

3.3.6. Providing a Branch for User Own-Code Exits

Sort/merge locates and enters each own-code routine via a branch table entry, which must also be the first coding of the own-code load module. Table 3-3 indicates the table format and the phases with which each exit code is associated. The right half of Table 3-3 represents the actual user coding required to build the branch table. (See lines 9 through 11 in Figure 3-7 for an illustration of this coding.)

Table 3—3. Branch Table Format

Applicable Phase of Sort/Merge Operation	Typical Table Format
Data input and internal sort (phase 1)	entry B E11 B E15 B E18
Final merge and output (phase 3)	entry B E31 B E32 B E35 B E38 B E39

When sort/merge gives control to your own-code routine, it loads register 15 with the address of the first branch table entry and then enters your routine at the appropriate branch table entry. Own-code routines for the same phase of the sort must be linked together as one common load module. Each routine used at a given exit must have its own point of entry (exit code) listed in the branch table.

Several exit codes (E65, E75, E84) link the sort to your own-code routines differently. Because functions provided at these exits are common to all phases of sort/merge, they are linked as independent load modules rather than as one common load module by phase association. The point of entry for exit codes E65 and E75 is the first position in the load module. Exit code E84, however, has a unique problem. It is used for entering either a user-defined, alternate collating sequence (USQ) or a user-defined collating sequence for sorting two or more different characters having equal collating values. Because exit code E84 has no executable code of its own, your coding must show the address for entry to your translation tables as the first word of the load module. If your own-code routine is for an alternate collating sequence, you must provide two table entry addresses; one for the input translation table and one for the output translation table. For MC (multiple character) sorting, you need only one table entry address because this function uses only the input table for comparison purposes (conversion is not performed during this operation thereby eliminating the need for the output translation table). The format for exit code E84 is:

```
entry   DC  A(convto-address)

        DC  A(convfrm-address)
```

3.3.7. Formatting the Exit Parameter List

Sort/merge uses information it finds in the parameter list to locate your response (action code). The action code you place in the action word tells sort/merge how to process your records. Register 1 points to the first entry in the parameter list when control passes to your own-code routine. Each entry in the parameter list is a 1-word (4-byte) entry. Table 3–4 describes the parameter list information required and the parameter positions it occupies in the list.

Some exits do not use the parameter list. These exits work according to data management requirements, using data management imperative macros to locate own-code routines and return to the sort. Current versions of the consolidated data management macro language user guide/programmer reference, UP-9979, and the basic data management user guide, UP-8068, discuss these imperative macros in more detail.

Table 3—4. Parameter List Format

Exits	Parameter Position No.	Function of Parameter
E11, E31	None	Interfaces conform to data management conventions for user label routines.*
E15	1 2	Address of record in the input buffer Address of action word
E32	1	Address of record in input buffer
E35	1 2 3 4	Address of record next scheduled for the output buffer Address of last record in the output buffer Address of action word Address of sequence check word
E18, E38	None	See data management conventions for error routines.*
E39	None	See conventions for data management error routines.*
E65	None	See E65 description (3.3.3.7).
E75	None	See E75 description (3.3.3.8).
E84	None	No executable code at this exit

* Refer to the consolidated data management macro language user guide/programmer reference, UP-9979 or basic data management user guide, UP-8068 (current versions).

The parameter list is used by three exits: E15, E32, and E35. E15 uses a 2-word parameter list, E32 uses a 1-word parameter list, and E35 uses a 4-word parameter list. All other exits use other interface conventions.

3.3.8. Job Control for the Own-Code Routine

After you have written your own-code routine, you must assemble and link it (see Figure 3-7, lines 3-5 and 88-93) before you can use the routine in your sort/merge program. Perhaps you want to assemble and link your routine and execute the sort/merge in a single run as described in Figure 3-7. In this case, sort/merge finds the load module in the job run library file ($Y$RUN). However, you may want to save your own-code routine in the form of a load module that you can use again.

If you decide to save the load module for future use, you again have two choices. You can store the module in the system load library file ($Y$LOD), where the sort/merge modules also reside, or in a private library file. If you store the module in $Y$LOD, you have a little less coding to do and sort/merge can retrieve the module slightly faster at execution time.

Example 1 gives the job control stream needed for storing your own-code routine in $Y$LOD.

Example 1:

```
      1         10    16
     ----------------------------------
 1. | // JOB OWNCODE
 2. | // DVC 2Ø  // LFD PRNTR
 3. | // ASM
 4. | /$
 5. | * \
 6. | *  }your program coding
 7. | * /
 8. | /*
 9. | // LINK PHASE1,OUT=(RES,$Y$LOD)
10. | /&
11. | // FIN
```

Notice that we have used the job control procedure (jproc) calls for both the assembler (line 3) and the linkage editor (line 9). This saves a considerable amount of coding. The first parameter on the LINK jproc tells the linkage editor to include the object module called PHASE1 in the load module it is creating. Since the label field is omitted, the name of the load module will also be PHASE1 by default. The OUT parameter tells the linkage editor to place the load module in the $Y$LOD file on your SYSRES volume.

When you want to execute the sort/merge, you use the job control stream in example 2.

Example 2:

```
     1        10    16
 1. // JOB SRTEXMPL,,6ØØØ,8ØØØ
 2. // DVC 2Ø  // LFD PRNTR
 3. // DVC 5Ø  // VOL DSPØØ1
 4. // LBL MYFILE1  // LFD INPUT
 5. // DVC 5Ø  // VOL DSPØØ1
 6. // EXT ,,,CYL,4
 7. // LBL MYFILE2
 8. // LFD SORTOUT
 9. //DMØ1 WORK1
10. //DMØ2 WORK2
11. // EXEC SORT
12. /$
13.  SORT FIELDS=(1,8)
14.  INPFIL EXIT
15.  MODS PH1=(PHASE1,3588,E15)
16.  END
17. /*
18. /&
19. // FIN
```

There are three indications in example 2 that an own-code routine is being provided. On line 4, the LFD name for the input file is INPUT, instead of the standard input file name SORTIN1. This matches the label you used on the input CDIB when you wrote the program. (See Figure 3-7, line 20.) The EXIT parameter on the INPFIL control statement (line 14) indicates that you are providing the input routine, and the MODS control statement (line 15) specifies that your load module is named PHASE1 and is to be called from the data input phase at exit E15. Job control automatically looks for your load module in $Y$LOD.

If you want to store your program in an alternate library file, you might use the job control stream in example 3.

Example 3:

```
// JOB OWNCODE
// DVC 2Ø  // LFD PRNTR
// ASM
/$
.
. } your program coding
.
/*
// DVC 5Ø  // VOL DSPØØ2
// EXIT ST,,,CYL,1
// LBL OWNCODE  // LFD OWNCODE
// WORK1
// EXEC LNKEDT
/$
 LINKOP OUT=OWNCODE
 LOADM PHASE1
 INCLUDE PHASE1
/*
/&
// FIN
```

The device assignment set in lines 9 through 11 sets up a file labeled OWNCODE on volume DSP002 to contain the load module PHASE1. If you wanted to add PHASE1 to a program file that existed, you would omit the EXT statement (line 10). In this example, we have elected to use the WORK jproc, the EXEC LNKEDT statement, and linkage editor control statements (lines 12 through 18) in place of the LINK jproc. The OUT keyword parameter of the LINKOP control statement tells the linkage editor to store the load module in the file with the LFD name OWNCODE. The LOADM and INCLUDE statements tell the linkage editor to name the load module PHASE1 and to include the object module PHASE1.

At execution time, you have to tell job control where to find the load module you have placed in an alternate library file. The same coding is needed as in example 2, except for an additional device assignment set and a change in the EXEC statement.

Example 4:

```
      1        10      16
10a. // DVC 51    // VOL DSP002
10b. // LBL OWNCODE    // LFD OWNCODE
11.  // EXEC SORT,OWNCODE
```

Lines 10a and 10b represent the device assignment set for the load module containing your own-code routine. The second parameter of the EXEC statement (line 11) must give the LFD name of the alternate library file on which the load module, PHASE1, is stored. Actually, placing OWNCODE in the EXEC statement will cause job control to search the alternate library for all the needed sort modules. When they are not found in OWNCODE, job control will automatically go to $Y$LOD, where the sort/merge modules reside. It takes slightly longer to retrieve modules this way than if you had stored PHASE1 in $Y$LOD, but the difference in total sort time is negligible.

3.4. USING THE MERGE-ONLY PROCESS

You need the merge-only process when you have previously sorted or sequenced files and want only to combine or merge them. The merge-only operation can combine 2 to 16 similarly ordered files into one final output file arranged in the same sequence as the input files. When sort/merge performs the merge-only process, control goes only to the final merge phase and bypasses the internal sort and preliminary merge phases. The same sort control statements used for the sort/merge operation may be used for the merge-only operation except you replace the SORT control statement with the MERGE control statement. User own-code exit routines for a merge-only operation; i.e., exit routines E32 and E38, are associated with final merge phase (phase 3) of the sort. Thus, when sort/merge performs a merge-only operation, it begins with the sort initialization and assignment phase (phase 0), skips the next two phases (data input and preliminary sort), and ends with final merge and output (phase 3), where it enters your own-code routine via exit codes E32 and E38 (if specified on your MODS sort control statement).

3.4.1. Defining the Merge-Only Operation

Independent sort/merge needs information about key fields, their formats, and the number of files to be merged. The MERGE control statement specifies this information as the SORT statement does for a sort/merge operation. It replaces the SORT control statement when you specify a merge-only operation. The MERGE control statement format is:

LABEL	△OPERATION△	OPERAND
	MERGE	[FIELDS={([strt-pos-1][,lgth-1][,form-1][,seq-1] [,...,strt-pos-n,lgth-n[,form-n] [,seq-n]]) ([strt-pos-1][,lgth-1][,seq-1] [,...,strt-pos-n,lgth-n [,seq-n]]),FORMAT=code}] [,{FILES}={number}] {ORDER} {**16**} [,MERGEP=output-file-number, input-file-number] See note.

NOTE:

The MERGEP parameter is provided and accepted for compatibility with other systems; no action is performed.

- Specifying key fields (FIELDS)

  The FIELDS parameter specifies the key field starting position *(strt-pos-1),* the length of the key field *(lgth-1),* the data format code *(form-1),* and the merging sequence *(seq-1),* ascending or descending. The FORMAT subparameter is used to specify the data format code when the data formats for all key fields are the same. Data format codes for this subparameter are the same as those for the SORT control statement (Table 3–1). Descriptions for positional subparameters of the FIELDS keyword parameter on the MERGE control statement are the same as those for the SORT control statement (3.2.1).

  If you omit the FIELDS keyword parameter, a character field is assumed beginning in position 1, the record length is assumed as the field length up to 256 bytes, and records will be merged in ascending sequence. If you specify FIELDS and omit any of its subparameters, you must retain their associated commas, except for trailing commas.

- Specifying the number of files (FILES and ORDER)

  The keyword parameters FILES and ORDER can be used interchangeably to specify the number of data input files you want to merge. This number must not exceed 16. If you elect not to include either the FILES or ORDER parameter in the MERGE control statement, sort/merge assumes 16 input files. Remember, input files are defined via LFD statements in your job's control stream. Therefore, you must use the system standard file names SORTIN1,...,SORTIN9 for the first nine input files defined and SORTINA,...,SORTING for the remaining seven input files defined. The file names must be defined in sequence.

3.4.2. Merge-Only Exit Code for Input File Processing

You can use exit code E32 to enter your own-code routine from the final merge phase of the merge-only operation. Your own-code routine may modify the contents, including control fields, of each record in the merge-only input files; however, it may not change the record size or insert or delete records from the input merge-only files. In your own-code routine, you may replace one record with another but you must be careful to avoid changing the sequence of the record in the merge or a sequence error will result. To specify that you want merge-only to enter your own-code routine, you indicate exit code E32, phase 3, and the load module name of your own-code routine on the MODS control statement.

Figure 3–8 illustrates a typical job control stream required for a merge-only operation. In the following discussion, we step through these statements to describe the processing involved.

```
      1         10    16
 1. // JOB MRGEXMP2,,7000,9000,2
 2. // OPR 'MERGE EXAMPLE 2'
 3. // OPR 'VARYING BLOCK SIZE'
 4. // DVC 50
 5. // VOL DSP028
 6. // LBL MYLIB1
 7. // LFD SORTIN1
 8. // DVC 50
 9. // VOL DSP028
10. // LBL MYLIB2
11. // LFD SORTIN2
12. // DVC 50
13. // VOL DSP028
14. // LBL MYLIB3
15. // LFD SORTIN3
16. // DVC 50
17. // VOL DSP028
18. // LBL MYLIB4
19. // LFD SORTOUT,,INIT
20. // EXEC SORT
21. /$
22.  MERGE FIELDS=(1,8,PD),FILES=3
23.  RECORD TYPE=F,RCSZ=80
24.  INPFIL BLKSIZE=(800,400,1600)
25.  OUTFIL BLKSIZE=800
26.  END
27. /*
28. /&
29. // FIN
```

Figure 3—8. Typical Job Control Stream for a Merge-Only Operation

The JOB statement supplies the name of your job, a minimum main storage requirement of 7000 hexadecimal bytes, a maximum of 9000 hexadecimal bytes of main storage requested for your program, and a maximum of 2 tasks simultaneously active during your program execution. The two OPR control statements (lines 2 and 3) display the messages enclosed in quotation marks to the operator at the system console.

Lines 4, 8, 12, and 16 specify that device 50 is used for input and output files. Lines 5, 9, 13, and 17 specify that the same volume (DSP028) is used for input and output files.

The file identifier for input file 1 is MYLIB1 (line 6); for input file 2, MYLIB2 (line 10); and for input file 3, MYLIB3 (line 14). The output file identifier is MYLIB4 (line 18). File names for the three input files to be merged and the output file to receive the merged data record are the standard names SORTIN1 (line 7), SORTIN2 (line 11), SORTIN3 (line 15), and SORTOUT (line 19), respectively.

The INIT parameter on the LFD statement (line 19) indicates that this output file is to be initialized starting at the first record the first time the file is opened.

The EXEC statement (line 20) tells OS/3 job control to execute the SORT program. Your sort/merge control statements follow, beginning with the /$ and ending with the /* delimiters. The MERGE control statement (line 22) specifies that the key field begins in byte 1, extends eight bytes, and is in packed decimal data format. There are three files being merged.

Records are 80 bytes, fixed length according to the RECORD control statement (line 23).

Input files 1, 2, and 3 are blocked at 800, 400, and 1600 bytes, respectively, and the output file has a block size of 800 bytes. These specifications appear in the INPFIL and OUTFIL control statements (lines 24 and 25).

Finally the END, /*, /&, and FIN control statements (lines 26 through 29) indicate the end of your sort control statements, end of job step, end of job, and end of card reader operations.

Note that on first runs, the EXT job control statement is needed immediately after each VOL statement to allocate each file; however, it should be removed on all succeeding runs after the files have been allocated. The job control user guide, UP-9986 (current version) explains the EXT statement in more detail.

3.4.3. Merge-Only Exit Code for Input File Read Error Processing

If you decide to write your own-code routine for processing input file read errors during the merge-only operation, use exit code E38. You write only the BR 14 instruction to return to the sort.

4. Program and Control Stream Examples

4.1. GENERAL

This section contains examples that illustrate program coding and job control streams for independent sort/merge operation. The first group of examples (4.2) illustrate sort/merge control statements only and the succeeding examples show complete job control streams including job control and sort control statements required for performing disk (4.3), tape (4.4), format label diskette (4.5) sorts, and a default sort (4.6). These job control streams can be punched on cards, written to a diskette, or built from a workstation. Building and initiating job control streams from a workstation is discussed in Section 1 and an example is shown in 4.3.

4.2. SORT/MERGE CONTROL STATEMENT EXAMPLES

The following six examples illustrate the sort/merge control statements needed to supply information to sort/merge or merge-only for their functions. In each example, the sort control statements are preceded by a /$ delimiter statement and followed by a /* delimiter statement. The sort control statements within these delimiter statements represent a data set.

Example 1 shows specifications for a tape sort/merge on fixed-length records.

Example 1:

```
/$
 SORT FIELDS=(1,4,CH,A,10,12,BI,A),WORK=3,SIZE=3500
 RECORD TYPE=F,RCSZ=82
 INPFIL BLKSIZE=820,OPEN=RWD,CLOSE=UNLD,DATA=E
 OUTFIL BLKSIZE=820,OPEN=RWD CLOSE=UNLD
 OPTION PRINT=ALL,STORAGE=20000,LABEL=(S,S)
 END
/*
```

The SORT statement defines the key fields, the number of work files, and the number of records to be sorted. The first key field begins in record position 1, is four bytes long, has a character format, and is to be sorted in ascending sequence. The second key field begins in position 10, is 12 positions long, is in binary format, and is to be sorted in ascending sequence. Three work files are indicated by the WORK keyword parameter, and the input file contains approximately 3500 records. The RECORD control statement defines the record type as fixed with a record size of 82 bytes. The INPFIL control statement specifies that the records are blocked at 820 characters per block, that the input file is on tape, and that the tape is to be rewound to load point upon opening the rewound with interlock on closing. Data is in EBCDIC format (DATA=E). The parameters specified in the OPTION control statement provide for the printing of all messages, define the available main storage as 20,000 bytes, and identify the input and output file labels as being standard tape labels. The end of sort/merge control statements is indicated by the END control statement in the job control stream.

Example 2 shows a tag sort on variable-length records.

Example 2:

```
/$
 SORT FIELDS=(6,1Ø,CH,A,12,1Ø,CH,D),WORK=3,SIZE=35ØØ
 RECORD TYPE=V,LENGTH=(4ØØ,3Ø,,65,65)
 OPTION ADDROUT=D
 END
/*
```

The FIELDS parameter says that the first sort key begins in byte 6 and is 10 bytes long, in character format, sorted in ascending sequence. The second sort key begins in byte 12 and is 10 bytes long, in character format, sorted in descending sequence. The WORK parameter indicates three work files, and the SIZE parameter indicates approximately 3500 records in the input file. The length specifications of these records for each phase of sort/merge operation, required for a tag sort, are: maximum input record length, 400 bytes; maximum length of records released to the internal sort, 30 bytes; maximum output record length, 30 bytes by default; minimum input record length, 65 bytes; and the record length appearing most frequently in the input file, 65 bytes. The OPTION control statement defines a tag sort in which output records are to include both the direct access address and the key fields. The new tag sort records will be 30 bytes in length, including the 10-byte address field and 20 bytes for the key fields. This length is reflected in the second and third subparameters of the LENGTH parameter. The END control statement indicates the end of sort/merge control cards.

Example 3 shows fixed-length record processing with user-written modification exists.

Example 3:

```
/$
 SORT FIELDS=(2,4,CH,A),WORK=3,SIZE=9ØØØ
 RECORD TYPE=F,RCSZ=82
 INPFIL EXIT
 OUTFIL EXIT
 OPTION STORAGE=21ØØØ
 MODS PH1=(PHASE1,45ØØ,E11,E15),                                 C
      PH3=(PHASE3,4ØØØ,E31,E35),                                 C
      PH7=(PHASE7,1ØØØ,E75)
 END
/*
```

The FIELDS parameter describes the sort key beginning in byte 2, extending four bytes in character format, and being sorted in ascending sequence. There are three work files (WORK) and approximately 9000 records in the input file (SIZE). The INPFIL and OUTFIL control statements both state the EXIT keyword parameter indicating that user own-code routines will provide the coding for reading the input file and writing the output file. Exit codes E11 and E15 (approximately 4500 bytes long) provide the entry points to your read routines, and exit codes E31 and E35 (approximately 4500 bytes long) provide the entry points to your write routines. In addition, this coding indicates that you will also provide a routine (approximately 1000 bytes long) for processing records with equal key fields (specified by exit code E75 in the MODS control statement).

Example 4 illustrates the use of CALCAREA in the OPTION statement.

Example 4:

```
/$
 SORT FIELDS=(2,4,,,12,1Ø,,D),SIZE=658ØØ
 RECORD TYPE=F,RCSZ=82
 INPFIL BLKSIZE=82Ø
 OUTFIL BLKSIZE=82Ø
 OPTION CALCAREA
 END
/*
```

The FIELDS parameter describes the sort keys. The first key in byte 2 is four bytes long in character format and is sorted in ascending sequence. The second key begins in byte 12, extends 10 bytes in character format, and is sorted in descending sequence. There are approximately 65,800 records in the input file (SIZE=65800). This example will not perform a sort, but will give you the estimated sort time in minutes and the number of cylinders sort/merge requires for disk work space.

Example 5 defines a merge-only operation that processes three input files of fixed-length records (165 bytes), which are blocked 10 records per block. The sort key begins in byte 20, extends 10 bytes in character format, and is sorted in ascending sequence.

Example 5:

```
/$
 MERGE FIELDS=(20,10,CH,A),FILES=3
 RECORD TYPE=F,RCSZ=165
 INPFIL BLKSIZE=1650
 OUTFIL BLKSIZE=1650
 END
/*
```

Example 6 shows the same merge-only operation as example 5 except for the file blocking specified by the INPFIL control statement. The first file is blocked at 1650 bytes (10 records per block), the second at 825 bytes (5 records per block), and the third at 2475 bytes (15 records per block).

Example 6:

```
/$
 MERGE FIELDS=(20,10),FILES=3
 RECORD TYPE=F,RCSZ=165
 INPFIL BLKSIZE=(1650,825,2475)
 OUTFIL BLKSIZE=1650
 END
/*
```

4.3. JOB CONTROL STREAMS TO PERFORM DISK SORTS

The two examples that follow illustrate complete job streams to perform disk sorts where:

- disk input files and disk work files are used to create a disk output file; and
- multiple input files and one disk work file are used to create a single disk output file.

Example 1 illustrates a typical job control stream required for performing a sort using a disk input file, disk work files, and a disk output file. The job named SRTEXMP1 performs a sort of the data records contained in disk input file SORTIN1. Three disk work files (DM01 through DM03) are assigned to the sort. The first key field for sorting starts at byte 9 of the record and is one byte long. The second key field starts at byte 1 and extends for eight bytes. The records are character formatted and are sorted in ascending order (specified by default in the sort statement). The records are 80 bytes long and are fixed length. Both the input file and the output file are blocked at 800 bytes. Approximately 10,000 records are involved in the sort.

Example 1:

```
     1        10    16
 1.  // JOB SRTEXMP1,,7000,9000
 2.  // DVC 50                                   }
 3.  // VOL DSP001                               } Input file device
 4.  // LBL MYLIB1                               } assignment set
 5.  // LFD SORTIN1                              }
 6.  // DVC 50                                   }
 7.  // VOL DSP001                               } Output file device
 8.  // LBL MYLIB2                               } assignment set
 9.  // LFD SORTOUT,,INIT                        }
10.  // DVC 51                                   }
11.  // VOL DSP111                               } Work file 1 device
12.  // LBL SRTWK1                               } assignment set
13.  // LFD DM01,,INIT                           }
14.  // DVC 52                                   }
15.  // VOL DSP112                               } Work file 2 device
16.  // LBL SRTWK2                               } assignment set
17.  // LFD DM02,,INIT                           }
18.  // DVC 53                                   }
19.  // VOL DSP113                               } Work file 3 device
20.  // LBL SRTWK3                               } assignment set
21.  // LFD DM03,,INIT                           }
22.  // EXEC SORT
23.  /$                                          }
24.   SORT FIELDS=(9,1,,,1,8),WORK=3,SIZE=10000  }
25.   RECORD LENGTH=(80),TYPE=F                  }
26.   INPFIL BLKSIZE=800                         } Sort program
27.   OUTFIL BLKSIZE=800                         }
28.   END                                        }
29.  /*                                          }
30.  /&
31.  // FIN
```

Line Number	Explanation
1	The JOB statement defines the job named SRTEXMP1 to the system and defines the minimum and maximum main storage required for the job.
2-5	Assigns the input file. (See 3.1.1.)
6-9	Assigns the output file. (See 3.1.1.)
10-21	Assigns the disk work files. (See 3.1.1.) These files have been previously created (indicated by lack of EXT statement); however, the INIT option on the LFD statement allows the sort to access them as though they were new files.

Line Number — Explanation

22 Initiates the execution of sort/merge.

23–29 The data set containing the sort/merge control statements.

24 The SORT statement specifies:

- a 1-byte character key field at byte 9 of the data records to be sorted and an 8-byte field at byte 1;
- three work files; and
- the input file contains approximately 10,000 records to be sorted.

25 The RECORD statement defines an 80-byte, fixed-length record.

26–27 The INPFIL and OUTFIL statements define the input and output block sizes as 800 bytes.

28–29 Marks the end of the sort control statements.

30 Marks the end of the job stream.

31 Marks the end of reader operations.

This control stream can be submitted when you want to execute this sort program.

You can also use·the general editor to create this control stream from a workstation. To do this, proceed as follows:

1. Turn on your workstation and logon to the systern by entering the LOGON command (refer to interative services commands and facilities, UP-9972).
2. After you have successfully logged on, press the FUNCTION key and (while holding it down) press the SYSMODE key.
3. Key in EDT, then transmit (press the XMIT key).

   This will activate EDT.
4. Now keyin as follows:

```
Job name          1.ØØØØ // JOB SRTEXMP1,,7ØØØ,9ØØØ
Device            2.ØØØØ // DVC 5Ø
assignment        3.ØØØØ // VOL DSPØØ1
set               4.ØØØØ // LBL MYLIB1
                  5.ØØØØ // LFD SORTIN1
                  6.ØØØØ // DVC 5Ø
                  7.ØØØØ // VOL DSPØØ1
                  8.ØØØØ // LBL MYLIB2
                  9.ØØØØ // LFD SORTOUT,,INIT
                 1Ø.ØØØØ // DVC 51
                 11.ØØØØ // VOL DSP111
                 12.ØØØØ // LBL SRTWK1
                 13.ØØØØ // LFD DMØ1,,INIT
                 14.ØØØØ // DVC 52
Job              15.ØØØØ // VOL DSP112
control          16.ØØØØ // LBL SRTWK2
stream           17.ØØØØ // LFD DMØ2,,INIT
                 18.ØØØØ // DVC 53
                 19.ØØØØ // VOL DSP113
                 2Ø.ØØØØ // LBL SRTWK3
                 21.ØØØØ // LFD DMØ3,,INIT
                 22.ØØØØ // EXEC SORT
                 23.ØØØØ /$
                 24.ØØØØ  SORT FIELDS=(9,1,,,1,8),WORK=3,SIZE=1ØØØØ
                 25.ØØØØ  RECORD LENGTH=(8Ø),TYPE=F
                 26.ØØØØ  INPFIL BLKSIZE=8ØØ
                 27.ØØØØ  OUTFIL BLKSIZE=8ØØ
                 28.ØØØØ  END
                 29.ØØØØ /*
                 3Ø.ØØØØ /&
Store job        31.ØØØØ @WRITE△MO=SRTEXMP1,FIL=$Y$JCS
control stream
Terminates       32.ØØØØ @HALT
EDT
```

NOTE:

A '△' indicates a space.

At this point, the control stream has been stored on $Y$JCS. You can either logoff the system by entering the LOGOFF command or you can excecute your program by entering RV△SRTEXMP1. If you logoff the system, you can execute your program at a later time by first logging on. Then, press the FUNCTION key and (while holding that key down) press the SYSMODE key and enter RV△SRTEXMP1.

You can also use the general editor to create the control stream for examples 2, 3, 4, 5, and 6 by following the procedure described for example 1.

Example 2 illustrates the job control stream required for performing an independent disk sort using multiple input files, a disk work file, and a disk output file. The job named SRTEXMP2 is to sort the data records of the three input files SORTIN1, SORTIN2, and SORTIN3. The key fields are in packed decimal format and are to be sorted in ascending order. Input files 1 and 3 are blocked at 800 bytes each and input file 2 at 400 bytes. The output file is blocked at 800 bytes. Approximately 50,000 records are sorted.

Example 2:

```
     1            10      16
 1.  // JOB SRTEXMP2,,7ØØØ,9ØØØ
 2.  // OPR ' SORT EXAMPLE 7'
 3.  // OPR 'MULTIPLE INPUT FILES'
 4.  // DVC 5Ø
 5.  // VOL DSP1ØØ                         Input file 1 device
 6.  // LBL SORTIN1                        assignment set
 7.  // LFD SORTIN1
 8.  // DVC 5Ø
 9.  // VOL DSP1ØØ                         Input file 2 device
10.  // LBL INPUTØ2                        assignment set
11.  // LFD SORTIN2
12.  // DVC 5Ø
13.  // VOL DSP1ØØ                         Input file 3 device
14.  // LBL INPUTØ3                        assignment set
15.  // LFD SORTIN3
16.  // DVC 5Ø
17.  // VOL DSP1ØØ                         Output file device
18.  // LBL OUTPUT                         assignment set
19.  // LFD SORTOUT,,INIT
20.  // DVC 51
21.  // VOL DSP1Ø1                         Work file device
22.  // LBL WORK                           assignment set
23.  // LFD DMØ1,,INIT
24.  // EXEC SORT
25.  /$
26.   SORT FIELDS=(4,8,PD),FILE=3,SIZE=5ØØØØ
27.   INPFIL BLKSIZE=(8ØØ,4ØØ,8ØØ)                   Sort program
28.   OUTFIL BLKSIZE=(8ØØ)
29.   END
30.  /*
31.  /&
32.  // FIN
```

Line Number	Explanation
1	The JOB statement defines the job named SRTEXMP2 to the system and the minimum and maximum main storage bytes required for the job.
2–3	Gives messages to operator at system console.
4–15	Assigns the three input files. (See 3.1.1.)
16–19	Assigns the output file. (See 3.1.1.)

Line Number Explanation

20-23 Assigns the disk work file. (See 3.1.1.) This file has been previously created (indicated by lack of EXT statement); however, the INIT option on the LFD statement allows the sort to access it as though it were a new file.

24 Initiates the execution of sort/merge.

25-30 The data set containing the sort/merge control statements.

26 The SORT statement specifies:

- an 8-byte packed decimal key field at byte 4 of the data records to be sorted;
- three input files; and
- the input files contain approximately 50,000 records to be sorted.

27 The INPFIL statement defines the input block sizes for each input file: 800 bytes for input files 1 and 3 and 400 bytes for input file 2.

28 The OUTFIL statement defines the output block size as 800 bytes.

29-30 Marks the end of sort/merge control statements.

31 Marks the end of the job stream.

32 Marks the end of card reader operations.

4.4. JOB CONTROL STREAM TO PERFORM TAPE SORTS

Both examples 3 and 4 use tape input and work files to create tape output files. They illustrate the use of SHARE, RESERV, RESTART, and CSPRAM parameters in the OPTION sort control statement and the use of the PARAM job control statement to enter parameters from the control stream.

Example 3 illustrates a typical job control stream required to perform a sort/merge operation using tape for the input, output, and work files. The job named SRTEXMP3 sorts the character-formatted data records to input file SORTIN1 into ascending order and then writes those records to the output tape file SORTOUT. The records are fixed-length and 80 bytes long, with a 10-byte sort key field starting in byte 8. The data block size for both the input and output records is 800 bytes. The input and output files are rewound to their starting point upon opening and rewound with interlock upon closing. Tape device SM01 is shared as an input device during input operations and as a work storage device during sort operations. Tape device SM03 is specified as a reserved device used for working storage during the first two phases of the sort and for output file during the final merge phase.

Example 3:

```
     1           10      16
 1. // JOB SRTEXMP3,,7ØØØ,9ØØØ
 2. // DVC 9Ø          }
 3. // VOL MASTER      } Input file device
 4. // LFD SORTIN1     } assignment set
 5. // DVC 9Ø,IGNORE   } Redefined input device
 6. // VOL TAPEØ1      } assignment set to work file
 7. // LFD SMØ1        }
 8. // DVC 91          }
 9. // VOL TAPEØ2      } Work file device
10. // LFD SMØ2        } assignment set
11. // DVC 92          }
12. // VOL TAPEØ3      } Work file device
13. // LFD SMØ3        } assignment set
14. // DVC 92,IGNORE   } Redefined work file device
15. // VOL MASTER      } assignment set to output file
16. // LFD SORTOUT     }
17. // EXEC SORT
18. /$                                                 }
19.  SORT FIELDS=(8,1Ø,CH)                             }
20.  RECORD LENGTH=(8Ø),TYPE=F                         }
21.  INPFIL BLKSIZE=8ØØ,OPEN=RWD,CLOSE=RWI             } Sort program
22.  OUTFIL BLKSIZE=8ØØ,OPEN=RWD,CLOSE=RWI             }
23.  OPTION SHARE=SMØ1,RESERV=SMØ3                     }
24.  END                                               }
25. /*                                                 }
26. /&
27. // FIN
```

Line Number	Explanation
1	The JOB statement defines the job named SRTEXMP3 to the system and minimum and maximum main storage required to run the job.
2–4	Assigns the input file. (See 3.1.1.)
5–7	Redefines the device assigned to the input file as a work file, using the IGNORE option of the DVC statement.
8–13	Assigns the remaining work files.

Line Number	Explanation
14-16	Redefines the device assigned to SM03 as the output file, using the IGNORE option of the DVC statement.
17	Initiates the execution of the sort.
18-25	Is the data set containing the sort control statements.
19	The SORT statement defines a 10-character key field which begins in byte 8 of the record.
20	The RECORD statement defines an 80-byte, fixed-length record.
21-22	The INPFIL and OUTFIL statements define the input and output block sizes as 800 bytes and indicate that these files are to be set to load point on opening and to interlock on closing.
23	The OPTION statement specifies SM01 as the SHARE file and SM03 as the RESERV file. (See 3.2.6.)
24-25	Indicates the end of the sort control statements.
26	Marks the end of the job stream.
27	Marks the end of card reader operations.

Example 4 illustrates a typical job control stream required for restarting an interrupted tape sort performed by sort/merge. The sort itself is identical with that described in example 3. (See example 3 for program and coding details.) By specifying the RESTART and CSPRAM keyword parameters in the OPTION statement (line 25) included in the user data set, the tape sort can be resumed. The system console displays the most recent pass number, and the PARAM statement shown in line 28 of example 4 gives sort/merge the pass recovery point at which the sort is resumed.

Example 4:

```
1          10     16
// JOB SRTEXM4,,7000,9000
// OPR 'SORT EXAMPLE 4'
// OPR 'TAPE SORT WITH RESTART'
// DVC 90                                                  } Input file device
// VOL MASTER                                              } assignment set
// LFD SORTIN1                                             }
// DVC 90,IGNORE                                           } Redefined input device
// VOL TAPE01                                              } assignment set to work file
// LFD SM01                                                }
// DVC 91                                                  } Work file device
// VOL TAPE02                                              } assignment set
// LFD SM02                                                }
// DVC 92                                                  } Work file device
// VOL TAPE03                                              } assignment set
// LFD SM03                                                }
// DVC 92,IGNORE                                           } Redefined work file device
// VOL MASTER                                              } assignment set to output file
// LFD SORTOUT                                             }
// EXEC SORT
/$                                                         }
 SORT FIELDS=(8,10,CH)                                     }
 RECORD LENGTH=(80),TYPE=F                                 }
 INPFIL BLKSIZE=800,OPEN=RWD,CLOSE=RWI                     } Sort program
 OUTFIL BLKSIZE=800,OPEN=RWD,CLOSE=RWI                     }
 OPTION SHARE=SM01,RESERV=SM03,RESTART,CSPRAM=YES          }
 END                                                       }
/*                                                         }
// PARAM RESUME=(PASS,023)
/&
// FIN
```

4.5. JOB CONTROL STREAM TO PERFORM A FORMAT LABEL DISKETTE SORT

Example 5 illustrates a typical job control stream for performing a sort using a format label diskette input file, disk work files, and a format label diskette output file. The job named SRTDKT performs a sort of the data records contained in the format label diskette input file SORTIN1. Two disk work files (DM01 and DM02) are assigned to the sort. The first key field for sorting starts at byte 8 of the record and is one byte long. The second key field starts at byte 3 and extends for four bytes. The records are character formatted and are sorted in ascending order (specified by default in the SORT statement). The records are 80 bytes long and are fixed length. Both the input and output file are blocked at 400 bytes. Approximately 6000 records are involved in the sort. The system DELETE data reduction routine is used to automatically delete duplicate records from the files (specified by the MODS statement).

Example 5:

```
     1            10     16
 1. // JOB SRTDKT,,7ØØØ,9ØØØ
 2. // DVC 13Ø                                   )
 3. // VOL DKTØØ1                                 }  Input file device assignment set
 4. // LBL DIN                                    }
 5. // LFD SORTIN1                                )
 6. // DVC 13Ø                                   )
 7. // VOL DKTØØ1                                 }  Output file device assignment set
 8. // LBL DOUT                                   }
 9. // LFD SORTOUT,,INIT                          )
10. // DVC 51                                    )
11. // VOL DSP111                                 }  Work file 1 device assignment set
12. // LBL SRTWK1                                 }
13. // LFD DMØ1                                   )
14. // DVC 52                                    )
15. // VOL DSP112                                 }  Work file 2 device assignment set
16 // LBL SRTWK2                                  }
17. // LFD DMØ2                                   )
18. // EXEC SORT
19. /$                                                )
20.   SORT FIELDS=(8,1,,,3,4),WORK=2,SIZE=6ØØØ         }
21.   RECORD LENGTH=(8Ø),TYPE=F                        }
22.   INPFIL BLKSIZE=4ØØ                               }  Sort program
23.   OUTFIL BLKSIZE=4ØØ                               }
24.   MODS PH7=DELETE,,E75)                            }
25.   END                                              }
26.   /*                                              )
27.   /&
28.   // FIN
```

Line Number	Explanation
1	The JOB statement defines the job named SRTDKT to the system and the minimum and maximum main storage required for the job.
2-5	Assigns the input file. (See 3.1.1.)
6-9	Assigns the output file. (See 3.1.1.)
10-17	Assigns the disk work files. (See 3.1.1.) These files have been previously created (indicated by lack of EXT statement); however, the INIT option on the LFD statement allows the sort to access them as though they were new files.
18	Initiates the execution of sort/merge.
19-26	The data set containing the sort/merge control statements.

Line Number Explanation

20 The SORT statement specifies:

- a 1-byte character key field at byte 8 of the data records to be sorted and a 4-byte field at byte 3.
- two work files; and
- the input file contains approximately 6,000 records to be sorted.

21 The RECORD statement defines an 80-byte, fixed-length record.

22–23 The INPFIL and OUTFIL statements define the input and output block sizes as 400 bytes.

24 The MODS statement specifies that the DELETE data reduction routine is used to delete duplicate records.

25–26 Marks the end of the sort control statements.

27 Marks the end of the job stream.

28 Marks end of card reader operations.

4.6. JOB CONTROL STREAM TO PERFORM A DEFAULT SORT

The default sort is so named because all information supplied to sort/merge is automatically defaulted in the absence of sort control statements. In example 6, there are no sort control statements. The only indication of a sort is the EXEC SORT job control statement in the control stream. This example illustrates a typical job control stream required to perform a default disk sort operation. When a default sort is performed, sort/merge takes the record size, block size, and record type specifications from the volume-table-of-contents (VTOC) for the input file. The output file is structured from the specifications for the input file. If the input file happens to be a partitioned disk file, sort/merge assumes the first partition of the file as the input partition. The output file is always a single partition file in a default sort operation. The data records are assumed to be character formatted and to have one sort key field the same length as the record but not to exceed 256 bytes. In a default sort, only one input file can be processed, and all input, output, and work files assigned in the job control stream must be disk files.

Example 6:

```
     1          10     16
 1.  // JOB SRTEXMP6,,7000,9000
 2.  // OPR 'SORT EXAMPLE 6'
 3.  // OPR 'DEFAULT SORT'
 4.  // DVC 50                          )
 5.  // VOL DSP100                      |
 6.  // LBL INPUT                       }  Input file device assignment set
 7.  // LFD SORTIN1                     )
 8.  // DVC 50                          )
 9.  // VOL DSP100                      |
10.  // LBL OUTPUT                      }  Output file device assignment set
11.  // LFD SORTOUT,,INIT               )
12.  // DVC 51                          )
13.  // VOL DSP101                      |
14.  // LBL WORK                        }  Work file device assignment set
15.  // LFD DM01,,INIT                  )
16.  // EXEC SORT
17.  /&
18.  // FIN
```

Line Number	Explanation
1	The JOB statement defines the job named SRTEXMP6 and minimum and maximum main storage required to run the job.
2–3	Gives message to the operator at the system console.
4–7	Defines the input file. (See 3.1.1.)
8–11	Defines the output file. (See 3.1.1.)
12–15	Defines the work file. (See 3.1.1.)
16	Initiates the execution of a default sort.
17	Marks the end of control stream.
18	Marks the end of card reader operations.

Appendix A. Statement Conventions

A.1. GENERAL FORMAT RULES

The following general conventions apply to the coding formats illustrated in this manual for sort/merge control statements.

- Lowercase letters and words are generic terms representing information that must be supplied by you. Such lowercase terms may contain hyphens and acronyms (for readability).

  Example:

```
┌,BIN=┌bytes                                            ┐┐
└     └(min-bytes,size-1,freq-1[,...,size-n,freq-1])   ┘┘
```

- Capital letters, commas, equal signs, and parentheses must be coded exactly as shown.

  Example:

```
┌FIELDS=┌([strt-pos-1][,lgth-1][,form-1][,seq-1]                ┐┐
│       │   [,...,strt-pos-n,lgth-n[,form-n][seq-n]])            ││
│       │([strt-pos-1][,lgth-1][,seq-1][,...,strt-pos-n,lgth-n  ││
└       └   [,seq-n]]),FORMAT=code                               ┘┘
```

  Sample Coding:

```
1            10      16
________________________________________________
             SORT    FIELDS=(1,4,,1Ø,12),FORMAT=CH
```

- Information contained within braces { } represents alternate choices, of which only one may be chosen.

  Example:

```
┌,┌DISC┐=number┐
│ │TAPE│       │
└ └WORK┘       ┘
```

- Information contained within brackets [ ] represents optional entries that (depending upon program requirements) are included or omitted. Braces within brackets [{}] signify that one of the specified entries must be chosen if that parameter is included.

  Examples:

  Brackets:

  ```
  [,FILE=number]
  ```

  Braces within brackets:

  ```
  ⎡,CLOSE=⎧NORWD⎫⎤
  ⎢       ⎪RWD  ⎪⎥
  ⎢       ⎨RWI  ⎬⎥
  ⎣       ⎩UNLD ⎭⎦
  ```

- Optional parameters having lists of optional entries may have default specifications supplied by the operating system when the parameters are not specified by you. Although the default may be specified by you with no adverse effect, it is considered inefficient to do so. For easy reference, when a default specification occurs in the sort macro or sort control statement format, it is printed on a shaded background. If, by parameter omission, the operating system performs some complex processing other than parameter insertion, it is explained in text.

  Example:

  ```
  ⎡TYPE=⎧D⎫⎤
  ⎢     ⎨F⎬⎥
  ⎣     ⎩V⎭⎦
  ```

- An ellipsis (series of three periods) indicates the presence of a variable number of entries.

  Example:

  ```
  ⎡,BLKSIZE=⎧bytes                   ⎫⎤
  ⎣         ⎩(bytes-1[,....,bytes-8])⎭⎦
  ```

- A keyword parameter consists of a word or a code usually, but not always, followed by an equal sign and a specification. Keyword parameters can be written in any order in the operand field and are separated by commas.

Examples:

Assume that INPFIL is an independent sort/merge control statement with four optional keyword parameters; OPEN, CLOSE, DATA, and BYPASS.

```
1        10    16
--------------------------------------------
         INPFIL OPEN=RWD,CLOSE=RWD,DATA=A
         INPFIL CLOSE=RWD,DATA=A,OPEN=RWD
         INPFIL DATA=A,CLOSE=RWD
         INPFIL CLOSE=RWD,BYPASS
```

- Positional parameter specifications are presented in lowercase letters and require insertion of a value.

  Example:

```
SIZE=number
```

- A keyword parameter may contain a sublist of parameters called subparameters, which are separated by commas and enclosed in parentheses. The parentheses must be coded as part of the specification. All subparameters presented in this manual are positional and can be up to eight alphanumeric characters. They must be coded in the order shown, and the comma must be retained when a subparameter is omitted, except for trailing commas.

  Examples:

```
1        10    16
------------------------------
         SORT  FIELDS=(1,4,CH,A)
         SORT  FIELDS=(1,4,,A)
         SORT  FIELDS=(1,4)
```

A.2. CODING RULES

Sort/merge control statements are placed in the control stream between the start-of-data (/$) and end-of-data (/*) job control statements. Each control statement can appear only once in your program. Sort/merge control statements should appear in this general format on the coding form:

COL. 1	ΔOPERATIONΔ	OPERAND	ΔCOMMENTS	COL. 72	SEQUENCE 73 80
blank	variable	variable	optional	blank or character	optional

- Column 1

  The first column or position must be blank for all sort/merge control statements.

- Operation

  Use field for defining the operation to be performed. The permissible entries are:

  END

  INPFIL

  MERGE

  MODS

  OPTION

  OUTFIL

  RECORD

  SORT

  The operation field must be the first field you define in a control statement. It starts in any column after column 1 but cannot extend into column 72 (continuation field).

- Operand

  The operands specify the parameters needed to define and execute the sort/merge. Separate this field from the operation field by one or more blanks and begin the first operand on the same card as your operation field. Operands are composed of keyword parameters equated to a value, a set of values (positional parameters), or used as an indicator without associated values. Embedded blanks are not allowed; anything after a blank is regarded as a comment.

- Comment

  Use the comment field to annotate the sort/merge program. Leave one or more blanks between the operand field and the comment field.

- Continuation

  Column 72 is used for the continuation indicator. A blank indicates that this is the last image of the statement. You may use any other character to indicate that the statement is continued on the following image or card. The continuation character is not included in any operation or operand field.

- Sequence

  You may use the sequence field (positions 73 through 80) as you wish; however, it is usually used for sequence numbering and identification.

Continuation statements are logical extensions of the preceding operand or comment. The statement preceding a continuation statement contains characters through column 71 or the operand may be discontinued by a comma followed by a blank. In either case, a character must appear in column 72 to establish a continuation statement. The continuation of an operand starts in column 16; the continuation of a comment starts in column 17.

1 15	16	17 STATEMENT CONTINUATION 71	72	73 80
blank	operand	operand	continuation character	sequence-id
blank	blank	comment	continuation character	sequence-id

- Column 16

  Contains the first characters or delimiter of the continued operand (never a comma).

- Column 17

  Contains the first character of the continued comment.

- Column 71

  Is the last column of the continuation statement.

- Column 72

  Is any nonblank character; indicates that the next statement is a continuation statement.

- Columns 73 through 80

  Is the sequence identification.

Appendix B. Standard EBCDIC and ASCII Collating Sequences

B.1. GENERAL

This appendix provides three useful tables containing collating sequences. The first (Table B-1) presents a cross-reference table that enables you to compare the following standard codes commonly used in data processing and OS/3:

- Hollerith punched card code
- EBCDIC (Extended Binary Coded Decimal Interchange Code)
- ASCII (American National Standard Code for Information Interchange)
- Binary bit-pattern (bit-configuration) representation for an 8-bit system
- Hexadecimal representation

Table B-2 provides a convenient chart of OS/3 EBCDIC graphics only, and Table B-3 lists OS/3 ASCII graphics only.

B.2. EBCDIC/ASCII/HOLLERITH CORRESPONDENCE

Table B-1 is a cross-reference table depicting the correspondences among the Hollerith punched card code, ASCII, and EBCDIC. The table is arranged in the sorting (or collating) sequence of the binary bit patterns which have been assigned to the codes, with 0000 0000 being the lowest value in the sequence and 1111 1111 the highest. These binary bit patterns are sorted in a left to right sequence (most significant to least significant bit).

Note that the column headed *Decimal* uses decimal numbers to represent the positions of the codes and bit patterns in this sequence, but counts the position of the lowest value as the zero position rather than the first. Thus, the position of the highest value bit pattern 1111 1111 is represented in the decimal column by 255, whereas it is actually the 256th in the sequence. This scheme corresponds to the common convention for numbering bytes, in which the first byte of a group is byte 0, and is convenient when you are constructing a 256-byte translation table.

The column headed *Decimal* also represents the collating sequence for the EBCDIC graphic characters shown in the fourth column of the table; the fifth column, *Hollerith Punched Card Code*, contains the hole patterns assigned to these EBCDIC graphics. Empty space in the fourth column represents the positions of the EBCDIC control characters; the EBCDIC space character is represented in the fourth column by the conventional notation SP at decimal position 64, and the corresponding card code is *no punches.*

The ASCII graphic characters, listed in the sixth column of Table B–1, are also in their collating sequence, and the hole patterns in the seventh column correspond to the ASCII graphics. The ASCII space character is represented by the notation *SP* in the sixth column at decimal position 32; the corresponding card code is, again, *no punches.* The empty space in the sixth column represents the positions of the ASCII control characters. The shading in the ASCII graphic character column indicates where the 128-character ASCII code leaves off: there are no ASCII graphic or control characters that correspond to the bit patterns higher in collating sequence than 0111 1111 (the 128th in Table B–1).

B.2.1. Hollerith Punched Card Code

The Standard Hollerith punched card code specifies 256 hole-patterns in 12-row punched cards. Hole-patterns are assigned to the 128 characters of ASCII and to 128 additional characters for use in 8-bit coded systems. These include the EBCDIC set. Note that no sorting sequence is implied by the Hollerith code itself.

B.2.2. EBCDIC

EBCDIC is an extension of Hollerith coding practices. It comprises 256 characters, each of which is represented by an 8-bit pattern. Table B–1 shows the EBCDIC graphic characters only; the EBCDIC control characters are not indicated.

B.2.3. ASCII

ASCII comprises 128 coded characters, each represented by an 8-bit pattern, and includes both control characters and graphic characters. Only the latter are shown in Table B–1.

Table B—1. Cross-Reference Table: EBCDIC/ASCII/Hollerith (Part 1 of 5)

EBCDIC					ASCII	
Decimal	Hexadecimal	Binary	EBCDIC Graphic Character	Hollerith Punched Card Code	ASCII Graphic Character	Hollerith Punched Card Code
0	00	0000 0000		12-0-9-8-1		12-0-9-8-1
1	01	0000 0001		12-9-1		12-9-1
2	02	0000 0010		12-9-2		12-9-2
3	03	0000 0011		12-9-3		12-9-3
4	04	0000 0100		12-9-4		9-7
5	05	0000 0101		12-9-5		0-9-8-5
6	06	0000 0110		12-9-6		0-9-8-6
7	07	0000 0111		12-9-7		0-9-8-7
8	08	0000 1000		12-9-8		11-9-6
9	09	0000 1001		12-9-8-1		12-9-5
10	0A	0000 1010		12-9-8-2		0-9-5
11	0B	0000 1011		12-9-8-3		12-9-8-3
12	0C	0000 1100		12-9-8-4		12-9-8-4
13	0D	0000 1101		12-9-8-5		12-9-8-5
14	0E	0000 1110		12-9-8-6		12-9-8-6
15	0F	0000 1111		12-9-8-7		12-9-8-7
16	10	0001 0000		12-11-9-8-1		12-11-9-8-1
17	11	0001 0001		11-9-1		11-9-1
18	12	0001 0010		11-9-2		11-9-2
19	13	0001 0011		11-9-3		11-9-3
20	14	0001 0100		11-9-4		9-8-4
21	15	0001 0101		11-9-5		9-8-5
22	16	0001 0110		11-9-6		9-2
23	17	0001 0111		11-9-7		0-9-6
24	18	0001 1000		11-9-8		11-9-8
25	19	0001 1001		11-9-8-1		11-9-8-1
26	1A	0001 1010		11-9-8-2		9-8-7
27	1B	0001 1011		11-9-8-3		0-9-7
28	1C	0001 1100		11-9-8-4		11-9-8-4
29	1D	0001 1101		11-9-8-5		11-9-8-5
30	1E	0001 1110		11-9-8-6		11-9-8-6
31	1F	0001 1111		11-9-8-7		11-9-8-7
32	20	0010 0000		11-0-9-8-1	SP	No punches
33	21	0010 0001		0-9-1	!	12-8-7
34	22	0010 0010		0-9-2	"	8-7
35	23	0010 0011		0-9-3	#	8-3
36	24	0010 0100		0-9-4	$	11-8-3
37	25	0010 0101		0-9-5	%	0-8-4
38	26	0010 0110		0-9-6	&	12
39	27	0010 0111		0-9-7	'	8-5
40	28	0010 1000		0-9-8	(	12-8-5
41	29	0010 1001		0-9-8-1	)	11-8-5
42	2A	0010 1010		0-9-8-2	*	11-8-4
43	2B	0010 1011		0-9-8-3	+	12-8-6
44	2C	0010 1100		0-9-8-4	,	0-8-3
45	2D	0010 1101		0-9-8-5	–	11
46	2E	0010 1110		0-9-8-6	.	12-8-3
47	2F	0010 1111		0-9-8-7	/	0-1
48	30	0011 0000		12-11-0-9-8-1	0	0
49	31	0011 0001		9-1	1	1
50	32	0011 0010		9-2	2	2
51	33	0011 0011		9-3	3	3
52	34	0011 0100		9-4	4	4
53	35	0011 0101		9-5	5	5
54	36	0011 0110		9-6	6	6

Table B—1. Cross-Reference Table: EBCDIC/ASCII/Hollerith (Part 2 of 5)

EBCDIC					ASCII	
Decimal	Hexa-decimal	Binary	EBCDIC Graphic Character	Hollerith Punched Card Code	ASCII Graphic Character	Hollerith Punched Card Code
55	37	0011 0111		9-7	7	7
56	38	0011 1000		9-8	8	8
57	39	0011 1001		9-8-1	9	9
58	3A	0011 1010		9-8-2	:	8-2
59	3B	0011 1011		9-8-3	;	11-8-6
60	3C	0011 1100		9-8-4	<	12-8-4
61	3D	0011 1101		9-8-5	=	8-6
62	3E	0011 1110		9-8-6	>	0-8-6
63	3F	0011 1111		9-8-7	?	0-8-7
64	40	0100 0000	SP	No punches	@	8-4
65	41	0100 0001		12-0-9-1	A	12-1
66	42	0100 0010		12-0-9-2	B	12-2
67	43	0100 0011		12-0-9-3	C	12-3
68	44	0100 0100		12-0-9-4	D	12-4
69	45	0100 0101		12-0-9-5	E	12-5
70	46	0100 0110		12-0-9-6	F	12-6
71	47	0100 0111		12-0-9-7	G	12-7
72	48	0100 1000		12-0-9-8	H	12-8
73	49	0100 1001		12-8-1	I	12-9
74	4A	0100 1010	[	12-8-2	J	11-1
75	4B	0100 1011	.	12-8-3	K	11-2
76	4C	0100 1100	<	12-8-4	L	11-3
77	4D	0100 1101	(	12-8-5	M	11-4
78	4E	0100 1110	+	12-8-6	N	11-5
79	4F	0100 1111	!	12-8-7	O	11-6
80	50	0101 0000	&	12	P	11-7
81	51	0101 0001		12-11-9-1	Q	11-8
82	52	0101 0010		12-11-9-2	R	11-9
83	53	0101 0011		12-11-9-3	S	0-2
84	54	0101 0100		12-11-9-4	T	0-3
85	55	0101 0101		12-11-9-5	U	0-4
86	56	0101 0110		12-11-9-6	V	0-5
87	57	0101 0111		12-11-9-7	W	0-6
88	58	0101 1000		12-11-9-8	X	0-7
89	59	0101 1001		11-8-1	Y	0-8
90	5A	0101 1010	]	11-8-2	Z	0-9
91	5B	0101 1011	$	11-8-3	[	12-8-2
92	5C	0101 1100	*	11-8-4	\	0-8-2
93	5D	0101 1101	)	11-8-5	]	11-8-2
94	5E	0101 1110	;	11-8-6	∧	11-8-7
95	5F	0101 1111	∧	11-8-7	—	0-8-5
96	60	0110 0000	–	11	`	8-1
97	61	0110 0001	/	0-1	a	12-0-1
98	62	0110 0010		11-0-9-2	b	12-0-2
99	63	0110 0011		11-0-9-3	c	12-0-3
100	64	0110 0100		11-0-9-4	d	12-0-4
101	65	0110 0101		11-0-9-5	e	12-0-5
102	66	0110 0110		11-0-9-6	f	12-0-6
103	67	0110 0111		11-0-9-7	g	12-0-7
104	68	0110 1000		11-0-9-8	h	12-0-8
105	69	0110 1001		0-8-1	i	12-0-9
106	6A	0110 1010	¦	12-11	j	12-11-1
107	6B	0110 1011	,	0-8-3	k	12-11-2
108	6C	0110 1100	%	0-8-4	l	12-11-3
109	6D	0110 1101	—	0-8-5	m	12-11-4

Table B—1. Cross-Reference Table: EBCDIC/ASCII/Hollerith (Part 3 of 5)

EBCDIC					ASCII	
Decimal	Hexadecimal	Binary	EBCDIC Graphic Character	Hollerith Punched Card Code	ASCII Graphic Character	Hollerith Punched Card Code
110	6E	0110 1110	>	0-8-6	n	12-11-5
111	6F	0110 1111	?	0-8-7	o	12-11-6
112	70	0111 0000		12-11-0	p	12-11-7
113	71	0111 0001		12-11-0-9-1	q	12-11-8
114	72	0111 0010		12-11-0-9-2	r	12-11-9
115	73	0111 0011		12-11-0-9-3	s	11-0-2
116	74	0111 0100		12-11-0-9-4	t	11-0-3
117	75	0111 0101		12-11-0-9-5	u	11-0-4
118	76	0111 0110		12-11-0-9-6	v	11-0-5
119	77	0111 0111		12-11-0-9-7	w	11-0-6
120	78	0111 1000		12-11-0-9-8	x	11-0-7
121	79	0111 1001	`	8-1	y	11-0-8
122	7A	0111 1010	:	8-2	z	11-0-9
123	7B	0111 1011	#	8-3	{	12-0
124	7C	0111 1100	@	8-4	¦	12-11
125	7D	0111 1101	'	8-5	}	11-0
126	7E	0111 1110	=	8-6	~	11-0-1
127	7F	0111 1111	"	8-7		12-9-7
128	80	1000 0000		12-0-8-1		11-0-9-8-1
129	81	1000 0001	a	12-0-1		0-9-1
130	82	1000 0010	b	12-0-2		0-9-2
131	83	1000 0011	c	12-0-3		0-9-3
132	84	1000 0100	d	12-0-4		0-9-4
133	85	1000 0101	e	12-0-5		11-9-5
134	86	1000 0110	f	12-0-6		12-9-6
135	87	1000 0111	g	12-0-7		11-9-7
136	88	1000 1000	h	12-0-8		0-9-8
137	89	1000 1001	i	12-0-9		0-9-8-1
138	8A	1000 1010		12-0-8-2		0-9-8-2
139	8B	1000 1011		12-0-8-3		0-9-8-3
140	8C	1000 1100		12-0-8-4		0-9-8-4
141	8D	1000 1101		12-0-8-5		12-9-8-1
142	8E	1000 1110		12-0-8-6		12-9-8-2
143	8F	1000 1111		12-0-8-7		11-9-8-3
144	90	1001 0000		12-11-8-1		12-11-0-9-8-1
145	91	1001 0001	j	12-11-1		9-1
146	92	1001 0010	k	12-11-2		11-9-8-2
147	93	1001 0011	l	12-11-3		9-3
148	94	1001 0100	m	12-11-4		9-4
149	95	1001 0101	n	12-11-5		9-5
150	96	1001 0110	o	12-11-6		9-6
151	97	1001 0111	p	12-11-7		12-9-8
152	98	1001 1000	q	12-11-8		9-8
153	99	1001 1001	r	12-11-9		9-8-1
154	9A	1001 1010		12-11-8-2		9-8-2
155	9B	1001 1011		12-11-8-3		9-8-3
156	9C	1001 1100		12-11-8-4		12-9-4
157	9D	1001 1101		12-11-8-5		11-9-4
158	9E	1001 1110		12-11-8-6		9-8-6
159	9F	1001 1111		12-11-8-7		11-0-9-1

Table B—1. Cross-Reference Table: EBCDIC/ASCII/Hollerith (Part 4 of 5)

EBCDIC					ASCII	
Decimal	Hexa-deci-mal	Binary	EBCDIC Graphic Character	Hollerith Punched Card Code	ASCII Graphic Character	Hollerith Punched Card Code
160	A0	1010 0000		11-0-8-1		12-0-9-1
161	A1	1010 0001	~	11-0-1		12-0-9-2
162	A2	1010 0010	s	11-0-2		12-0-9-3
163	A3	1010 0011	t	11-0-3		12-0-9-4
164	A4	1010 0100	u	11-0-4		12-0-9-5
165	A5	1010 0101	v	11-0-5		12-0-9-6
166	A6	1010 0110	w	11-0-6		12-0-9-7
167	A7	1010 0111	x	11-0-7		12-0-9-8
168	A8	1010 1000	y	11-0-8		12-8-1
169	A9	1010 1001	z	11-0-9		12-11-9-1
170	AA	1010 1010		11-0-8-2		12-11-9-2
171	AB	1010 1011		11-0-8-3		12-11-9-3
172	AC	1010 1100		11-0-8-4		12-11-9-4
173	AD	1010 1101		11-0-8-5		12-11-9-5
174	AE	1010 1110		11-0-8-6		12-11-9-6
175	AF	1010 1111		11-0-8-7		12-11-9-7
176	B0	1011 0000		12-11-0-8-1		12-11-9-8
177	B1	1011 0001		12-11-0-1		11-8-1
178	B2	1011 0010		12-11-0-2		11-0-9-2
179	B3	1011 0011		12-11-0-3		11-0-9-3
180	B4	1011 0100		12-11-0-4		11-0-9-4
181	B5	1011 0101		12-11-0-5		11-0-9-5
182	B6	1011 0110		12-11-0-6		11-0-9-6
183	B7	1011 0111		12-11-0-7		11-0-9-7
184	B8	1011 1000		12-11-0-8		11-0-9-8
185	B9	1011 1001		12-11-0-9		0-8-1
186	BA	1011 1010		12-11-0-8-2		12-11-0
187	BB	1011 1011		12-11-0-8-3		12-11-0-9-1
188	BC	1011 1100		12-11-0-8-4		12-11-0-9-2
189	BD	1011 1101		12-11-0-8-5		12-11-0-9-3
190	BE	1011 1110		12-11-0-8-6		12-11-0-9-4
191	BF	1011 1111		12-11-0-8-7		12-11-0-9-5
192	C0	1100 0000	{	12-0		12-11-0-9-6
193	C1	1100 0001	A	12-1		12-11-0-9-7
194	C2	1100 0010	B	12-2		12-11-0-9-8
195	C3	1100 0011	C	12-3		12-0-8-1
196	C4	1100 0100	D	12-4		12-0-8-2
197	C5	1100 0101	E	12-5		12-0-8-3
198	C6	1100 0110	F	12-6		12-0-8-4
199	C7	1100 0111	G	12-7		12-0-8-5
200	C8	1100 1000	H	12-8		12-0-8-6
201	C9	1100 1001	I	12-9		12-0-8-7
202	CA	1100 1010		12-0-9-8-2		12-11-8-1
203	CB	1100 1011		12-0-9-8-3		12-11-8-2
204	CC	1100 1100		12-0-9-8-4		12-11-8-3
205	CD	1100 1101		12-0-9-8-5		12-11-8-4
206	CE	1100 1110		12-0-9-8-6		12-11-8-5
207	CF	1100 1111		12-0-9-8-7		12-11-8-6
208	D0	1101 0000	}	11-0		12-11-8-7
209	D1	1101 0001	J	11-1		11-0-8-1

Table B—1. Cross-Reference Table: EBCDIC/ASCII/Hollerith (Part 5 of 5)

EBCDIC					ASCII	
Decimal	**Hexa-decimal**	**Binary**	**EBCDIC Graphic Character**	**Hollerith Punched Card Code**	**ASCII Graphic Character**	**Hollerith Punched Card Code**
210	D2	1101 0010	K	11-2		11-0-8-2
211	D3	1101 0011	L	11-3		11-0-8-3
212	D4	1101 0100	M	11-4		11-0-8-4
213	D5	1101 0101	N	11-5		11-0-8-5
214	D6	1101 0110	O	11-6		11-0-8-6
215	D7	1101 0111	P	11-7		11-0-8-7
216	D8	1101 1000	Q	11-8		12-11-0-8-1
217	D9	1101 1001	R	11-9		12-11-0-1
218	DA	1101 1010		12-11-9-8-2		12-11-0-2
219	DB	1101 1011		12-11-9-8-3		12-11-0-3
220	DC	1101 1100		12-11-9-8-4		12-11-0-4
221	DD	1101 1101		12-11-9-8-5		12-11-0-5
222	DE	1101 1110		12-11-9-8-6		12-11-0-6
223	DF	1101 1111		12-11-9-8-7		12-11-0-7
224	E0	1110 0000	\	0-8-2		12-11-0-8
225	E1	1110 0001		11-0-9-1		12-11-0-9
226	E2	1110 0010	S	0-2		12-11-0-8-2
227	E3	1110 0011	T	0-3		12-11-0-8-3
228	E4	1110 0100	U	0-4		12-11-0-8-4
229	E5	1110 0101	V	0-5		12-11-0-8-5
230	E6	1110 0110	W	0-6		12-11-0-8-6
231	E7	1110 0111	X	0-7		12-11-0-8-7
232	E8	1110 1000	Y	0-8		12-0-9-8-2
233	E9	1110 1001	Z	0-9		12-0-9-8-3
234	EA	1110 1010		11-0-9-8-2		12-0-9-8-4
235	EB	1110 1011		11-0-9-8-3		12-0-9-8-5
236	EC	1110 1100		11-0-9-8-4		12-0-9-8-6
237	ED	1110 1101		11-0-9-8-5		12-0-9-8-7
238	EE	1110 1110		11-0-9-8-6		12-11-9-8-2
239	EF	1110 1111		11-0-9-8-7		12-11-9-8-3
240	F0	1111 0000	0	0		12-11-9-8-4
241	F1	1111 0001	1	1		12-11-9-8-5
242	F2	1111 0010	2	2		12-11-9-8-6
243	F3	1111 0011	3	3		12-11-9-8-7
244	F4	1111 0100	4	4		11-0-9-8-2
245	F5	1111 0101	5	5		11-0-9-8-3
246	F6	1111 0110	6	6		11-0-9-8-4
247	F7	1111 0111	7	7		11-0-9-8-5
248	F8	1111 1000	8	8		11-0-9-8-6
249	F9	1111 1001	9	9		11-0-9-8-7
250	FA	1111 1010		12-11-0-9-8-2		12-11-0-9-8-2
251	FB	1111 1011		12-11-0-9-8-3		12-11-0-9-8-3
252	FC	1111 1100		12-11-0-9-8-4		12-11-0-9-8-4
253	FD	1111 1101		12-11-0-9-8-5		12-11-0-9-8-5
254	FE	1111 1110		12-11-0-9-8-6		12-11-0-9-8-6
255	FF	1111 1111		12-11-0-9-8-7		12-11-0-9-8-7

B.3. OS/3 COLLATING SEQUENCE FOR EBCDIC GRAPHIC CHARACTERS

The following table shows the OS/3 collating sequence for EBCDIC characters and unsigned decimal data. The collating sequence ranges from low (0000 0000) to high (1111 1111). The bit configurations that do not correspond to symbols (e.g., 0–73, 81–89, etc) are not shown. Some of these correspond to control commands for printers and other devices.

Packed decimal, zoned decimal, fixed-point, and normalized floating-point data is collated algebraically; i.e., each quantity is interpreted as having a sign.

Table B—2. OS/3 Collating Sequence: EBCDIC Graphics (Part 1 of 2)

Collating Sequence	Bit Configuration	Symbol	Meaning
0	0000 0000		
⋮			
64	0010 0000	SP	Space
⋮			
74	0100 1010	[	Opening bracket
75	0100 1011	.	Period, decimal point
76	0100 1100	<	Less than sign
77	0100 1101	(	Left parenthesis
78	0100 1110	+	Plus sign
79	0100 1111	!	Exclamation point
80	0101 0000	&	Ampersand
⋮			
90	0101 1010	]	Closing bracket
91	0101 1011	$	Dollar sign
92	0101 1100	*	Asterisk
93	0101 1101	)	Right parenthesis
94	0101 1110	;	Semicolon
95	0101 1111	¬	Logical NOT
96	0110 0000	-	Minus sign, hyphen
97	0110 0001	/	Slash
⋮			
106	0110 1010	|	Vertical bar
107	0110 1011	,	Comma
108	0110 1100	%	Percent sign
109	0110 1101	–	Underscore
110	0110 1110	>	Greater than sign
111	0110 1111	?	Question mark
⋮			
122	0111 1010	:	Colon
123	0111 1011	#	Number sign
124	0111 1100	@	At sign
125	0111 1101	'	Apostrophe, prime
126	0111 1110	=	Equals sign
127	0111 1111	''	Quotation marks
⋮			
129	1000 0001	a	
130	1000 0010	b	
131	1000 0011	c	
132	1000 0100	d	
133	1000 0101	e	
⋮			
134	1000 0110	f	
135	1000 0111	g	
136	1000 1000	h	
137	1000 1001	i	
⋮			
145	1001 0001	j	
146	1001 0010	k	
147	1001 0011	l	
148	1001 0100	m	

Table B—2. OS/3 Collating Sequence: EBCDIC Graphics (Part 2 of 2)

Collating Sequence	Bit Configuration	Symbol	Meaning
149	1001 0101	n	
150	1001 0110	o	
151	1001 0111	p	
152	1001 1000	q	
153	1001 1001	r	
:			
161	1010 0001	~	Tilde
162	1010 0010	s	
163	1010 0011	t	
164	1010 0100	u	
165	1010 0101	v	
166	1010 0110	w	
167	1010 0111	x	
168	1010 1000	y	
169	1010 1001	z	
:			
192	1100 0000	{	Opening brace
193	1100 0001	A	
194	1100 0010	B	
195	1100 0011	C	
196	1100 0100	D	
197	1100 0101	E	
198	1100 0110	F	
199	1100 0111	G	
200	1100 1000	H	
:			
201	1100 1001	I	
:			
208	1101 0000	}	Closing brace
209	1101 0001	J	
210	1101 0010	K	
211	1101 0011	L	
212	1101 0100	M	
213	1101 0101	N	
214	1101 0110	O	
215	1101 0111	P	
216	1101 1000	Q	
217	1101 1001	R	
:			
224	1110 0000	\	Reverse slant
226	1110 0010	S	
227	1110 0011	T	
228	1110 0100	U	
229	1110 0101	V	
230	1110 0110	W	
231	1110 0111	X	
232	1110 1000	Y	
233	1110 1001	Z	
:			
240	1111 0000	0	
241	1111 0001	1	
242	1111 0010	2	
243	1111 0011	3	
244	1111 0100	4	
245	1111 0101	5	
246	1111 0110	6	
247	1111 0111	7	
248	1111 1000	8	
249	1111 1001	9	

B.4. OS/3 COLLATING SEQUENCE FOR ASCII GRAPHIC CHARACTERS

Table B-3 shows the OS/3 collating sequence for ASCII characters and unsigned decimal data. The collating sequence ranges from low (0000 0000) to high (0111 1111). Bit configurations that do not correspond to symbols are not shown.

Packed decimal, zoned decimal, fixed-point normalized floating-point data, and the signed numeric data formats are collated algebraically; i.e., each quantity is interpreted as having a sign.

Table B—3. OS/3 Collating Sequence: ASCII Graphics (Part 1 of 2)

Collating Sequence	Bit Configuration	Symbol	Meaning
0	0000 0000		Null
32	0010 0000	SP	Space
33	0010 0001	!	Exclamation mark
34	0010 0010	"	Quotation mark
35	0010 0011	#	Number sign
36	0010 0100	$	Dollar sign
37	0010 0101	%	Percent sign
38	0010 0110	&	Ampersand
39	0010 0111	'	Apostrophe, prime
40	0010 1000	(	Opening parenthesis
41	0010 1001	)	Closing parenthesis
42	0010 1010	*	Asterisk
43	0010 1011	+	Plus sign
44	0010 1100	,	Comma
45	0010 1101	-	Hyphen, minus sign
46	0010 1110	.	Period, decimal point
47	0010 1111	/	Slant
48	0011 0000	0	
49	0011 0001	1	
50	0011 0010	2	
51	0011 0011	3	
52	0011 0100	4	
53	0011 0101	5	
54	0011 0110	6	
55	0011 0111	7	
56	0011 1000	8	
57	0011 1001	9	
58	0011 1010	:	Colon
59	0011 1011	;	Semicolon
60	0011 1100	<	Less than sign
61	0011 1101	=	Equals sign
62	0011 1110	>	Greater than sign
63	0011 1111	?	Question mark
64	0100 0000	@	Commercial at sign
65	0100 0001	A	
66	0100 0010	B	
67	0100 0011	C	
68	0100 0100	D	
69	0100 0101	E	
70	0100 0110	F	
71	0100 0111	G	
72	0100 1000	H	
73	0100 1001	I	
74	0100 1010	J	
75	0100 1011	K	
76	0100 1100	L	
77	0100 1101	M	

Table B—3. OS/3 Collating Sequence: ASCII Graphics (Part 2 of 2)

Collating Sequence	Bit Configuration	Symbol	Meaning
78	0100 1110	N	
79	0100 1111	O	
80	0101 0000	P	
81	0101 0001	Q	
82	0101 0010	R	
83	0101 0011	S	
84	0101 0100	T	
85	0101 0101	U	
86	0101 0110	V	
87	0101 0111	W	
88	0101 1000	X	
89	0101 1001	Y	
90	0101 1010	Z	
91	0101 1011	[	Opening bracket
92	0101 1100	\	Reverse slant
93	0101 1101	]	Closing bracket
94	0101 1110	∧	Circumflex
95	0101 1111	_	Underscore
96	0110 0000	`	Grave accent
97	0110 0001	a	
98	0110 0010	b	
99	0110 0011	c	
100	0110 0100	d	
101	0110 0101	e	
102	0110 0110	f	
103	0110 0111	g	
104	0110 1000	h	
105	0110 1001	i	
106	0110 1010	j	
107	0110 1011	k	
108	0110 1100	l	
109	0110 1101	m	
110	0110 1110	n	
111	0110 1111	o	
112	0111 0000	p	
113	0111 0001	q	
114	0111 0010	r	
115	0111 0011	s	
116	0111 0100	t	
117	0111 0101	u	
118	0111 0110	v	
119	0111 0111	w	
120	0111 1000	x	
121	0111 1001	y	
122	0111 1010	z	
123	0111 1011	{	Opening brace
124	0111 1100	|	Vertical line
125	0111 1101	}	Closing brace
126	0111 1110	~	Tilde

Appendix C. Control Statement Summary

C.1. GENERAL

This appendix summarizes the sort/merge control statements (in alphabetical order) and is provided for quick reference only. Section 3 describes the sort/merge control statements in more detail.

C.2. END

Function:

Notifies sort/merge that all sort/merge control statements have been processed and that program execution may take place.

Format:

LABEL	△OPERATION△	OPERAND
	END	

The END statement is an optional sort/merge control statement. It is not to be used when sort/merge specifications are embedded in a jproc. Otherwise, the run processor will mistakenly interpret the END statement as the END directive for the jproc.

C.3. INPFIL

Function:

Defines the input files to sort/merge and specifies the procedures to be followed when input tape files are opened and closed. This statement is not required if disk input files are the source of data. However, if the RECORD control statement is used, the INPFIL statement must also be included.

Format:

```
LABEL          △OPERATION△          OPERAND

               INPFIL               ┌BLKSIZE=┌bytes                       ┐┐
                                    └        └(bytes-1[,...,bytes-8])    ┘┘
                                    [,BUFOFF=n]
                                    [,BYPASS]
                                    ┌,CLOSE=┌NORWD┐┐
                                    │       │RWD  ││
                                    │       │RWI  ││
                                    └       └UNLD ┘┘
                                    ┌,DATA=┌A┐┐
                                    └      └E┘┘
                                    [,EXIT]
                                    ┌,OPEN=┌NORWD┐┐
                                    └      └RWD  ┘┘
                                    [,SKIPBYTE=n]
                                    ┌,VOLUME=┌vol                     ┐┐  See note.
                                    └        └vol-1[,...,vol-8])      ┘┘
```

NOTE:

VOLUME is provided and accepted for compatibility with other systems; no action is performed by OS/3 sort/merge.

Keyword Parameters:

BLKSIZE=bytes

Defines the block size of the input files. For a sort procedure, specify the length of the largest input block. For a merge-only operation, use this format only if all input files have the same block size.

BLKSIZE=(bytes-1[,...,bytes-8])

This format is used for a merge-only operation when multiple input files have different block sizes.

If the BLKSIZE keyword parameter and the RCSZ parameter of the RECORD control statement are both omitted, the size of the first block processed is assumed to be the size of all input blocks.

BUFOFF=n

A decimal number of 0 to 99 defining the length of a block prefix for an ASCII data block structure.

BYPASS

Directs the sort/merge input phase to ignore or bypass all unreadable blocks of data on the input file. Sort/merge maintains no record of the errors encountered.

CLOSE=NORWD

Specifies that input tape files are not to be rewound on closing.

CLOSE=RWD

Specifies that input tape files are to be rewound to load point on closing.

CLOSE=RWI or CLOSE=UNLD

Specifies that input tape files are to be rewound with interlock on closing.

DATA=A

Specifies that the data is recorded in ASCII.

DATA=E

Specifies that the data is recorded in EBCDIC.

EXIT

Required when the user provides his own input routine for reading the input file. No other parameters are to be specified when the EXIT keyword parameter is coded.

OPEN=NORWD

Specifies that input tape files are not to be rewound to load point on opening.

OPEN=RWD

Specifies that input tape files are to be rewound to load point on opening.

SKIPBYTE=n

Indicates the location of the first data record in relation to the beginning of the block. The *n* is the number of bytes preceding the first data record.

C.4. MERGE

Function:

Defines a merge-only operation. This statement is used in place of the SORT statement for merging files that have been previously sequenced. It defines the key fields, their formats, and the number of input files involved.

Format:

LABEL	△OPERATION△	OPERAND
	MERGE	[FIELDS={([strt-pos-1][,lgth-1][,form-1] [seq-1][,...,strt-pos-n,lgth-n [,form-n][,seq-n]]) ([strt-pos-1][,lgth-1][,seq-1] [,...,strt-pos-n,lgth-n [,seq-n]]),FORMAT=code}] [,{FILES ORDER}={number 16}] [,MERGEP=output-file-number, input-file-number] See note.

NOTE:

The MERGEP parameter is provided and accepted for compatibility with other systems; no action is performed.

Keyword Parameters:

```
FIELDS={([strt-pos-1][,lgth-1][,form-1][,seq-1]
          [,...,strt-pos-n,lgth-n[,form-n][,seq-n]])
        ([strt-pos-1][,lgth-1][,seq-1][,...,strt-pos-n,lgth-n][,seq-n]]),
          FORMAT=code                                                      }
```

Defines the merge key fields. The data may vary for each key field, or it may be the same for all key fields. A maximum of 12 fields may be specified. If omitted, one key field is assumed, beginning in position 1, the same length as the record up to 256 bytes, with character format. Ascending sequence is assumed.

If any of the subparameters are omitted, their associated commas must be retained, except for trailing commas.

Positional Subparameters:

strt-pos-n

A decimal number specifying the starting point of a key field relative to the beginning of the record.

Key fields are numbered consecutively, starting with 1 for the most significant key field, 2 for the next, and so on. All key fields, with the exception of binary key fields, must start on a full byte boundary. The starting point is defined by specifying the number of that byte relative to the beginning of the record. For example, subparameter *strt-pos-1* specified as 9 indicates that the most significant key field begins at byte 9 of the record.

Binary key fields are permitted to start on a bit boundary. In this case, the *strt-pos-n* subparameter is specified in a byte.bit format. As an example, assume that key field 1 starts at bit 2 of byte 9 of the record. The *strt-pos-1* subparameter is specified as 9.2.

lgth-n

A decimal number specifying the length of a key field. Key field lengths are specified in full bytes with the exception of binary key fields, which can begin on a bit boundary. Key field lengths expressed in full bytes are defined by whole numbers written in any of the following formats:

n
n.
n.0

Binary key fields starting on a bit boundary require a byte.bit format for defining key field length. The number of bits specified must not exceed 7. For example, a key field length of 6 bits would be written as 0.6. If the key field extends from bit 2 of byte 10 through bit 5 of byte 12, the *lgth-n* subparameter would be specified as 2.4.

form-n

A code consisting of two or three alphabetic characters specifying the data format of the key field. This subparameter is used when the data format varies for each key field. If this optional subparameter is not specified, the format is assumed to be character (CH). If all key fields have the same format, the FORMAT=code subparameter can be used. The format codes and their maximum allowable field lengths are:

Format Code	Description	Allowable Field Length
AC	EBCDIC data in ASCII collating sequence	1 - 256 bytes
ASL	ASCII numeric data leading sign	2 - 256 bytes
AST	ASCII numeric data trailing sign	2 - 256 bytes
BI	Unsigned binary	1 bit to 256 bytes
CH	Character (EBCDIC or (ASCII)	1 - 256 bytes
CLO	Numeric data overpunched leading sign	1 - 256 bytes
CSL	Leading sign numeric	2 - 256 bytes
CST	Trailing sign numeric	2 - 256 bytes
CTO	Numeric data overpunched trailing sign	1 - 256 bytes

Format Code	Description	Allowable Field Length
FI	Fixed-point integer	1 - 256 bytes
FL	Floating-point	1 - 256 bytes
MC	Multiple character, user-specified collating sequence	1 - 256 bytes
PD	Packed decimal	1 - 32 bytes
USQ	Character, user-specified collating sequence	1 - 256 bytes
ZD	Zoned decimal	1 - 32 bytes

seq-n

An alphabetic character specifying the sorting sequence of the key field, A for ascending order and D for descending order. If omitted, ascending order (A) is assumed. In a merge-only application, the output file must be sequenced in the same order as the input file.

FORMAT=code

A code consisting of two or three alphabetic characters specifying the data format of the key fields. This subparameter is used when the data format for all key fields is the same. The code specifications for this subparameter are the same as those listed for the *form-n* subparameter. The *form-n* subparameter must not be specified when FORMAT=code is used.

FILES=number or ORDER=number

The FILES and ORDER keywords may be used interchangeably. Specifies the number of input files that are to be merged. Independent sort/merge can merge data from 2 to 16 previously sequenced input files. Input files are defined by the LFD job control statement and are assigned the names SORTIN1 through SORTIN9 for the first nine input files and SORTINA through SORTING for the last seven input files.

If FILES and ORDER are both omitted, sort/merge assumes that 16 files are being merged.

C.5. MODS

Function:

Defines the user own-code routines that are to be included in the sort/merge operation. Also causes sort/merge to load and execute the system-supplied DELETE routine to perform automatic data reduction.

Format 1:

LABEL	△OPERATION△	OPERAND
	MODS	PHn=(module-name,[length],exit-code [,...,exit-code])[,...,PHn=(module-name, [length],exit-code[,...,exit-code])]

Keyword Parameters:

`PHn=(module-name,[length],exit-code[,...,exit-code])`

Identifies a load module containing one or more user own-code routines to be accessed during a specific operational phase or during all operational phases.

This keyword parameter must be repeated for each phase requiring user own-code exits and for those routines which are accessed during all phases. Multiple calls of the keyword are separated by commas, with a continuation character coded in column 72, if necessary.

`PHn`

Specifies the sort/merge phase in which the user own-code exit routine is to be executed or an identifying code for an exit routine which is to be accessed from all operational phases. The values for *n* are:

n	Description
1	Phase 1 (input internal sort). Exit codes are E11, E15, and E18.
3	Phase 3 (final merge-output). Exit codes are E31, E32, E35, E38, and E39.
6	All phases (record sequencing routine). Exit code is E65.
7	All phases (data reduction routine). Exit code is E75.
8	All phases (user-defined collation sequencing). Exit code is E84.

Positional Subparameters:

`module-name`

Specifies the name of the load module for the user own-code routine. The module name may be one to eight characters in length.

length

A decimal number specifying the length (in bytes) of the user own-code load module.

If omitted, the length is obtained from the load module header record.

exit-code

Specifies the program modification exit code to be used; for example, E11, E35, etc. All exit codes applicable to a particular phase must be specified as subparameters to that phase. Exit codes are listed in Table 3–2.

Format 2:

LABEL	△OPERATION△	OPERAND
	MODS	PH7=(DELETE,,E75)

Keyword Parameters:

PH7=(DELETE,,E75)

Identifies a system-supplied load module that contains an automatic data reduction routine to delete record duplication from your files.

PH7

Specifies the identifying code that allows the load module to be accessed from all operational phases of the sort program.

Positional Subparameters:

DELETE

Specifies the name of the load module.

E75

Specifies the exit code that the sort/merge uses to exit from the sort program and pass control to the DELETE routine.

C.6. OPTION

Function:

Specifies sort/merge options that do not apply to any of the other control statements. These are tag sort specification, key length specifications for direct access input records, label specifications for input, output, and work files, working-storage specifications, error message printing options, restart specification, output block verification for direct access devices, and calculation of the working-storage requirements by the sort/merge.

Format:

```
LABEL      △OPERATION△    OPERAND
                          ┌ADDROUT=┌A┐┐
           OPTION         └        └D┘┘

                          ┌,┌CALCAREA           ┐┐
                          │ │CALCAREA=┌NO  ┐    ││
                          └ └         └YES ┘    ┘┘

                          ┌,CSPRAM=┌NO  ┐┐
                          └        └YES ┘┘

                          [,KEYLEN=bytes]

                          [,LABEL=(output,input-1[,...,input-n],work)]

                          ┌,PRINT=┌ALL      ┐┐
                          │       │CRITICAL ││
                          └       └NONE     ┘┘

                          ┌,RESERV=┌work-file-name                          ┐┐
                          └        └(work-file-name[,output-file-name])     ┘┘

                          [,RESTART]

                          ┌,SHARE=┌work-file-name                         ┐┐
                          └       └(work-file-name[,input-file-name])     ┘┘

                          [,STORAGE=bytes]

                          [,VERIFY]

                          [,ALTWK]   ┐
                          [,DUMP]    │
                          [,ERASE]   │
                          [,ROUTE]   ├  See note.
                          [,SORTIN]  │
                          [,SORTOUT] │
                          [,SORTWK]  ┘
```

NOTE:

The ALTWK, DUMP, ERASE, ROUTE, SORTIN, SORTOUT, and SORTWK parameters are provided and accepted for compatibility with other systems; however, no action is performed by OS/3 sort/merge.

Keyword Parameters:

```
ADDROUT=┌A┐
        └D┘
```

Specifies that a tag sort is to be performed. If the user is providing the input through an own-code routine, each record must be preceded by a 10-byte field containing its direct access address. A tag sort may be specified only when input is from nonindexed disk files.

ADDROUT=A

Specifies that only the direct access addresses of the input records are to appear in the output file.

ADDROUT=D

Specifies that both the direct access addresses and the sort key fields of each record are to comprise the final output.

The following restrictions apply when ADDROUT is used:

1. Output block size must be a multiple of:

   a. 10 bytes for ADDROUT=A

   b. The sum of the sort key field lengths plus 10 bytes for ADDROUT=D

2. The *lgth-2* and *lgth-3* values in the LENGTH keyword of the RECORD control statement must be used. The *lgth-2* value must be 10 bytes plus the sum of the sort key field lengths. The *lgth-3* value must be:

   a. 10 bytes for ADDROUT=A

   b. 10 bytes plus the sum of the sort key field lengths (after any user modification at exit E35) for ADDROUT=D

CALCAREA or CALCAREA=NO

Specifies that sort/merge is to calculate the optimum working-storage area in a disk sort, display the estimated sort time in minutes and the number of cylinders required for work space, and then terminate the job step.

CALCAREA=YES

Specifies that sort/merge is to calculate optimum working-storage area, display information, and then execute the sort.

If the CALCAREA parameter is used, the SIZE parameter on the SORT control statement should be specified; otherwise, the default value of 25,000 records will be used in calculating the working-storage area.

CSPRAM=YES

Specifies that sort/merge parameters may be accepted from the job control stream at run time through PARAM control statements. The keyword parameters that can be entered through the control stream are BIN, DISC, NOCKSM, RESERV, RESUME, SHARE, and TAPE.

If CSPRAM is omitted, sort/merge parameters will not be accepted from the control stream.

KEYLEN=bytes

Required by data management when direct access input blocks have leading keys. Defines the length of the key field.

LABEL=(output,input-1[,...,input-n],work)

Indicates the type of labels for output, and work files: S for standard, N for nonstandard, or U for unlabeled tapes. Files with nonstandard labels must be opened and closed by the user via user exits E11 and E31.

If omitted, sort/merge assumes standard output, input, and work file labels.

PRINT=ALL

Specifies that all error messages are to be written to the job log for subsequent printing.

PRINT=CRITICAL

Specifies that ony fatal error messages are to be written to the job log.

PRINT=NONE

Specifies that no error messages are to be written to the job log. The sort control statements will always be written, however.

```
RESERV={work-file-name                               }
       {(work-file-name[,output-file-name])}
```

Allows a tape unit to function as a work file device during the input and intermediate phases of sort/merge operation and as the device for the output data file during the output phase. Messages instructing the operator when to unload the scratch tape and mount the output tape are displayed on the system console. The same device cannot be used for both RESERV and SHARE.

Positional Subparameters:

work-file-name

Identifies the reserved tape unit by a standard sort work file name (SMO1,...,SMO6). The tape unit is associated with this file name by an LFD statement in the job control stream.

output-file-name

Identifies the reserved tape unit by the standard output file name SORTOUT or, if the user is supplying his own output routine, by the user-specified file name assigned on the LFD job control statement. If this subparameter is coded, the information displayed on the console will include the name of the output file the operator is to mount.

RESTART

Required when an interrupted tape sort is to be restarted at the last check point. Restart information identifying each check point is displayed on the system console. To restart an interrupted sort/merge, the control stream must be updated and resubmitted. The updated control stream must include the PARAM job control statement containing the proper specification for the RESUME keyword parameter.

```
SHARE={work-file-name                          }
      {(work-file-name[,input-file-name])}
```

Allows a tape unit assigned to sort/merge as an input device to be used for a sort work file during the intermediate and ouput phases. Messages instructing the operator when to unload the input tape and mount a scratch tape are displayed on the system console. The same device cannot be used for RESERV and SHARE.

Positional Subparameters:

work-file-name

Identifies the shared tape unit by a standard sort work file name (SMO1,...,SMO6). The tape unit is asociated with this file name by an LFD statement in the job control stream.

input-file-name

Identifies the shared tape unit by the standard input file name SORTINn or, if the user is supplying his own input routine, by a user-specified file name assigned on an LFD job control statement. If this subparameter is coded, the information displayed on the system console will include the name of the input file the operator is to demount.

STORAGE=bytes

Specifies the number of bytes of main storage allocated to sort/merge. If this parameter is omitted, sort/merge obtains this information from job control.

VERIFY

Specifies that each output block is to be checked to ensure that it is written correctly when the output file is on a direct access device.

C.7. OUTFIL

Function:

Defines the output file to sort/merge and specifies the procedures for opening and closing output tapes. The OUTFIL control statement is not required if:

- both input and output files are on disk;
- the output file is to have the same block size and record size as the input file; and
- the output file is a single-partition file or a predefined multipartitioned file.

If the output file has been predefined, the first optional parameter of the LFD statement, specifying the maximum number of extents in the file, should be omitted.

If the OUTFIL control statement is used to define a predefined output file, all file specifications must be the same as when the file was created, or an error will result.

Format:

LABEL	△OPERATION△	OPERAND
	OUTFIL	[BLKSIZE=bytes] [,BUFOFF=n] [,CLOSE= {NORWD RWD RWI UNLD}] [,EXIT] [,FILTYPE={IRAM NI SAM}] See note. [,NOTPMK] [,NPTN={number 1}] See note. [,OPEN={NORWD RWD}] [RCSZ=bytes] [SIZE=percentage]} See note. [TYPE=type]} [UOS=ext-percent]

NOTE:

The FILTYPE, NPTN, SIZE, and TYPE parameters are provided and accepted for compatibility with other systems; no action is performed by OS/3 sort/merge.

Keyword Parameters:

BLKSIZE=bytes

Defines the output file block size when the output file is a tape or disk file.

If the BLKSIZE keyword parameter is omitted and the RCSZ parameter is not specified elsewhere, sort/merge assumes a block size equal to the first output block.

BUFOFF=n

A decimal number from 0 to 99 defining the length of a block prefix for an ASCII data block structure.

CLOSE=NORWD

Specifies that the tape output file is not to be rewound on closing.

CLOSE=RWD

Specifies that the tape output file is to be rewound on closing.

CLOSE=RWI or CLOSE=UNLD

Specifies that the tape output file is to be rewound with interlock on closing.

EXIT

Required when a user output routine is provided for writing the output file. No other parameters should be specified when the EXIT parameter is used.

NOTPMK

Specifies that no tape mark is to be written before the first data record on each volume in the tape output file.

OPEN=NORWD

Specifies that the tape output file is not to be rewound to load point on opening.

OPEN=RWD

Specifies that the tape output file is to be rewound to load point on opening.

RCSZ=bytes

Indicates the maximum size of the records written to a disk output file.

If RCSZ is omitted, sort/merge supplies the same number of bytes as the input file record size.

TYPE=type

Defines the type of data written to a specific partition in the output file: D for ASCII variable-length records; F for fixed-length records; V for variable-length records.

If omitted, fixed-length records are assumed.

UOS=ext-percent

Specifies the percentage of secondary storage allocation assigned to a disk output file. Up to 100 percent may be specified.

C.8. RECORD

Function:

Defines the type and length of the data records to be sorted or merged and provides the capability of deleting records from a file by character identification. This statement is required if input is from a tape file. It is not required for disk input files unless files contain variable-length records, length modifications are to be made, or input is by a user exit routine. However, if the INPFIL control statement is included, the RECORD control statement must also be included.

Format:

```
LABEL          △OPERATION△   OPERAND
               RECORD        ⎧LENGTH=(lgth-1[,lgth-2][,lgth-3]⎫
                             ⎨   [,lgth-4][,lgth-5])          ⎬
                             ⎩RCSZ=bytes                      ⎭
                             ⎡,TYPE=⎧D⎫⎤
                             ⎢      ⎨F⎬⎥
                             ⎣      ⎩V⎭⎦
                             ⎡,BIN=⎧bytes                         ⎫⎤
                             ⎢     ⎨(min-bytes,size-1,freq-1      ⎬⎥
                             ⎣     ⎩   [,....,size-n,freq-n])     ⎭⎦
                             [,DEBLANK=(delete-char,byte-position)]
```

Keyword Parameters:

LENGTH=(lgth-1[,lgth-2][,lgth-3][,lgth-4][,lgth-5])

Establishes a sublist that specifies record length information. This sublist can describe either fixed-length or variable-length records for input, internal sort, and output phases of sort/merge operation.

Positional Subparameters:

lgth-1

A decimal number specifying the input record length in bytes for fixed-length records or the maximum input record length for variable records. The length specified must not exceed 32,767 bytes.

lgth-2

A decimal number specifying the length (in bytes) of each record released to the internal sort phase for fixed-length records or the maximum length record for variable-length records.

If omitted, sort/merge assumes *lgth-1* by default.

This subparameter should not be specified for a merge-only operation. However, its associated comma must be retained.

lgth-3

A decimal number specifying the output record length in bytes for fixed-length records or maximum output record length for variable-length records written to tape or single-partition disk output files. Output record lengths written to multipartitioned disk files are specified by use of the RCSZ keyword parameter in the OUTFIL control statement.

If omitted, *lgth-2* is assumed for sort operations, and *lgth-1* is assumed for merge-only operations by default.

lgth-4

A decimal number specifying the minimum input record length in bytes for variable-length records.

If omitted, this information is obtained from the BIN specification.

lgth-5

A decimal number specifying the input record length (in bytes) for the variable-length records that appear most frequently in the input file.

If omitted, this information is obtained from the BIN specification.

RCSZ=bytes

Specifies the record length for fixed-length records or the maximum record size for variable-length records.

If input is from sequential or nonindexed disk files and both of these parameters are omitted along with the BLKSIZE parameter on the INPFIL sort control statement, sort/merge defaults to the input record size supplied by data management.

```
TYPE=⎧D⎫
     ⎨F⎬
     ⎩V⎭
```

Specifies the type of data records to be processed by sort/merge. The specifications provided in this keyword parameter apply only to tape and single-partition disk files. The specifications for data record types contained in multipartitioned disk files are defined in the TYPE keyword parameter of the OUTFIL control statement.

TYPE=D

Specifies that the data records are ASCII format variable-length records.

TYPE=F

Specifies that the data records are fixed length.

TYPE=V

Specifies that the data records are variable length.

```
BIN=⎧bytes                                          ⎫
    ⎩(min-bytes,size-1,freq-1[,...,size-n,freq-n])⎭
```

Specifies the size of fixed-length subrecords (BIN size) when variable-length records are to be sorted, or provides the information from which sort/merge can calculate the subrecord size. The BIN keyword parameter should be coded if the *lgth-4* and *lgth-5* subparameters of the LENGTH keyword are omitted or if RCSZ is used in place of the LENGTH keyword.

BIN=bytes

Specifies the number of bytes into which variable-length records are to be subdivided. The BIN size must be large enough to contain all sort key fields within each record, plus the 4-byte record length field.

`BIN=(min-bytes,size-1,freq-1[,...,size-n,freq-n])`

Positional Subparameters:

`min-bytes`
Specifies the minimum number of bytes into which variable-length records may be subdivided. The number must be large enough to accommodate all sort key fields within each record, plus the 4-byte record length field.

`size-1`
Defines the record length (in bytes) that appears most frequently in the file.

`freq-1`
Specifies either the frequency (percentage) or estimated number of size-1 records in the file. If the number is less than 100, sort/merge assumes that it is a percentage; if greater than 100, it is assumed to be an estimate of the number of records in the file.

`size-n`
Optionally defines additional record lengths (in bytes) that appear frequently in the file. Up to six record lengths may be defined.

`freq-n`
Specifies either the frequency (percentage) or estimated number of size-n records in the file. The sum of the records specified does not have to equal 100 percent of the file.

If the BIN keyword is omitted, BIN size is calculated from *lgth-4* of the LENGTH parameter or from the FIELDS parameter on the SORT statement.

`DEBLANK=(delete-char,byte-position)`
Deletes specific records from the file. The deleted records are identified by defining a specific character contained within a particular byte of the record.

Positional Subparameters:

`delete-char`
The identifying character that, when found in the byte specified by the *byte-position* subparameter, causes the record to be deleted from the file.

`byte-position`
The position of the character (in bytes) relative to the start of the record; for example, 1, indicating the first byte of the record, 2, the second byte, and so on.

C.9. SORT

Function:

Defines the sort key fields and their sorting sequence. It also defines the type and number of auxiliary storage devices to be used, the approximate number of records to be sorted, and the number of input files.

Format:

```
LABEL          △OPERATION△     OPERAND
                SORT            ⌈FIELDS=⎧([strt-pos-1][,lgth-1][,form-1]        ⎫⌉
                                ⎢       ⎪  [,seq-1][,...,strt-pos-n,lgth-n       ⎪⎥
                                ⎢       ⎨  [,form-n][,seq-n]])                    ⎬⎥
                                ⎢       ⎪([strt-pos-1][,lgth-1][,seq-1]          ⎪⎥
                                ⎢       ⎪  [,...,strt-pos-n,                      ⎪⎥
                                ⌊       ⎩  lgth-n[,seq-n]]),FORMAT=code          ⎭⌋

                                ⌈,COPY=⎧ALL                     ⎫⌉  See note.
                                ⎢      ⎨input-file-number,      ⎬⎥
                                ⌊      ⎩  output-file-number    ⎭⌋

                                ⌈,⎧DISC⎫=number⌉
                                ⎢ ⎨TAPE⎬       ⎥
                                ⌊ ⎩WORK⎭       ⌋

                                ⌈,FILE=⎧number⎫⌉
                                ⌊      ⎩1     ⎭⌋

                                ⌈,NOCKSM=⎧D⎫⌉
                                ⌊        ⎩T⎭⌋

                                [,SIZE=number]

                                ⌈,SORTP=output-file-number,⌉  See note.
                                ⌊          input-file-number⌋

                                ⌈,⎧CHPT ⎫⌉  See note.
                                ⌊ ⎩CHKPT⎭⌋
```

NOTE:

The COPY, SORTP, and CHPT/CHKPT parameters are provided and accepted for compatibility with other systems; however, no action will be performed by sort/merge.

Keyword Parameters:

```
FIELDS=⎧([strt-pos-1][,lgth-1][,form-1][,seq-1][,...,strt-pos-n,lgth-n     ⎫
       ⎪  [,form-n][,seq-n]])                                                ⎪
       ⎨([strt-pos-1][,lgth-1][,seq-1][,...,strt-pos-n,lgth-n[,seq-n]]),    ⎬
       ⎩  FORMAT=code                                                        ⎭
```

Defines the sort key fields. The data format may vary for each key field, or it may be the same for all key fields. A maximum of 12 key fields may be specified. If omitted, one key field is assumed, beginning in position 1, the same length as the record up to 256 bytes, with character format. Sorting is in ascending sequence.

If any of the subparameters are omitted, their associated commas must be retained, except for trailing commas.

Positional Subparameters:

strt-pos-n

A decimal number specifying the starting point of a key field relative to the beginning of the record.

Key fields are numbered consecutively, starting with 1 for the most significant key field, 2 for the next, and so on. All key fields, with the exception of binary key fields, must start on a full byte boundary. The starting point is defined by specifying the number of that byte relative to the beginning of the record. For example, subparameter *strt-pos-1* specified as 9 indicates that the most significant key field begins at byte 9 of the record.

Binary key fields are permitted to start on a bit boundary. In this case, the *strt-pos-n* subparameter is specified in a byte.bit format. As an example, assume that key field 1 starts at bit 2 of byte 9 of the record. The *strt-1* subparameter is specified as 9.2.

lgth-n

A decimal number specifying the length of a key field. Key field lengths are specified in full bytes, with the exception of binary key fields, which can begin on a bit boundary. Key field lengths expressed in full bytes are defined by whole numbers written in any of the following formats:

n
n.
n.0

Binary key fields starting on a bit boundary require a byte.bit format for defining key field length. The number of bits specified must not exceed 7. For example, a key field of 6 bits would be written as 0.6. If the key field extends from bit 2 of byte 10 through bit 5 of byte 12, the *lgth-n* subparameter would be specified as 2.4.

form-n

A code consisting of two or three alphabetic characters specifying the data format of the key field. This subparameter is used when the data format varies for each key field. If this optional subparameter is not specified, the format is assumed to be character (CH). If all key fields have the same format, the *FORMAT=code* subparameter can be used. The format codes and their maximum allowable field lengths are:

Format Code	Description	Allowable Field Length
AC	EBCDIC data in ASCII collating sequence	1 - 256 bytes
ASL	ASCII numeric data leading sign	2 - 256 bytes

Format Code	Description	Allowable Field Length
AST	ASCII numeric data trailing sign	2 - 256 bytes
BI	Unsigned binary	1 bit to 256 bytes
CH	Character (EBCDIC or ASCII)	1 - 256 bytes
CLO	Numeric data overpunched leading sign	1 - 256 bytes
CSL	Leading sign numeric	2 - 256 bytes
CST	Trailing sign numeric	2 - 256 bytes
CTO	Numeric data overpunched trailing sign	1 - 256 bytes
FI	Fixed-point integer	1 - 256 bytes
FL	Floating point	1 - 256 bytes
MC	Multiple character, user-specified collating sequence	1 - 256 bytes
PD	Packed decimal	1 - 32 bytes
USQ	Character, user-specified collating sequence	1 - 256 bytes
ZD	Zoned decimal	1 - 32 bytes

seq-n

An alphabetic character specifying the sorting sequence of the key field, A for ascending order and D for descending order. If omitted, ascending order is assumed.

FORMAT=code

A code consisting of two or three alphabetic characters specifying the data format of the key fields. This subparameter is used when the data format for all key fields is the same. The code specifications are the same as those listed for the *form-n* subparameter. The *form-n* subparameter must not be specified when *FORMAT=code* is used.

DISC=number

Indicates the number of disk files available to sort/merge for working storage. Disk files are assigned in LFD job control statements or in WORK jproc calls by means of the standard disk file names DM01,...,DM08 or $SCR1,...,$SCR8.

TAPE=number

Indicates the number of tape files available to sort/merge for working storage. Tape files are assigned in the LFD job control statements by means of the standard sort tape file names SM01,...,SM06.

WORK=number

May alternately be used for specifying the number of disk or tape files available to sort/merge for working storage.

A maximum of eight disk files or six tape files may be assigned for working storage. If the DISC, TAPE, and WORK keyword parameters are omitted, sort/merge determines the number and type of work files assigned from the PUBS list generated by job control when devices are assigned to the job. If no work files are assigned to the job, an internal (main storage) sort is performed.

FILE=number

Indicates the total number of input files to be sorted. The input files must be specified as SORTIN1,...,SORTIN9 in the LFD job control statements, unless a user input routine is provided.

If omitted, one input file is assumed.

NOCKSM=D

Suppresses the calculation of a checksum word for disk work files. The checksum word is normally calculated and written for each output data block, then verified for each input block read to ensure data integrity.

NOCKSM=T

Suppresses the checksum calculation for tape work files.

SIZE=number

Specifies the approximate number of records in the input file. This information is used for optimizing sort performance and for calculating optimum working-storage area when the CALCAREA parameter is specified on the sort/merge OPTION control statement. If SIZE is omitted, a file of 25,000 records is assumed.

Index

CUT

SPERRY

USER COMMENT SHEET

We will use your comments to improve subsequent editions.

NOTE: Please do not use this form as an order blank.

____

(Document Title)

____________	____________	____________
(Document No.)	*(Revision No.)*	*(Update No.)*

Comments:

From:

____

(Name of User)

____

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

CUT

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY CORPORATION

ATTN.: SOFTWARE SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

FOLD

SPERRY

USER COMMENTS

We will use your comments to improve subsequent editions.

NOTE: Please do not use this form as an order blank.

____

(Document Title)

____________	____________	____________
(Document No.)	*(Revision No.)*	*(Update Level)*

Comments:

From:

____

(Name of User)

____

(Business Address)

Fold on dotted lines, and mail. (No postage is necessary if mailed in the U.S.A.)
Thank you for your cooperation

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY CORPORATION

ATTN: SYSTEM PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19422-9990

FOLD

UNISYS

USER COMMENTS

We will use your comments to improve subsequent editions.

NOTE: Please do not use this form as an order blank.

____

(Document Title)

____________________ ____________________ ____________________

(Document No.) *(Revision No.)* *(Update Level)*

Comments:

From:

____

(Name of User)

____

(Business Address)

Fold on dotted lines, and mail. (No postage is necessary if mailed in the U.S.A.)
Thank you for your cooperation

CUT

FOLD

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

Unisys Corporation
E/MSG Product Information Development
PO Box 500 C1-NE6
Blue Bell, PA 19422-9990

FOLD